НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-застосунок для онлайн бібліотеки

Виконав студент IV курсу, групи ІТ-94
(шифр групи)
Кравченко Даніл Олегович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст.викладач Вітковська І.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доц.каф.ІСТ, доц, к.т.н. Писаренко А.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Кравченку Данілу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-застосунок для онлайн бібліотеки
керівник проєкту Ліщук Катерина Ігорівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: опис і аналіз предметної області, аналіз існуючих технологій, успішних ІТ-проектів, алгоритмічних та технічних рішень, аналіз вимог до програмного забезпечення, постановка задач.
2) Моделювання та аналіз програмного забезпечення: архітектура програмного забезпечення, конструювання ПЗ та аналіз безпеки даних.
3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування та опис контрольного прикладу.
4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення та підтримка програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема бази даних _____

3) Схема структурна класів програмного забезпечення _____

4) Креслення вигляду екранних форм _____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	05.04.2023	
3	Постановка та формалізація задачі	05.04.2023	
4	Розробка інформаційного забезпечення	19.04.2023	
5	Алгоритмізація задачі	05.05.2023	
6	Обґрунтування вибору використаних технічних засобів	05.05.2023	
7	Розробка програмного забезпечення	19.05.2023	
8	Налагодження програми	24.05.2023	
9	Виконання графічних документів	04.06.2023	
10	Оформлення пояснювальної записки	02.06.2023	
11	Подання ДП на попередній захист	24.05.2023	
12	Подання ДП рецензенту		
13	Подання ДП на основний захист		

Студент _____ Даніл КРАВЧЕНКО

(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 86 таблиць, 38 рисунків та 13 джерел – загалом 98 сторінки.

Дипломний проєкт присвячений розробці веб-застосунку для онлайн бібліотеки.

Метою розробки є популяризація електронних форматів книжок та спрощення процесу їх отримання.

Об'єкт дослідження: Веб-застосунок для онлайн бібліотеки.

Предмет дослідження: Розробка та впровадження веб-застосунка для онлайн бібліотеки з метою популяризації електронних форматів книжок та спрощення процесу їх отримання.

У розділі «АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» розглянуто основні вимоги до техніки для підтримки застосунку та використання. Також описано основні вимоги до функціоналу серверної та клієнтської частин і інтерфейсу користувача.

У розділі «МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» проведено аналіз та розроблено моделі, що стосуються архітектури та конструювання програмного забезпечення, розглянуто архітектуру серверної та клієнтської частин та застосунку в цілому.

Розділ «АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» присвячений аналізу якості програмного забезпечення з використанням таких метрик як: функціональність, ефективність, зручність використання, сумісність, покриття тестами та підтримка працездатності

Програмне забезпечення впроваджено за допомогою Azure App Service для серверної частини та Vercel для клієнтської.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, VISUAL STUDIO, AZURE, MSSQL, РОБОТА З ФАЙЛАМИ, КЛІЄНТ-СЕРВЕР, .NET, NUXT, АДМІН ПАНЕЛЬ, CLOUDINARY

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 86 tables, 38 figures and 13 sources – in total 98 pages.

The diploma project is dedicated to the development of a web application for an online library.

The aim of the development is to popularize electronic book formats and simplify the process of obtaining them.

The object of research: A web application for an online library.

The subject of research: Development and implementation of a web application for an online library to popularize electronic book formats and simplify the process of obtaining them.

In the section "АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ", the basic requirements for hardware to support the application and use are considered. It also describes the basic requirements for the functionality of the server and client parts and the user interface.

The section "МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ" analyzes and develops models related to software architecture and design, and considers the architecture of the server and client parts and the application as a whole.

The section "АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ" is devoted to the analysis of software quality using such metrics as functionality, efficiency, usability, compatibility, test coverage and maintainability.

The software was implemented using Azure App Service for the server side and Vercel for the client sideand Vercel for the client.

KEYWORDS: WEB APPLICATION, VISUAL STUDIO, AZURE, MSSQL, FILE MANAGEMENT, CLIENT-SERVER, .NET, NUXT, ADMIN PANEL, CLOUDINARY

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9409.045440.01.91	Технічне завдання	16	
3	А4	КПІ.ІТ-9409.045440.02.81	Пояснювальна записка	97	
4	А4	КПІ.ІТ-9409.045440.03.13	Текст програми	74	
5	А4	КПІ.ІТ-9409.045440.04.51	Програма та методика тестування	5	
6	А4	КПІ.ІТ-9409.045440.05.34	Керівництво користувача	8	
7	А3	КПІ.ІТ-9409.045440.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІТ-9409.045440.07.99	Схема бази даних	1	
9	А3	КПІ.ІТ-9409.045440.08.99	Схема структурна класів	1	
			програмного забезпечення		
10	А3	КПІ.ІТ-9409.045440.09.99	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9409.045440.00.90		
Розроб.		Кравченко Д.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-94	
Н.контр.		Вітковська І.І.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ОНЛАЙН БІБЛІОТЕКИ

Технічне завдання

КПІ.ІТ-9409.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
Катерина ЛІЩУК

Нормоконтроль:
Ірина ВІТКОВСЬКА

Виконавець:
Даніл КРАВЧЕНКО

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосунок для онлайн бібліотеки

Галузь застосування: інформаційні технології та бібліотечна справа.

Наведене технічне завдання поширюється на розробку програмного забезпечення «Веб-застосунок для онлайн бібліотеки» [ВЗОБ], котра використовується для швидкого й зручного доступу до книжкової колекції, перегляду інформації про книги, авторів та жанри, створення та редагування персональних колекцій, залишення відгуків та оцінок, скачування книг у різних форматах, а також адміністрування бібліотеки через адміністративну панель та призначена для широкого кола користувачів, які мають доступ до Інтернету та зацікавлені в читанні та використанні онлайн бібліотек, включаючи школярів, студентів, викладачів, науковців, книголюбів та інших зацікавлених осіб.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Веб-застосунок для онлайн бібліотеки» є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для полегшення пошуку та перегляду та популяризації електронних версій книжок для користувачів, які мають доступ до Інтернету.

Метою розробки є забезпечення користувачам можливості перегляду, створення власного списку книжок, залишення відгуків та скачування книжок у різних форматах. Та полегшення управління контентом та користувачами за допомогою адміністративної панелі.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– початкова сторінка з популярними книжками, авторами, жанрами та останніми відгуками(рис 4.1);

– перегляд книжок з можливістю фільтрації(рис 4.2);

– відображення додаткової інформації про книжку та відгуків користувачів(рис 4.3);

– завантаженням книжки у певному форматі(рис 4.3);

– відображення додаткової інформації про автора, його книжок та відгуків користувачів (рис 4.4);

– перегляд профілю та власних колекцій (рис 4.5).

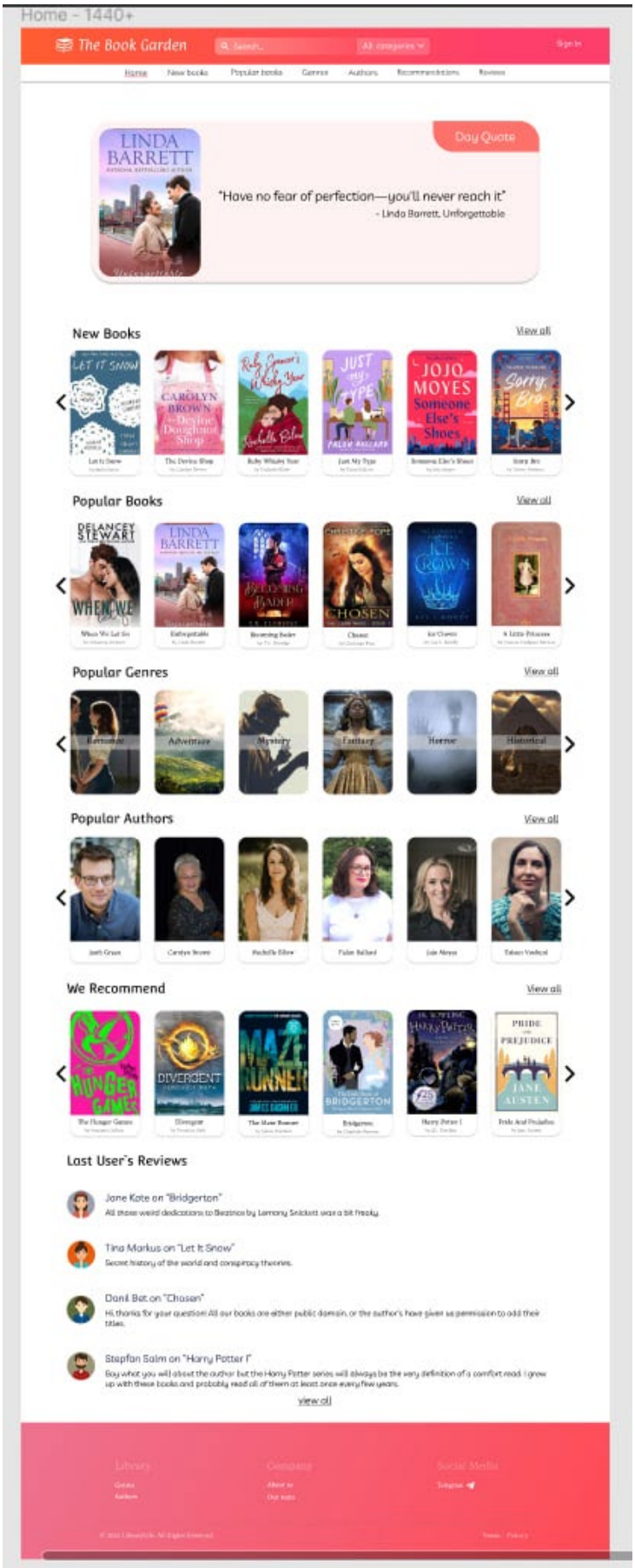


Рисунок 4.1 – Прототип початкової сторінки

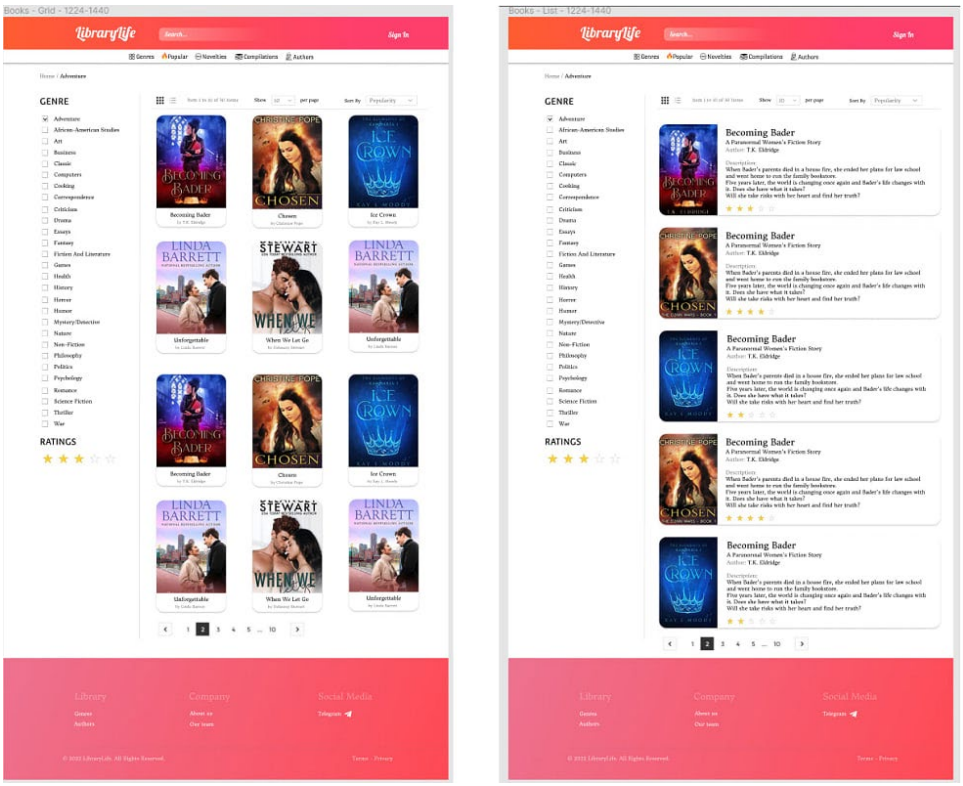


Рисунок 4.2 – Прототип відображення книжок у вигляді таблиці й списку

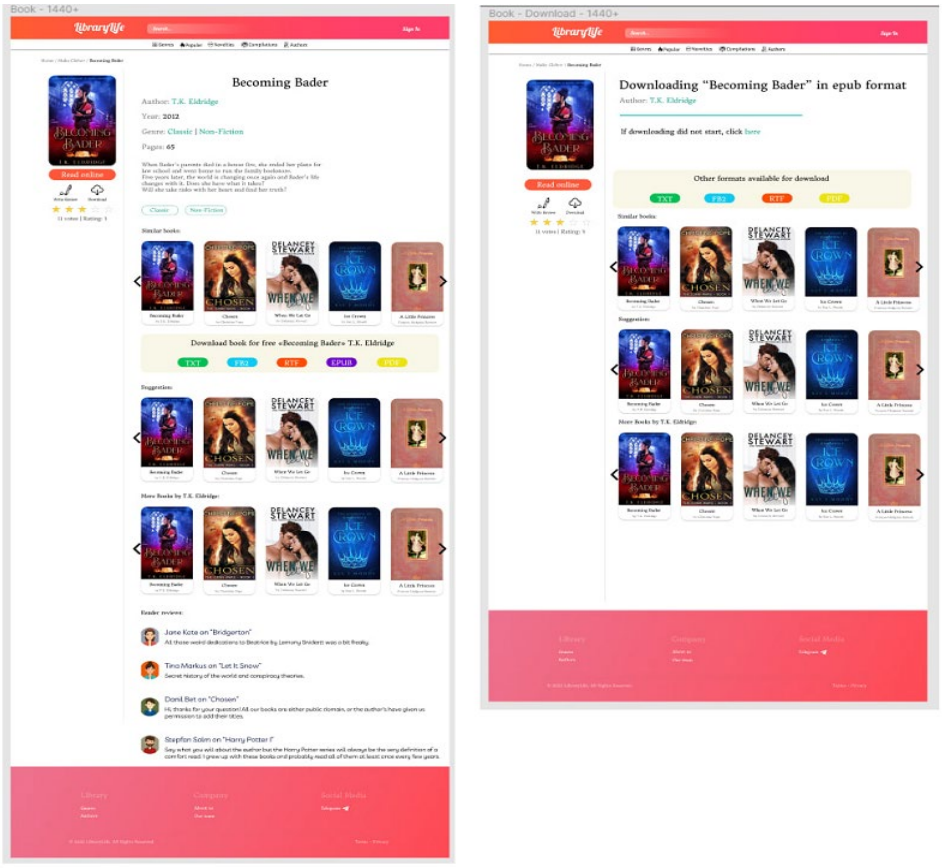


Рисунок 4.3 – Прототип сторінок з додатковою інформацією про книжку та завантаження

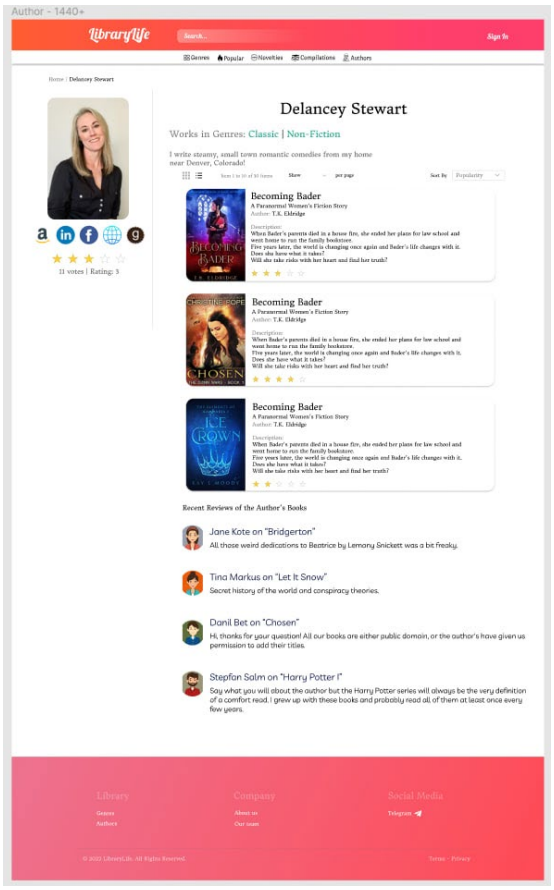


Рисунок 4.4 – Прототип сторінки автора

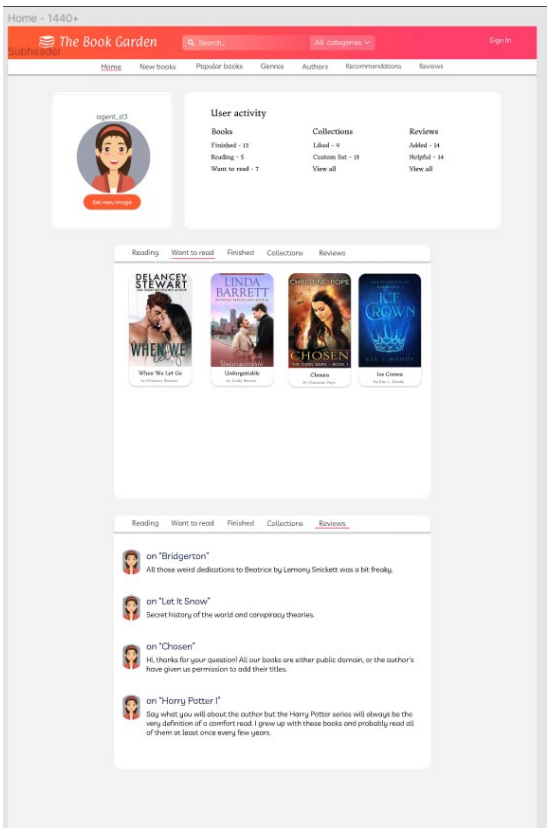


Рисунок 4.5 – Прототип сторінки профілю користувача

4.1.2 Для користувача:

– перегляд, фільтрування та сортування книжок;
– перегляд, фільтрування та сортування авторів;
– перегляд, фільтрування та сортування жанрів;
– перегляд, фільтрування та сортування збірок;
– обирати спосіб відображення між таблицею та списком;
– відображення детальної інформації та книжок відповідного автора;
– залишити відгук та оцінку автору;
– відображення детальної інформації про відповідну книжку;
– завантажувати книжку в електронному форматі;
– залишити відгук та оцінку на книжку;
– додати до колекцій читаю/хочу прочитати/прочитав;
– відображення книжок відповідної збірки;
– залишити відгук та оцінку на збірку;
– створити акаунт у системі;
– зайти у існуючий акаунт;
– переглянути власну активність(колекції, відгуки);
– змінити фото профілю.

4.1.3 Для адміністратора системи:

– створення, редагування, видалення та перегляд книжок;
– створення, редагування, видалення та перегляд авторів;
– створення, редагування, видалення та перегляд жанрів;
– створення, редагування, видалення та перегляд збірок;
– редагування, видалення та перегляд користувачів;
– перевірка, видалення та перегляд відгуків користувачів.

4.2 Вимоги до надійності

– передбачити контроль введення інформації та захист від некоректних дій користувача;

– забезпечити безпеку паролів користувачів за допомогою шифрування;

– забезпечити приватний доступ до файлів та передбачити можливості зламу системи збереження;

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Отримання та аналіз відгуків користувачів, у випадку звернень, забезпечити надійну та швидку підтримку та відлагодження продукту

4.3.2 Обслуговуючий персонал

Для відлагодження системи та внесення швидких правок необхідний один Full-Stack програміст зі знанням .Net та Vue.js технологій, або два окремих для серверної та клієнтської частин відповідно.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на усіх пристроях з останніми версіями веб-браузеру

Мінімальна конфігурація технічних засобів:

– будь-яка операційна система: Linux, Windows, MacOS;

– підключення до інтернету;

– будь-який веб-браузер;

– підключення до інтернету.

Рекомендована конфігурація технічних засобів:

– процесор: Intel Pentium III, Celeron або AMD Athlon, з тактовою частотою 500 МГц;

– оперативна память: 2 Гб;

– відеокарта: с 1 Гб відеопам'яті;

– підключення до інтернету.

4.5 Вимоги до інформаційної та програмної сумісності

Веб-сервіс повинен бути сумісний з веб-браузерами (наприклад, Google Chrome, Mozilla Firefox, Microsoft Edge)

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі:

Книга – для збереження даних про книжку

– назва книжки;

– опис;

– автор;

– мова, на якій написано;

– посилання на обкладинку книжки.

Автор – для збереження даних про автора

– ім’я автора;

– опис;

– посилання на фото автора;

– публічні посилання(Amazon, Facebook, GoodReads).

Жанр – для збереження даних про жанр

– назва жанру;

– опис;

– посилання на фото жанру.

Збірка – для збереження даних про збірку

– назва збірки;
– опис;
– мова книжок у збірці;
– посилання на фото збірки.

Відгук – для збереження даних про відгуки від користувачів

– ім’я користувача, що опублікував;
– електронна пошта користувача;
– текст відгуку;
– оцінка(від 1 до 5);
– дата й час публікації;
– книжка, на яку було опубліковано відгук.

Користувач – для збереження даних про користувача

– ім’я користувача;
– посилання на головне фото;
– пароль(захешований);
– роль.

4.5.2 Вимоги до вихідних даних

Вихідні дані програмного забезпечення відсутні.

4.5.3 Вимоги до мови розробки

Розробка складається з двох частин: серверна та клієнтська частини.

Розробку серверної частини виконати на мові програмування C#, а клієнтської частини виконати на мові програмування JavaScript за допомогою фреймворку Nuxt.js

4.5.4 Вимоги до середовища розробки

Розробку клієнтської частини виконати на платформі Visual Studio Code, а серверну - у середовищі Visual Studio.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути написаний на мовах програмування C# та TypeScript, добре структуровані та збережені у системі контролю версій.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні та особисті вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема бази даних;

– схема структурна класів програмного забезпечення;

– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	20.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	05.05	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	19.05	Тексти програмного забезпечення
6.	Розробка матеріалів текстової частини проекту	19.05	Пояснювальна записка
7.	Тестування програмного забезпечення	25.05	Тести, результати тестування
8.	Розгортання програмного забезпечення	28.05	Пояснювальна записка
9.	Розробка матеріалів графічної частини проекту	30.05	Графічний матеріал проекту

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка

до дипломного проекту

на тему: Веб-застосунок для онлайн бібліотеки

КПІ.ІТ-9409.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЕБ	–	Електронна бібліотека
JVM	–	Java Virtual Machine
JS	–	JavaScript
SSR	–	Server Side Rendering
SEO	–	Search Engine Optimization
IDE	–	Integrated Development Environment
ОС	–	Операційна система.
БД	–	База даних
РСУБД	–	Реляційні системи управління базами даних
JWT	–	JSON Web Token

ВСТУП

Від появи писемності у світі люди почали записувати і зберігати інформацію, яку бажали передати майбутнім поколінням та поділитися знаннями. Таким чином з'являлися бібліотеки, які і зараз служать місцем зберігання книг, журналів, рукописів, літописи та інше.

Проте, з плином часу книги почали псуватися і потребували реставрації та переписування, що займало багато років. З розвитком технологій люди отримали можливість друкувати книги, що збільшило їх доступність. Бібліотеки залишалися зручними завдяки системі каталогів, яка дозволяла легко знаходити і переглядати книги за авторами, жанрами, роками видання та іншими параметрами.

Завдяки комп'ютерним технологіям і цифровому зберіганню інформації, паперові формати книги поступово витісняються електронними, такими як електронні книги та інтернет-джерела. Зберігання інформації у цифровому вигляді є надійнішим, оскільки електронні ресурси можна зберігати на різних пристроях або у хмарному сховищі. Навіть якщо станеться щось з одним носієм, інформацію можна відновити з іншого пристрою або хмари, за умови доступу до Інтернету.

Проаналізувавши все вищесказане, бачимо, що ідея створення онлайн бібліотеки, як спосіб збереження та отримання безліч книг у будь-який час, є чудовою. Це дозволить ефективно зберігати і систематизувати різноманітну інформацію в одному місці, що економить час при пошуку потрібних матеріалів в Інтернеті.

Саме тому, темою для моєї дипломної роботи буде проектування та реалізація такої бібліотеки. Головна мета якої – популяризація електронних форматів книжок та спрощення процесу їх отримання

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Робота присвячена розробці веб-застосунку, що має полегшити пошук книжкових ресурсів у електронному форматі в Інтернеті. На думку автора електронні ресурси є одним із ключових рішень поширення та отримання інформації.

Електронна книга – це версія книги в електронному (цифровому) вигляді. Такі книжки можна читати за допомогою комп'ютерів, мобільних телефонів чи спеціалізованих пристроїв.[1]

Проте виникли вони ще давно, за часів першого комп’ютера, в наслідок можливості збільшення ринку розповсюдження та споживання книг. Електронні книги стали порятунком для віддалених від великих друкарень регіонів і країн, хоча вони й не є заміщенням традиційних паперових книг, а створені задля поліпшення та покращення процесу розповсюдження літератури, що у свій час допомагає зменшити кількість посередників між автором і читачем[2], саме для цього існують електронні бібліотека

Електронна бібліотека – комплексна інформаційна система, в якій документи зберігаються і можуть використовуватися в машиночитаній ("електронній") формі, програмними засобами забезпечується єдиний інтерфейс доступу з однієї точки до електронних документів, що містять тексти і зображення. База даних ЕБ може складатися з різного виду електронних колекцій документів.[3]

Онлайн бібліотеки з’явилися як результат поєднання новітніх технологій та бібліотечної науки. Вони забезпечують не тільки зручний доступ до книжкових ресурсів, але й розширюють функціонал за допомогою новинок й особливостей, які покращують пошук використанням ключових слів, а не тільки назви, або можливістю залишати коментарі та рецензії на

книжки, збірки або навіть авторів, що покращує популяризацію та додає інтерактив у прочитання електронної книги.

Розвиток не зупиняється навіть й зараз, створюючи нові й нові напрямки розробок, тож розглянемо одні з найвідоміших:

– веб-напрямок. Веб-застосунки для онлайн бібліотек можуть розвиватися в напрямку додавання нових функцій і можливостей, за допомогою впровадження системи рейтингу та відгуків на книги, можливості створення списків побажань, можливість додавання закладок у тексті електронної книги, покращений пошук та фільтрація книжок за різними параметрами, інтеграція з соціальними медіа тощо;

– мобільні додатки. Популярність мобільних пристроїв зростає, що надає можливість створювати мобільні додатки задля задоволення потреб користувачів й охвату більшого кола користувачів, та дозволяє користувачам зручно переглядати та читати книжки на своїх смартфонах або планшетах, забезпечуючи доступ до бібліотеки з будь-якого місця і в будь-який час;

– покращення інтерфейсу користувача. Можна не тільки додавати новий функціонал, а працювати над існуючим оптимізуючі та покращуючи інтерфейс для вдосконалення дизайну, розробку інтуїтивно зрозумілих елементів управління. Наприклад, створення темної та світлої теми застосунку або вдосконалення навігації на сайті;

– інтеграція з іншими сервісами. Онлайн бібліотеки можуть інтегруватися з іншими сервісами, наприклад, з платформами для покупки паперових книг. Це дозволяє користувачам зручно й швидко придбати книжку, просто переходячи за посиланням. Або інтеграція з соціальними мережами, для поширення власних прочитаних книжок, або для рекомендації.

1.2 Змістовний опис і аналіз предметної області

У сучасному світі онлайн бібліотека є зручним інструментом для доступу до книжкових ресурсів через Інтернет. Вона допомагає зберігати, організовувати, шукати та поширювати знання, та розширювати використання користувачами задовольняючи їх потреби у читанні та отримання знань з літератури.

Програмні забезпечення не стоять у сторонці й активно використовують знання та концепції пов’язані з сучасною бібліотекою. У більшості веб-застосунків, наприклад, реалізовано класифікація книг, пошук та фільтрація по жанрам, авторам та збіркам, навігація по книжковим ресурсам, обробка запитів користувача. ІТ-розробники використовують ці знання для створення функціональності та інтерфейсу веб-застосунків для онлайн бібліотек.

Незважаючи на значний прогрес у розвитку веб-застосунків для онлайн бібліотек, існують певні недоліки та проблеми:

– неефективний пошук та фільтрація, що ускладнює користувачеві отримання бажаного результату;

– незручна навігація. Наприклад, неможливий перехід з певної книги на сторінку автора, для перегляду всіх його книжок;

– відсутність відгуків чи оцінок, що сприятиме збільшенню часу на пошук чогось особистого й бажаного;

– незрозумілий інтерфейс або застарілий дизайн, що може призвести до втрати уваги користувача.

Усі вище розглянуті недоліки не з’явилися сьогодні чи вчора, вони проявлялися з часом та розвитком застосунків, а отже й існують шляхи покращення ситуації з розробками у сфері ІТ:

– розширення функціональності пошуку та фільтрації, наприклад, включаючи фільтрування за автором, жанром, літературними напрями або мовами написання;

– впровадження інтелектуальних систем рекомендацій, які базуються на інтересах користувачів, що надасть змогу відображати реальні й корисні рекомендації книжкового асортименту;

– покращення інтерактивності та соціальної взаємодії веб-застосунків, шляхом додавання можливостей обміну враженнями, коментування, рецензування книг та спілкування з іншими читачами;

– розробка мобільних додатків та адаптація веб-застосунків для різних платформ та пристроїв, що дозволить забезпечити зручний доступ до онлайн бібліотек у будь-який час і в будь-якому місці.

Підсумовуючи різноплановість шляхів покращення, в рамках дипломного проекту було обрано шлях розробки та реалізації веб-застосунку для онлайн бібліотеки, який буде фокусуватись на розширенні соціальної взаємодії, а саме можливості поділитися, залишити відгук та оцінити книгу. Це дозволить створити активну спільноту користувачів, яка буде обмінюватись думками та рекомендаціями, залучати нових читачів та підтримувати інтерес до читання. А за допомогою колекцій «Читаю», «Хочу прочитати» та «Прочитав» користувачі завжди будуть мати власні списки та легко ділитися ними з іншими.

До того ж даний шлях включає в себе покращення пошуку необхідного ресурсу за допомогою ключових слів пошукової системи.

1.3 Аналіз існуючих технологій та успішних ІТ-проектів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації веб-застосунку онлайн бібліотеки. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Під час порівняльного аналізу рішень аутентифікації та авторизації, обрано використання JWT (JSON Web Tokens) через його переваги, такі як легкість використання, масштабованість, безстанність та підтримка розподіленої аутентифікації. JWT забезпечує безпеку та інтеграцію з різними платформами.

Не менш важливим є робота з файлами, завантаження, отримання доступу та організація, саме для цього необхідно мати зручну файлову систему, на щастя сучасні технології мають готові рішення, серед яких було обрано Cloudinary для збереження фотографій та Azure Storage - для файлів книжок у різних форматах.

Cloudinary було обрано через зручну інтеграцію з C#, зручний інтерфейс для організації та перегляду файлів та безкоштовні 25 ГБ простору для збереження ресурсів, що достатньо для збереження необхідної кількості фотографій.

Azure Storage має переваги у безпечному й надійному збереженні файлів, так як має вбудовані механізми реплікації та резервування, та зручну інтеграцію з екосистемою Azure, а так як використовується багато технологій Microsoft, то це може бути плюсом у майбутньому

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Оскільки основні конпоненти веб-застосунку були виділені, тому для їх реалізації мають бути використані мови для розробки серверної частини, бази даних та клієнтської частини.

1.3.2.1 Мови програмування для серверної частини

У сучасному світі серверна частина може бути реалізована багатьма універсальними мовами, серед яких популярними є Python[4], NodeJS[5], .Net[6] та Java[7]. Кожна з цих мов має свої переваги та недоліки, які можуть

впливати на вибір для розробки онлайн бібліотеки. Давайте розглянемо їх детальніше.

Python є інтерпретованою, високо рівневою мовою програмування, з фокусом на простоту та читабельність коду. Він має широкі можливості для розробки веб-застосунків, включаючи підтримку різних фреймворків, таких як Django і Flask.

Переваги: Простий синтаксис, велика кількість бібліотек і модулів для швидкого розробки, широка спільнота розробників.

Недоліки: Менша продуктивність порівняно з деякими іншими мовами, особливо в області високонавантажених систем.

NodeJS є середовищем виконання JavaScript на стороні сервера. Воно побудоване на двигуні V8 Chrome і пропонує асинхронний підхід до програмування.[8]

Переваги: відкритий код платформи, багато модулів вже реалізовано та можна реалізувати клієнт та серверну частини однією мовою, що дозволить зменшити кількість технологій для підтримки.

Недоліки: Через швидкий розвиток та часті оновлення треба постійно слідкувати за новими версіями та витрачати ресурси на підтримку робочого функціоналу.

.Net є найбільш популярною та швидкою платформою для розробки веб-застосунків та має фреймворк ASP.Net для реалізації серверної частини. Мова програмування C# є основною мовою для розробки на платформі .Net.

Переваги: Велика екосистема .Net, потужність та гнучкість фреймворку ASP.Net, велика кількість інструментів та ресурсів для розробки, ефективність, типізованість мови.

Недоліки: Багато бібліотек або додаткових сервісів потребують додаткових матеріальних витрат на ліцензійне програмне забезпечення порівняно з деякими іншими мовами.

Java є високорівневою, об'єктно-орієнтованою мовою програмування, яка є дуже популярною в сфері розробки веб-додатків. Вона відома своєю платформою JVM, що дає можливість запускати програми на будь-якій операційній системі без необхідності перекомпілювати їх для кожної ОС окремо.

Переваги: багатоплатформність, велика кількість додаткових бібліотек та фреймворків, безпечність

Недоліки: через використання віртуальної машини, Java-програми можуть запускатися повільніше ніж наприклад на .Net

Проаналізувавши всі переваги й недоліки мов для реалізації серверної частини, було обрано .Net платформу з використанням мови C# через швидку розробку основного функціонала та інтеграцію з фреймворками клієнтської частини, вбудовану систему захисту даних, масштабованість, що може потім дозволити розширити розробкою мобільного додатку та через велику спільноту та кількість форумів де користувачі допомагають один одному.

1.3.2.2 Технології для клієнтської частини

Для розробки клієнтської частини часто використовують JS фреймворки, який дуже велика кількість, у кожного є свої плюси й мінуси, але зосередитись на найпопулярніших фреймворках, таких як Angular[9], React[10], Vue.js[11] та Nuxt.js[12] який створений на основі Vue.

Angular є повноцінним фреймворком для розробки веб-додатків, який використовує мову TypeScript.

Переваги: масштабованість, розширюваність та надає багато вбудованих функціональних можливостей, таких як система модулів, компоненти, директиви, сервіси та інші.

Недоліки: важкий(згенерований код може мати великий розмір), складність структури та залежність від ресурсів у власній екосистемі

React є бібліотекою для створення інтерфейсів користувача, що зосереджується на компонентах. Він дозволяє розробникам створювати та

використовувати UI-компоненти та ефективно керувати станом додатка. Переваги: простота та гнучкість, яка дає розробникам більшу свободу. Недоліки: для повнофункціонального додатку може знадобитися додаткова настройка та використання сторонніх бібліотек.

Vue.js є прогресивним фреймворком для розробки користувацьких інтерфейсів, який поєднує переваги Angular та React. Він простий у використанні, має зрозумілу документацію та добре структурований код.

Переваги: плавна реактивна розробка, висока продуктивність та легкість згенерованого коду.

Недоліки: менша екосистема сторонніх бібліотек та ресурсів.

Nuxt.js є фреймворком на основі Vue.js, спеціально розробленим для рендерингу на стороні сервера (SSR) веб-додатків. Основна перевага Nuxt.js полягає в його адаптивності до пошукової оптимізації (SEO). Nuxt.js забезпечує покращення SEO-показників завдяки SSR, що пришвидшує ініціальну загрузку сторінок. Так як Nuxt.js є розширенням Vue.js, то він включає в себе й його переваги, але також має свої: вбудовані функції для маршрутизації, кешування, підтримки мета-тегів.

Отже, обрання Nuxt.js для реалізації клієнтської частини онлайн-бібліотеки обґрунтоване його адаптивністю до SEO. Завдяки вбудованою можливостю SSR, Nuxt.js забезпечує покращення SEO-показників, що дозволяє забезпечити кращу видимість додатку у пошукових системах. До того ж було обрано TypeScript як мову для написання логіки клієнтської частини, так як дана мова є об’єктно орієнтованою, що покращить зрозумілий зв’язок між серверною та клієнтською частинами.

1.3.2.3 Середа розробки

Обравши мови програмування, необхідно розглянути середи розробки, які підтримують мови C# та TypeScript, а саме Visual Studio + Visual Studio Code та IntelliJ IDEA.

У кожної середи є свої переваги, але було обрано само комбінацію Visual Studio + Visual Studio Code через практичну зручність розділення серверної та клієнтської частини на різні вікна.

1.3.2.4 Вибір бази даних

Основною частиною онлайн бібліотеки є база даних, а отже вибір її є не менш важливим, так як було обрано .Net з використанням мови C#, то треба розглядати реляційні бази даних, що дозволяє організувати дані у вигляді таблиць зі зв'язками між ними, що ж важливим для онлайн бібліотеки, так як потрібно зберігати й взаємодіяти з різними типами даних: книги, автори, колекції, жанри, користувачі та їх відгуки, тощо. Реляційні БД забезпечують структурованість та цілісність даних, що полегшує управління та запити до них.

Основні й найбільш популярні РСУБД: SQLite, MySql, PostgreSQL та Microsoft SQL Server. І так як було вже обрано мову програмування від Microsoft, то оберемо Microsoft SQL Server[13], який має значні переваги та причини обрати само його: масштабованість, надійність, підтримка, безпека даних та зручну інтеграцію з Visual Studio.

1.3.3 Аналіз відомих програмних продуктів

Як вже було сказано, існує багато відомих та дійсно якісних веб-застосунків онлайн бібліотеки, проти кожен продукт має свої недоліки, які може навіть і не є недоліками для розробників, але на думку автора мають бути виправлені або замінені.

Головні моменти, які будуть розглядатися та порівнюватися, це чи інтуїтивно зрозумілий інтерфейс, який функціонал дозволений гостю, зручність пошуку та фільтрації, навігація по сайту, наявність особових списків для користувача, наявність функціоналу оцінки та рецензії, приємність дизайну та чи безкоштовне користування застосунком.

1.3.3.1 Bookua

Перший погляд головної сторінки, яку можна побачити на рисунку 1.1 викликає негативні емоції, так як дизайн сайту застарілий, не зовсім зрозумілий інтерфейс та дивна навігація, що змушує користувача розбиратися куди треба натиснути задля переходу на детальну сторінку. Хоча якщо витратити більше часу на дослідження, то можемо помітити, що сторінки відображаються швидко, якість картинок гарна й наявна мобільна версія.

Можна залишати анонімні коментарі, але не наявні списки бажані

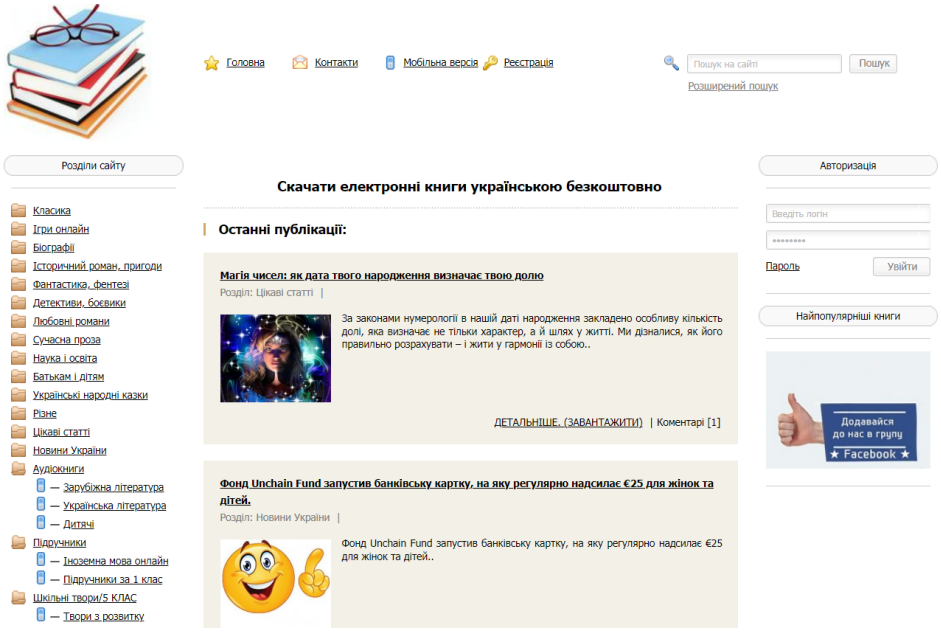


Рисунок 1.1 – Головна сторінка Bookua

1.3.3.2 ManyBooks

Приємний застосунок з гарним дизайном та інтерфейсом, який можна побачити на рисунку 1.2, пошук виконується швидко, як і сортування та фільтрація, до того ж все це супроводжується швидкою та зручною навігацією по сайту.

Наявна колекція бажань, але неможливо переглянути цю колекцію, що трохи плутає користувача. Іноді проскакує підгрузка сторінки й не подгружаются стилі сторінки, що змушує користувача робити додаткові кроки, такі як перезавантаження сторінки.

Дуже багато реклами, яка заважає насолоджуватися переглядом книжок, або залишенням відгука, який у свій час можна залишити тільки після входу до акаунту.

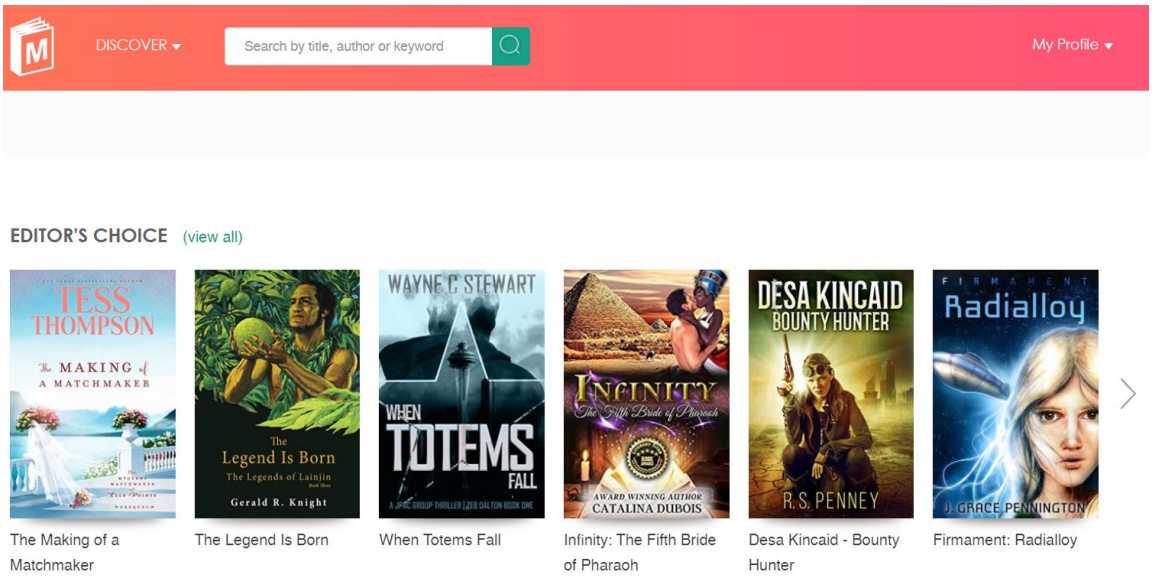


Рисунок 1.2 – Головна сторінка ManyBooks

1.3.3.3 Knigogo

Сучасний дизайн з інтуїтивно зрозумілим інтерфейсом одразу приваблює, наявність пошуку по ключовим словам та зручна навігація з швидким відображенням контенту додає симпатії до даного рішення. Головну сторінку даного застосунку можна побачити на рисунку 1.3

Проте, відсутній функціонал реєстрації та не всі книжки наявні у повній версії, і дізнатися це можна тільки за допомогою переходу на детальну сторінку книжки.

Відсутність змоги авторизуватися дозволяє залишати анонімні відгуки, проте автоматично лишає змоги створювати списків бажань та організовувати власні книги.

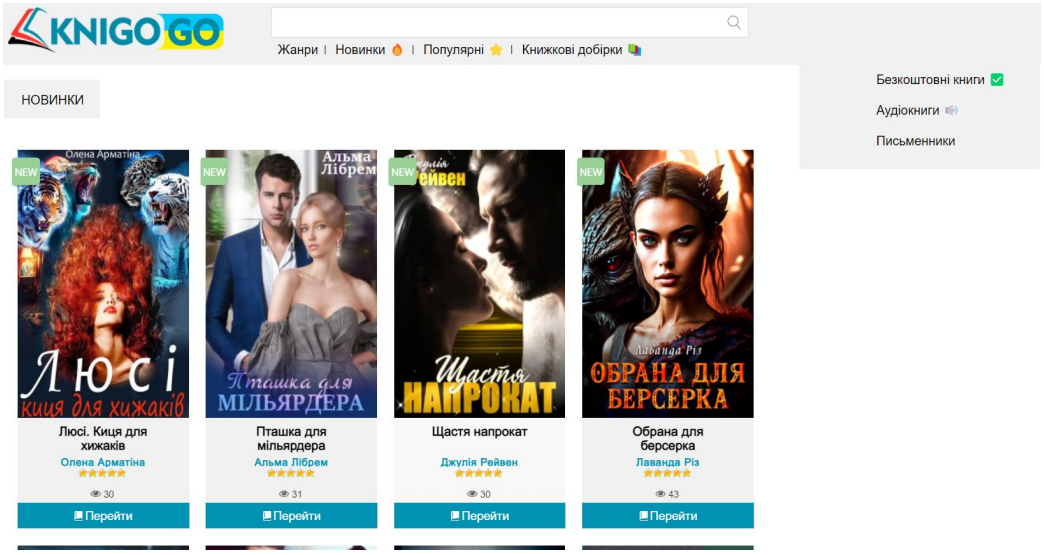


Рисунок 1.3 – Головна сторінка Knigogo

1.3.3.4 УкрЛіб

Українізований застосунок з творчістю українського народу та видатних письменників, головну сторінку якого можна побачити на рисунку 1.4. Застосунок має сучасний дизайн, швидку навігацію, гарні шрифти.

Наданий зручний функціонал читання книжки онлайн, але при змозі скачати більшість книжок виявляються платними.

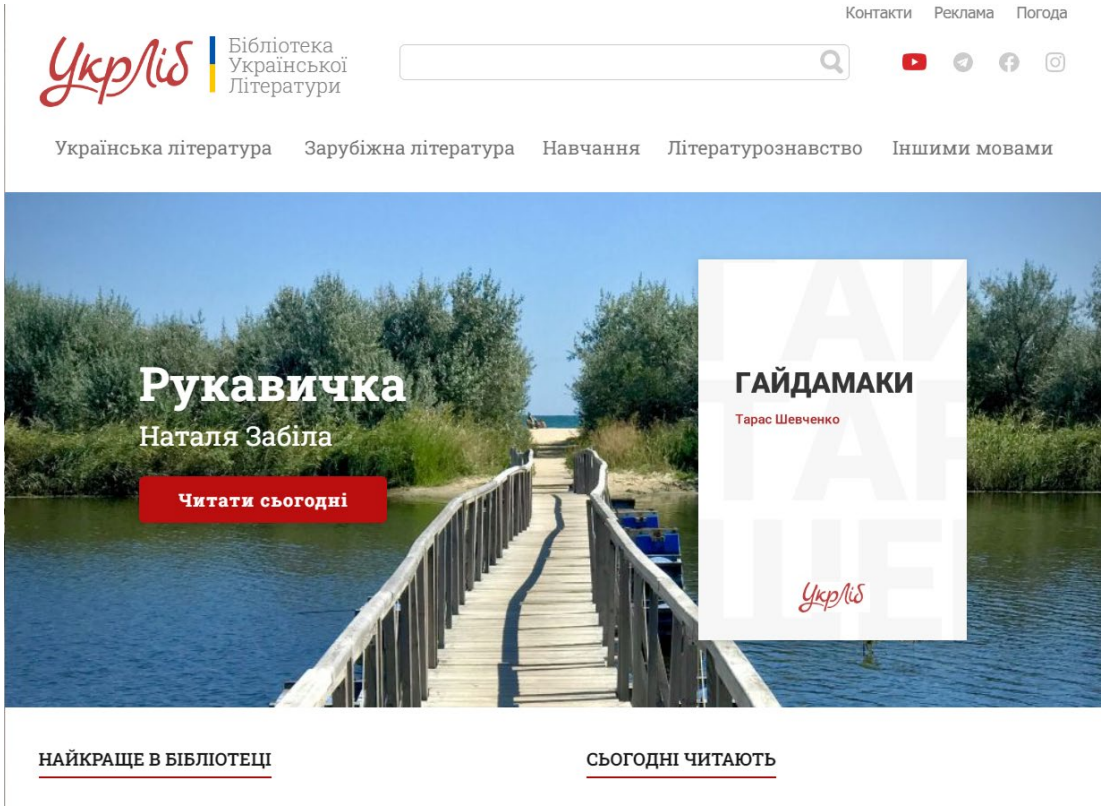


Рисунок 1.4 – Головна сторінка УкрЛіб

Для порівняння дипломної роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Book Garden	Bookua	Knigogo	УкрЛіб	Пояснення
Швидкість роботи	+	+	+	+	Застосунок має швидко отримувати й відображати дані користувачу
Зрозумілий інтерфейс	+	-	+	+	Інтерфейс застосунку має бути інтуїтивно зрозумілим
Приємний дизайн	+	-	+	+	Застосунок повинен мати комфортний для очей дизайн
Користування без реєстрації	+	+	+	+	Чи можна користуватися застосунком без реєстрації
Залишати відгук без акаунту	+	+	+	-	Чи можна залишати відгук без попереднього входу до акаунту
Зручний пошук та фільтрація	+	-	+	-	Застосунок має відображати тільки елементи, які задовольняють фільтри

Продовження таблиці 1.1

Навігація по сайту	+	-	+	-	Застосунок повинен містити зручну навігацію
Авторизація	+	+	-	+	Застосунок повинен містити функціонал створення акаунту
Наявність особових списків для користувача	+	-	-	-	Застосунок має функціонал для додавання книжок до власного списку планів
Відображення результату у вигляді списку або таблиці	+	-	-	-	Застосунок повинен мати вибір між таблицею та список при відображенні контенту
Завантаження книжки	+	-	+	+	Застосунок повинен надавати змоги завантажити книжку для самостійного прочитання
Повна версія книжки	+	+	-	+	Застосунок має включати в себе повну версію книжки, а не фрагмент

1.4 Аналіз вимог до програмного забезпечення

Головними функціями програмного забезпечення є перегляд, пошук та завантаження книжок, більше функцій можна побачити на рисунку 1.5.

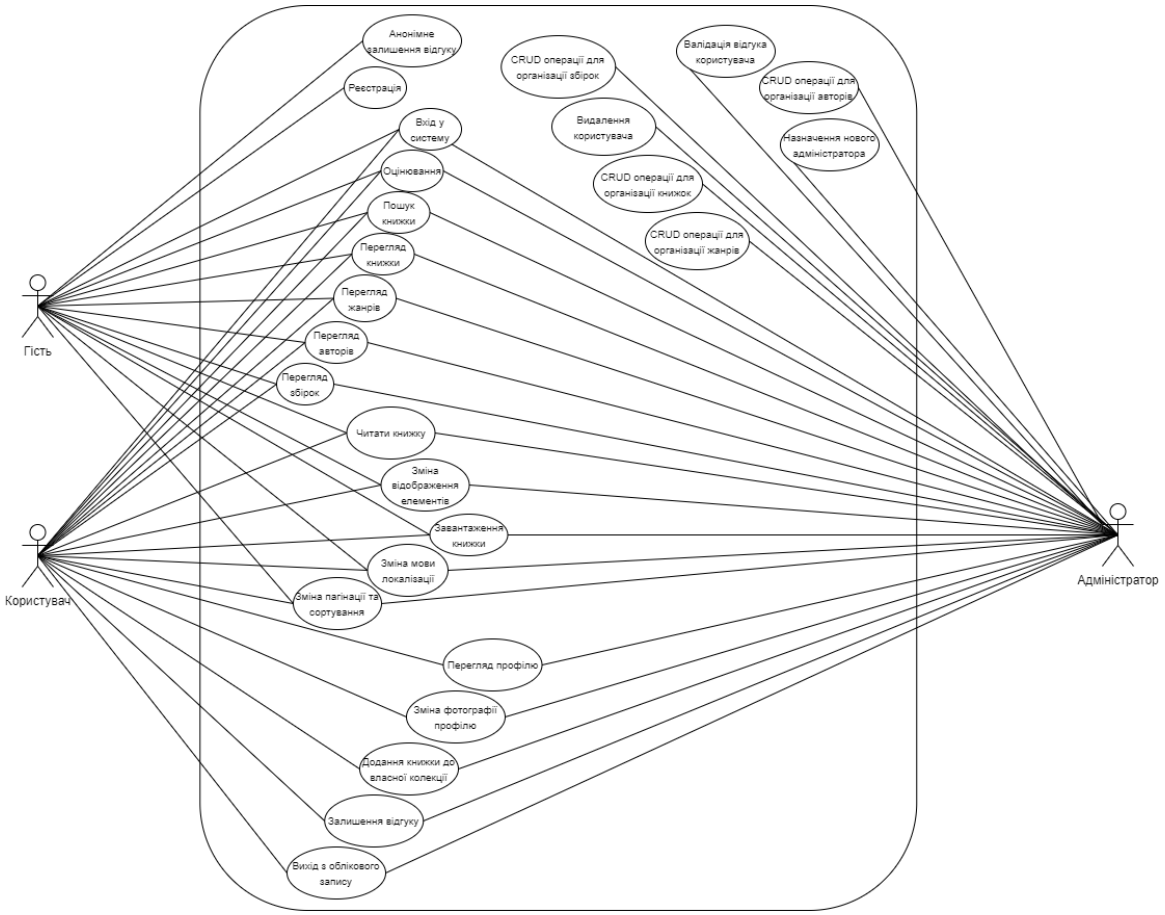


Рисунок 1.5 – Діаграма варіантів використання

В таблицях 1.2 - 1.26 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Створення облікового запису користувача
Actors	Гість

Продовження таблиці 1.2

Trigger	Гість бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. Вводить необхідні персональні дані – ім’я користувача, email та пароль. Після введення всіх коректних даних підсвічується кнопка “Sign up”. Користувач натискає кнопку для реєстрації.
Extension	У випадку введення некоректних даних, кнопка реєстрації стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно обводиться по контуру червоним кольором
Post-Condition	Створення сторінки користувача, перехід на сторінку входу

Таблиця 1.3 - Варіант використання UC-2

Use case name	Авторизація
Use case ID	UC-02
Goals	Вхід до застосунку за допомогою існуючого облікового запису
Actors	Гість
Trigger	Гість бажає авторизуватися
Pre-conditions	Гість має вже створений обліковий запис
Flow of Events	Користувач переходить на сторінку авторизації. Вводить email та пароль. Після введення всіх коректних даних підсвічується кнопка “Login”. Користувач натискає кнопку для авторизації.
Extension	У випадку введення пустих або некоректних даних кнопка неактивна. При неправильно введених даних повернеться помилка “User was not found”

Продовження таблиці 1.3

Post-Condition	Користувач авторизується у веб-застосунку та перенаправляється на головну сторінку

Таблиця 1.4 - Варіант використання UC-3

Use case name	Пошук книжки
Use case ID	UC-03
Goals	
Actors	Користувач
Trigger	Користувач хоче знайти необхідну йому книжку
Pre-conditions	-
Flow of Events	Користувач вводить ключові слова у поле пошуку. Система перевіряє введені дані й виконує пошук у базі даних. Система відображає книжки, які відповідають критеріям пошуку. Користувач обирає потрібну йому книжку.
Extension	Якщо жодна книжка не знайдена за введеними критеріями, система повідомляє користувача про відсутність результатів пошуку.
Post-Condition	Користувач знаходить потрібну йому книжку в онлайн бібліотеці.

Таблиця 1.5 - Варіант використання UC-4

Use case name	Перегляд книжок
Use case ID	UC-04
Goals	Переглянути наявні книжки в онлайн бібліотеці
Actors	Користувач
Trigger	Користувач бажає ознайомитись зі списком доступних книжок
Pre-conditions	-

Продовження таблиці 1.5

Flow of Events	Система відображає головну сторінку, на якій доступний перелік популярних книжок, авторів та жанрів. Користувач вибирає певну категорію, яка його цікавить. Система виконує запит до бази даних та відображає список книжок, що відповідають обраній фільтрації. Користувач переглядає перелік доступних книжок, відображених з назвами, авторами та обкладинками. Користувач може вибрати певну книжку для отримання більш детальної інформації.
Extension	Якщо жодна книжка не відповідає за обраній фільтрації, система повідомляє користувача про відсутність результатів.
Post-Condition	Користувач має можливість переглядати наявні книжки в онлайн бібліотеці та отримувати більш детальну інформацію про кожну книжку

Таблиця 1.6 - Варіант використання UC-5

Use case name	Зміна відображення елементів
Use case ID	UC-05
Goals	Змінити стиль відображення з таблиці на список або зі списку на таблицю
Actors	Користувач
Trigger	Користувач бажає змінити спосіб відображення елементів (з таблиці на список або зі списку на таблицю)
Pre-conditions	Система відстежує який спосіб відображення було обрано до цього

Продовження таблиці 1.6

Flow of Events	Користувач переглядає елементи у стилі таблиці та хоче більш детальну інформацію. Користувач клікає на іконку "Список". Система змінює відображення елементів на стиль списку. Користувач спостерігає змінений спосіб відображення елементів у стилі списку. Користувач клікає на іконку "Таблиця". Система змінює відображення елементів на стиль таблиці.
Extension	-
Post-Condition	Користувач має можливість переглядати елементи у зміненому способі відображення

Таблиця 1.7 - Варіант використання UC-6

Use case name	Зміна пагінації та сортування
Use case ID	UC-06
Goals	Організувати зміну способу сортування та пагінації результатів веб-застосунку для зручного перегляду користувачем
Actors	Користувач
Trigger	Користувач бажає відсортувати елементи за назвою, датою публікації або рейтингом, а також змінити кількість елементів на сторінці
Pre-conditions	Система відслідковує який спосіб сортування було обрано до цього, та який номер сторінки переглядав користувач

Продовження таблиці 1.7

Flow of Events	Користувач переглядає книжки. Користувач обирає потрібний спосіб сортування (за назвою, датою публікації або рейтингом). Система застосовує обраний спосіб сортування до списку елементів та оновлює їх відповідно до нового порядку. Користувач обирає бажану кількість елементів на сторінці (наприклад, 10, 20, 50 елементів). Система оновлює список елементів та відображає нову кількість елементів на сторінці.
Extension	-
Post-Condition	Користувач має можливість змінити спосіб сортування та пагінації результатів веб-застосунку для зручного перегляду списку елементів

Таблиця 1.8 - Варіант використання UC-7

Use case name	Оцінювання
Use case ID	UC-07
Goals	Забезпечити можливість користувачам оцінювати книжки за рейтингом від 1 до 5
Actors	Користувач
Trigger	Користувач бажає оцінити певну книжку за рейтингом
Pre-conditions	Користувач ще не оцінював відповідну книжку
Flow of Events	Користувач переходить до сторінки книжки, яку він бажає оцінити. Користувач обирає бажану оцінку для книжки на шкалі від 1 до 5. Система оновлює рейтинг та зберігає оцінку користувача, відображаючи її йому
Extension	-
Post-Condition	Користувач успішно оцінив книжку за рейтингом, а система зберегла оцінку та оновила рейтинг книжки.

Таблиця 1.9 - Варіант використання UC-8

Use case name	Анонімне залишення відгуку
Use case ID	UC-08
Goals	Забезпечити можливість гостям залишати відгуки про книжки
Actors	Гість
Trigger	Гість бажає залишити відгук про певну книжку
Pre-conditions	-
Flow of Events	Гість переходить до сторінки книжки, для якої він бажає залишити відгук. Гість натискає на іконку «Відгук». Гість вводить свою електронну пошту, ім'я та текст відгуку, обирає оцінку. Гість підтверджує залишення відгуку. Система зберігає відгук разом з відповідними даними гостя (email, ім'я) та пов'язує його зі відповідною книжкою
Extension	Якщо гість не введе обов'язкові поля, система повідомляє про необхідність їх заповнення перед підтвердженням. Якщо гість введе неправильний формат електронної пошти, система повідомляє про помилку та просить ввести правильну адресу
Post-Condition	Гість успішно залишив відгук про книжку, і система зберегла відгук та пов'язала його зі відповідною книжкою

Таблиця 1.10 - Варіант використання UC-9

Use case name	Залишення відгуку
Use case ID	UC-09
Goals	Забезпечити можливість зареєстрованим користувачам залишати відгуки про книжки.
Actors	Зареєстрований користувач
Trigger	Користувач бажає залишити відгук про певну книжку.

Продовження таблиці 1.10

Pre-conditions	-
Flow of Events	Користувач переходить до сторінки книжки, для якої він бажає залишити відгук. Користувач натискає на іконку «Відгук». Користувач вводить текст відгуку. Користувач підтверджує залишення відгуку. Система зберігає відгук разом з відповідними даними користувача та пов'язує його зі відповідною книжкою
Extension	Якщо користувач не введе текст відгуку, система повідомляє про необхідність ввести текст перед підтвердженням
Post-Condition	Користувач успішно залишив відгук про книжку, і система зберегла відгук та пов'язала його зі відповідною книжкою та обліковим записом користувача

Таблиця 1.11 - Варіант використання UC-10

Use case name	Завантаження книжки у певному форматі
Use case ID	UC-10
Goals	Забезпечити можливість користувачам завантажувати книжки у вибраному форматі на свій пристрій.
Actors	Користувач
Trigger	Користувач бажає завантажити книжку у певному форматі.
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку книжки, яку він бажає завантажити. Користувач переглядає доступні формати, у яких можна завантажити книжку. Користувач обирає бажаний формат для завантаження. Система перевіряє, чи обраний формат доступний для завантаження. Система генерує файл книжки у вибраному форматі, який автоматично завантажується у користувача через 10 секунд

Продовження таблиці 1.11

Extension	Якщо обраний формат не доступний для завантаження, система повідомляє користувача про неможливість завантаження у цьому форматі.
Post-Condition	Користувач успішно завантажив книжку у вибраному форматі на свій пристрій.

Таблиця 1.12 - Варіант використання UC-11

Use case name	Додавання книжки до власної колекції
Use case ID	UC-11
Goals	Забезпечити можливість зареєстрованим користувачам додавати книжки до своєї власної колекції з визначеними статусами
Actors	Зареєстрований користувач
Trigger	Користувач бажає додати книжку до своєї власної колекції з вказанням статусу
Pre-conditions	Користувач має обліковий запис та відповідна книжка не наявна вже у колекції
Flow of Events	Користувач переходить на сторінку книжки, яку він бажає додати до своєї колекції. Користувач натискає на іконку «+» під обкладинкою книжки та обирає необхідну колекцію з списку: "Want to read", "Reading" чи "Read". Система додає книжку до колекції користувача з обраним статусом. Користувач може переглянути цю книжку у профілі
Extension	-
Post-Condition	Користувач успішно додав книжку до своєї власної колекції з вказаним статусом.

Таблиця 1.13 - Варіант використання UC-12

Use case name	Перегляд жанрів
Use case ID	UC-12
Goals	Надати користувачеві можливість переглядати доступні жанри книг в онлайн бібліотеці
Actors	Користувач
Trigger	Користувач вибирає опцію для перегляду жанрів книг
Pre-conditions	-
Flow of Events	Користувач натискає на “Genres” у меню застосунку. Система відображає сторінку зі списком доступних жанрів книг. Користувач переглядає перелік жанрів та може вибрати конкретний жанр для отримання більш детальної інформації та фільтрації книг за жанром.
Extension	Якщо в системі немає жанрів або жанри відсутні, система повідомляє користувача про це
Post-Condition	Користувач успішно переглянув доступні жанри книг та може вибрати конкретний жанр для подальшої інтеракції з книжками вибраного жанру

Таблиця 1.14 - Варіант використання UC-13

Use case name	Перегляд авторів
Use case ID	UC-13
Goals	Надати користувачеві можливість переглядати доступні автори книг в онлайн бібліотеці
Actors	Користувач
Trigger	Користувач вибирає опцію для перегляду авторів книг
Pre-conditions	-

Продовження таблиці 1.14

Flow of Events	Користувач натискає на “Authors” у меню застосунку. Система відображає сторінку зі списком наявних авторів. Користувач переглядає перелік авторів та може перейти до сторінки конкретного автора для отримання більш детальної інформації та його книг.
Extension	Якщо в системі немає авторів або автори відсутні, система повідомляє користувача про це
Post-Condition	Користувач успішно переглянув наявних авторів книг та може вибрати конкретного автора для подальшої інтеракції з його книжками

Таблиця 1.15 - Варіант використання UC-14

Use case name	Перегляд збірок
Use case ID	UC-14
Goals	Надати користувачеві можливість переглядати доступні збірки книг в онлайн бібліотеці
Actors	Користувач
Trigger	Користувач обирає опцію для перегляду збірок книг
Pre-conditions	-
Flow of Events	Користувач натискає на “Compilations” у меню застосунку. Система відображає сторінку зі списком наявних збірок. Користувач переглядає їх та може перейти до сторінки конкретної збірки для отримання більш детальної інформації та книг наявних у збірці.
Extension	Якщо в системі немає збірок або вони відсутні, система повідомляє користувача про це
Post-Condition	Користувач успішно переглянув наявні збірки книг та може вибрати збірку для отримання детальної інформації

Таблиця 1.16 - Варіант використання UC-15

Use case name	Зміна мови локалізації
Use case ID	UC-15
Goals	Надати користувачеві можливість змінити мову локалізації веб-застосунку між українською та англійською
Actors	Користувач
Trigger	Користувач бажає змінити мову локалізації на українську або англійську
Pre-conditions	
Flow of Events	Користувач вибирає бажану мову локалізації у меню застосунку, українську або англійську. Система застосовує зміни і оновлює інтерфейс веб-застосунку у вибраній мові локалізації. Користувач бачить змінений інтерфейс у вибраній мові локалізації.
Extension	-
Post-Condition	Користувач успішно змінив мову локалізації веб-застосунку на українську або англійську та бачить змінений інтерфейс у вибраній мові

Таблиця 1.17 - Варіант використання UC-16

Use case name	Перегляд профілю
Use case ID	UC-16
Goals	Дозволити зареєстрованому користувачеві переглянути свій профіль
Actors	Зареєстрований користувач
Trigger	Користувач бажає переглянути свій профіль
Pre-conditions	Користувач має активний обліковий запис у веб-застосунку

Продовження таблиці 1.17

Flow of Events	Користувач натискає на своє ім'я користувача для перегляду профілю. Система відображає профіль користувача, який містить основну інформацію, таку як ім'я, статистику, фотографію профілю. Користувач може переглянути додаткові деталі, які включають, наприклад, список прочитаних книжок, список улюблених книжок та відгуки.
Extension	Якщо користувач не авторизован, система відображає повідомлення про потребу у реєстрації або увійти в систему
Post-Condition	Користувач успішно переглядає свій профіль і бачить відповідну інформацію, включаючи основні та додаткові дані

Таблиця 1.18 - Варіант використання UC-17

Use case name	Зміна фотографії профілю
Use case ID	UC-17
Goals	Дозволити зареєстрованому користувачеві змінити фотографію свого профілю
Actors	Зареєстрований користувач
Trigger	Користувач бажає змінити фотографію свого профілю
Pre-conditions	Користувач має активний обліковий запис у веб-застосунку
Flow of Events	Користувач натискає на своє ім'я користувача для перегляду профілю. Система відображає профіль користувача. Система надає можливість користувачу завантажити нову фотографію. Користувач завантажує нову фотографію з власного пристрою. Система зберігає нову фотографію та оновлює фотографію профілю користувача. Система відображає підтвердження про успішну зміну фотографії профілю

Продовження таблиці 1.18

Extension	Якщо користувач не авторизован, система відображає повідомлення про потребу у реєстрації або увійти в систему
Post-Condition	Користувач успішно змінює фотографію свого профілю, і нова фотографія відображається у профілі користувача

Таблиця 1.19 - Варіант використання UC-18

Use case name	CRUD операції для організації авторів
Use case ID	UC-18
Goals	Дозволити адміністратору веб-застосунку виконувати CRUD-операції (створення, читання, оновлення, видалення) для організації авторів
Actors	Адміністратор
Trigger	Адміністратор бажає здійснити операції створення, читання, оновлення або видалення даних про авторів
Pre-conditions	-
Flow of Events	Адміністратор переходить на адмін панель. Система відображає список авторів, які вже присутні в системі. Адміністратор має можливість виконувати такі операції: створення нового автора, читання інформації про автора, оновлення інформації про автора, видалення автора. Після виконання кожної операції система оновлює список авторів та відображає повідомлення про успішну виконану дію
Extension	У разі некоректного вводу адміністратором інформації про автора, система повідомляє про помилку та пропонує внести відповідні корективи
Post-Condition	Адміністратор успішно здійснює операції створення, читання, оновлення та видалення авторів у системі. Зміни відображаються у списку авторів та пов'язаній інформації.

Таблиця 1.20 - Варіант використання UC-19

Use case name	CRUD операції для організації жанрів
Use case ID	UC-19
Goals	Дозволити адміністратору веб-застосунку виконувати CRUD-операції (створення, читання, оновлення, видалення) для організації жанрів
Actors	Адміністратор
Trigger	Адміністратор бажає здійснити операції створення, читання, оновлення або видалення даних про жанрів
Pre-conditions	-
Flow of Events	Адміністратор переходить на адмін панель. Система відображає список жанрів, які вже присутні в системі. Адміністратор має можливість виконувати такі операції: створення нового жанру, читання інформації про жанр, оновлення інформації про жанр, видалення жанру. Після виконання кожної операції система оновлює список жанрів та відображає повідомлення про успішну виконану дію
Extension	У разі некоректного вводу адміністратором інформації про жанр, система повідомляє про помилку та пропонує внести відповідні корективи
Post-Condition	Адміністратор успішно здійснює операції створення, читання, оновлення та видалення даних про жанрів у системі. Зміни відображаються у списку жанрів та пов'язаній інформації.

Таблиця 1.21 - Варіант використання UC-20

Use case name	CRUD операції для організації збірок
Use case ID	UC-20

Продовження таблиці 1.21

Goals	Дозволити адміністратору веб-застосунку виконувати CRUD-операції (створення, читання, оновлення, видалення) для організації збірок
Actors	Адміністратор
Trigger	Адміністратор бажає здійснити операції створення, читання, оновлення або видалення даних про збірку
Pre-conditions	-
Flow of Events	Адміністратор переходить на адмін панель. Система відображає список збірок, які вже присутні в системі. Адміністратор має можливість виконувати такі операції: створення нової збірки, читання інформації про збірку, оновлення інформації про збірку, видалення збірки. Після виконання кожної операції система оновлює список збірок та відображає повідомлення про успішну виконану дію
Extension	У разі некоректного вводу адміністратором інформації про збірку, система повідомляє про помилку та пропонує внести відповідні корективи
Post-Condition	Адміністратор успішно здійснює операції створення, читання, оновлення та видалення даних про збірки у системі. Зміни відображаються у списку збірок та пов'язаній інформації.

Таблиця 1.22 - Варіант використання UC-21

Use case name	CRUD операції для організації книжок
Use case ID	UC-21
Goals	Забезпечити адміністратору можливість створення, читання, оновлення та видалення даних про книжки в системі.
Actors	Адміністратор

Продовження таблиці 1.22

Trigger	Адміністратор бажає здійснити операції створення, читання, оновлення або видалення даних про книжки
Pre-conditions	-
Flow of Events	Адміністратор переходить на адмін панель. Система відображає список книжок, які вже присутні в системі. Адміністратор має можливість виконувати такі операції: створення нової книжки, читання інформації про книжку, оновлення інформації про книжку, видалення книжки. Після виконання кожної операції система оновлює список книжок та відображає повідомлення про успішну виконану дію
Extension	У разі некоректного вводу адміністратором інформації про книжку, система повідомляє про помилку та пропонує внести відповідні корективи
Post-Condition	Адміністратор успішно здійснює операції створення, читання, оновлення та видалення даних про книжку у системі. Зміни відображаються у списку книжок та пов'язаній інформації.

Таблиця 1.23 - Варіант використання UC-22

Use case name	Валідація відгука користувача
Use case ID	UC-22
Goals	Перевірити та підтвердити відгуки користувачів перед публікацією у веб-застосунку онлайн бібліотеки
Actors	Адміністратор
Trigger	Адміністратор бажає провести валідацію відгука користувача перед його публікацією
Pre-conditions	-

Продовження таблиці 1.23

Flow of Events	Адміністратор переходить на адмін панель. Система відображає список останніх відгуків. Адміністратор має можливість подивитися контент та дозволити публікацію. Система оновлює список відгуків та відображає повідомлення про успішну перевірку
Extension	Якщо адміністратор виявляє неприпустимий вміст у відгуку, він може прийняти додаткові заходи, такі як видалення або редагування конкретних фрагментів тексту
Post-Condition	Відгуки користувачів, що успішно пройшли валідацію, стають видимими для перегляду та користування іншими користувачами веб-застосунку онлайн бібліотеки.

Таблиця 1.24 - Варіант використання UC-23

Use case name	Назначення нового адміністратора
Use case ID	UC-23
Goals	Назначити нового адміністратора для веб-застосунку онлайн бібліотеки
Actors	Адміністратор
Trigger	Адміністратор має права доступу до функціоналу назначення нового адміністратора
Pre-conditions	-
Flow of Events	Адміністратор переходить на адмін панель. Система відображає список облікових записів користувачів. Адміністратор має можливість призначити роль адміністратора новому користувачу. Після успішного назначення нового адміністратора система оновлює список користувачів та надає новому адміністратору відповідні права доступу

Продовження таблиці 1.24

Extension	-
Post-Condition	Новий адміністратор успішно назначений і має відповідні права доступу для управління веб-застосунком онлайн бібліотеки

Таблиця 1.25 - Варіант використання UC-24

Use case name	Видалення користувача
Use case ID	UC-24
Goals	Видалити обліковий запис користувача з веб-застосунку онлайн бібліотеки
Actors	Адміністратор
Trigger	Адміністратор бажає видалити обліковий запис користувача
Pre-conditions	-
Flow of Events	Адміністратор переходить на адмін панель. Система відображає список облікових записів користувачів. Адміністратор обирає користувача, обліковий запис якого потрібно видалити та підтверджує видалення. Система видаляє обліковий запис користувача, включаючи всю пов'язану з ним інформацію, включаючи відгуки та колекції книжок
Extension	-
Post-Condition	Обліковий запис користувача успішно видалений з веб-застосунку онлайн бібліотеки разом з усією пов'язаною з ним інформацією

Таблиця 1.26 - Варіант використання UC-25

Use case name	Вихід з облікового запису
Use case ID	UC-25
Goals	Завершити сесію користувача в системі

Продовження таблиці 1.26

Actors	Зарєстрований користувач
Trigger	Користувач бажає вийти з системи.
Pre-conditions	Користувач є авторизованим.
Flow of Events	Користувач натискає на власне ім’я користувача та у списку обирає «Logout»
Extension	-
Post-Condition	Користувач перенаправляється на сторінку авторизації.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. На рисунку 1.6 наведено загальну модель вимог, а в таблицях 1.27 – 1.51 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.7.

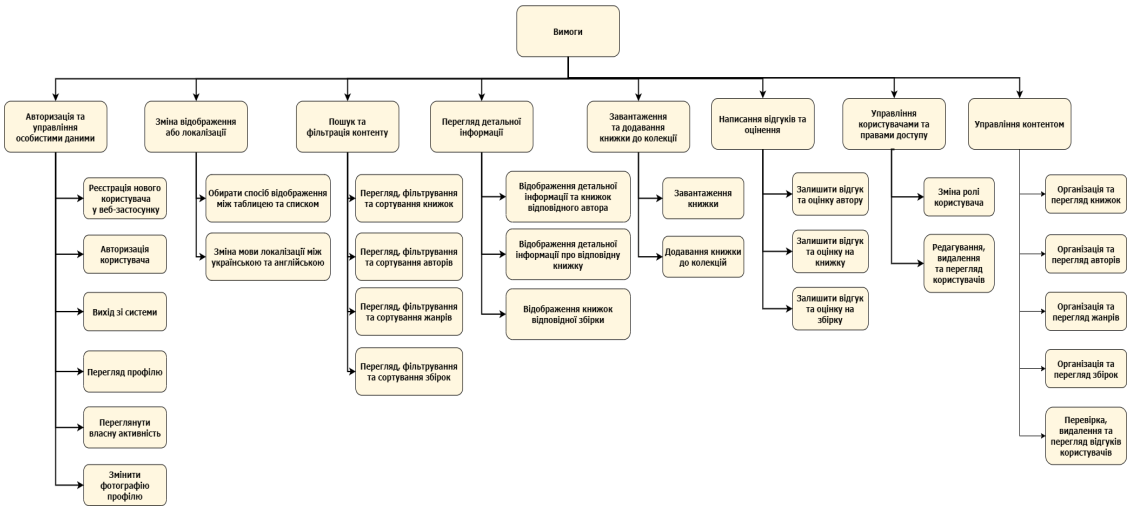


Рисунок 1.6 – Модель вимог у загальному вигляді

Таблиця 1.27 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Система повинна надати можливість користувачам зареєструватись в онлайн бібліотеці, створивши свій обліковий запис

Таблиця 1.28 – Функціональна вимога FR-2

Назва	Авторизація користувача
Опис	Система повинна надати можливість користувачам авторизуватись, використовуючи свої облікові дані

Таблиця 1.29 – Функціональна вимога FR-3

Назва	Перегляд, фільтрування та сортування книжок
Опис	Система повинна надати можливість користувачам переглядати книжки, а також фільтрувати та сортувати їх за різними параметрами

Таблиця 1.30 – Функціональна вимога FR-4

Назва	Перегляд, фільтрування та сортування авторів
Опис	Система повинна надати можливість користувачам переглядати авторів, а також фільтрувати та сортувати їх за різними параметрами

Таблиця 1.31 – Функціональна вимога FR-5

Назва	Перегляд, фільтрування та сортування жанрів
Опис	Система повинна надати можливість користувачам переглядати жанри, а також фільтрувати та сортувати їх за різними параметрами

Таблиця 1.32 – Функціональна вимога FR-6

Назва	Перегляд, фільтрування та сортування збірок
Опис	Система повинна надати можливість користувачам переглядати збірки, а також фільтрувати та сортувати їх за різними параметрами

Таблиця 1.33 – Функціональна вимога FR-7

Назва	Обирати спосіб відображення між таблицею та списком
Опис	Система повинна надати можливість користувачам обирати спосіб відображення даних між таблицею і списком в залежності від їх уподобань та потреб

Таблиця 1.34 – Функціональна вимога FR-8

Назва	Зміна мови локалізації між українською та англійською
Опис	Система повинна надати можливість користувачам обирати мову інтерфейсу між українською та англійською

Таблиця 1.35 – Функціональна вимога FR-9

Назва	Відображення детальної інформації та книжок відповідного автора
Опис	Система повинна надати можливість користувачам переглядати детальну інформацію про книжки, які належать певному автору

Таблиця 1.36 – Функціональна вимога FR-10

Назва	Залишити відгук та оцінку автору
Опис	Система повинна надавати можливість користувачам залишати відгуки та оцінювати авторів, які є частиною онлайн бібліотеки

Таблиця 1.37 – Функціональна вимога FR-11

Назва	Залишити відгук та оцінку на книжку
Опис	Система має надавати можливість користувачам залишати відгуки та оцінювати книжки, що знаходяться в онлайн бібліотеці

Таблиця 1.38 – Функціональна вимога FR-12

Назва	Відображення детальної інформації про відповідну книжку
Опис	Система має надавати можливість користувачам переглядати детальну інформацію про книжку в онлайн бібліотеці

Таблиця 1.39 – Функціональна вимога FR-13

Назва	Перегляд профілю
Опис	Користувачі повинні мати можливість переглядати свій профіль, де вони зможуть переглянути та редагувати свої особисті дані та колекції

Таблиця 1.40 – Функціональна вимога FR-14

Назва	Завантажувати книжку в електронному форматі
Опис	Система повинна надавати можливість користувачам завантажувати книжки в електронному форматі

Таблиця 1.41 – Функціональна вимога FR-15

Назва	Додавання книжки до колекцій "Читаю/Хочу прочитати/Прочитав"
Опис	Система повинна надавати користувачам можливість додавати книжки до власних колекцій з вказанням статусу "Читаю", "Хочу прочитати" або "Прочитав"

Таблиця 1.42 – Функціональна вимога FR-16

Назва	Відображення книжок відповідної збірки
Опис	Система повинна надавати можливість відображати книжки, які належать до певної збірки, для зручного ознайомлення користувачів зі збіркою книжок

Таблиця 1.43 – Функціональна вимога FR-17

Назва	Залишити відгук та оцінку на збірку
Опис	Користувачі повинні мати можливість залишити відгук та оцінку на певну збірку книжок, щоб поділитися своїми враженнями та допомогти іншим користувачам зробити вибір

Таблиця 1.44 – Функціональна вимога FR-18

Назва	Переглянути власну активність(колекції, відгуки)
Опис	Користувачі повинні мати можливість переглядати свою активність в своєму профілі, включаючи колекції, створені ними, та відгуки, які вони залишили на книжки

Таблиця 1.45 – Функціональна вимога FR-19

Назва	Змінити фотографію профілю
Опис	Користувачі повинні мати можливість змінювати свою фотографію профілю, вибираючи нове зображення для відображення на своєму профілі

Таблиця 1.46 – Функціональна вимога FR-20

Назва	Організація та перегляд книжок для адміністратора
Опис	Адміністратор системи повинен мати можливість організовувати та переглядати книжки з використанням адмін-панелі

Таблиця 1.47 – Функціональна вимога FR-21

Назва	Організація та перегляд авторів для адміністратора
Опис	Адміністратор системи повинен мати можливість організовувати та переглядати авторів з використанням адмін-панелі

Таблиця 1.48 – Функціональна вимога FR-22

Назва	Організація та перегляд жанрів для адміністратора
Опис	Адміністратор системи повинен мати можливість організовувати та переглядати жанри з використанням адмін-панелі

Таблиця 1.49 – Функціональна вимога FR-23

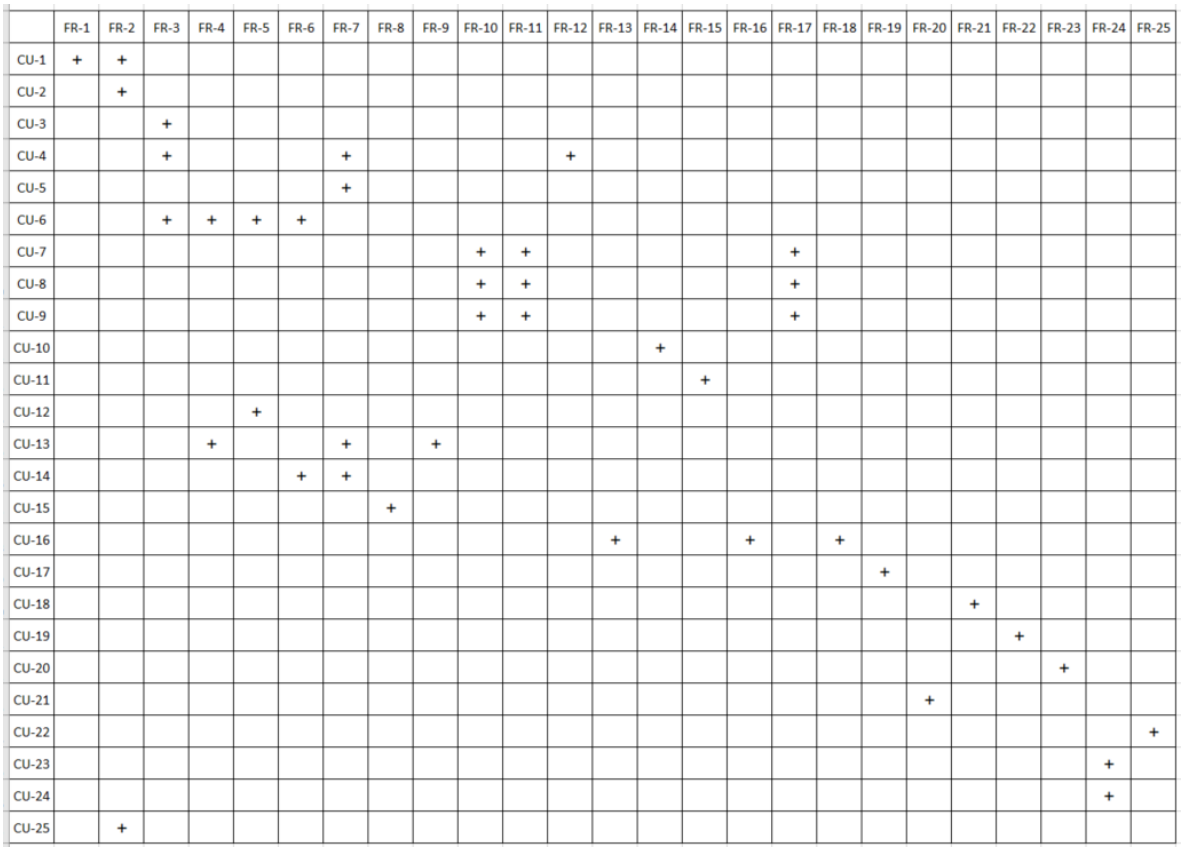
Назва	Організація та перегляд збірок для адміністратора
Опис	Адміністратор системи повинен мати можливість організовувати та переглядати збірки з використанням адмін-панелі

Таблиця 1.50 – Функціональна вимога FR-24

Назва	Редагування, видалення та перегляд користувачів
Опис	Адміністратор системи повинен мати можливість редагувати, видаляти та переглядати інформацію про користувачів з використанням адмін-панелі

Таблиця 1.51 – Функціональна вимога FR-25

Назва	Перевірка, видалення та перегляд відгуків користувачів
Опис	Адміністратор системи повинен мати можливість перевіряти, видаляти та переглядати відгуки, залишені користувачами, з використанням адмін-панелі



	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25
CU-1	+	+																							
CU-2		+																							
CU-3			+																						
CU-4			+				+					+													
CU-5							+																		
CU-6			+	+	+	+																			
CU-7										+	+						+								
CU-8										+	+						+								
CU-9										+	+						+								
CU-10														+											
CU-11															+										
CU-12					+																				
CU-13				+			+		+																
CU-14						+	+																		
CU-15								+																	
CU-16													+			+		+							
CU-17																			+						
CU-18																					+				
CU-19																						+			
CU-20																							+		
CU-21																				+					
CU-22																									+
CU-23																								+	
CU-24																								+	
CU-25		+																							

Рисунок 1.7 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення повинно мати:

- швидку навігацію та відображення контенту;
- інтуїтивно зрозумілий інтерфейс;
- сучасний дизайн;
- підтримуватися сучасними браузерами;
- локалізацію;
- забезпечення безпеки даних користувачів.

1.5 Постановка задачі

Метою реалізації онлайн бібліотеки є популяризація електронних форматів книжок та спрощення процесу їх отримання. Застосунок матиме зручний інтерфейс, дозволятиме користувачам створювати власні списки бажань, оцінювати книжки та залишати відгуки. Це стимулює людей більше читати та забезпечує зручну платформу для доступу до книжок.

Щоб реалізувати поставлену мету необхідно покроково виконати наступні задачі:

- організація контенту бібліотеки;
- пошук та фільтрація контенту;
- реалізація системи відгуків та оцінювання книжок та авторів;
- завантаження та онлайн читання книжки;
- авторизація, реєстрація і редагування особистого профілю користувача;
- зміна відображеня способу відображення контенту на інтерфейсі;
- зміна локалізації інтерфейсу.

Висновки до розділу

У першому розділі було проведено детальний аналіз вимог, необхідних для розробки веб-застосунку онлайн бібліотеки. Перш за все, було визначено загальні положення, пов'язані з розробкою такого застосунку.

Далі, було проведено змістовний опис і аналіз предметної області, що дозволило зрозуміти основні аспекти, принципи та особливості функціонування онлайн бібліотеки.

Також було виконано аналіз існуючих технологій та було обрано платформи та мови програмування для реалізації клієнтської та серверної частин, що дозволило виявити корисні технічні рішення та підходи, які можна використати при розробці застосунку.

У процесі аналізу було проведено оцінку відомих технічних рішень для збереження файлів та фотографій застосунку. Цей аналіз дозволив виявити найкращі практики та інструменти, які можуть бути використані для розробки веб-застосунку та покращити безпеку даних файлової системи.

Під час аналізу вимог до програмного забезпечення було розроблено функціональні вимоги, які описують основні функції та можливості, які має мати онлайн бібліотека. Також було розроблено нефункціональні вимоги, що визначають вимоги до інтерфейсу клієнтської частини та продуктивності, безпеки, надійності серверної частини.

Завершенням цього розділу була постановка задачі, яка визначає загальну мету розробки веб-застосунку онлайн бібліотеки та набір конкретних задач, які необхідно вирішити в процесі розробки.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису бізнес процесу програмного забезпечення використовується BPMN модель, основні процеси зображені на рисунках 2.1 - 2.10

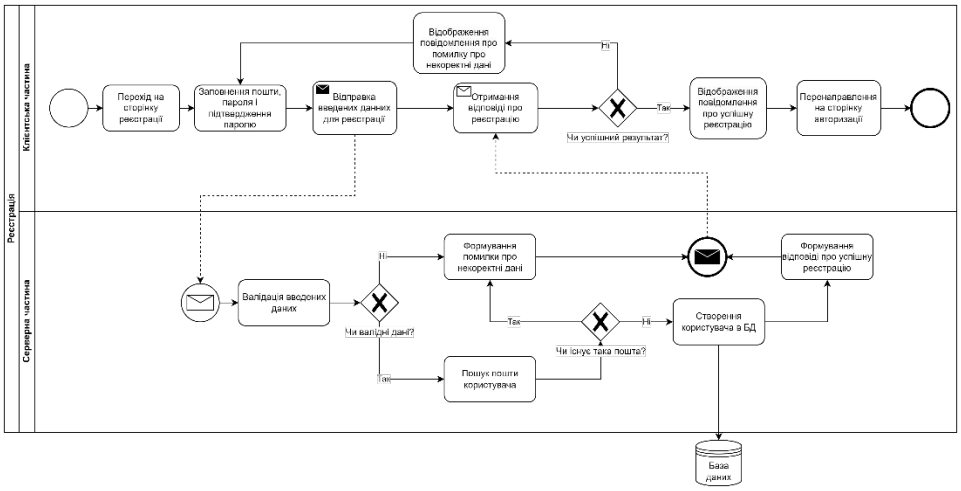


Рисунок 2.1 – Схема реєстрації

Опис послідовності створення облікового запису користувача:

– користувач переходить на сторінку реєстрації;

– користувач заповнює поля реєстрації: електронну пошту, пароль та підтвердження паролю;

– якщо введені поля, не відповідають шаблону заповнення на клієнтській стороні, відповідні поля підсвічуються помилкою;

– якщо введені поля, не валідні, то серверна частина повідомляє помилкою;

– у разі валідності, буде створено новий запис на перенаправлено на сторінку авторизації.

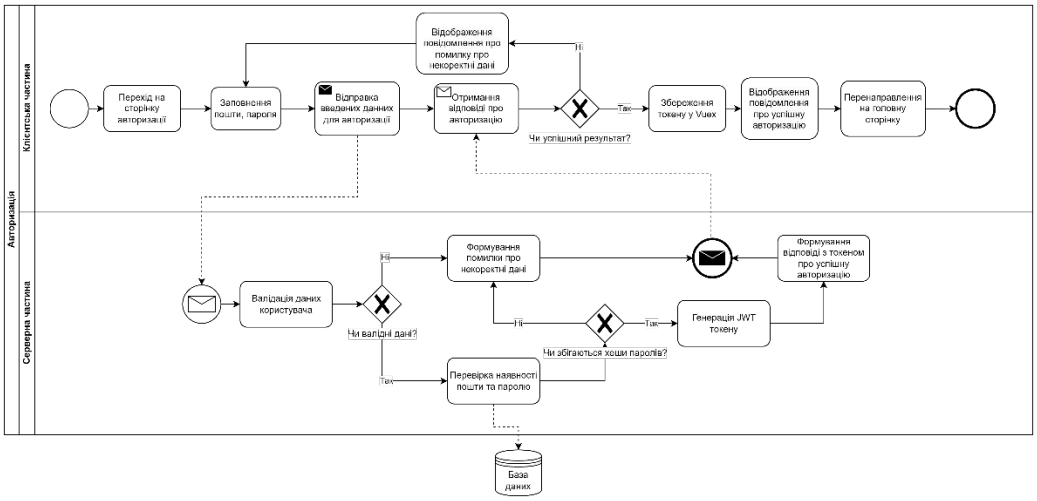


Рисунок 2.2 – Схема авторизації

Опис послідовності створення входу в обліковий запис:

– користувач переходить на сторінку авторизації;

– користувач заповнює поля авторизації: електронну пошту, пароль;

– якщо введені поля, не відповідають шаблону заповнення на клієнтській стороні, відповідні поля підсвічуються помилкою;

– якщо введені поля, не валідні, то серверна частина повідомляє помилкою;

– у разі валідності та наявності користувача у БД, буде згенеровано JWT та збережено у пам’яті для подальшого використання.

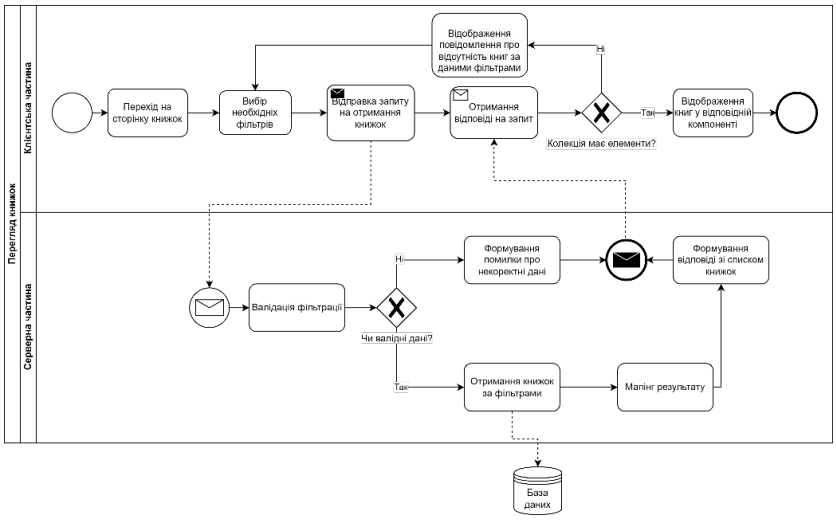


Рисунок 2.3 – Схема перегляду книжок

Опис послідовності перегляду книжок(авторів/жанрів/збірок):

– користувач переходить на сторінку книжок;

– користувач обирає необхідні фільтри;

– якщо фільтрація не валідна, серверна частина повідомляє помилкою;

– у разі валідності, буде повернено книжки які відповідають фільтрації;

– якщо було отримано пустий список, клієнтська частина повідомить про відсутність книжок;

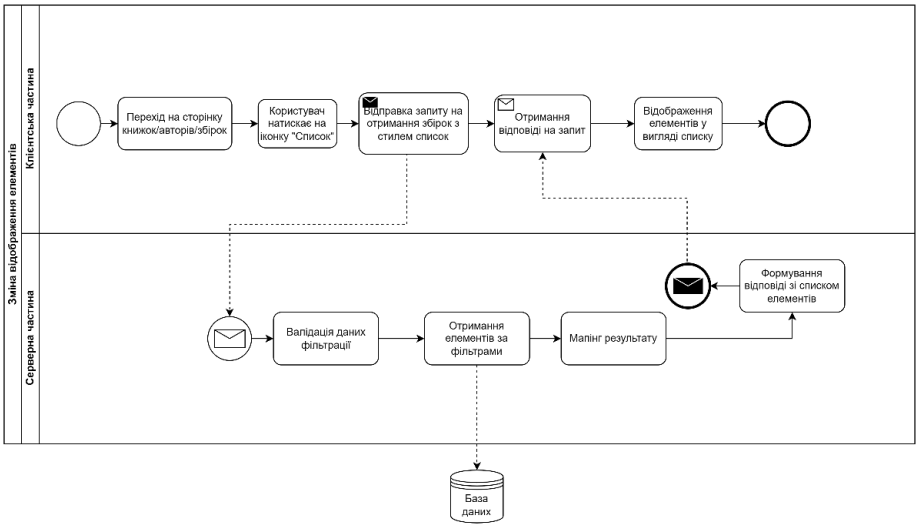


Рисунок 2.4 – Схема зміни відображення елементів

Опис послідовності зміни відображення елементів

– користувач переходить на сторінку книжок/авторів/збірок;
– користувач натискає на іконку «Список»;
– система відображає елементи у вигляді списку.

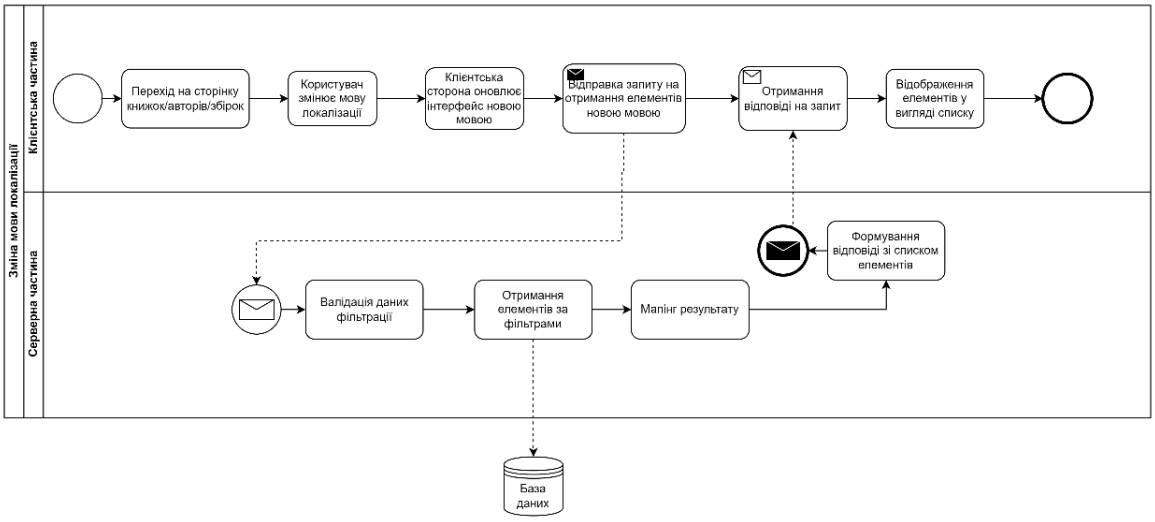


Рисунок 2.5 – Схема зміни мови локалізації

Опис послідовності зміни мови локалізації, детальна схема зображена на плакаті «Схема бізнес процесу «Оцінювання книги»:

– користувач обирає іншу мову локалізації у меню застосунку;
– клієнтська сторона оновлює інтерфейс новою мовою;
– виконує запит до сервера для отримання контенту новою мовою;
– система відображає контент новою мовою.

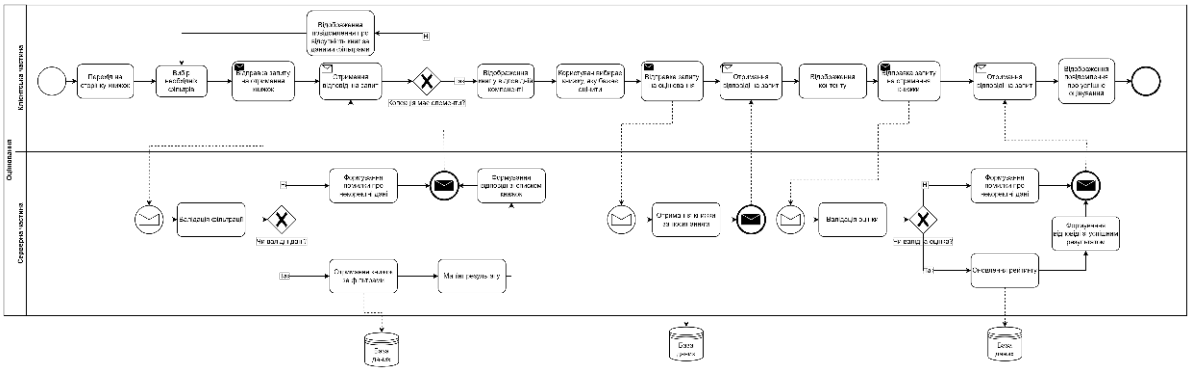


Рисунок 2.6 – Схема оцінювання

Опис послідовності оцінювання, додатково схему продубльовано у додатку Б:

– користувач виконує пошук необхідної книжки;

– користувач переходить на сторінку книжки;

– користувач оцінює книжку за допомогою рейтингу від 1 до 5;

– сервер перевіряє наявність оцінки від користувача та оновлює рейтинг у разі необхідності;

– клієнтська сторона відображає оцінку користувача.

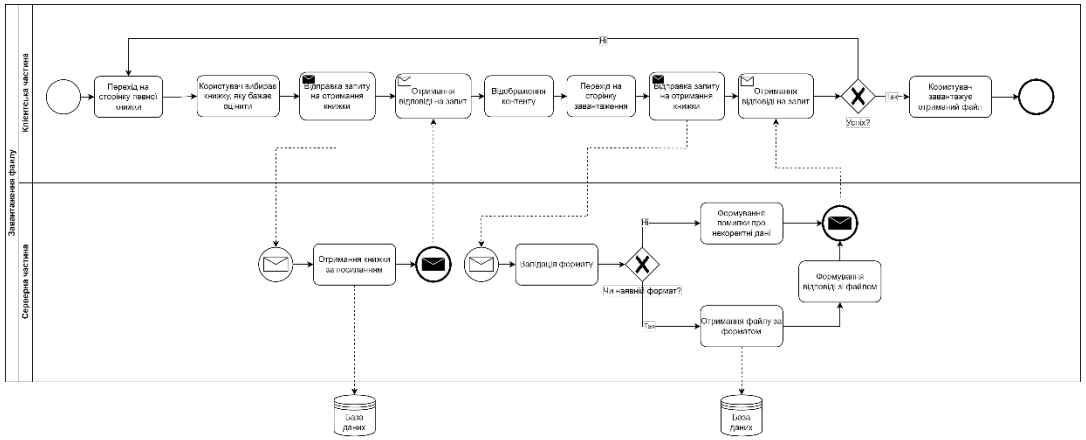


Рисунок 2.7 – Схема завантаження книжки

Опис послідовності завантаження книжки:

– користувач виконує пошук необхідної книжки;

– користувач переходить на сторінку книжки;

– користувач обирає необхідний формат з наявних та перенаправляється на сторінку завантаження;

– серверна частина перевіряє наявність файлу та у разі знаходження повертає файл користувачу;

– клієнтська сторона завантажує файл на стороні користувача.

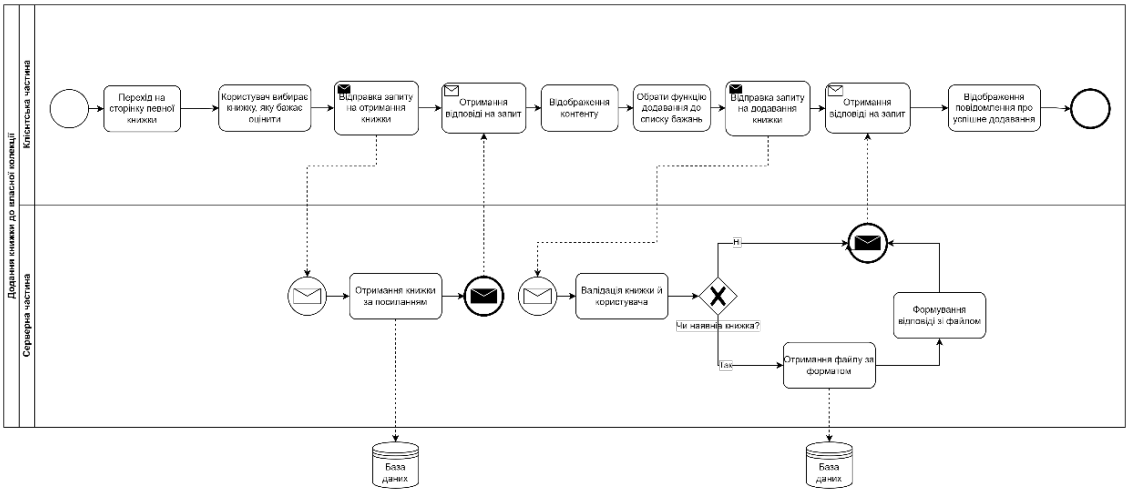


Рисунок 2.8 – Схема додавання книжки до власної колекції

Опис послідовності оцінювання:

– користувач виконує пошук необхідної книжки;
– користувач переходить на сторінку книжки;
– користувач обирає у які колекцію додати книжку;
– клієнтська частина надсилає запит на додавання до колекції;
– сервер додає книжку до відповідної колекції користувача;
– клієнтська сторона відображає наявність книжки у колекції.

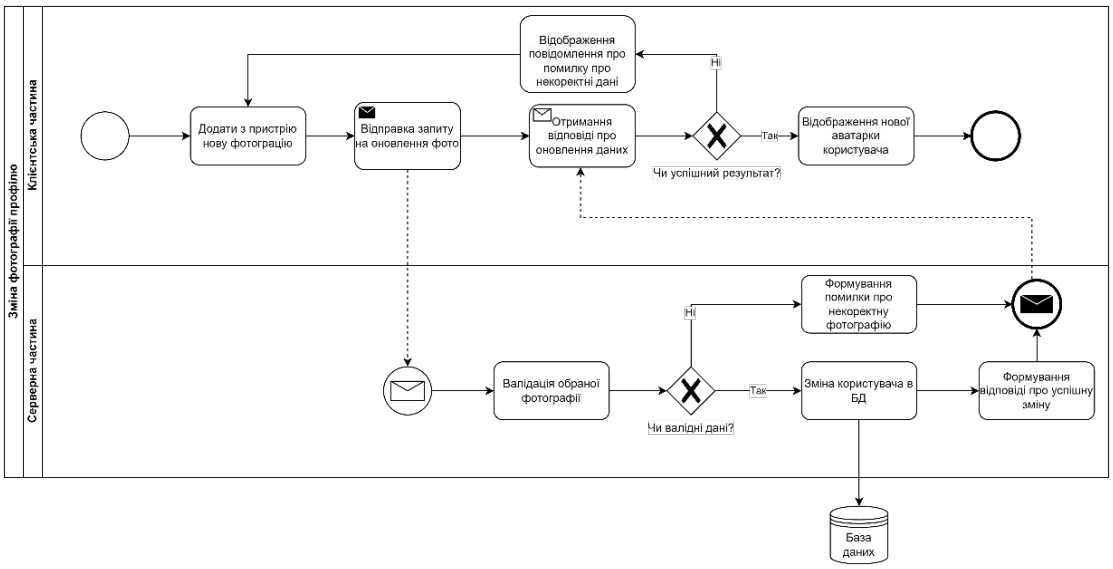


Рисунок 2.9 – Схема зміни фотографії профілю

Опис послідовності оцінювання:

– користувач переходить на сторінку профілю;
– користувач додає нову фотографію з пристрою;
– надсилається запит на сервер з новою фотографією;
– сервер завантажує фото на Cloudinary та отримує посилання, яке при успішності запиту використовує для оновлення даних користувача;
– за успішності повертається повідомлення про зміну фотографії;
– клієнтська сторона відображає нову фотографію користувача.

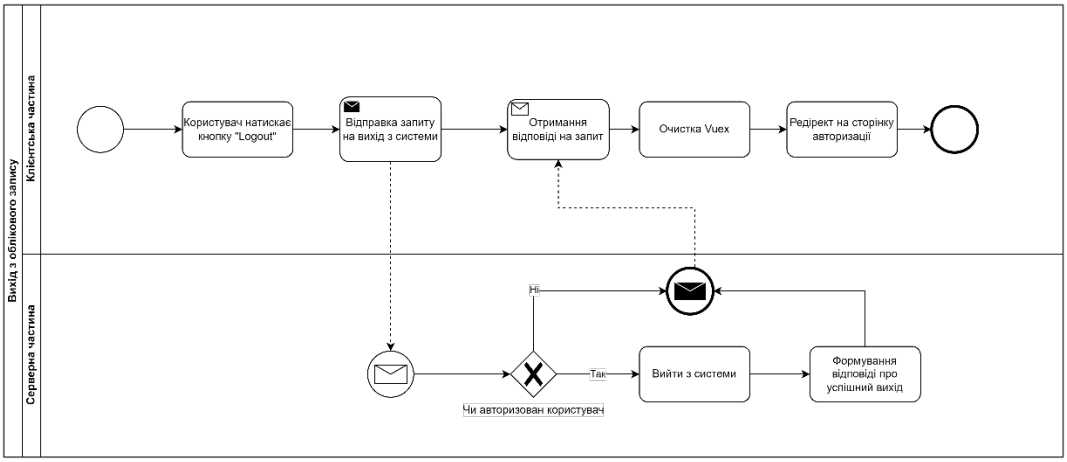


Рисунок 2.10 – Схема виходу з системи

Опис послідовності виходу з систем:

– користувач натискає кнопку «Logout»;
– клієнтська частина надсилає запит на вихід з системи до серверу;
– якщо користувач все ще авторизован, то система вийде зі запису;
– користувач буде перенаправлений на сторінку авторизації.

2.2 Архітектура програмного забезпечення

Так як вже було обговорено й обрано технології для клієнтською та серверної частин, веб-застосунок складається з двох основних частин та бази даних, то було обрано використання патерна Client-Server. На рисунку 2.11 наведено архітектурний шаблон, що використовується у веб-застосунку. Серверна частину реалізовано на платформі .Net за допомогою мови C#, клієнтська – за допомогою фреймворка Nuxt.js та база даних – SQL Server.

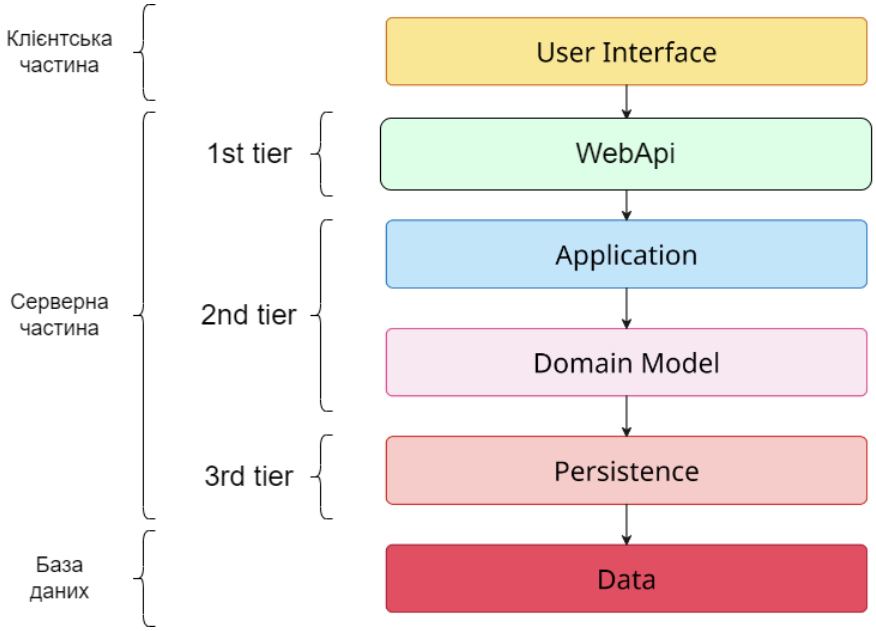


Рисунок 2.11 - Архітектурний шаблон веб-застосунку

За цим патерном фронтенд (клієнт) та бекенд (сервер) взаємодіють через мережу, де клієнтська частина звертається до сервера для отримання даних чи виконання певних операцій.

Щоб забезпечити модульність, чистоту коду та легкість супроводжуваність, серверна частина була поділена на різні слої, такі як API, Application, Domain Model та Persistence. Розглянемо детальніше:

API відповідає за надання API, через яке клієнтська частина взаємодіє з сервером. Він містить контролери, які обробляють HTTP-запити та викликають наступні елементи для обробки запиту.

Application, який ще називається Business Layer, містить бізнес-логіку застосунку, яка включає в себе обробку даних, валідацію та організацію сутностей.

Domain Model використовується для визначення основних сутностей, діаграма яких відображена на рисунку 2.12, та взаємодій між ними. Цей шар використовується як у Persistence так і у Application слоях.

Persistence, який ще називається Data Access Layer, служить для доступу до даних, включаючи модель даних та репозиторії, які виконують операції збереження та отримання даних.

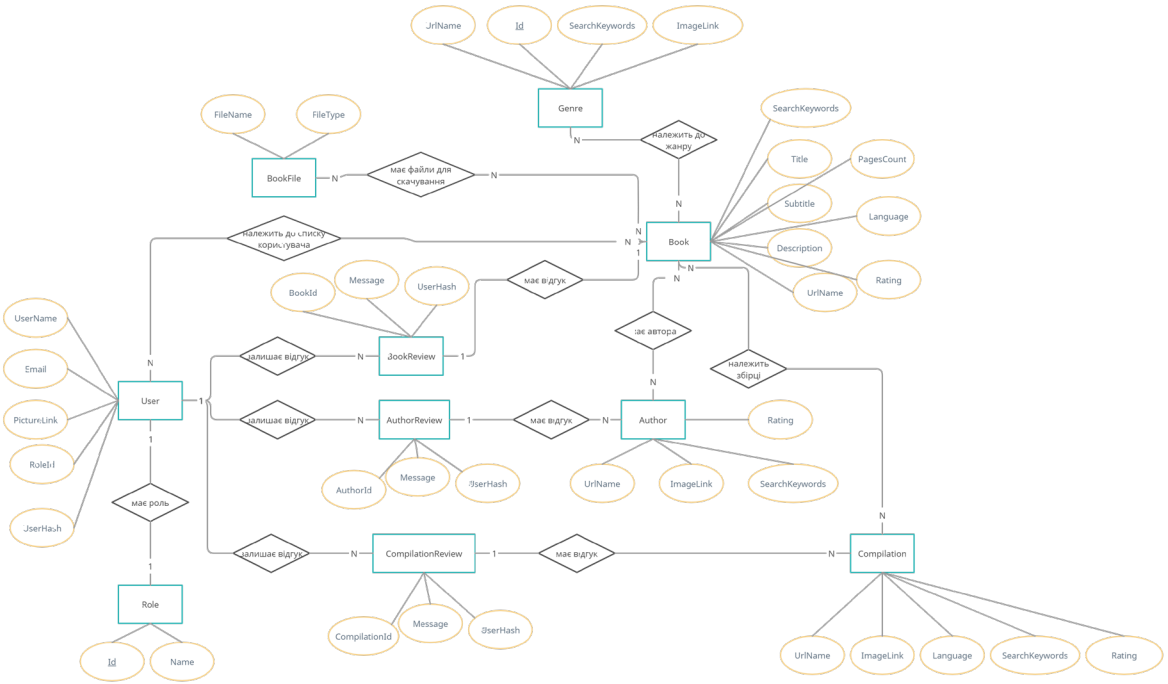


Рисунок 2.12 – ER діаграма сутностей онлайн бібліотеки

2.3 Конструювання програмного забезпечення

Клієнтська частина реалізується за допомогою фреймворку Nuxt.js, який має строгу структуру дерева файлів. Структуру папок можна побачити на рисунку 2.13, а опис призначення у таблиці 2.1.

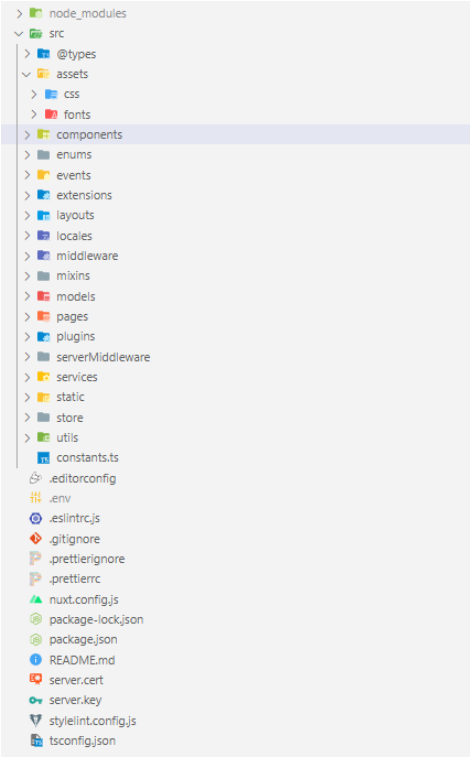


Рисунок 2.13 – Структура папок і файлів клієнтської частини

Таблиця 2.1 – Призначення кожної папки клієнтської частини

Назва папки	Призначення
assets	У папці assets зберігаються стилі CSS, шрифти
components	У папці components розміщуються Vue компоненти, які можуть бути використані в різних частинах проекту
layouts	В цій папці знаходяться файлові шаблони, які використовуються для оформлення загального макету сторінок
locales	У папці locales зберігаються локалізаційні файлі для підтримки англійської на української локалізації
middleware	В цій папці знаходяться middleware-функції, які будуть виконуватися перед завантаженням сторінки
models	Тут зберігаються усі класи та інтерфейси для роботи з API серверної частини.

Продовження таблиці 2.1

pages	Одна з основних папок проектів на Nuxt.js. Тут лежать окремі сторінки, які потім слугують для автоматичного рендерінгу роутінгу.
plugins	У цій папці розміщуються плагіни, які розширюють функціональність проєкту
services	Тут зберігаються файли сервіс, які відповідають за роботу з API серверної частини
static	У цій папці знаходяться файли, які доступні напряму з кореню сайту, фотографії чи якісь сервісні файли.
store	В папці store знаходиться Vuex файли який слугує для збереження та управління станом додатку.

Серверна частина не має настільки строгу структуру, але через розподілення на слої створені допоміжні бібліотеки(Class library), які виконують обов’язки певні частини застосунку. Структуру папок можна побачити на рисунку 2.14, а опис призначення у таблиці 2.2.

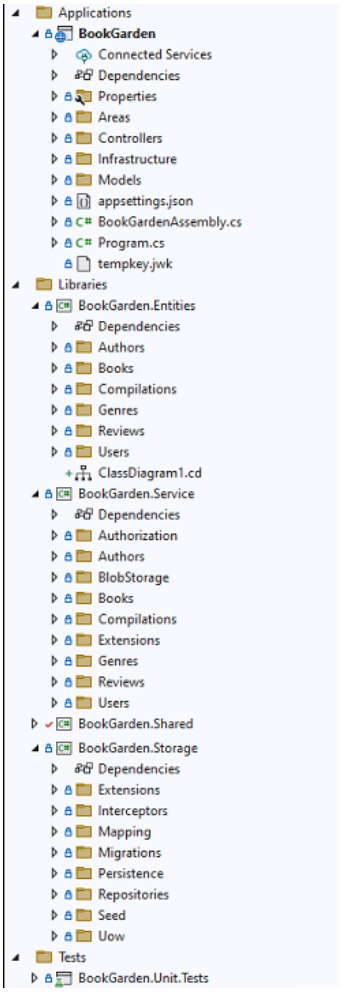


Рисунок 2.14 - Структура папок і файлів серверної частини

Таблиця 2.2 – Призначення проєктів серверної частини

Назва папки	Призначення
BookGarden(API)	Відповідає за надання API, через яке клієнтська частина взаємодіє з сервером за допомогою контролерів
Entities	Включає в себе сутності, які використовуються для отримання, збереження даних з БД або валідацію
Service	Містить бізнес-логіку додатку: сервіси, які виконують операції над даними, валідацію, обробку бізнес-правил та взаємодію з іншими компонентами системи
Shared	Містить загальний код, перераховування та утилітарні функції, які використовуються в різних частинах серверної частини

Продовження таблиці 2.2

Storage	Відповідає за взаємодію з базою даних. Він містить код для доступу до даних, збереження, оновлення та видалення даних у базі даних
Unit.Tests	Містить unit тести застосунку, які гарантують актуальність і вірність покритого функціоналу.

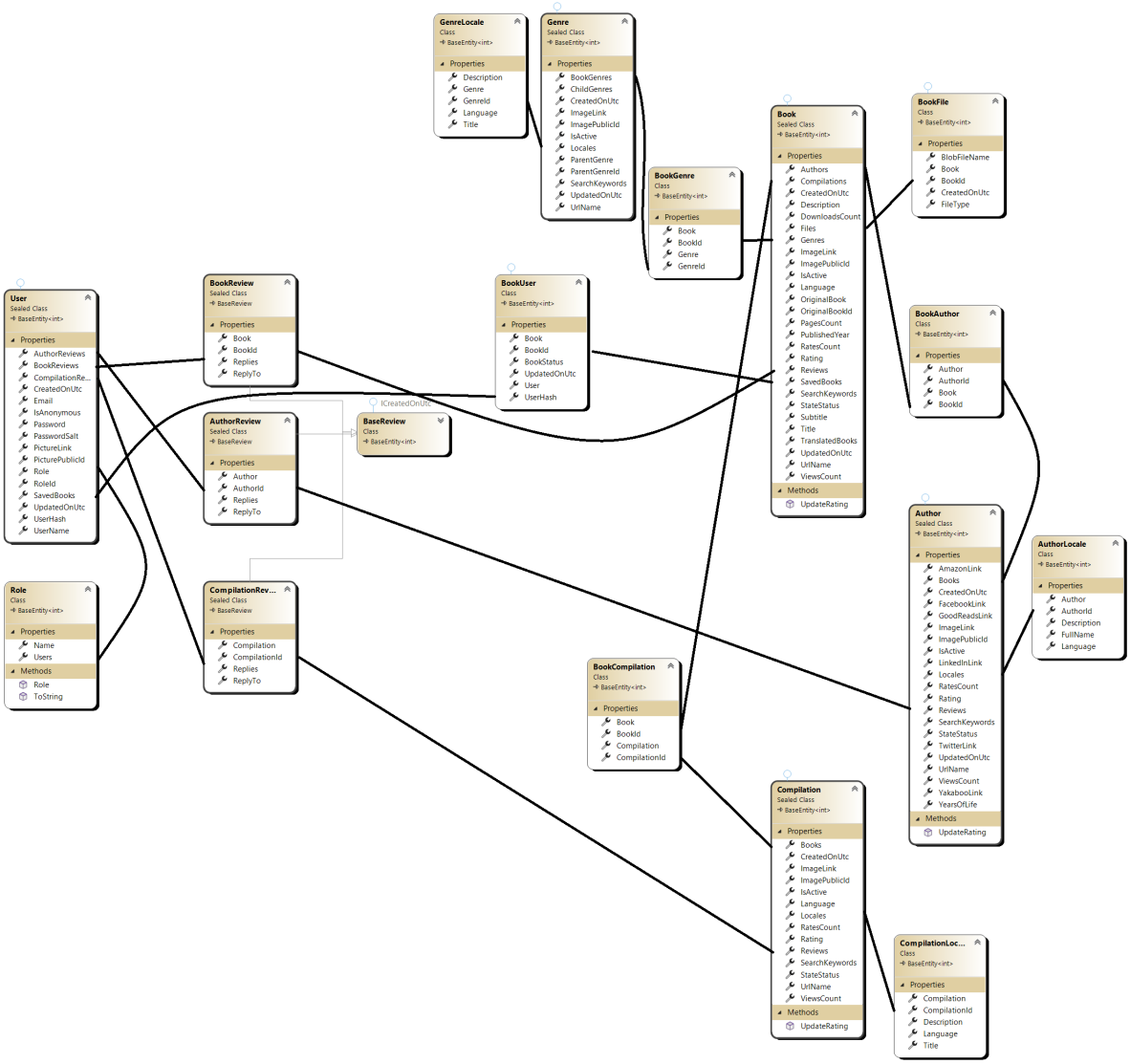


Рисунок 2.15 – Схема структурна класів

Клас User відповідає за створення та редагування даних про користувача та пов’язує відгуки та колекції.

Клас Role відповідає за позначення ролі користувача та включає в себе всіх користувачів відповідної ролі.

Метод ToString() – для отримання даних у форматі тексту

Клас BaseReview – виконує роль базового класу для відгуків, який потім буде використуватися при наслідування вже конкретних типів відгуків.

Класи BookReview, AuthorReview та CompilationReview – відповідають за створення та організацію відгуків на книжки, авторів та збірки відповідно. Унаслідують клас BaseReview.

Клас Book – використовується для створення, редагування книги та зв’язування її з авторами, жанрами та збірками.

Метод UpdateRating(int rating) – для перерахунку актуального рейтингу книжки.

Клас BookUser – виконує роль зв’язку книги й користувача для організації колекції.

Клас BookFile – використовується для створення формату для завантаження відповідної книжки.

Клас Genre – використовується для створення, редагування жанру та зв’язування його з книжками.

Клас BookGerne – виконує роль зв’язку книги й жанру

Клас GenreLocale – використовується для організації локалізованих даних про відповідний жанр

Клас Author – використовується для створення, редагування автора та зв’язування його з книжками.

Метод UpdateRating(int rating) – для перерахунку актуального рейтингу автора

Клас BookAuthor – виконує роль зв’язку книги й автора

Клас AuthorLocale – використовується для організації локалізованих даних про відповідного автора

Клас Compilation – використовується для створення, редагування збірки та зв’язування її з книжками.

Метод UpdateRating(int rating) – для перерахунку актуального рейтингу збірки

Клас BookCompilation – виконує роль зв’язку книги й збірки

Клас CompilationLocale – використовується для організації локалізованих даних про відповідну збірку

Аналіз системних вимог

У таблиці 2.3 описані системні вимоги для стабільної роботи застосунку

Таблиця 2.3 – Опис системних вимог

Назва вимоги	Рекомендовано
Тип процесору	Intel Pentium III, Celeron або AMD Athlon, з тактовою частотою 500 МГц
Об’єм ОЗП	2 ГБ
Швидкість підключення до мережі Інтернет	від 20 мегабіт
Браузер	Google Chrome версії 90+ Safari версії 14+ Firefox версії 88+
Операційна система	Windows, Linux, MacOS

В якості системи управління базами даних використовується MS SQL. База даних серверу призначена для зберігання книжок, авторів, жанрів, колекцій, користувачів та їх відгуків. Опис таблиць бази даних наведено у таблицях 2.4 - 2.20. Модель бази даних наведена на рисунку 2.16.

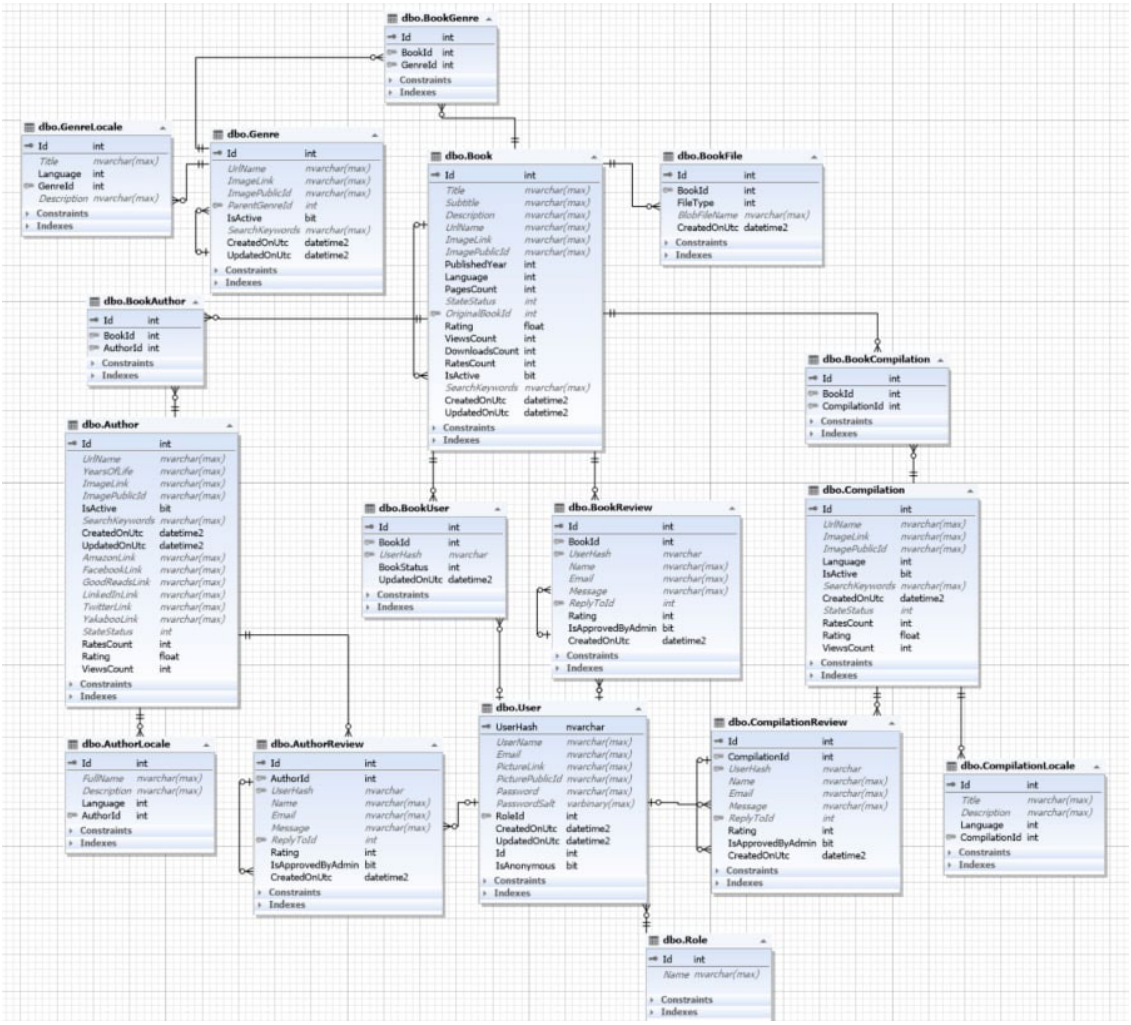


Рисунок 2.16 – Фізична модель бази даних

Таблиця 2.4 – Опис таблиці Author

Таблиця	Назва поля	Тип даних	Опис
Author	Id	Integer	Ідентифікаційний номер автора
	UrlName	String	Ім’я-посилання автора
	YearsOfLife	String	Роки життя автора
	ImageLink	String	Посилання на фотографію автора
	ImagePublicId	String	Публічний номер фотографії

Продовження таблиці 2.4

	IsActive	Boolean	Активність автора
	SearchKeywords	String	Ключові слова пошуку
	CreatedOnUtc	DateTime	Дата створення автора
	UpdatedOnUtc	DateTime	Дата останньої зміни
	AmazonLink	String	Посилання автора на Amazon
	FacebookLink	String	Посилання автора на Facebook
	GoodReadsLink	String	Посилання автора на GoodReads
	LinkedInLink	String	Посилання автора на LinkedIn
	TwitterLink	String	Посилання автора на Twitter
	YakabooLink	String	Посилання автора на Yakaboo
	StateStatus	Integer	Статус автора
	RatesCount	Integer	Кількість оцінок
	Rating	Float	Рейтинг від 1 до 5
	ViewsCount	Integer	Кількість переглядів

Таблиця 2.5 – Опис таблиці AuthorLocale

Таблиця	Назва поля	Тип даних	Опис
AuthorLocale	Id	Integer	Ідентифікаційний номер локалізації
	FullName	String	Ім’я автора
	Description	String	Опис автора
	Language	Integer	Мова локалізації
	AuthorId	Integer	Ідентифікаційний номер автора

Таблиця 2.6 – Опис таблиці AuthorReview

Таблиця	Назва поля	Тип даних	Опис
AuthorReview	Id	Integer	Ідентифікаційний номер відгуку
	AuthorId	Integer	Ідентифікаційний номер автора
	UserHash	String	Хек-ключ користувача
	Name	String	Ім’я анонімного відгуку
	Email	String	Пошта анонімного відгуку
	Message	String	Текст відгуку
	ReplyToId	Integer	Ідентифікаційний номер відгука на який відповів

Продовження таблиці 2.6

	Rating	Integer	Рейтинг від 1 до 5
	IsApprovedByAdmin	Boolean	Чи дозволена публікація адміном
	CreatedOnUtc	DateTime	Дата публікації

Таблиця 2.7 – Опис таблиці Book

Таблиця	Назва поля	Тип даних	Опис
Book	Id	Integer	Ідентифікаційний номер книжки
	Title	String	Назва книжки
	Subtitle	String	Підназва книжки
	Description	String	Опис книжки
	UrlName	String	Ім’я-посилання книжки
	ImageLink	String	Посилання на фото обкладинки
	ImagePublicId	String	Публічний номер фотографії
	PublishedYear	Integer	Рік публікації книжки
	Language	Integer	Мова книжки
	PagesCount	Integer	Кількість сторінок
	StateStatus	Integer	Статус книжки

Продовження таблиці 2.7

	OriginalBookId	Integer	Ідентифікаційний номер оригінальної книжки
	Rating	Float	Рейтинг від 1 до 5
	ViewsCount	Integer	Кількість переглядів
	DownloadsCount	Integer	Кількість скачувань
	RatesCount	Integer	Кількість оцінок
	IsActive	Boolean	Активність книжки
	SearchKeywords	String	Ключові слова пошуку
	CreatedOnUtc	DateTime	Дата створення

Таблиця 2.8 – Опис таблиці BookAuthor

Таблиця	Назва поля	Тип даних	Опис
BookAuthor	Id	Integer	Ідентифікаційний номер
	BookId	Integer	Ідентифікаційний номер книжки
	AuthorId	Integer	Ідентифікаційний номер автора

Таблиця 2.9 – Опис таблиці BookCompilation

Таблиця	Назва поля	Тип даних	Опис
BookCompilation	Id	Integer	Ідентифікаційний номер

Продовження таблиці 2.9

	BookId	Integer	Ідентифікаційний номер книжки
	CompilationId	Integer	Ідентифікаційний номер збірки

Таблиця 2.10 – Опис таблиці BookFile

Таблиця	Назва поля	Тип даних	Опис
BookFile	Id	Integer	Ідентифікаційний номер
	BookId	Integer	Ідентифікаційний номер книжки
	FileType	Integer	Формат файлу
	BlobFileName	String	Ім’я файлу в хмарі
	CreatedOnUtc	DateTime	Дата створення

Таблиця 2.11 – Опис таблиці BookGenre

Таблиця	Назва поля	Тип даних	Опис
BookGenre	Id	Integer	Ідентифікаційний номер
	BookId	Integer	Ідентифікаційний номер книжки
	GenreId	Integer	Ідентифікаційний номер жанру

Таблиця 2.12 – Опис таблиці BookReview

Таблиця	Назва поля	Тип даних	Опис
BookReview	Id	Integer	Ідентифікаційний номер відгуку
	BookId	Integer	Ідентифікаційний номер книжки
	UserHash	String	Хек-ключ користувача
	Name	String	Ім’я анонімного відгуку
	Email	String	Пошта анонімного відгуку
	Message	String	Текст відгуку
	ReplyToId	Integer	Ідентифікаційний номер відгука на який відповів
	Rating	Integer	Рейтинг від 1 до 5
	IsApprovedByAdmin	Boolean	Чи дозволена публікація адміністратором
	CreatedOnUtc	DateTime	Дата публікації

Таблиця 2.13 – Опис таблиці BookUser

Таблиця	Назва поля	Тип даних	Опис
BookUser	Id	Integer	Ідентифікаційний номер
	BookId	Integer	Ідентифікаційний номер книжки

Продовження таблиці 2.13

	UserHash	String	Хеш-ключ користувача
	BookStatus	Integer	Тип колекції
	UpdatedOnUtc	DateTime	Дата останньої зміни

Таблиця 2.14 – Опис таблиці Compilation

Таблиця	Назва поля	Тип даних	Опис
Compilation	Id	Integer	Ідентифікаційний номер збірки
	UrlName	String	Ім’я-посилання збірки
	ImageLink	String	Посилання на фотографію збірки
	ImagePublicId	String	Публічний номер фотографії
	Language	Integer	Мова книжок у збірці
	IsActive	Boolean	Активність збірки
	SearchKeywords	String	Ключові слова пошуку
	CreatedOnUtc	DateTime	Дата створення
	StateStatus	Integer	Статус збірки
	RatesCount	Integer	Кількість оцінок
	Rating	Float	Рейтинг від 1 до 5
	ViewsCount	Integer	Кількість переглядів

Таблиця 2.15 – Опис таблиці CompilationLocale

Таблиця	Назва поля	Тип даних	Опис
CompilationLocale	Id	Integer	Ідентифікаційний номер локалізації
	Title	String	Назва збірки
	Description	String	Опис збірки
	Language	Integer	Мова локалізації
	CompilationId	Integer	Ідентифікаційний номер збірки

Таблиця 2.16 – Опис таблиці CompilationReview

Таблиця	Назва поля	Тип даних	Опис
CompilationReview	Id	Integer	Ідентифікаційний номер відгуку
	CompilationId	Integer	Ідентифікаційний номер збірки
	UserHash	String	Хек-ключ користувача
	Name	String	Ім’я анонімного відгуку
	Email	String	Пошта анонімного відгуку
	Message	String	Текст відгуку
	ReplyToId	Integer	Ідентифікаційний номер відгука на який відповів

Продовження таблиці 2.16

	Rating	Integer	Рейтинг від 1 до 5
	IsApprovedByAdmin	Boolean	Чи дозволена публікація адміністратором
	CreatedOnUtc	DateTime	Дата публікації

Таблиця 2.17 – Опис таблиці Genre

Таблиця	Назва поля	Тип даних	Опис
Genre	Id	Integer	Ідентифікаційний номер жанру
	UrlName	String	Ім’я-посилання жанру
	ImageLink	String	Посилання на фотографію жанру
	ImagePublicId	String	Публічний номер фотографії
	ParentGenreId	Integer	Ідентифікаційний номер головного жанру
	IsActive	Boolean	Активність жанру
	SearchKeywords	String	Ключові слова пошуку
	CreatedOnUtc	DateTime	Дата створення
	UpdatedOnUtc	DateTime	Дата останньої зміни

Таблиця 2.18 – Опис таблиці GenreLocale

Таблиця	Назва поля	Тип даних	Опис
GenreLocale	Id	Integer	Ідентифікаційний номер локалізації
	Title	String	Назва жанру
	Language	Integer	Мова локалізації
	GenreId	Integer	Ідентифікаційний номер жанру
	Description	String	Опис жанру

Таблиця 2.19 – Опис таблиці Role

Таблиця	Назва поля	Тип даних	Опис
Role	Id	Integer	Ідентифікаційний номер ролі
	Name	String	Назва ролі

Таблиця 2.20 – Опис таблиці User

Таблиця	Назва поля	Тип даних	Опис
User	UserHash	String	Хеш-ключ користувача
	UserName	String	Ім’я користувача
	Email	String	Електронна пошта
	PictureLink	String	Посилання на фотографію

Продовження таблиці 2.20

	PicturePublicId	String	Публічний номер фотографії
	Password	String	Захешований пароль
	PasswordSalt	HexString	Додаткове значення для хешування
	RoleId	Integer	Ідентифікаційний номер ролі користувача
	CreatedOnUtc	DateTime	Дата створення
	UpdatedOnUtc	DateTime	Дата останньої зміни
	Id	Integer	Ідентифікаційний номер користувача
	IsAnonymous	Boolean	Чи користувач гість

Опис основних NuGet пакетів, що використовуються у розробці продемонстровано в таблиці 2.21

Таблиця 2.21 – Опис застосування NuGet пакетів

№ п/п	Назва пакету	Опис застосування
1	AutoMapper	Спрощує роботу з копіюванням значень між різними моделями різних слоїв серверної частини.
2	CloudinaryDotNet	Використовується для організації, збереження та отримання фотографій користувачів у зовнішньому сервісі Cloudinary.

Продовження таблиці 2.21

3	Azure.Storage.Blobs	Використовується для організації, збереження та отримання файлів для скачування користувачами у зовнішньому сервісі Azure Storage.

Опис утиліт що використовується у розробці наведено в таблиці 2.22.

Таблиця 2.22 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Visual Studio	Головне середовище розробки серверної частини веб-застосунку.
3	Microsoft Visual Studio Code	Середовище розробки клієнтської частини веб-застосунку.
4	Figma	Програмне забезпечення для створення темплейтів сторінок веб-застосунку
5	Microsoft SQL Server Management Studio	Середовище для управління та адміністрування бази даних Microsoft SQL Server.
6	Microsoft Azure Storage Explorer	Інструмент для управління та взаємодії зі сховищами даних Azure, а саме файлів книжок.

2.4 Аналіз безпеки даних

Хешування паролів:

Паролі користувачів зберігаються у захешованому вигляді. Використання хеш-функцій забезпечує безпеку паролів, оскільки оригінальний пароль неможливо відновити з хешу. При авторизації порівнюються хеші, а не самі паролі.

Доступ до файлів:

Доступ до файлів неможливий без запиту від клієнтської частини, тобто користувач не отримує прямого посилання на файл та ім’я файлу, при переході на посилання завантаження користувач отримує тільки файл, що гарантує безпеку файловій системі та неможливість атаки на неї

Висновки до розділу

У другому розділі було проведено аналіз та розроблено моделі, що стосуються архітектури та конструювання програмного забезпечення.

Початково було застосовано модель BPMN для опису бізнес-процесу програмного забезпечення, що дозволяє зрозуміти послідовність та взаємозв'язки між різними етапами розробки та покроково зрозуміти основний функціонал.

У контексті архітектури програмного забезпечення було розглянуто архітектурні рішення та було обрано патерн Client-Server, оскільки веб-застосунок складається з клієнтської та серверної частини.

Конструювання програмного забезпечення включає опис структури папок клієнтської та серверної частин, що сприяє організації коду та управлінню проектом. Була розроблена схема структурних класів, що дозволяє зрозуміти ієрархію та залежності між класами у системі.

Для бази даних була створена фізична модель з детальним описом кожної таблиці, що визначає структуру та взаємозв'язки даних, які зберігаються в системі.

Також було проведено опис та розглянуто використані NuGet пакети, що використовуються у веб-застосунку, а також були наведені описи утиліт, що використовуються під час розробки.

Ці кроки з моделювання та конструювання програмного забезпечення є важливими етапами у розробці веб-застосунку онлайн бібліотеки, оскільки вони допомагають визначити структуру, архітектуру та організацію проекту, а також забезпечують належний рівень безпеки даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Написання автоматизованих автоматизованих сценаріїв тестування та покриття програмного коду тестами надає можливість покращити якість ПЗ.

Оцінювалися такі моменти програмного забезпечення:

– функціональність: ПЗ виконує свої основні функції, а серверна та клієнтська частини працюють належним чином, вчасно повідомляючи про помилки вхідних даних користувача;

– ефективність: за допомогою розробницької консолі Google Chrome було виміряно час генерації базових сторінок застосунку. За 20 секунд активного використання застосунку, бо згенеровано 11 сторінок, середній час отримання даних – 1.4 с, генерації – 0.3 с, відображення стилів – 0.1 с;

– зручність використання: користуватися ПЗ зручно та інтуїтивно зрозуміло через приємний та сучасний інтерфейс, що відображає користувачу усі дії та процеси для зрозумілої комунікації між користувачем та продуктом;

– сумісність: продукт є веб-застосунком та є сумісний з популярними браузерами;

– покриття тестами: частина коду, яка відповідає за операції створення, редагування та видалення контенту були покриті тестами на 100%. Також було покрито основні методи локалізації застосунку;

– підтримка працездатності: багатошарова архітектура та розбиття відповідальностей надає можливість легко підтримувати основний функціонал та виправлення помилок.

3.2 Опис процесів тестування

Було використано обидва типи тестування: ручне та автоматичне. Автоматичне було використано для покриття серверної реалізації, а саме частини, яка відповідає за CRUD операції з контентом, дерево тестів відображено на рисунку 3.1, розгорнуте та детальне дерево тестів наведено у додатку А.

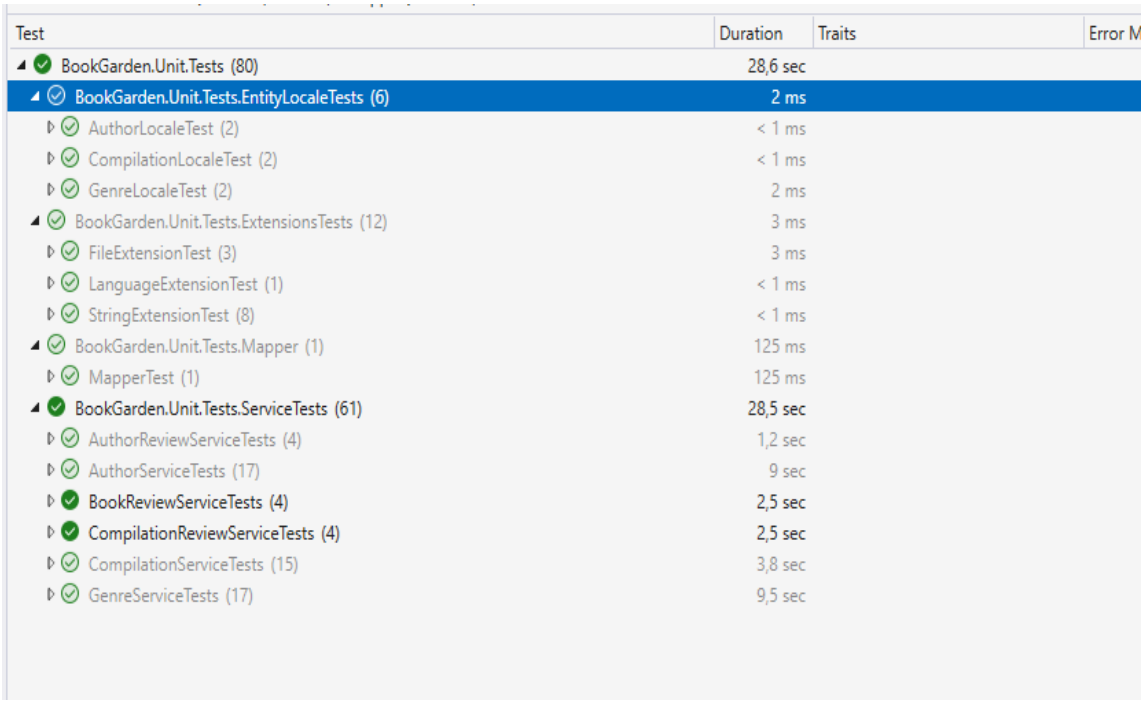


Рисунок 3.1 – Дерево автоматизованих тестів

Ручне тестування використовувалося для тестування клієнтської частини, а саме взаємодії між користувачем та додатком, опис відповідних тестів наведено у таблицях 3.1 – 3.8.

Таблиця 3.1 – Тест 1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації

Продовження таблиці 3.1

Вхідні дані	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 6 до 64 символів, який відповідає правилам паролю, підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього натискається кнопка реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.

Таблиця 3.2 – Тест 2

Тест	Авторизація користувача
Модуль	Авторизація користувача
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці авторизація
Вхідні дані	Електронна пошта та пароль
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка була зареєстрована в системі та пароль від 6 до 64 символів, який було використано при регістрації. Після цього натискається кнопка авторизації.
Очікуваний результат	Авторизація проходить успішно, користувач входить у обліковий запис і перенаправляється на головну сторінку
Фактичний результат	Авторизація проходить успішно, користувач входить у обліковий запис і перенаправляється на головну сторінку

Таблиця 3.3 – Тест 3

Тест	Зміна відображення елементів
Модуль	Відображення елементів
Номер тесту	1.3
Початковий стан системи	Користувач переглядає книжки
Вхідні дані	-
Опис проведення тесту	Користувач клікає на іконку "Список" у верхній частині відображення у блоці з сортуванням та пагінацією
Очікуваний результат	Система змінює відображення елементів на стиль списку. Користувач спостерігає змінений спосіб відображення елементів у стилі списку
Фактичний результат	Система змінює відображення елементів на стиль списку. Користувач спостерігає змінений спосіб відображення елементів у стилі списку

Таблиця 3.4 – Тест 4

Тест	Зміна локалізації
Модуль	Відображення елементів
Номер тесту	1.4
Початковий стан системи	Користувач переглядає контент застосунку
Вхідні дані	-
Опис проведення тесту	Користувач вибирає бажану мову локалізації у меню застосунку, українську, російську або англійську

Продовження таблиці 3.4

Очікуваний результат	Система застосовує зміни і оновлює інтерфейс веб-застосунку у вибраній мові локалізації. Користувач бачить змінений інтерфейс у вибраній мові локалізації.
Фактичний результат	Система застосовує зміни і оновлює інтерфейс веб-застосунку у вибраній мові локалізації. Користувач бачить змінений інтерфейс у вибраній мові локалізації.

Таблиця 3.5 – Тест 5

Тест	Пошук книжки
Модуль	Перегляд контенту
Номер тесту	1.5
Початковий стан системи	Користувач переглядає контент застосунку
Вхідні дані	Ключові слова пошуку
Опис проведення тесту	Користувач вводить ключові слова, які наявні у як мінімум одній книжці в системі, у поле пошуку у верхній частині сторінки
Очікуваний результат	Система відображає книжки, які відповідають критеріям пошуку. Користувач обирає потрібну йому книжку
Фактичний результат	Система відображає книжки, які відповідають критеріям пошуку. Користувач обирає потрібну йому книжку.

Таблиця 3.6 – Тест 6

Тест	Завантаження книжки у певному форматі
Модуль	Завантаження файлу книжки
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на сторінці певної книжки, яка має файли для завантаження

Продовження таблиці 3.6

Вхідні дані	-
Опис проведення тесту	Користувач переглядає доступні формати, у яких можна завантажити книжку. Користувач обирає бажаний формат для завантаження
Очікуваний результат	Система генерує файл книжки у вибраному форматі, який автоматично завантажується на стороні користувача через 10 секунд
Фактичний результат	Система генерує файл книжки у вибраному форматі, який автоматично завантажується на стороні користувача через 10 секунд

Таблиця 3.7 – Тест 7

Тест	Оцінювання книжки по рейтингу від 1 до 5
Модуль	Оцінювання книжки
Номер тесту	1.7
Початковий стан системи	Користувач знаходиться на сторінці певної книжки, яка не була оцінена
Вхідні дані	Оцінка користувача
Опис проведення тесту	Користувач обирає бажану оцінку для книжки на шкалі від 1 до 5 у блоці для оцінювання під фотографією обкладинки книжки
Очікуваний результат	Система оновлює рейтинг та зберігає оцінку користувача, відображаючи її під фотографією
Фактичний результат	Система оновлює рейтинг та зберігає оцінку користувача, відображаючи її під фотографією

Таблиця 3.8 – Тест 8

Тест	Додавання книжки до власної колекції
Модуль	Додавання книжки до власної колекції
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на сторінці певної книжки, яка не належить відповідній колекції
Вхідні дані	-
Опис проведення тесту	Користувач натискає на «+» біля кнопки читати онлайн, та у списку обирає колекцію у яку хоче додати книжку та клікає на неї
Очікуваний результат	Система додає книжку до колекції, та відображає наявність даної книжки в обраній колекції

3.3 Опис контрольного прикладу

У таблицях 3.9 – 3.12 описано мануальне тестування процесу публікації відгуку, а саме від гостя та від авторизованого користувача.

Таблиця 3.9 – Тест 9

Тест	Публікація відгуку авторизованим користувачем
Модуль	Публікація відгука
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на сторінці певної книжки, яка не була оцінена
Вхідні дані	Оцінка або відгук
Опис проведення тесту	Користувач переходить до блоку відгуків, вводить відгук довжиною більше 25 та менше 5000 символів та підтверджує публікацію відгука.
Очікуваний результат	Система не дозволяє публікацію через необрану оцінку

Продовження таблиці 3.9

Фактичний результат	Система не дозволяє публікацію блокуючи натискання кнопки для підтвердження, не активна кнопка зображена на рисунку 3.2 та 3.3.

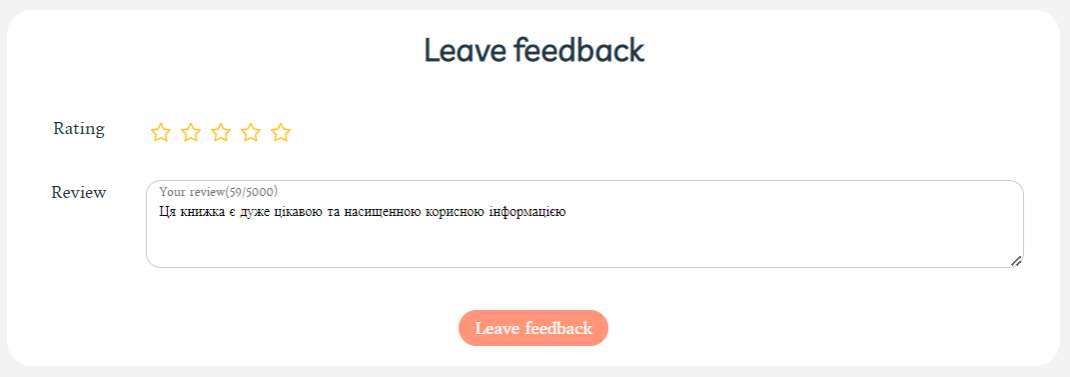


Рисунок 3.2 – Неактивність кнопки при необраній оцінці

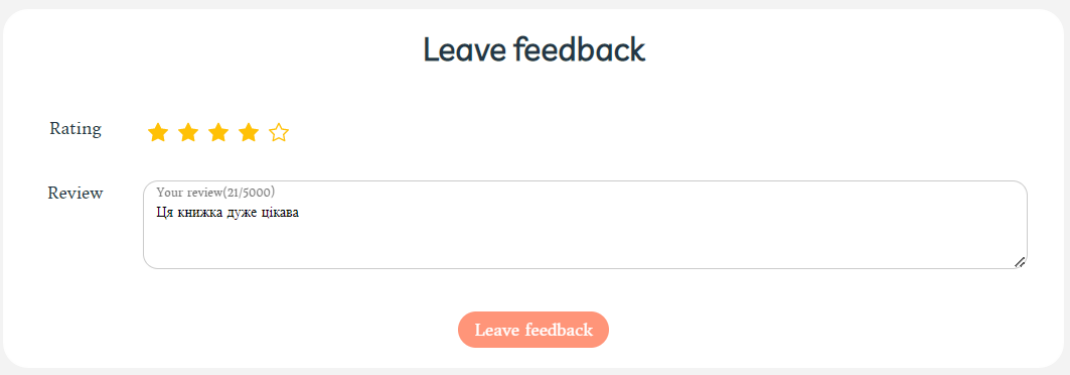


Рисунок 3.3 – Неактивність кнопки при незадовільній довжині

Таблиця 3.10 – Тест 10

Тест	Публікація відгуку авторизованим користувачем
Модуль	Публікація відгука
Номер тесту	1.9
Початковий стан системи	Користувач знаходиться на сторінці певної книжки, яка не була оцінена
Вхідні дані	Оцінка та відгук

Продовження таблиці 3.10

Опис проведення тесту	Користувач переходить до блоку відгуків, обирає оцінку, вводить відгук довжиною більше 25 та менше 5000 символів та підтверджує публікацію відгука.
Очікуваний результат	Система дозволяє публікацію(рис 3.4), зберігає відгук та надсилає адміністратору запит на підтвердження
Фактичний результат	Система дозволяє публікацію, зберігає відгук та надсилає адміністратору запит на підтвердження

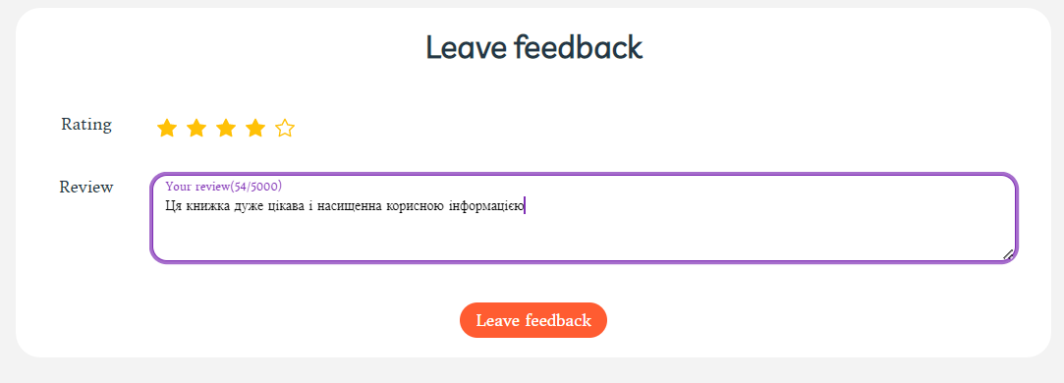


Рисунок 3.4 – Успішна валідація та можливість публікації

Таблиця 3.11 – Тест 11

Тест	Затвердження відгуку адміністратором
Модуль	Публікація відгука
Номер тесту	1.10
Початковий стан системи	Користувач авторизован та має роль адміністратора, знаходиться на сторінці відгуків на книжки, де відображено попередній відгук
Вхідні дані	-
Опис проведення тесту	Адміністратор передивляється контент відгуку та натискає кнопку у колонці «Затверджений», таблицю відображено на рисунку 3.5
Очікуваний результат	Відгук отримує статус затвердженого(рис 3.6) та відображається на сторінці відовідної книжки(рис 3.7)

Продовження таблиці 3.11

Фактичний результат	Відгук отримує статус затвердженого та відображається на сторінці відповідної книжки

Рисунок 3.5 – Не затверджений відгук користувача

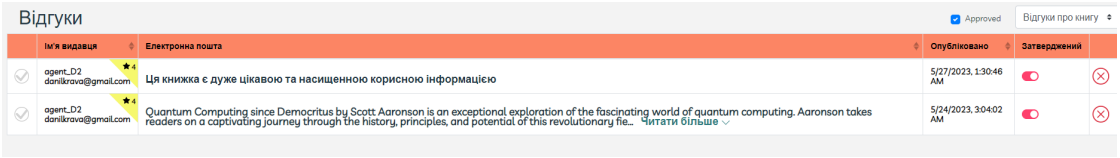


Рисунок 3.6 – Затверджений відгук користувача

Таблиця 3.12 – Тест 12

Тест	Публікація не першого відгука на одну книжку
Модуль	Публікація відгука
Номер тесту	1.11
Початковий стан системи	Користувач знаходиться на сторінці певної книжки, яка вже була оцінена або було опубліковано відгук від поточного користувача
Вхідні дані	Відгук
Опис проведення тесту	Користувач переходить до блоку відгуків, вводить відгук довжиною більше 25 та менше 5000 символів та підтверджує публікацію відгука.
Очікуваний результат	Система дозволяє публікацію, зберігає відгук використовуючи попередню оцінку(рис 3.7) та надсилає адміністратору запит на підтвердження
Фактичний результат	Система дозволяє публікацію, зберігає відгук використовуючи попередню оцінку та надсилає адміністратору запит на підтвердження

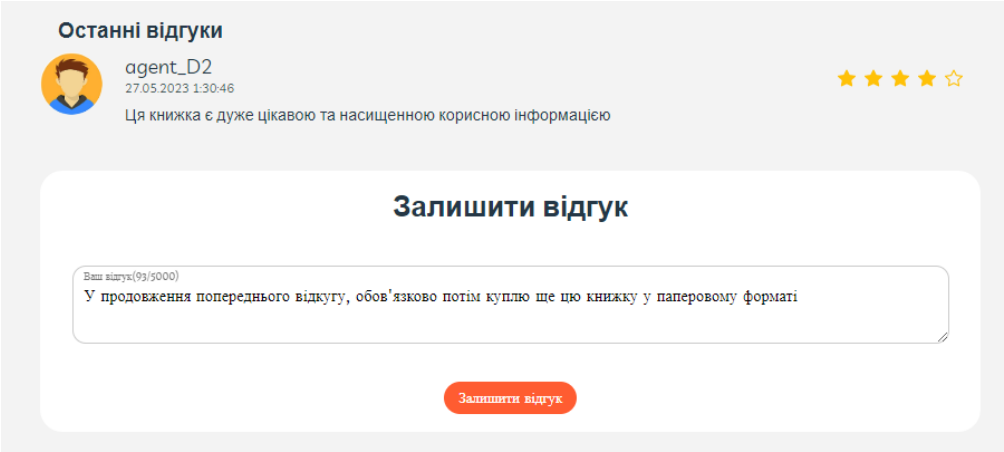


Рисунок 3.7 – Публікація не першого відгуку

Таблиця 3.13 – Тест 13

Тест	Публікація анонімного відгука
Модуль	Публікація відгука
Номер тесту	1.12
Початковий стан системи	Гість знаходиться на сторінці певної книжки
Вхідні дані	Ім’я, електронна пошта, оцінку та відгук
Опис проведення тесту	Користувач переходить до блоку відгуків, вводить ім’я, валідну електронну пошту та відгук довжиною більше 25 та менше 5000 символів та підтверджує публікацію відгука.
Очікуваний результат	Система дозволяє публікацію(рис 3.8), зберігає відгук та надсилає адміністратору запит на підтвердження. А після підтвердження адміністратором відображається з введеним іменем та анонімною фотографією, як на рис 3.9
Фактичний результат	Система дозволяє публікацію, зберігає відгук та надсилає адміністратору запит на підтвердження. А після підтвердження адміністратором відображається з введеним іменем та анонімною фотографією, як на рис 3.9

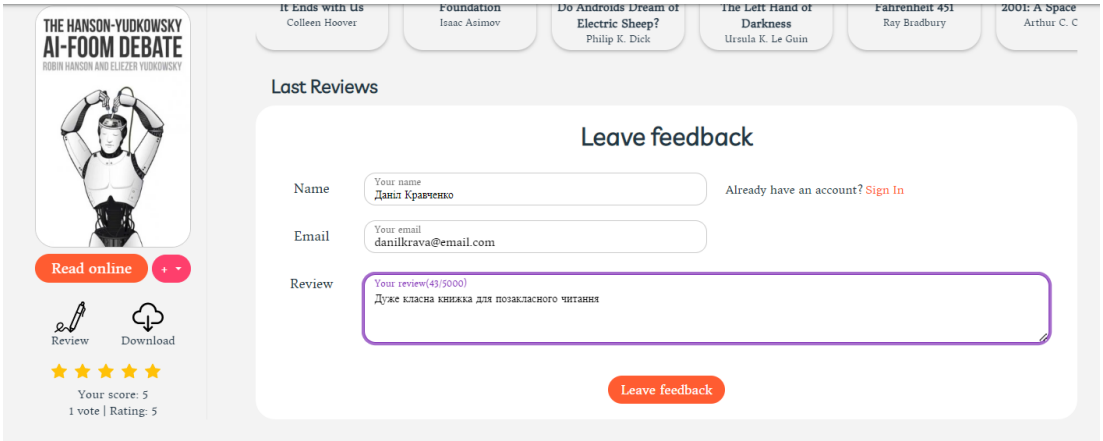


Рисунок 3.8 – Публікація анонімного відгуку

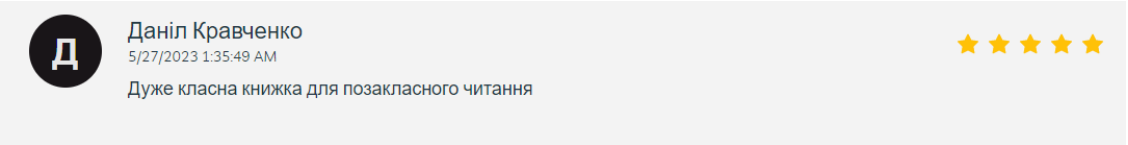


Рисунок 3.9 – Наявність анонімного відгуку

Висновки до розділу

У третьому розділі було проведено аналіз якості програмного забезпечення з використанням таких метрик як: функціональність, ефективність, зручність використання, сумісність, покриття тестами та підтримка працездатності. В рамках цього аналізу було розглянуто як автоматизоване, так і мануальне тестування. Було описано процеси тестування, включаючи опис контрольного прикладу.

Особлива увага була приділена ручному тестуванню основного функціоналу. Було проведено детальний аналіз функціоналу публікації відгука на книжку з урахуванням дій анонімних та авторизованих користувачів. Що пройшло перевірку на працездатність основний функціонал користувача та які обмеження і функціональні можливості доступні в різних режимах.

Аналіз ПЗ показав, що основний функціонал, зокрема функціонал публікації відгука на книжку, працює належним чином і задовольняє вимоги користувачів. Тестування дозволило виявити та усунути деякі помилки та недоліки, що сприяло покращенню якості ПЗ.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Серверна та клієнтська частини хоч і існують обидві разом, але вони реалізовані за допомогою критично різних технологій, що дозволяє розгорнути різні частини на різних сервісах.

Так як серверна частина використовує стек технологій Azure, то було вирішено розгорнути її за допомогою Azure App Service. Для початку необхідно створити SQL Server та базу даних, результат успішного створення зображено на рисунках 4.1 та 4.2.

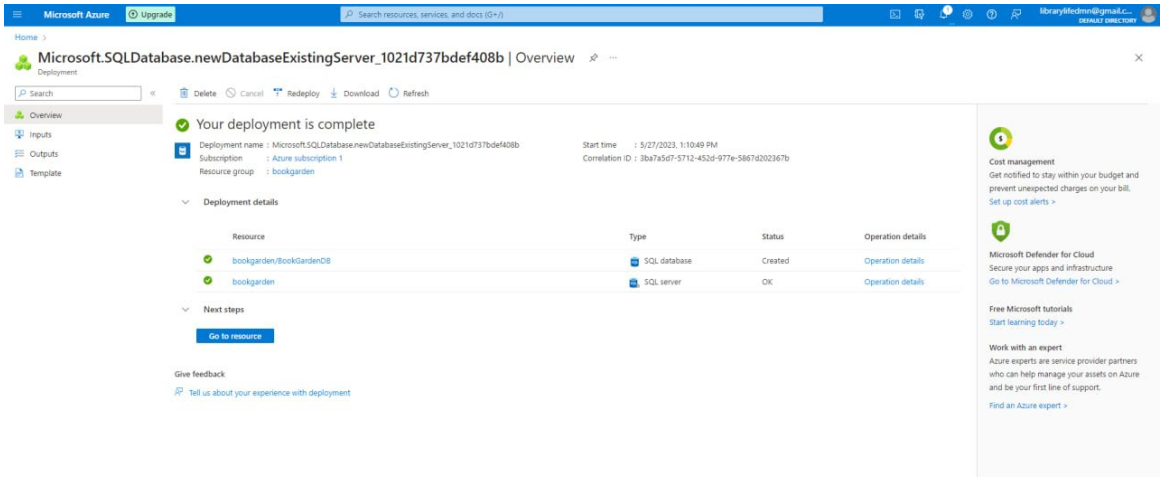


Рисунок 4.1 – Успішне створення база даних

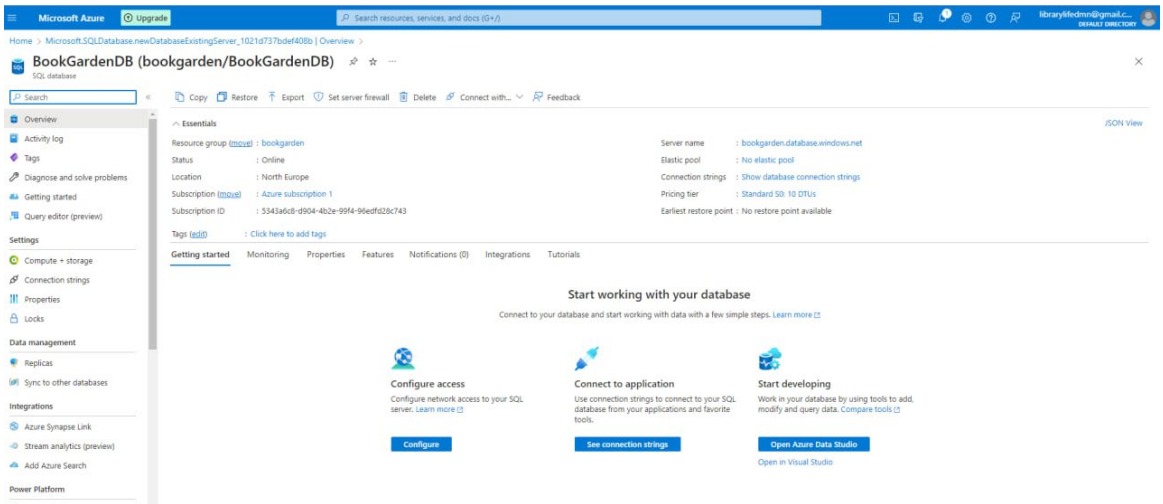


Рисунок 4.2 – Детальна інформація про створену базу даних

Тепер необхідно скопіювати текст підключення до бази даних та вставити його у appsetting.json у блок Connection:Source. Тепер необхідно опублікувати, натискаємо правою кнопкою миші на проект API у браузері рішень і обираємо пункт «Опублікувати» і покроково створити усе необхідне:

1. Натискаємо «New profile» для створення конфігурацій розгортання
2. Обираємо «Azure» та «Azure App Service (Linux)»
3. Тепер необхідно створити нову App Service, можна використати дані за замовчуванням або ввести власні
4. Тепер необхідно створити нову API Management
5. Оберемо звичайну публікацію та почекаємо на розгортання, яке може зайняти декілька хвилин, після цього з'явиться екран із інформацією про публікацію (рисунок 4.3).

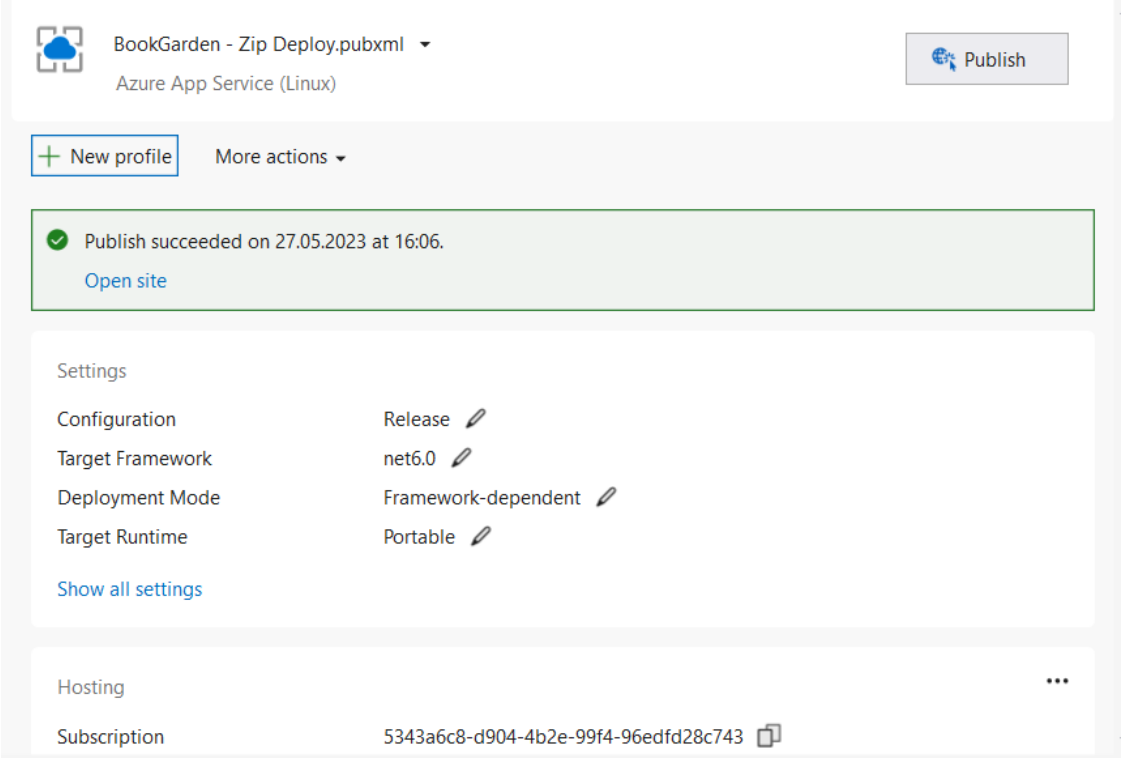


Рисунок 4.3 – Успішне розгортання серверної частини

Перейдемо за посиланням, та переглянемо сторінку Swagger, як результат можемо впевнитись у працездатності серверної частини та переглянути усі можливі запити на рисунку 4.4.

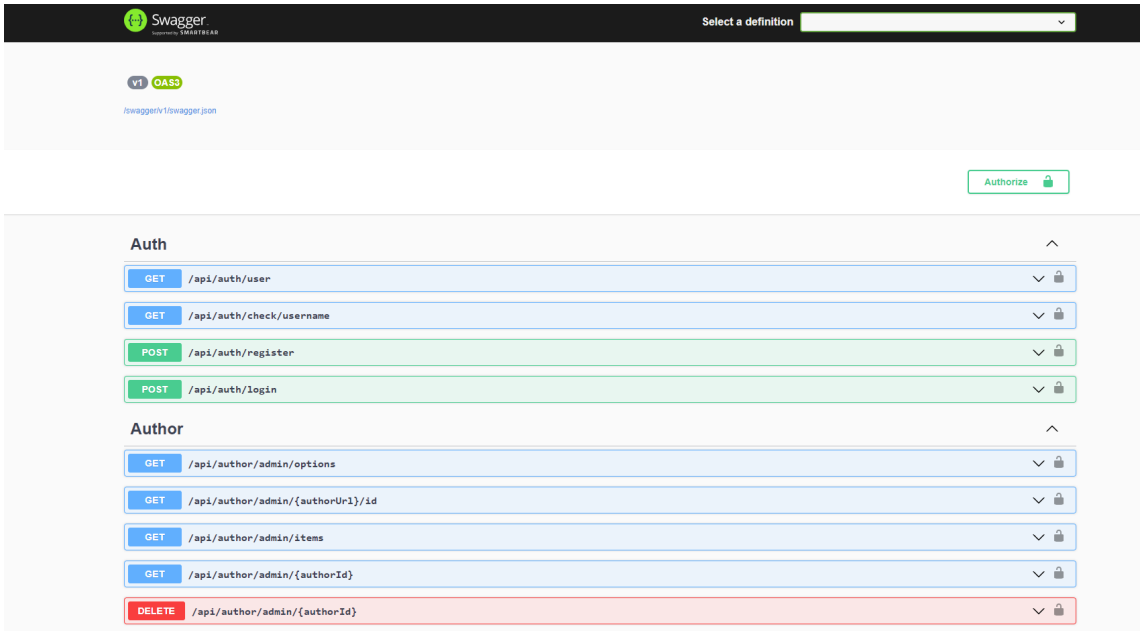


Рисунок 4.4 – Swagger сторінка серверної частини

Тепер перейдемо до розгортання клієнтської частини, для цього використовуємо сервіс Vercel для зручної публікації front-end частини, так як Nuxt використовує SRR, то небхідно сконфігурувати розгортання за допомогою команди «nuxt generate», що зображено на рисунку 4.5.

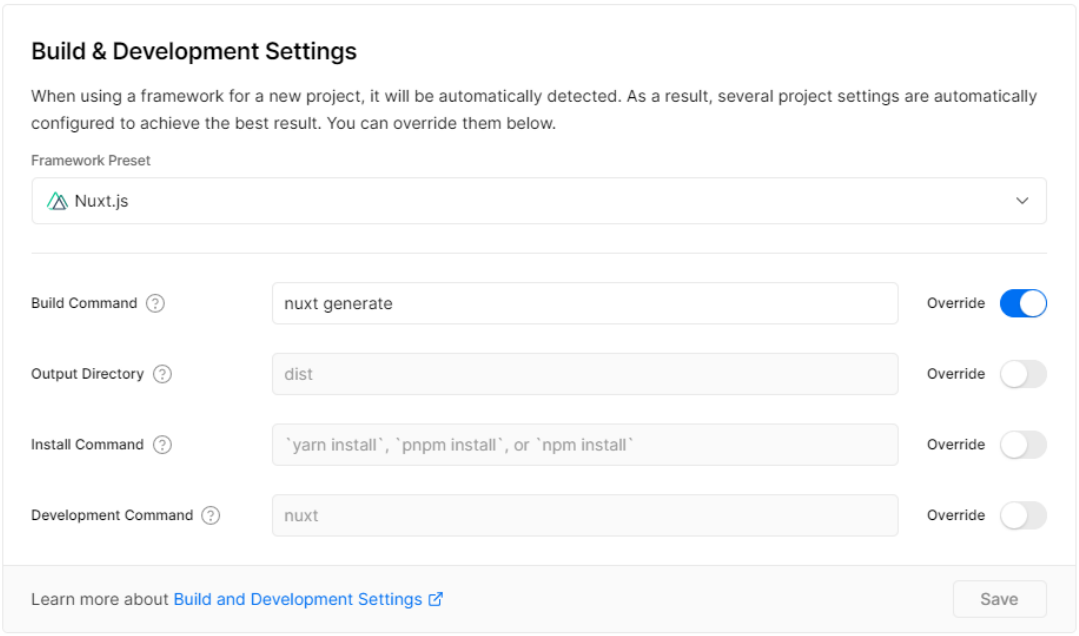


Рисунок 4.5 – Конфігурація розгортання клієнтської частини

Підтверджуючи деплой, необхідно почекати приблизно 5 хв для генерації усіх необхідних файлів, після чого відобразиться сторінка успішного розгортання, що відображено на рисунку 4.6

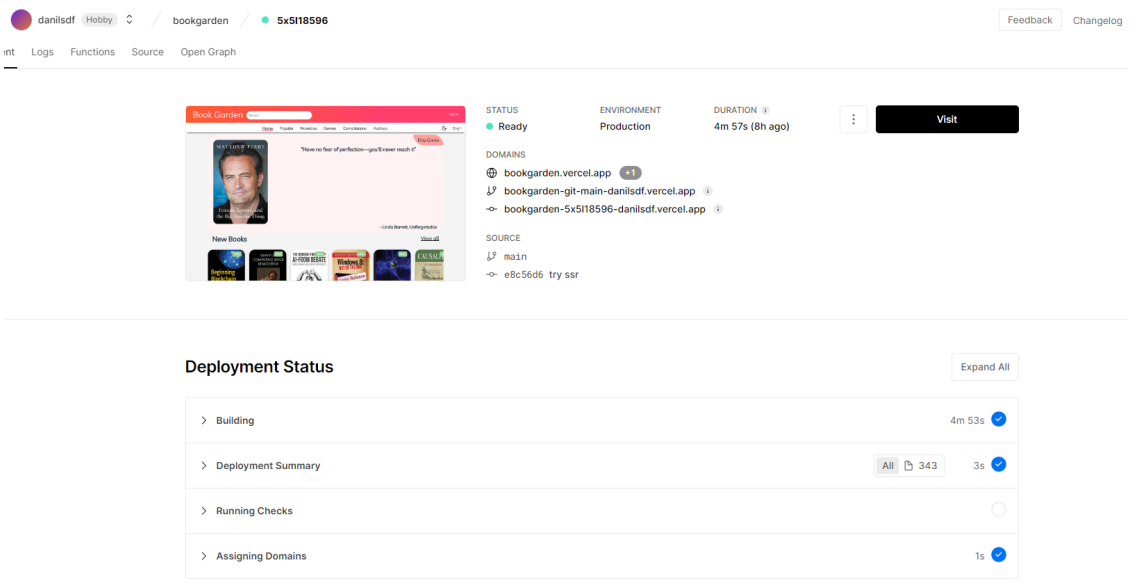


Рисунок 4.6 – Успішне розгортання клієнтської частини

4.2 Підтримка програмного забезпечення

У плані розробника більше ніж реалізовано на цей час, тому з часом буде додано новий функціонал як для клієнтської та серверної частин. Саме через це було впроваджено систему оновлення продукту.

Серверна частина оновлюється ручним способом, за допомогою кліку на кнопку «Publish», так як розробник має змогу перевірити увесь функціонал, усі зміни в коді та оновити серверну частину.

Клієнтська частина оновлюється у більш простий спосіб, на кожну зміну на гілці main буде створюватися автоматичний деплоймент нової версії

Висновки до розділу

У цьому розділі було проведено розгортання ПЗ та надано підтримку для нього.

В рамках розгортання ПЗ було розглянуто серверну та клієнтську частини. Виявлено, що вони реалізовані з використанням різних технологій і

можуть бути розгорнуті на різних сервісах. Серверна частина використовує технології Azure і була розгорнута за допомогою Azure App Service. Для цього було створено SQL Server і базу даних. Для розгортання клієнтської частини було використано сервіс Vercel. Була сконфігурована опція розгортання та допоміжні файли.

У плані підтримки програмного забезпечення було зазначено, що є плани на розширення функціональності для обох частин ПЗ. Система оновлення була впроваджена, де серверна частина оновлюється вручну шляхом публікації, а клієнтська частина оновлюється автоматично при змінах на гілці main.

Таким чином, в цьому розділі було успішно розгорнуто серверну та клієнтську частини ПЗ з використанням відповідних технологій і забезпечено їх підтримку для майбутнього розширення функціональності.

ВИСНОВКИ

У першому розділі було проведено детальний аналіз вимог до програмного забезпечення для розробки веб-застосунку онлайн бібліотеки. Це дозволило зорієнтуватися у потребах користувачів і визначити, що від їх боку вимагається зручний доступ до електронних книг та спрощений процес отримання цих книг.

У другому розділі було проведено моделювання та конструювання програмного забезпечення. Розроблено веб-застосунок, який надає можливість швидкого та зручного доступу до електронних книг, а в якості середовища розробки обрано сімейство Visual Studio. Користувачі можуть швидко знайти бажану книгу за допомогою пошуку, а також здійснити її отримання у електронному форматі без зайвих труднощів.

У третьому розділі було проведено аналіз якості та тестування програмного забезпечення. Програмне забезпечення пройшло успішне тестування, що підтвердило його працездатність та надійність. Користувачі можуть впевнено використовувати веб-застосунок для отримання електронних книжок зручним способом.

У четвертому розділі було проведено розгортання та аналіз підтримки програмного забезпечення. Розроблений веб-застосунок був успішно розгорнутий, що дозволяє забезпечити стабільну роботу системи та зручне використання електронних книг.

В результаті виконання дипломного проєкту було реалізовано та протестовано веб-застосунок для онлайн бібліотеки та на основі проведених етапів розробки та аналізу застосунку можна зробити висновок, що розроблений веб-застосунок спрощує перегляд та отримання інформації про необхідну книжку, а зручний та гарний інтерфейс полегшує використання застосунку, що створює всі умови для популяризації електронних форматів книжок та спрощення процесу їх отримання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Електронна книга: сучасне читання [Електронний ресурс] // Міська спеціалізована молодіжна бібліотека (МСМБ). – 2012. – Режим доступу до ресурсу: https://msmb.org.ua/actions/projects/elektrona-kniga/.
2) Шевченко А. Електронні книги – історія виникнення [Електронний ресурс] / Анатолій Шевченко. – 2013. – Режим доступу до ресурсу: https://cikavosti.com/elektronni-knigi-istoriya-viniknennya/.
3) Культура України [Електронний ресурс] – Режим доступу до ресурсу: https://elib.nlu.org.ua/content.html?id=3.
4) Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/
5) NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en
6) .Net [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/
7) Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/en/.
8) The V8 JavaScript Engine [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/en/learn/the-v8-javascript-engine/
9) Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/.
10) React [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/
11) Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/
12) Nuxt.js [Електронний ресурс] – Режим доступу до ресурсу: https://nuxtjs.org/
13) Microsoft SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/en-au/sql-server/sql-server-2019

ДОДАТКИ

ДОДАТОК А Дерево unit-тестів

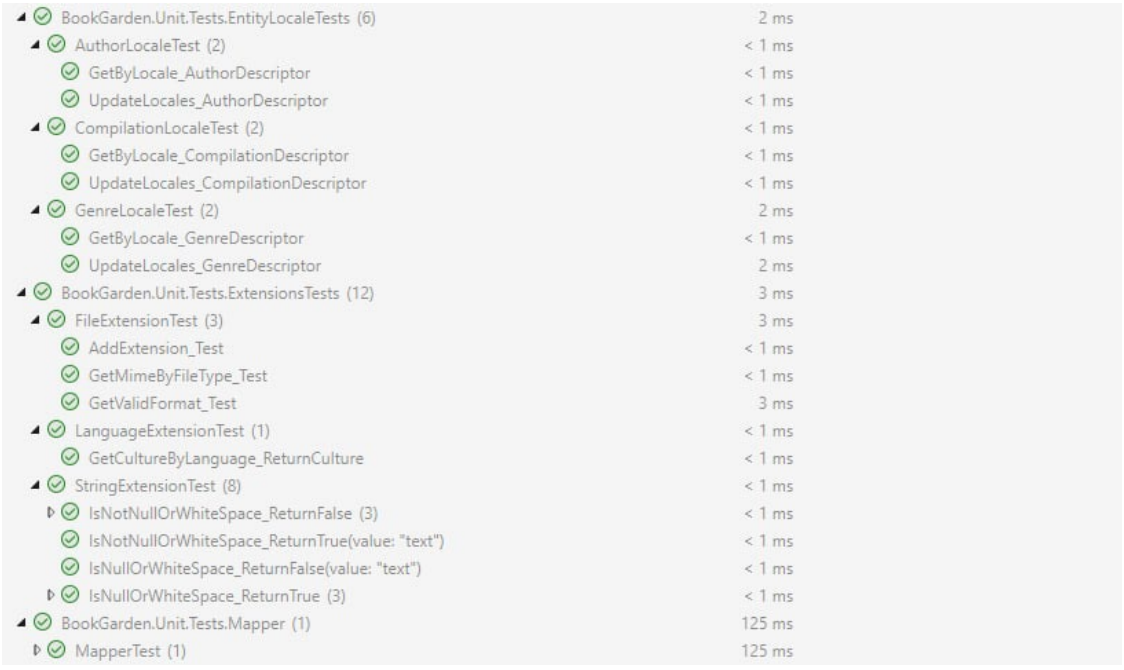


Рисунок А.1 – Unit-тести допоміжних методів

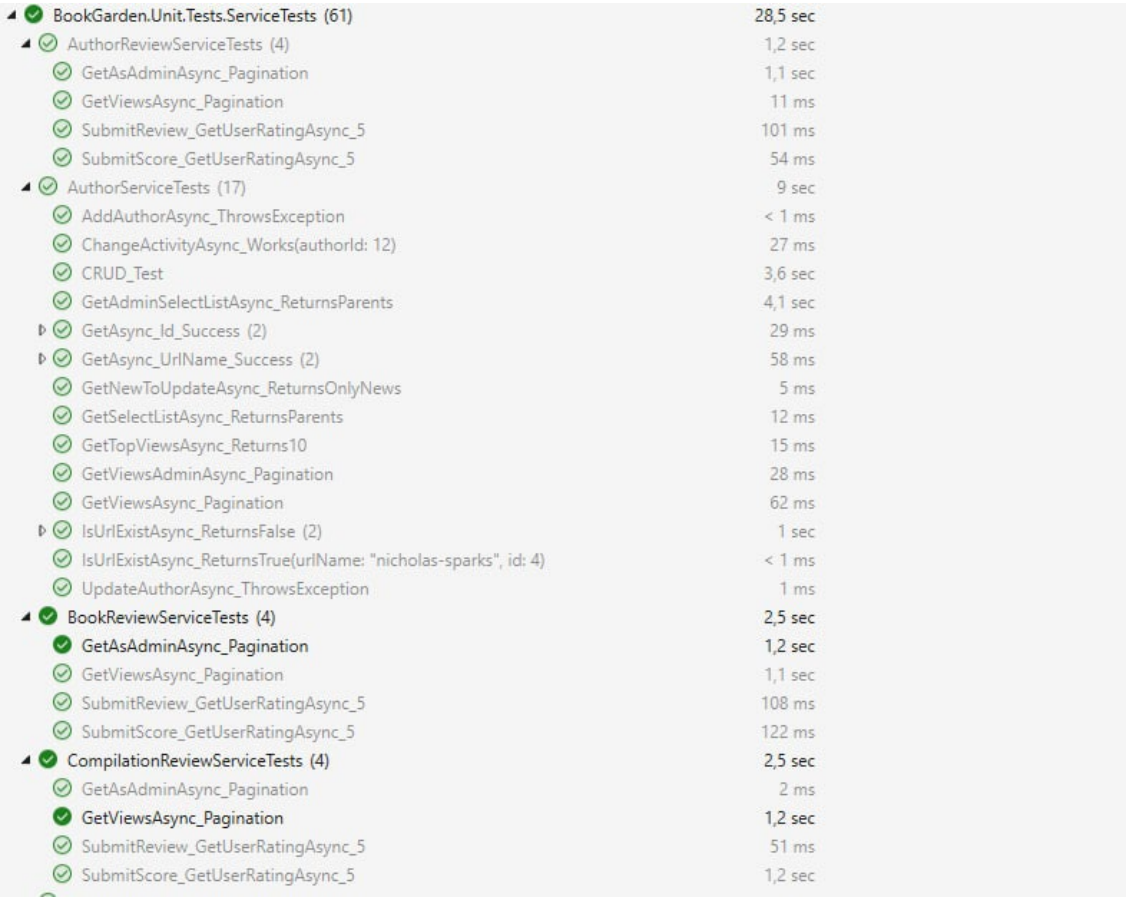
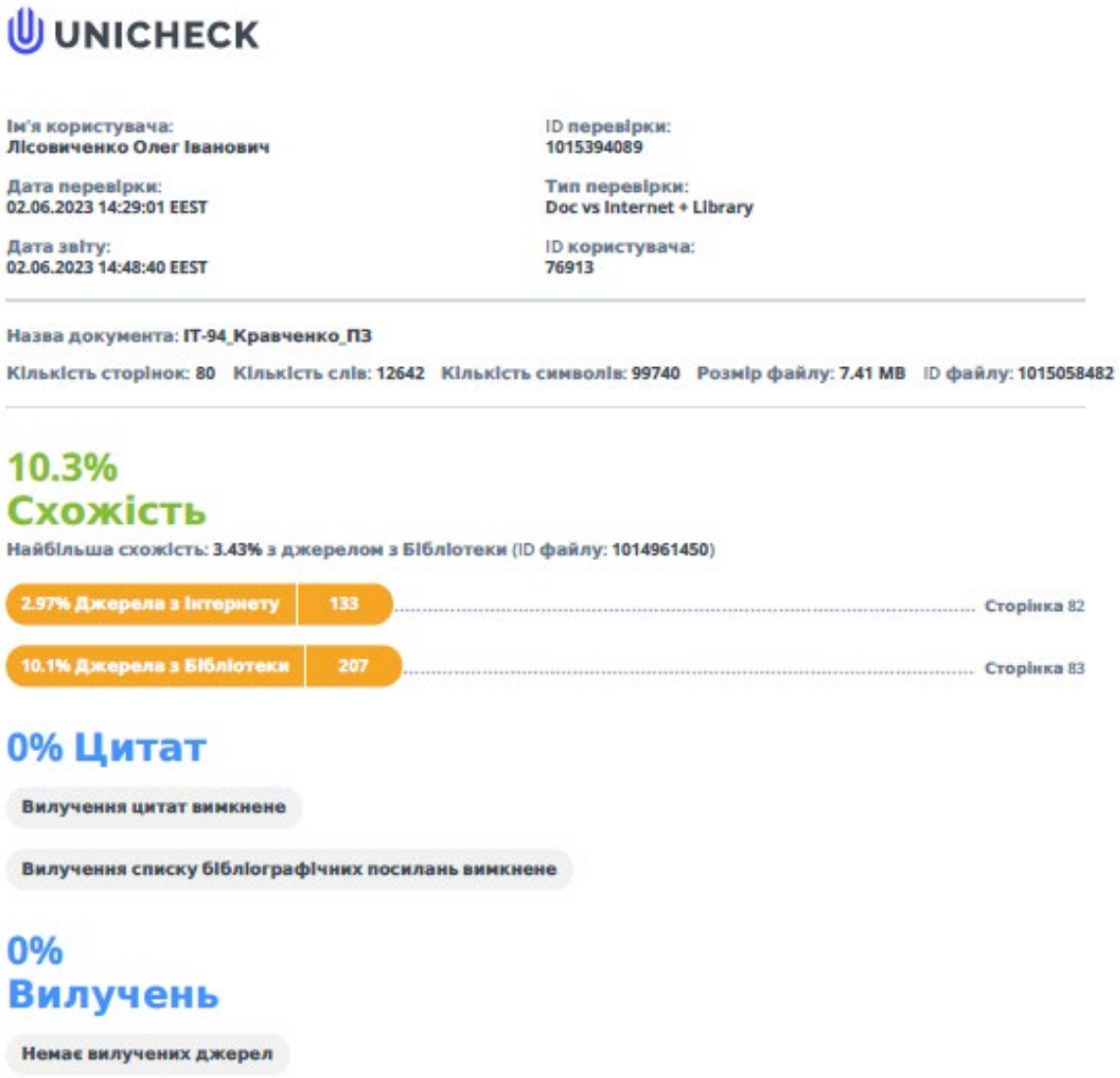


Рисунок А.2 – Unit-тести бізнес логіки

Рисунок А.3 – Unit-тести бізнес логіки

ДОДАТОК Б Звіт подібності



UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015394089

Дата перевірки:
02.06.2023 14:29:01 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 14:48:40 EEST

ID користувача:
76913

Назва документа: IT-94_Кравченко_ПЗ

Кількість сторінок: 80 Кількість слів: 12642 Кількість символів: 99740 Розмір файлу: 7.41 MB ID файлу: 1015058482

10.3%
Схожість

Найбільша схожість: 3.43% з джерелом з Бібліотеки (ID файлу: 1014961450)

2.97% Джерела з Інтернету 133 ..... Сторінка 82

10.1% Джерела з Бібліотеки 207 ..... Сторінка 83

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ОНЛАЙН БІБЛІОТЕКИ

Текст програми

КПІ.ІТ-9409.045440.03.13

“ПОГОДЖЕНО”
Керівник проєкту:
Катерина ЛІЩУК

Нормоконтроль:
Ірина ВІТКОВСЬКА

Виконавець:
Даніл КРАВЧЕНКО

Київ – 2023

Файл Program.cs

```
using BookGarden.Infrastructure.DependencyInjection;
using System.Runtime.CompilerServices;
[assembly: InternalsVisibleTo("BookGarden.Unit.Tests")]
const string allCors = "All";
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddSwaggerCustom<Program>(builder.Configuration, builder.Environment);
builder.Services.AddHttpContextAccessor();
builder.Services.AddCors(options => options.AddPolicy(allCors, build => build.AllowAnyHeader()
   .AllowAnyOrigin()
   .AllowAnyMethod()));
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddAutoMapperCustom();
builder.Services.AddDatabaseContexts(builder.Configuration);
builder.Services.AddServicesCustom();
builder.Services.AddAuthenticationCustom(builder.Configuration, builder.Environment);
builder.Services.AddCloudinaryOptions(builder.Configuration);
builder.Services.AddAzureBlobStorage(builder.Configuration);
builder.Services.AddQuartzCustom();
var app = builder.Build();
app.UseSwaggerCustom<Program>(app.Configuration, app.Environment);
app.UseExceptionMiddleware();
app.UseHttpsRedirection();
app.UseForwardedHeadersCustom();
app.UseCors(allCors);
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Файл AuthController.cs

```
using AutoMapper;
using BookGarden.Models.Auth;
using BookGarden.Service.Authorization;
using BookGarden.Service.Authorization.Descriptors;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using BookGarden.Infrastructure.Extensions;
namespace BookGarden.Controllers
{
   [Route("api/auth")]
   [AllowAnonymous]
   public class AuthController : BookGardenController
   {
      private readonly IAuthService _authService;
      private readonly IMapper _mapper;
      public AuthController(
         IAuthService authService,
         IMapper mapper)
      {
         _authService = authService;
         _mapper = mapper;
      }
      [HttpGet("user")]
      [Authorize]
      public AuthInfoModel GetUser()
      {
         if (User.TryGetAsString(ClaimTypes.Email, out var email))
         {
            return new AuthInfoModel
```

```
            {
                User = new AuthInfoModel.UserInfo
                {
                    Success = true,
                    Email = email,
                    Username = User.GetUsername(),
                    UserHash = User.GetUserHash(),
                    Role = User.GetRole(),
                    IsAdmin = User.IsAdmin(),
                }
            };
        }
        return null!;
    }
    [HttpGet("check/username")]
    public async Task<bool> CheckUsername([FromQuery] string username)
    {
        return await _authService.CheckUsernameAsync(username);
    }
    [HttpPost("register")]
    public async Task Register(RegistrationModel model)
    {
        var descriptor = _mapper.Map<RegistrationDescriptor>(model);
        await _authService.AddUserAsync(descriptor);
    }
    [HttpPost("login")]
    public async Task<string> Login(LoginModel model)
    {
        var descriptor = _mapper.Map<LoginDescriptor>(model);
        var token = await _authService.LoginAndGetTokenAsync(descriptor);
        return token;
    }
  }
}
```

Файл AuthorController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.To.Authors;
using BookGarden.Models.Base;
using BookGarden.Service.Authors;
using BookGarden.Service.Authors.Descriptors;
using BookGarden.Models.Admin.From.UpsertModels;
using Microsoft.AspNetCore.Authorization;
using BookGarden.Shared;
using BookGarden.Models.BookGarden.To.Authors;
using BookGarden.Models.BookGarden.From.RequestModels;
using BookGarden.Service.Reviews.AuthorReviews;
namespace BookGarden.Controllers
{
  [Route("api/author")]
  public class AuthorController : BookGardenController
  {
    private readonly IAuthorService _authorService;
    private readonly IAuthorReviewService _authorReviewService;
    private readonly IMapper _mapper;
    public AuthorController(
      IAuthorService authorService,
      IMapper mapper,
      IAuthorReviewService authorReviewService)
    {
```

```
            _authorService = authorService;
            _mapper = mapper;
            _authorReviewService = authorReviewService;
        }
        #region Admin-Gets
        [HttpGet("admin/options")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<ItemResponse<int>>> GetOptionsAdmin()
        {
            var listItems = await
_authorService.GetOptionsAsAdminAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<ItemResponse<int>>>(listItems);
        }
        [HttpGet("admin/{authorUrl}/id")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> GetAuthorIdAsAdmin([FromRoute] string authorUrl)
        {
            var authorId = await _authorService.GetIdAsync(authorUrl);
            return authorId;
        }
        [HttpGet("admin/items")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<PaginatedResponseList<AuthorAdminResponse>>
GetItemsAsAdmin([FromQuery] AuthorRequestAdminModel model)
        {
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<AuthorRequestAdminDescriptor>(model);
            var listItems = await _authorService.GetAuthorViewsAsAdminAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<AuthorAdminResponse>>(listItems);
        }
        [HttpGet("admin/{authorId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<FullAuthorAdminResponse> GetAuthorAsAdmin([FromRoute] int authorId)
        {
            var author = await _authorService.GetAsync(authorId);
            return _mapper.Map<FullAuthorAdminResponse>(author);
        }
        [HttpGet("admin/checkUrl")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<bool> CheckUrl([FromQuery] string urlName, int authorId)
        {
            return await _authorService.IsUrlExistAsync(urlName, authorId);
        }
        #endregion
        #region User-Gets
        [HttpGet("items/top")]
        public async Task<List<AuthorLightResponse>> GetTop()
        {
            var authors = await _authorService.GetTopViewsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<AuthorLightResponse>>(authors);
        }
        [HttpGet("options")]
        public async Task<List<ItemResponse<string>>> GetOptions()
        {
            var listItems = await _authorService.GetOptionsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<ItemResponse<string>>>(listItems);
        }
        [HttpPost("items")]
        public async Task<PaginatedResponseList<AuthorResponse>> GetItems([FromBody]
AuthorRequestModel model)
        {
            var descriptor = _mapper.Map<AuthorRequestDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
```

```
            var authors = await _authorService.GetViewsAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<AuthorResponse>>(authors);
        }
        [HttpGet("{authorUrl}")]
        public async Task<FullAuthorResponse> Get([FromRoute] string authorUrl)
        {
            var author = await _authorService.GetFullViewAsync(authorUrl,
HttpContext.GetCurrentLanguage());
            author.Genres = author.Genres.DistinctBy(g=>g.GenreUrlName).ToArray();
            var response = _mapper.Map<FullAuthorResponse>(author);
            if (User.TryGetAsString(BookGardenConstants.Claims.UserHash, out var userHash))
            {
                response.UserRating = await _authorReviewService.GetUserRatingAsync(userHash, authorUrl);
            }
            return response;
        }
        #endregion
        #region Admin-Post
        [HttpPost("admin/add")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> AddAsAdminAuthor([FromBody] AuthorModel model)
        {
            var descriptor = _mapper.Map<AuthorDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            return await _authorService.AddAuthorAsync(descriptor);
        }
        #endregion
        #region User-Post
        #endregion
        #region Admin-Put
        [HttpPut("admin/update")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task UpdateAsAdminAuthor([FromBody] AuthorModel model)
        {
            var descriptor = _mapper.Map<AuthorDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            await _authorService.UpdateAuthorAsync(descriptor);
        }
        [HttpPut("admin/{authorId}/activity")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task ChangeActivity([FromRoute] int authorId)
        {
            await _authorService.ChangeActivityAsync(authorId);
        }
        #endregion
        #region User-Put
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{authorId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminAuthor([FromRoute] int authorId)
        {
            await _authorService.DeleteAuthorAsync(authorId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл AuthorReviewController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
```

```
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.Admin.To.Reviews;
using BookGarden.Models.Base;
using BookGarden.Models.BookGarden.From.RequestModels;
using BookGarden.Models.BookGarden.To;
using BookGarden.Service.Reviews.AuthorReviews;
using BookGarden.Service.Reviews.Descriptors;
using BookGarden.Shared;
using Microsoft.AspNetCore.Authorization;
using BookGarden.Models.BookGarden.From;
using System.Security.Claims;
namespace BookGarden.Controllers
{
    [Route("api/review/author")]
    public class AuthorReviewController : BookGardenController
    {
        private readonly IAuthorReviewService _reviewService;
        private readonly IMapper _mapper;
        public AuthorReviewController(
            IAuthorReviewService reviewService,
            IMapper mapper)
        {
            _reviewService = reviewService;
            _mapper = mapper;
        }
        #region Admin-Gets
        [HttpGet("admin/items")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<PaginatedResponseList<ReviewAdminResponse>>
GetItemsAsAdmin([FromQuery] ReviewRequestAdminModel model)
        {
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<ReviewRequestAdminDescriptor>(model);
            var listItems = await _reviewService.GetAsAdminAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<ReviewAdminResponse>>(listItems);
        }
        #endregion
        #region User-Gets
        [HttpGet("items")]
        public async Task<PaginatedResponseList<ReviewResponse>> GetItems([FromQuery]
AuthorReviewRequestModel model)
        {
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<AuthorReviewRequestDescriptor>(model);
            var listItems = await _reviewService.GetViewsAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<ReviewResponse>>(listItems);
        }
        [HttpGet("{authorUrl}/rating/{userHash}")]
        public async Task<int?> GetUserScore([FromRoute] string authorUrl, [FromRoute] string userHash)
        {
            return await _reviewService.GetUserRatingAsync(userHash, authorUrl);
        }
        #endregion
        #region Admin-Post
        #endregion
        #region User-Post
        [HttpPost("{authorUrl}/submit")]
        public async Task SubmitReview([FromRoute] string authorUrl, [FromBody] ReviewModel model)
        {
            var descriptor = _mapper.Map<AuthorReviewDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
```

```
            descriptor.AuthorUrl = authorUrl;
            if (User.TryGetAsString(ClaimTypes.Email, out var email))
            {
                descriptor.IsUser = true;
                descriptor.Email = email;
            }
            await _reviewService.SubmitReviewAsync(descriptor);
        }
        [HttpPost("{authorUrl}/submit/{score}")]
        public async Task SubmitScore([FromRoute] string authorUrl, [FromQuery] string userHash,
[FromRoute] int score)
        {
            await _reviewService.SubmitScoreAsync(userHash, authorUrl, score);
        }
        #endregion
        #region Admin-Put
        [HttpPut("admin/{reviewId}/approve")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task ApproveReviewAsAdmin([FromRoute] int reviewId)
        {
            await _reviewService.ApproveReviewAsync(reviewId);
        }
        #endregion
        #region User-Put
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{reviewId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminReview([FromRoute] int reviewId)
        {
            await _reviewService.DeleteAsync(reviewId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл BookController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.Admin.To.Books;
using BookGarden.Models.Base;
using BookGarden.Models.BookGarden.From.RequestModels;
using BookGarden.Service.Books;
using BookGarden.Service.Books.Descriptors;
using BookGarden.Shared;
using BookGarden.Shared.Enums;
using Microsoft.AspNetCore.Authorization;
using BookGarden.Models.BookGarden.To.Books;
using BookGarden.Shared.Pagination;
using BookGarden.Service.Reviews.BookReviews;
using BookGarden.Service.Users;
using BookGarden.Shared.Extensions;
using BookGarden.Shared.Localization;
namespace BookGarden.Controllers
{
    [Route("api/book")]
    public class BookController : BookGardenController
```

```
{
    private readonly IBookService _bookService;
    private readonly IBookReviewService _bookReviewService;
    private readonly IUserService _userService;
    private readonly IMapper _mapper;
    private readonly LocalizationService _localizationService;
    public BookController(
        IBookService bookService,
        IMapper mapper,
        IBookReviewService bookReviewService,
        LocalizationService localizationService,
        IUserService userService)
    {
        _bookService = bookService;
        _mapper = mapper;
        _bookReviewService = bookReviewService;
        _localizationService = localizationService;
        _userService = userService;
    }
    #region Admin-Gets
    [HttpGet("admin/{bookId}")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<FullBookAdminResponse> GetAsAdmin([FromRoute] int bookId)
    {
        var book = await _bookService.GetAsync(bookId);
        return _mapper.Map<FullBookAdminResponse>(book);
    }
    [HttpGet("admin/{bookId}/files")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<List<FileType>> GetFilesAsAdmin([FromRoute] int bookId)
    {
        return await _bookService.GetFilesAsync(bookId);
    }
    [HttpGet("admin/{bookUrl}/id")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<int> GetBookIdAsAdmin([FromRoute] string bookUrl)
    {
        var bookId = await _bookService.GetIdAsync(bookUrl);
        return bookId;
    }
    [HttpGet("admin/items")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<PaginatedResponseList<BookAdminResponse>> GetItemsAsAdmin([FromQuery]
BookRequestAdminModel model)
    {
        var descriptor = _mapper.Map<BookRequestAdminDescriptor>(model);
        descriptor.SetHeaders(HttpContext.GetHeaderModel());
        var books = await _bookService.GetViewsAsAdmin(descriptor);
        return _mapper.Map<PaginatedResponseList<BookAdminResponse>>(books);
    }
    [HttpGet("admin/search")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<PaginatedResponseList<BookAdminLightResponse>>
SearchBookAsAdmin([FromQuery] BookSearchAdminModel model)
    {
        var descriptor = _mapper.Map<BookSearchAdminDescriptor>(model);
        descriptor.SetHeaders(HttpContext.GetHeaderModel());
        var books = await _bookService.SearchBookAsAdminAsync(descriptor);
        return _mapper.Map<PaginatedResponseList<BookAdminLightResponse>>(books);
    }
    [HttpGet("admin/checkUrl")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<bool> CheckUrl([FromQuery] string urlName, [FromQuery] int bookId)
```

```
        {
            return await _bookService.CheckUrlAsync(urlName, bookId);
        }
        [HttpGet("admin/by/author/{authorId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<BookAdminLightResponse>> GetBooksByAuthor([FromRoute] int authorId)
        {
            var books = await _bookService.GetViewsByAuthorAsAdminAsync(authorId,
HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookAdminLightResponse>>(books);
        }
        [HttpGet("admin/by/compilation/{compilationId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<BookAdminLightResponse>> GetBooksByCompilation([FromRoute] int
compilationId)
        {
            var books = await _bookService.GetViewsByCompilationAsAdminAsync(compilationId,
HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookAdminLightResponse>>(books);
        }
        [HttpGet("admin/by/translation/{bookId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<BookAdminLightResponse>> GetTranslatedBooksByBook([FromRoute] int
bookId)
        {
            var books = await _bookService.GetTranslatedViewsAsAdminAsync(bookId,
HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookAdminLightResponse>>(books);
        }
        [HttpGet("admin/searchOption")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<ItemResponse<int>>> SearchOptionAsAdmin([FromQuery] string search)
        {
            var lang = HttpContext.GetCurrentLanguage();
            var books = await _bookService.SearchOptionAsAdminAsync(search, lang);
            return _mapper.Map<List<ItemResponse<int>>>(books);
        }
        #endregion
        #region User-Gets
        [HttpPost("items")]
        public async Task<PaginatedResponseList<BookGridResponse>> GetItems([FromBody]
BookRequestModel model)
        {
            var descriptor = _mapper.Map<BookRequestDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            var books = await _bookService.GetViewsAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<BookGridResponse>>(books);
        }
        [HttpGet("items/compilation/{compilationUrl}")]
        public async Task<PaginatedResponseList<BookGridResponse>> GetByCompilation([FromRoute]
string compilationUrl, [FromQuery] SortPaginationModel model)
        {
            model.SetHeaders(HttpContext.GetHeaderModel());
            var books = await _bookService.GetByCompilationAsync(compilationUrl, model);
            return _mapper.Map<PaginatedResponseList<BookGridResponse>>(books);
        }
        [HttpGet("items/new")]
        public async Task<List<BookLightResponse>> GetNew()
        {
            var books = await _bookService.GetNewViewsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookLightResponse>>(books);
        }
        [HttpGet("items/top")]
```

```
        public async Task<List<BookLightResponse>> GetTop()
        {
            var books = await _bookService.GetTopViewsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookLightResponse>>(books);
        }
        [HttpGet("items/{bookUrl}/author/{authorUrl}")]
        public async Task<List<BookLightResponse>> GetByAuthor([FromRoute] string bookUrl,
[FromRoute] string authorUrl)
        {
            var books = await _bookService.GetByAuthorAsync(bookUrl, authorUrl,
HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookLightResponse>>(books);
        }
        [HttpGet("items/{bookUrl}/genre/{genreUrl}")]
        public async Task<List<BookLightResponse>> GetByGenre([FromRoute] string bookUrl,
[FromRoute] string genreUrl)
        {
            var books = await _bookService.GetByGenreAsync(bookUrl, genreUrl,
HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookLightResponse>>(books);
        }
        [HttpGet("{bookUrl}")]
        public async Task<FullBookResponse> Get([FromRoute] string bookUrl)
        {
            var bookView = await _bookService.GetAsync(bookUrl, HttpContext.GetCurrentLanguage());
            var response = _mapper.Map<FullBookResponse>(bookView);
            if (User.TryGetAsString(BookGardenConstants.Claims.UserHash, out var userHash))
            {
                response.UserRating = await _bookReviewService.GetUserRatingAsync(userHash, bookUrl);
                response.UserStatus = await _userService.GetBookStatusAsync(bookUrl, userHash);
            }
            return response;
        }
        [HttpGet("{bookUrl}/read")]
        public async Task<BookReadResponse> GetReadView([FromRoute] string bookUrl)
        {
            var bookView = await _bookService.GetReadViewAsync(bookUrl);
            return _mapper.Map<BookReadResponse>(bookView);
        }
        [HttpGet("{bookUrl}/file/{format}")]
        public async Task<FileStreamResult> GetFile([FromRoute] string bookUrl, [FromRoute] FileType
format)
        {
            var view = await _bookService.GetFileLinkAsync(bookUrl, format);
            return HttpContext.FileStreamResult(view.Item1, bookUrl, format);
        }
        [HttpGet("{bookUrl}/similar")]
        public async Task<List<BookLightResponse>> GetSimilar([FromRoute] string bookUrl)
        {
            var genre = await _bookService.GetSimilarAsync(bookUrl, HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<BookLightResponse>>(genre);
        }
        #endregion
        #region Admin-Post
        [HttpPost("admin/add")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> AddAsAdmin([FromBody] BookModel model)
        {
            var descriptor = _mapper.Map<BookDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            return await _bookService.AddAsync(descriptor);
        }
        [HttpPost("admin/author/add/{bookId}/to/{authorId}")]
```

```
[Authorize(Roles = BookGardenConstants.Roles.Admin)]
public async Task AddBookToAuthor([FromRoute] int authorId, [FromRoute] int bookId)
{
    await _bookService.AddBookToAuthorAsync(authorId, bookId);
}
[HttpPost("admin/compilation/add/{bookId}/to/{compilationId}")]
[Authorize(Roles = BookGardenConstants.Roles.Admin)]
public async Task AddBookToCompilation([FromRoute] int compilationId, [FromRoute] int bookId)
{
    await _bookService.AddBookToCompilationAsync(compilationId, bookId);
}
[HttpPost("admin/addFile")]
[Authorize(Roles = BookGardenConstants.Roles.Admin)]
public async Task<int> AddFileAsAdmin([FromForm] BookFileModel model)
{
    var lang = HttpContext.GetCurrentLanguage();
    if (model.IsDocumentEmpty())
    {
        throw _localizationService.GetBusinessException(LocalizationConstants.FileEmpty, lang);
    }
    if (model.IsDocumentTooLarge())
    {
        throw _localizationService.GetBusinessException(LocalizationConstants.FileTooLarge, lang);
    }
    var format = model.Document?.FileName.GetValidFormat();
    if (format == null)
    {
        throw _localizationService.GetBusinessException(LocalizationConstants.FileUnsupported, lang);
    }
    var valid = await _bookService.IsFileFormatFreeAsync(model.BookId, format.Value);
    if (!valid)
    {
        throw _localizationService.GetBusinessException(LocalizationConstants.FileFormatExist, lang);
    }
    try
    {
        return await _bookService.AddFileAsync(model.BookId, model.Document, format.Value);
    }
    catch
    {
        return 0;
    }
}
#endregion
#region User-Post
#endregion
#region Admin-Put
[HttpPut("admin/{bookId}/activity")]
[Authorize(Roles = BookGardenConstants.Roles.Admin)]
public async Task ChangeActivity([FromRoute] int bookId)
{
    await _bookService.ChangeActivityAsync(bookId);
}
[HttpPut("admin/update")]
[Authorize(Roles = BookGardenConstants.Roles.Admin)]
public async Task UpdateAsAdmin([FromBody] BookModel model)
{
    var descriptor = _mapper.Map<BookDescriptor>(model);
    descriptor.SetHeaders(HttpContext.GetHeaderModel());
    await _bookService.UpdateAsync(descriptor);
}
#endregion
#region User-Put
```

```
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{bookId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminAuthor([FromRoute] int bookId)
        {
            await _bookService.DeleteAsync(bookId);
        }
        [HttpDelete("admin/file")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteFileAsAdminAuthor([FromQuery] int bookId, [FromQuery] FileType
fileType)
        {
            await _bookService.DeleteFileAsync(bookId, fileType);
        }
        [HttpDelete("admin/author/delete/{bookId}/from/{authorId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteBookFromAuthor([FromRoute] int authorId, [FromRoute] int bookId)
        {
            await _bookService.DeleteFromAuthorAsync(authorId, bookId);
        }
        [HttpDelete("admin/compilation/delete/{bookId}/from/{compilationId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteBookFromCompilation([FromRoute] int compilationId, [FromRoute] int
bookId)
        {
            await _bookService.DeleteFromCompilationAsync(compilationId, bookId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл BookGardenController.cs

```
using Microsoft.AspNetCore.Mvc;
namespace BookGarden.Controllers
{
    [ApiController]
    public abstract class BookGardenController : ControllerBase
    {
    }
}
```

Файл BookReviewController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.Admin.To.Reviews;
using BookGarden.Models.Base;
using BookGarden.Models.BookGarden.From;
using BookGarden.Models.BookGarden.From.RequestModels;
using BookGarden.Models.BookGarden.To;
using BookGarden.Service.Reviews.BookReviews;
using BookGarden.Service.Reviews.Descriptors;
using BookGarden.Shared;
using Microsoft.AspNetCore.Authorization;
using System.Security.Claims;
namespace BookGarden.Controllers
{
    [Route("api/review/book")]
    public class BookReviewController : BookGardenController
```

```
{
    private readonly IBookReviewService _reviewService;
    private readonly IMapper _mapper;
    public BookReviewController(
        IBookReviewService reviewService,
        IMapper mapper)
    {
        _reviewService = reviewService;
        _mapper = mapper;
    }
    #region Admin-Gets
    [HttpGet("admin/items")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task<PaginatedResponseList<ReviewAdminResponse>>
GetItemsAsAdmin([FromQuery] ReviewRequestAdminModel model)
    {
        model.RequestLanguage = HttpContext.GetCurrentLanguage();
        var descriptor = _mapper.Map<ReviewRequestAdminDescriptor>(model);
        var listItems = await _reviewService.GetAsAdminAsync(descriptor);
        return _mapper.Map<PaginatedResponseList<ReviewAdminResponse>>(listItems);
    }
    #endregion
    #region User-Gets
    [HttpGet("items")]
    public async Task<PaginatedResponseList<ReviewResponse>> GetItems([FromQuery]
BookReviewRequestModel model)
    {
        model.RequestLanguage = HttpContext.GetCurrentLanguage();
        var descriptor = _mapper.Map<BookReviewRequestDescriptor>(model);
        var listItems = await _reviewService.GetViewsAsync(descriptor);
        return _mapper.Map<PaginatedResponseList<ReviewResponse>>(listItems);
    }
    [HttpGet("items/last")]
    public async Task<List<ReviewResponse>> GetLastReviews()
    {
        var listItems = await _reviewService.GetLastReviewsAsync(HttpContext.GetCurrentLanguage());
        return _mapper.Map<List<ReviewResponse>>(listItems);
    }
    [HttpGet("{bookUrl}/rating/{userHash}")]
    public async Task<int?> GetUserScore([FromRoute] string bookUrl, [FromRoute] string userHash)
    {
        return await _reviewService.GetUserRatingAsync(userHash, bookUrl);
    }
    #endregion
    #region Admin-Post
    #endregion
    #region User-Post
    [HttpPost("{bookUrl}/submit")]
    public async Task SubmitReview([FromRoute] string bookUrl, [FromBody] ReviewModel model)
    {
        var descriptor = _mapper.Map<BookReviewDescriptor>(model);
        descriptor.SetHeaders(HttpContext.GetHeaderModel());
        descriptor.BookUrl = bookUrl;
        if (User.TryGetAsString(ClaimTypes.Email, out var email))
        {
            descriptor.IsUser = true;
            descriptor.Email = email;
            descriptor.UserHash = User.GetUserHash();
        }
        await _reviewService.SubmitReviewAsync(descriptor);
    }
    [HttpPost("{bookUrl}/submit/{score}")]
```

```
        public async Task SubmitScore([FromRoute] string bookUrl, [FromQuery] string userHash, [FromRoute] int score)
        {
            await _reviewService.SubmitScoreAsync(userHash, bookUrl, score);
        }
        #endregion
        #region Admin-Put
        [HttpPut("admin/{reviewId}/approve")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task ApproveReviewAsAdmin([FromRoute] int reviewId)
        {
            await _reviewService.ApproveReviewAsync(reviewId);
        }
        #endregion
        #region User-Put
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{reviewId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminReview([FromRoute] int reviewId)
        {
            await _reviewService.DeleteAsync(reviewId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл CompilationController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.Admin.To.Compilations;
using BookGarden.Models.Base;
using BookGarden.Service.Compilations;
using BookGarden.Service.Compilations.Descriptors;
using BookGarden.Shared;
using Microsoft.AspNetCore.Authorization;
using BookGarden.Models.BookGarden.From.RequestModels;
using BookGarden.Models.BookGarden.To.Compilations;
using BookGarden.Service.Reviews.CompilationReviews;
namespace BookGarden.Controllers
{
    [Route("api/compilation")]
    public class CompilationController : BookGardenController
    {
        private readonly ICompilationService _compilationService;
        private readonly ICompilationReviewService _compilationReviewService;
        private readonly IMapper _mapper;
        public CompilationController(
            ICompilationService compilationService,
            IMapper mapper,
            ICompilationReviewService compilationReviewService)
        {
            _compilationService = compilationService;
            _mapper = mapper;
            _compilationReviewService = compilationReviewService;
        }
        #region Admin-Get
        [HttpGet("admin/items")]
```

```
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<PaginatedResponseList<CompilationAdminResponse>>
GetItemsAsAdmin([FromQuery] CompilationRequestAdminModel model)
        {
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<CompilationRequestAdminDescriptor>(model);
            var listItems = await _compilationService.GetAsAdminAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<CompilationAdminResponse>>(listItems);
        }
        [HttpGet("admin/options")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<ItemResponse<int>>> GetOptions()
        {
            var listItems = await _compilationService.GetOptionsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<ItemResponse<int>>>(listItems);
        }
        [HttpGet("admin/{compilationUrl}/id")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> GetCompilationIdAsAdmin([FromRoute] string compilationUrl)
        {
            var compilationId = await _compilationService.GetIdAsync(compilationUrl);
            return compilationId;
        }
        [HttpGet("admin/{compilationId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<FullCompilationAdminResponse> GetCompilationAsAdmin([FromRoute] int
compilationId)
        {
            var compilation = await _compilationService.GetAsync(compilationId);
            return _mapper.Map<FullCompilationAdminResponse>(compilation);
        }
        [HttpGet("admin/checkUrl")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<bool> CheckUrl([FromQuery] string urlName, int compilationId)
        {
            return await _compilationService.IsUrlExistAsync(urlName, compilationId);
        }
        #endregion
        #region User-Get
        [HttpGet("items/top")]
        public async Task<List<CompilationLightResponse>> GetTop()
        {
            var genres = await _compilationService.GetTopViewsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<CompilationLightResponse>>(genres);
        }
        [HttpGet("{compilationUrl}")]
        public async Task<FullCompilationResponse> Get([FromRoute] string compilationUrl)
        {
            var compilation = await _compilationService.GetFullViewAsync(compilationUrl,
HttpContext.GetCurrentLanguage());
            var response = _mapper.Map<FullCompilationResponse>(compilation);
            if (User.TryGetAsString(BookGardenConstants.Claims.UserHash, out var userHash))
            {
                response.UserRating = await _compilationReviewService.GetUserRatingAsync(userHash,
compilationUrl);
            }
            return response;
        }
        [HttpPost("items")]
        public async Task<PaginatedResponseList<CompilationResponse>> GetItems([FromBody]
CompilationRequestModel model)
        {
            var descriptor = _mapper.Map<CompilationRequestDescriptor>(model);
```

```
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            var books = await _compilationService.GetViewsAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<CompilationResponse>>(books);
        }
        #endregion
        #region Admin-Post
        [HttpPost("admin/add")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> AddAsAdminCompilation([FromBody] CompilationModel model)
        {
            var descriptor = _mapper.Map<CompilationDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            return await _compilationService.AddAsync(descriptor);
        }
        #endregion
        #region User-Post
        #endregion
        #region Admin-Put
        [HttpPut("admin/update")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task UpdateAsAdminCompilation([FromBody] CompilationModel model)
        {
            var descriptor = _mapper.Map<CompilationDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            await _compilationService.UpdateAsync(descriptor);
        }
        [HttpPut("admin/{compilationId}/activity")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task ChangeActivity([FromRoute] int compilationId)
        {
            await _compilationService.ChangeActivityAsync(compilationId);
        }
        #endregion
        #region User-Put
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{compilationId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminCompilation([FromRoute] int compilationId)
        {
            await _compilationService.DeleteAsync(compilationId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл CompilationReviewController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.Admin.To.Reviews;
using BookGarden.Models.Base;
using BookGarden.Models.BookGarden.From.RequestModels;
using BookGarden.Models.BookGarden.To;
using BookGarden.Service.Reviews.CompilationReviews;
using BookGarden.Service.Reviews.Descriptors;
using BookGarden.Shared;
using Microsoft.AspNetCore.Authorization;
using BookGarden.Models.BookGarden.From;
```

```
using System.Security.Claims;
namespace BookGarden.Controllers
{
    [Route("api/review/compilation")]
    public class CompilationReviewController : BookGardenController
    {
        private readonly ICompilationReviewService _reviewService;
        private readonly IMapper _mapper;
        public CompilationReviewController(
            ICompilationReviewService reviewService,
            IMapper mapper)
        {
            _reviewService = reviewService;
            _mapper = mapper;
        }
        #region Admin-Gets
        [HttpGet("admin/items")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<PaginatedResponseList<ReviewAdminResponse>>
GetItemsAsAdmin([FromQuery] ReviewRequestAdminModel model)
        {
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<ReviewRequestAdminDescriptor>(model);
            var listItems = await _reviewService.GetAsAdminAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<ReviewAdminResponse>>(listItems);
        }
        #endregion
        #region User-Gets
        [HttpGet("items")]
        public async Task<PaginatedResponseList<ReviewResponse>> GetItems([FromQuery]
CompilationReviewRequestModel model)
        {
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<CompilationReviewRequestDescriptor>(model);
            var listItems = await _reviewService.GetViewsAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<ReviewResponse>>(listItems);
        }
        [HttpGet("{compilationUrl}/rating/{userHash}")]
        public async Task<int?> GetUserScore([FromRoute] string compilationUrl, [FromRoute] string
userHash)
        {
            return await _reviewService.GetUserRatingAsync(userHash, compilationUrl);
        }
        #endregion
        #region Admin-Post
        #endregion
        #region User-Post
        [HttpPost("{compilationUrl}/submit")]
        public async Task SubmitReview([FromRoute] string compilationUrl, [FromBody] ReviewModel
model)
        {
            var descriptor = _mapper.Map<CompilationReviewDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            descriptor.CompilationUrl = compilationUrl;
            if (User.TryGetAsString(ClaimTypes.Email, out var email))
            {
                descriptor.IsUser = true;
                descriptor.Email = email;
            }
            await _reviewService.SubmitReviewAsync(descriptor);
        }
        [HttpPost("{compilationUrl}/submit/{score}")]
```

```
        public async Task SubmitScore([FromRoute] string compilationUrl, [FromQuery] string userHash,
[FromRoute] int score)
        {
            await _reviewService.SubmitScoreAsync(userHash, compilationUrl, score);
        }
        #endregion
        #region Admin-Put
        [HttpPut("admin/{reviewId}/approve")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task ApproveReviewAsAdmin([FromRoute] int reviewId)
        {
            await _reviewService.ApproveReviewAsync(reviewId);
        }
        #endregion
        #region User-Put
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{reviewId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminReview([FromRoute] int reviewId)
        {
            await _reviewService.DeleteAsync(reviewId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл GenreController.cs

```
using Microsoft.AspNetCore.Mvc;
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
using BookGarden.Models.Admin.To.Genres;
using BookGarden.Models.Base;
using BookGarden.Service.Genres;
using BookGarden.Service.Genres.Descriptors;
using BookGarden.Shared;
using Microsoft.AspNetCore.Authorization;
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.BookGarden.To.Genres;
using BookGarden.Models.BookGarden.From.RequestModels;
namespace BookGarden.Controllers
{
    [Route("api/genre")]
    public class GenreController : BookGardenController
    {
        private readonly IGenreService _genreService;
        private readonly IMapper _mapper;
        public GenreController(
            IGenreService genreService,
            IMapper mapper)
        {
            _genreService = genreService;
            _mapper = mapper;
        }
        #region Admin-Get
        [HttpGet("admin/items")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<PaginatedResponseList<GenreAdminResponse>>
GetItemsAsAdmin([FromQuery] GenreRequestAdminModel model)
        {
```

```
            model.RequestLanguage = HttpContext.GetCurrentLanguage();
            var descriptor = _mapper.Map<GenreRequestAdminDescriptor>(model);
            var listItems = await _genreService.GetGenreViewsAsAdminAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<GenreAdminResponse>>(listItems);
        }
        [HttpGet("admin/options")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<List<ItemResponse<int>>> GetAdminSelectOptions()
        {
            var lang = HttpContext.GetCurrentLanguage();
            var listItems = await _genreService.GetAdminSelectListAsync(lang);
            return _mapper.Map<List<ItemResponse<int>>>(listItems);
        }
        [HttpGet("admin/{genreId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<FullGenreAdminResponse> GetGenreAsAdmin([FromRoute] int genreId)
        {
            var genre = await _genreService.GetAsync(genreId);
            return _mapper.Map<FullGenreAdminResponse>(genre);
        }
        [HttpGet("admin/{genreUrl}/id")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> GetGenreIdAsAdmin([FromRoute] string genreUrl)
        {
            var genreId = await _genreService.GetIdAsync(genreUrl);
            return genreId;
        }
        [HttpGet("admin/checkUrl")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<bool> CheckUrl([FromQuery] string urlName, int genreId)
        {
            return await _genreService.IsUrlExistAsync(urlName, genreId);
        }
        #endregion
        #region User-Get
        [HttpGet("items/top")]
        public async Task<List<GenreLightResponse>> GetTop()
        {
            var genres = await _genreService.GetTopViewsAsync(HttpContext.GetCurrentLanguage());
            return _mapper.Map<List<GenreLightResponse>>(genres);
        }
        [HttpGet("{genreUrl}")]
        public async Task<FullGenreResponse> Get([FromRoute] string genreUrl)
        {
            var genre = await _genreService.GetViewAsync(genreUrl, HttpContext.GetCurrentLanguage());
            return _mapper.Map<FullGenreResponse>(genre);
        }
        [HttpPost("items")]
        public async Task<PaginatedResponseList<GenreResponse>> GetItems([FromBody]
GenreRequestModel model)
        {
            var descriptor = _mapper.Map<GenreRequestDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            var books = await _genreService.GetViewsAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<GenreResponse>>(books);
        }
        [HttpGet("options")]
        public async Task<List<ComplexLocaleGenreResponse>> GetSelectOptions()
        {
            var lang = HttpContext.GetCurrentLanguage();
            var listItems = await _genreService.GetOptionsTreeAsync(lang);
            return _mapper.Map<List<ComplexLocaleGenreResponse>>(listItems);
        }
```

```
        [HttpGet("parent-options")]
        public async Task<List<ItemResponse<string>>> GetParentSelectOptions()
        {
            var lang = HttpContext.GetCurrentLanguage();
            var listItems = await _genreService.GetParentOptionsAsync(lang);
            return _mapper.Map<List<ItemResponse<string>>>(listItems);
        }
        #endregion
        #region Admin-Post
        [HttpPost("admin/add")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<int> AddAsAdminGenre([FromBody] GenreModel model)
        {
            var descriptor = _mapper.Map<GenreDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            return await _genreService.AddGenreAsync(descriptor);
        }
        #endregion
        #region User-Post
        #endregion
        #region Admin-Put
        [HttpPut("admin/update")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task UpdateAsAdminGenre([FromBody] GenreModel model)
        {
            var descriptor = _mapper.Map<GenreDescriptor>(model);
            descriptor.SetHeaders(HttpContext.GetHeaderModel());
            await _genreService.UpdateGenreAsync(descriptor);
        }
        [HttpPut("admin/{genreId}/activity")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task ChangeActivity([FromRoute] int genreId)
        {
            await _genreService.ChangeActivityAsync(genreId);
        }
        #endregion
        #region User-Put
        #endregion
        #region Admin-Delete
        [HttpDelete("admin/{genreId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task DeleteAsAdminGenre([FromRoute] int genreId)
        {
            await _genreService.DeleteGenreAsync(genreId);
        }
        #endregion
        #region User-Delete
        #endregion
    }
}
```

Файл HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
namespace BookGarden.Controllers
{
    public sealed class HomeController : Controller
    {
    }
}
```

Файл UserController.cs

```
using AutoMapper;
using BookGarden.Infrastructure.Extensions;
using BookGarden.Models.Admin.From.RequestModels;
```

```
using BookGarden.Models.Admin.From.UpsertModels;
using BookGarden.Models.Admin.To.Users;
using BookGarden.Models.Base;
using BookGarden.Service.Users;
using BookGarden.Service.Users.Descriptors;
using BookGarden.Shared;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using BookGarden.Models.BookGarden.From.Users;
using BookGarden.Models.BookGarden.To.Users;
using BookGarden.Shared.Enums;
using BookGarden.Models.BookGarden.To.Books;
using BookGarden.Models.BookGarden.To;
using BookGarden.Shared.Pagination;
namespace BookGarden.Controllers
{
    [Route("api/user")]
    public class UserController : BookGardenController
    {
        private readonly IUserService _userService;
        private readonly IMapper _mapper;
        public UserController(
            IUserService authService,
            IMapper mapper)
        {
            _userService = authService;
            _mapper = mapper;
        }
        [HttpGet("admin/users")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<PaginatedResponseList<UserAdminResponse>> GetUsers([FromQuery]
UserRequestAdminModel model)
        {
            var descriptor = _mapper.Map<UserRequestAdminDescriptor>(model);
            descriptor.RequestAdminId = User.GetUserId();
            var users = await _userService.GetViewsAsAdminAsync(descriptor);
            return _mapper.Map<PaginatedResponseList<UserAdminResponse>>(users);
        }
        [HttpGet("admin/{userId}")]
        [Authorize(Roles = BookGardenConstants.Roles.Admin)]
        public async Task<FullUserAdminResponse> GetUserAdminInfo([FromRoute] int userId)
        {
            var user = await _userService.GetFullUserAdminViewAsync(userId);
            return _mapper.Map<FullUserAdminResponse>(user);
        }
        [HttpGet("me")]
        [Authorize]
        public async Task<FullUserResponse?> GetMeInfo()
        {
            if (!User.TryGetAsString(ClaimTypes.Email, out var email)) return null;
            var user = await _userService.GetFullUserViewAsync(email);
            return _mapper.Map<FullUserResponse>(user);
        }
        [HttpGet("me/book-status/{status}/{pageSize}/{page}")]
        [Authorize]
        public async Task<PaginatedResponseList<BookLightResponse>> GetSavedBooks([FromRoute]
BookStatus status, int pageSize, int page)
        {
            if (User.TryGetAsString(BookGardenConstants.Claims.UserHash, out var userHash))
            {
                var model = new PaginationModel
                {
```

```
                Page = page,
                PageSize = pageSize,
                RequestLanguage = HttpContext.GetCurrentLanguage()
            };
            var books = await _userService.GetSavedBooksAsync(userHash, status, model);
            return _mapper.Map<PaginatedResponseList<BookLightResponse>>(books);
        }
        return new PaginatedResponseList<BookLightResponse>();
    }
    [HttpGet("me/reviews/last/{pageSize}/{page}")]
    [Authorize]
    public async Task<PaginatedResponseList<ReviewResponse>> GetMyLastReviews(int pageSize, int
page)
    {
        if (User.TryGetAsString(BookGardenConstants.Claims.UserHash, out var userHash))
        {
            var model = new PaginationModel
            {
                Page = page,
                PageSize = pageSize,
                RequestLanguage = HttpContext.GetCurrentLanguage()
            };
            var listItems = await _userService.GetLastReviewsAsync(userHash, model);
            return _mapper.Map<PaginatedResponseList<ReviewResponse>>(listItems);
        }
        return new PaginatedResponseList<ReviewResponse>();
    }
    [HttpPut("admin/update")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task UpdateAsAdmin([FromBody] UserAdminModel model)
    {
        var descriptor = _mapper.Map<UserAdminDescriptor>(model);
        descriptor.SetHeaders(HttpContext.GetHeaderModel());
        await _userService.UpdateAsAdminAsync(descriptor);
    }
    [HttpPut("update/avatar")]
    [Authorize]
    public async Task UpdateAvatar([FromBody] UserAvatarModel model)
    {
        var descriptor = _mapper.Map<UserAvatarDescriptor>(model);
        descriptor.SetHeaders(HttpContext.GetHeaderModel());
        descriptor.UserId = User.GetUserId();
        await _userService.UpdateAvatarAsync(descriptor);
    }
    [HttpPut("{bookUrl}/status/{status}")]
    [Authorize]
    public async Task SetBookStatus([FromRoute] string bookUrl, [FromRoute] BookStatus status)
    {
        await _userService.SetBookStatusAsync(bookUrl, User.GetUserHash(), status);
    }
    [HttpDelete("admin/{userId}")]
    [Authorize(Roles = BookGardenConstants.Roles.Admin)]
    public async Task DeleteUser([FromRoute] int userId)
    {
        await _userService.DeleteAsync(userId, HttpContext.GetCurrentLanguage());
    }
    [HttpDelete("{bookUrl}/status")]
    [Authorize]
    public async Task DeleteBookStatus([FromRoute] string bookUrl)
    {
        await _userService.DeleteBookStatusAsync(bookUrl, User.GetUserHash());
    }
}
```

```
}
```

Файл AuthService.cs

```
using BookGarden.Service.Authorization.Descriptors;
using BookGarden.Storage.Uow;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using Newtonsoft.Json;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System;
using System.Linq;
using BookGarden.Entities.Users;
using BookGarden.Shared;
using BookGarden.Shared.Exceptions;
using BookGarden.Shared.Options;
using CloudinaryDotNet;
using Microsoft.Extensions.Options;
namespace BookGarden.Service.Authorization;
public class AuthService : IAuthService
{
    private readonly IConfiguration _configuration;
    private readonly IUnitOfWork _uow;
    private readonly Cloudinary _cloudinary;
    public AuthService(
        IConfiguration configuration,
        IUnitOfWork uow,
        IOptions<CloudinaryStorageOptions> options)
    {
        _configuration = configuration;
        _uow = uow;
        var account = new Account(
            options.Value.Cloud,
            options.Value.ApiKey,
            options.Value.ApiSecret);
        _cloudinary = new Cloudinary(account);
    }
    public async Task CreateIfNotExistAsync(string userHash)
    {
        var user = await _uow.UserRepository.GetAsync(userHash);
        if (user != null)
        {
            return;
        }
        var userId = await _uow.UserRepository.GetUserNextIdAsync();
        user = new User
        {
            Id = userId + 1,
            UserHash = userHash,
            RoleId = BookGardenConstants.Roles.UserId,
            IsAnonymous = true,
        };
        _uow.UserRepository.Add(user);
        await _uow.CompleteAsync();
    }
    public async Task AddUserAsync(RegistrationDescriptor descriptor)
    {
        if (await _uow.UserRepository.UserWithThisEmailExists(descriptor.Email))
        {
            throw new BookGardenBusinessException("User with such Login is already exist.");
```

```
            }
            if (await _uow.UserRepository.CheckUsernameAsync(descriptor.UserName))
            {
                throw new BookGardenBusinessException("User with such username is already exist.");
            }
            var userId = await _uow.UserRepository.GetUserNextIdAsync();
            CreatePasswordHash(descriptor.Password, out byte[] passwordHash, out byte[] passwordSalt);
            var pictureLink = string.Empty;
            var picturePublicId = string.Empty;
            try
            {
                var images = await _cloudinary.ListResourcesByTagAsync(BookGardenConstants.Cloudinary.Tag.BaseAvatar);
                var index = new Random().Next(0, images.Resources.Length);
                var randomImage = images.Resources[index];
                pictureLink = BookGardenConstants.Cloudinary.BlobUrl + randomImage.Url.AbsolutePath;
                picturePublicId = randomImage.PublicId;
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
            var user = new User
            {
                Id = userId + 1,
                UserHash = Guid.NewGuid().ToString(),
                Email = descriptor.Email,
                UserName = descriptor.UserName,
                Password = JsonConvert.SerializeObject(passwordHash),
                PasswordSalt = passwordSalt,
                RoleId = BookGardenConstants.Roles.UserId,
                IsAnonymous = false,
                PictureLink = pictureLink,
                PicturePublicId = picturePublicId,
            };
            _uow.UserRepository.Add(user);
            await _uow.CompleteAsync();
        }
        public Task<bool> CheckUsernameAsync(string username)
        {
            return _uow.UserRepository.CheckUsernameAsync(username);
        }
        public async Task<string> LoginAndGetTokenAsync(LoginDescriptor descriptor)
        {
            var user = await _uow.UserRepository.FindByEmailAsync(descriptor.Email);
            if (user == null)
            {
                throw new BookGardenBusinessException("User with such Login not found.");
            }
            if (!VerifyPasswordHash(descriptor.Password,
                    JsonConvert.DeserializeObject<byte[]>(user.Password),
                    user.PasswordSalt))
            {
                throw new BookGardenBusinessException("Wrong password.");
            }
            return CreateToken(user);
        }
        private static void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
        {
            using var hmac = new HMACSHA512();
            passwordSalt = hmac.Key;
            passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
```

```
    }
    private static bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)
    {
        using var hmac = new HMACSHA512(passwordSalt);
        var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
        return computedHash.SequenceEqual(passwordHash);
    }
    private string CreateToken(User user)
    {
        var claims = new List<Claim>
            {
                new(BookGardenConstants.Claims.UserId, user.Id.ToString()),
                new(ClaimTypes.Role, user.Role.ToString()),
                new(ClaimTypes.Email, user.Email),
                new(BookGardenConstants.Claims.Username, user.UserName),
                new(BookGardenConstants.Claims.UserHash, user.UserHash),
            };
        var key = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(
            _configuration.GetSection("AppSettings:Token").Value!));
        var cred = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);
        var token = new JwtSecurityToken(
            claims: claims,
            expires: DateTime.Now.AddDays(1),
            signingCredentials: cred
        );
        var jwt = new JwtSecurityTokenHandler().WriteToken(token);
        return jwt;
    }
}
```

Файл AuthorService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BookGarden.Entities.Authors;
using BookGarden.Service.Authors.Descriptors;
using BookGarden.Service.Extensions;
using BookGarden.Shared;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Localization;
using BookGarden.Shared.Options;
using BookGarden.Shared.Pagination;
using BookGarden.Storage.Repositories.AuthorModule.Authors.Filters;
using BookGarden.Storage.Repositories.AuthorModule.Authors.Views;
using BookGarden.Storage.Uow;
using CloudinaryDotNet;
using Microsoft.Extensions.Options;
namespace BookGarden.Service.Authors
{
    public class AuthorService : IAuthorService
    {
        private readonly IUnitOfWork _uow;
        private readonly IMapper _mapper;
        private readonly LocalizationService _localizationService;
        private readonly Cloudinary _cloudinary;
        public AuthorService(
            IUnitOfWork uow,
            IMapper mapper,
            IOptions<CloudinaryStorageOptions> options,
            LocalizationService localizationService)
```

```
        {
            _uow = uow;
            _mapper = mapper;
            _localizationService = localizationService;
            var account = new Account(
                options.Value.Cloud,
                options.Value.ApiKey,
                options.Value.ApiSecret);
            _cloudinary = new Cloudinary(account);
        }
        public async Task<PaginatedList<AuthorAdminView>> GetAuthorViewsAsAdminAsync(AuthorRequestAdminDescriptor descriptor)
        {
            var filter = _mapper.Map<AuthorRequestAdminFilter>(descriptor);
            var items = await _uow.AuthorRepository.GetViewsAsAdminAsync(filter);
            var count = await _uow.AuthorRepository.GetViewsCountAsAdminAsync(filter);
            return new PaginatedList<AuthorAdminView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public Task<Author> GetAsync(int authorId)
        {
            return _uow.AuthorRepository.GetAsync(authorId);
        }
        public Task<FullAuthorView> GetFullViewAsync(string authorUrl, Language lang)
        {
            return _uow.AuthorRepository.GetAsync(authorUrl, lang);
        }
        public Task<int> GetIdAsync(string authorUrl)
        {
            return _uow.AuthorRepository.GetIdAsync(authorUrl);
        }
        public async Task<PaginatedList<AuthorView>> GetViewsAsync(AuthorRequestDescriptor descriptor)
        {
            var filter = _mapper.Map<AuthorRequestFilter>(descriptor);
            var items = await _uow.AuthorRepository.GetViewsAsync(filter);
            var count = await _uow.AuthorRepository.GetViewsCountAsync(filter);
            return new PaginatedList<AuthorView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public Task<List<AuthorLightView>> GetTopViewsAsync(Language lang)
        {
            return _uow.AuthorRepository.GetTopViewsAsync(lang);
        }
        public Task<List<ListItem<int>>> GetOptionsAsAdminAsync(Language lang)
        {
            return _uow.AuthorRepository.GetOptionsAsAdminAsync(lang);
        }
        public Task<List<ListItem<string>>> GetOptionsAsync(Language lang)
        {
            return _uow.AuthorRepository.GetOptionsAsync(lang);
        }
        public Task<List<Author>> GetNewToUpdateAsync(DateTime validDate)
        {
            return _uow.AuthorRepository.GetNewToUpdateAsync(validDate);
        }
        public async Task<int> AddAuthorAsync(AuthorDescriptor descriptor)
```

```
{
    if (descriptor.Image == null)
    {
        throw _localizationService.GetNotFoundException(
            LocalizationConstants.ImageNotFound,
            descriptor.RequestLanguage);
    }
    // todo check name and url
    var entity = _mapper.Map<Author>(descriptor);
    entity.StateStatus = StateStatus.New;
    var result = await _cloudinary.CustomUploadImage(
        descriptor.Image,
        descriptor.UrlName,
        CloudinaryImageType.Author);
    entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
    entity.ImagePublicId = result.PublicId;
    _uow.AuthorRepository.Add(entity);
    var localeList = new[] {
        descriptor.GetByLocale(entity, Language.Russian),
        descriptor.GetByLocale(entity, Language.English),
        descriptor.GetByLocale(entity, Language.Ukrainian)
    };
    _uow.AuthorLocaleRepository.AddRange(localeList);
    await _uow.CompleteAsync();
    return entity.Id;
}
public async Task UpdateAuthorAsync(AuthorDescriptor descriptor)
{
    if (descriptor.Image == null)
    {
        throw _localizationService.GetNotFoundException(
            LocalizationConstants.ImageNotFound,
            descriptor.RequestLanguage);
    }
    // todo check name and url
    if (!descriptor.Id.HasValue)
    {
        return;
    }
    var entity = await _uow.AuthorRepository.GetAsync(descriptor.Id.GetValueOrDefault());
    if (entity == null)
    {
        throw _localizationService.GetNotFoundException(
            LocalizationConstants.AuthorNotFound,
            descriptor.RequestLanguage);
    }
    if (descriptor.Image.Path != entity.ImageLink)
    {
        await _cloudinary.DeleteResourcesAsync(entity.ImagePublicId);
        var result = await _cloudinary.CustomUploadImage(
            descriptor.Image,
            descriptor.UrlName,
            CloudinaryImageType.Author);
        entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
        entity.ImagePublicId = result.PublicId;
    }
    _mapper.Map(descriptor, entity);
    var languages = new[] { Language.English, Language.Russian, Language.Ukrainian };
    foreach (var language in languages)
    {
        var locale = entity.Locales.FirstOrDefault(locale => locale.Language == language);
        if (locale == null)
        {
```

```
                locale = descriptor.GetByLocale(entity, language);
                _uow.AuthorLocaleRepository.Add(locale);
            }
            else
            {
                locale.UpdateLocales(descriptor);
            }
        }
        await _uow.CompleteAsync();
    }
    public async Task ChangeActivityAsync(int authorId)
    {
        var item = await _uow.AuthorRepository.FindAsync(authorId);
        item.IsActive = !item.IsActive;
        await _uow.CompleteAsync();
    }
    public Task<bool> IsUrlExistAsync(string urlName, int authorId)
    {
        return _uow.AuthorRepository.IsUrlExistAsync(urlName, authorId);
    }
    public async Task DeleteAuthorAsync(int authorId)
    {
        var item = await _uow.AuthorRepository.FindAsync(authorId);
        await _cloudinary.DeleteResourcesAsync(item.ImagePublicId);
        _uow.AuthorRepository.Remove(item);
        await _uow.CompleteAsync();
    }
    public Task<int> CompleteAsync()
    {
        return _uow.CompleteAsync();
    }
  }
}
```

Файл AzureBlobStorage.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Azure;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using BookGarden.Service.BlobStorage.Models;
using BookGarden.Shared.Options;
using Microsoft.AspNetCore.Http;
namespace BookGarden.Service.BlobStorage
{
  public class AzureBlobStorage : IAzureBlobStorage
  {
    private readonly string _storageConnectionString;
    private readonly string _storageContainerName;
    public AzureBlobStorage(AzureBlobOptions options)
    {
      _storageConnectionString = options.ConnectionString;
      _storageContainerName = options.ContainerName;
    }
    public async Task<List<BlobModel>> ListAsync()
    {
      var container = new BlobContainerClient(_storageConnectionString, _storageContainerName);
      var files = new List<BlobModel>();
      await foreach (var file in container.GetBlobsAsync())
      {
        var uri = container.Uri.ToString();
        var name = file.Name;
        var fullUri = $"{uri}/{name}";
```

```
                files.Add(new BlobModel
                {
                    Uri = fullUri,
                    Name = name,
                    ContentType = file.Properties.ContentType
                });
            }
            return files;
        }
        public async Task<BlobResponse> UploadAsync(IFormFile file, string fileName)
        {
            BlobResponse response = new();
            var container = new BlobContainerClient(_storageConnectionString, _storageContainerName);
            try
            {
                var client = container.GetBlobClient(fileName);
                await using var data = file.OpenReadStream();
                await client.UploadAsync(data);
                response.Status = $"File {fileName} Uploaded Successfully";
                response.Error = false;
                response.Blob.Uri = client.Uri.AbsoluteUri;
                response.Blob.Name = client.Name;
            }
            catch (RequestFailedException ex) when (ex.ErrorCode == BlobErrorCode.BlobAlreadyExists)
            {
                response.Status = $"File with name {fileName} already exists. Please use another name to store
your file.";
                response.Error = true;
                return response;
            }
            catch (RequestFailedException ex)
            {
                response.Status = $"Unexpected error: {ex.StackTrace}. Check log with StackTrace ID.";
                response.Error = true;
                return response;
            }
            return response;
        }
        public async Task<BlobModel> DownloadAsync(string blobFilename)
        {
            var client = new BlobContainerClient(_storageConnectionString, _storageContainerName);
            var file = client.GetBlobClient(blobFilename);
            if (await file.ExistsAsync())
            {
                var data = await file.OpenReadAsync();
                var blobContent = data;
                var content = await file.DownloadContentAsync();
                var name = blobFilename;
                var contentType = content.Value.Details.ContentType;
                return new BlobModel { Content = blobContent, Name = name, ContentType = contentType };
            }
            return null;
        }
        public async Task<BlobResponse> DeleteAsync(string blobFilename)
        {
            var client = new BlobContainerClient(_storageConnectionString, _storageContainerName);
            var file = client.GetBlobClient(blobFilename);
            try
            {
                await file.DeleteAsync();
            }
            catch (RequestFailedException ex) when (ex.ErrorCode == BlobErrorCode.BlobNotFound)
            {
```

```
                return new BlobResponse { Error = true, Status = $"File with name {blobFilename} not found." };
            }
            return new BlobResponse { Error = false, Status = $"File: {blobFilename} has been successfully deleted." };
        }
    }
}
```

Файл BookService.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BookGarden.Entities.Books;
using BookGarden.Service.BlobStorage;
using BookGarden.Service.Books.Descriptors;
using BookGarden.Service.Extensions;
using BookGarden.Shared;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Localization;
using BookGarden.Shared.Options;
using BookGarden.Shared.Pagination;
using BookGarden.Storage.Repositories.BookModule.Books.Filters;
using BookGarden.Storage.Repositories.BookModule.Books.Views;
using BookGarden.Storage.Uow;
using CloudinaryDotNet;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Options;
namespace BookGarden.Service.Books
{
    public class BookService : IBookService
    {
        private readonly IUnitOfWork _uow;
        private readonly IMapper _mapper;
        private readonly LocalizationService _localizationService;
        private readonly Cloudinary _cloudinary;
        private readonly IAzureBlobStorage _blobStorage;
        public BookService(
            IUnitOfWork uow,
            IMapper mapper,
            IOptions<CloudinaryStorageOptions> options,
            LocalizationService localizationService,
            IAzureBlobStorage blobStorage)
        {
            _uow = uow;
            _mapper = mapper;
            _localizationService = localizationService;
            _blobStorage = blobStorage;
            var account = new Account(
                options.Value.Cloud,
                options.Value.ApiKey,
                options.Value.ApiSecret);
            _cloudinary = new Cloudinary(account);
        }
        public async Task<PaginatedList<BookViewWithAuthors>> SearchBookAsAdminAsync(BookSearchAdminDescriptor descriptor)
        {
            var filter = _mapper.Map<BookSearchAdminFilter>(descriptor);
            var items = await _uow.BookRepository.SearchBookAsAdminAsync(filter);
```

```
            return items.GetPaginationList(descriptor);
        }
        public Task<List<ListItem<int>>> SearchOptionAsAdminAsync(string search, Language lang)
        {
            return _uow.BookRepository.SearchOptionAsAdminAsync(search, lang);
        }
        public async Task<PaginatedList<BookView>> GetViewsAsAdmin(BookRequestAdminDescriptor
descriptor)
        {
            var filter = _mapper.Map<BookRequestAdminFilter>(descriptor);
            var items = await _uow.BookRepository.GetViewsAsAdminAsync(filter);
            var count = await _uow.BookRepository.GetViewsCountAsAdminAsync(filter);
            return new PaginatedList<BookView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public async Task<PaginatedList<BookGridView>> GetViewsAsync(BookRequestDescriptor
descriptor)
        {
            var filter = _mapper.Map<BookRequestFilter>(descriptor);
            var items = await _uow.BookRepository.GetViewsAsync(filter);
            var count = await _uow.BookRepository.GetViewsCountAsync(filter);
            return new PaginatedList<BookGridView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public Task<Book> GetAsync(int bookId)
        {
            return _uow.BookRepository.GetAsync(bookId);
        }
        public Task<FullBookView> GetAsync(string bookUrl, Language lang)
        {
            return _uow.BookRepository.GetAsync(bookUrl, lang);
        }
        public Task<int> GetIdAsync(string bookUrl)
        {
            return _uow.BookRepository.GetIdAsync(bookUrl);
        }
        public Task<BookReadView> GetReadViewAsync(string bookUrl)
        {
            return _uow.BookRepository.GetReadViewAsync(bookUrl);
        }
        public async Task<Tuple<Stream, string>> GetFileLinkAsync(string bookUrl, FileType format)
        {
            var fileView = await _uow.BookFileRepository.GetFileLinkAsync(bookUrl, format);
            var result = await _blobStorage.DownloadAsync(fileView.Item2);
            return new Tuple<Stream, string>(result.Content, fileView.Item1);
        }
        public async Task<PaginatedList<BookGridView>> GetByCompilationAsync(string compilationUrl,
SortPaginationModel model)
        {
            var compilationId = await _uow.CompilationRepository.GetIdAsync(compilationUrl);
            var                items                =                await
_uow.BookCompilationRepository.GetViewsByCompilationAsync(compilationId, model);
            var                count                =                await
_uow.BookCompilationRepository.GetViewsCountByCompilationAsync(compilationId, model);
            return new PaginatedList<BookGridView>
            {
                Items = items,
```

```
            TotalCount = count
        };
    }
    public async Task<List<BookViewWithAuthors>> GetSimilarAsync(string bookUrl, Language lang)
    {
        var genreIds = await _uow.BookRepository.GetGenresAsync(bookUrl);
        return await  _uow.BookRepository.GetMoreByGenre(genreIds, bookUrl, lang);
    }
    public async Task<List<BookViewWithAuthors>> GetByAuthorAsync(string bookUrl, string
authorUrl, Language lang)
    {
        var authorId = await _uow.AuthorRepository.GetIdAsync(authorUrl);
        return await _uow.BookRepository.GetMoreByAuthor(authorId, bookUrl, lang);
    }
    public async Task<List<BookViewWithAuthors>> GetByGenreAsync(string bookUrl, string
genreUrl, Language lang)
    {
        var genreId = await _uow.GenreRepository.GetIdAsync(genreUrl);
        return await _uow.BookRepository.GetMoreByGenre(genreId, bookUrl, lang);
    }
    public Task<List<BookViewWithAuthors>> GetViewsByAuthorAsAdminAsync(int authorId,
Language lang)
    {
        return _uow.BookRepository.GetViewsByAuthorAsAdminAsync(authorId, lang);
    }
    public Task<List<BookViewWithAuthors>> GetViewsByCompilationAsAdminAsync(int
compilationId, Language lang)
    {
        return _uow.BookRepository.GetViewsByCompilationAsAdminAsync(compilationId, lang);
    }
    public Task<List<BookViewWithAuthors>> GetTranslatedViewsAsAdminAsync(int bookId,
Language lang)
    {
        return _uow.BookRepository.GetTranslatedViewsAsAdminAsync(bookId, lang);
    }
    public Task<List<FileType>> GetFilesAsync(int bookId)
    {
        return _uow.BookFileRepository.GetFilesAsync(bookId);
    }
    public Task<List<Book>> GetNewToUpdateAsync(DateTime validDate)
    {
        return _uow.BookRepository.GetNewToUpdateAsync(validDate);
    }
    public Task<List<BookViewWithAuthors>> GetNewViewsAsync(Language language)
    {
        return _uow.BookRepository.GetNewViewsAsync(language);
    }
    public Task<List<BookViewWithAuthors>> GetTopViewsAsync(Language language)
    {
        return _uow.BookRepository.GetTopViewsAsync(language);
    }
    public async Task<int> AddAsync(BookDescriptor descriptor)
    {
        if (descriptor.Image == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.ImageNotFound,
                descriptor.RequestLanguage);
        }
        // todo validation
        var entity = _mapper.Map<Book>(descriptor);
        entity.StateStatus = StateStatus.New;
        var result = await _cloudinary.CustomUploadImage(
```

```
            descriptor.Image,
            descriptor.UrlName,
            CloudinaryImageType.Book);
        entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
        entity.ImagePublicId = result.PublicId;
        _uow.BookRepository.Add(entity);
        AddGenre(descriptor.GenreIds, entity);
        AddAuthor(descriptor.AuthorIds, entity);
        AddCompilation(descriptor.CompilationIds, entity);
        await _uow.CompleteAsync();
        return entity.Id;
    }
    public async Task UpdateAsync(BookDescriptor descriptor)
    {
        if (descriptor.Image == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.ImageNotFound,
                descriptor.RequestLanguage);
        }
        if (!descriptor.Id.HasValue)
        {
            return;
        }
        // todo validation
        var entity = await _uow.BookRepository.GetAsync(descriptor.Id.GetValueOrDefault());
        if (entity == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.BookNotFound,
                descriptor.RequestLanguage);
        }
        if (descriptor.Image.Path != entity.ImageLink)
        {
            await _cloudinary.DeleteResourcesAsync(entity.ImagePublicId);
            var result = await _cloudinary.CustomUploadImage(
                descriptor.Image,
                descriptor.UrlName,
                CloudinaryImageType.Book);
            entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
            entity.ImagePublicId = result.PublicId;
        }
        _mapper.Map(descriptor, entity);
        await UpdateGenreAsync(descriptor, entity);
        await UpdateAuthorAsync(descriptor, entity);
        await UpdateCompilationAsync(descriptor, entity);
        await _uow.CompleteAsync();
    }
    public async Task<int> AddFileAsync(int bookId, IFormFile file, FileType format)
    {
        var fileName = $"{bookId}/{file.FileName}";
        var result = await _blobStorage.UploadAsync(file, fileName);
        var bookFile = new BookFile
        {
            BookId = bookId,
            BlobFileName = result.Blob.Name,
            BlobUrl = result.Blob.Uri,
            FileType = format
        };
        _uow.BookFileRepository.Add(bookFile);
        var entity = await _uow.BookRepository.GetAsync(bookId);
        entity.StateStatus = StateStatus.New;
        return await _uow.CompleteAsync();
```

```
        }
        public async Task AddBookToAuthorAsync(int authorId, int bookId)
        {
            var isExist = await _uow.BookAuthorRepository.CheckAsync(authorId, bookId);
            if (isExist) return;
            var entity = new BookAuthor
            {
                AuthorId = authorId,
                BookId = bookId,
            };
            _uow.BookAuthorRepository.Add(entity);
            await _uow.CompleteAsync();
        }
        public async Task DeleteFromAuthorAsync(int authorId, int bookId)
        {
            var entity = await _uow.BookAuthorRepository
                .FirstOrDefaultAsync(item => item.AuthorId == authorId && item.BookId == bookId);
            if (entity == null) return;
            _uow.BookAuthorRepository.Remove(entity);
            await _uow.CompleteAsync();
        }
        public async Task AddBookToCompilationAsync(int compilationId, int bookId)
        {
            var isExist = await _uow.BookCompilationRepository.CheckAsync(compilationId, bookId);
            if (isExist) return;
            var entity = new BookCompilation
            {
                CompilationId = compilationId,
                BookId = bookId,
            };
            _uow.BookCompilationRepository.Add(entity);
            await _uow.CompleteAsync();
        }
        public async Task DeleteFromCompilationAsync(int compilationId, int bookId)
        {
            var entity = await _uow.BookCompilationRepository
                .FirstOrDefaultAsync(item => item.CompilationId == compilationId && item.BookId ==
bookId);
            if (entity == null) return;
            _uow.BookCompilationRepository.Remove(entity);
            await _uow.CompleteAsync();
        }
        public async Task ChangeActivityAsync(int bookId)
        {
            var item = await _uow.BookRepository.FindAsync(bookId);
            item.IsActive = !item.IsActive;
            await _uow.CompleteAsync();
        }
        public Task<bool> CheckUrlAsync(string urlName, int bookId)
        {
            return _uow.BookRepository.CheckUrlAsync(urlName, bookId);
        }
        public Task<bool> IsFileFormatFreeAsync(int bookId, FileType format)
        {
            return _uow.BookFileRepository.IsFileFormatFreeAsync(bookId, format);
        }
        public async Task DeleteAsync(int bookId)
        {
            var item = await _uow.BookRepository.GetAsync(bookId);
            foreach (var book in item.TranslatedBooks)
            {
                book.OriginalBookId = null;
            }
```

```
        await _cloudinary.DeleteResourcesAsync(item.ImagePublicId);
        _uow.BookUserRepository.RemoveRange(item.SavedBooks);
        _uow.BookRepository.Remove(item);
        await _uow.CompleteAsync();
    }
    public async Task DeleteFileAsync(int bookId, FileType type)
    {
        var file = await _uow.BookFileRepository
            .FirstOrDefaultAsync(item => item.BookId == bookId && item.FileType == type);
        try
        {
            await _blobStorage.DeleteAsync(file.BlobFileName);
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        _uow.BookFileRepository.Remove(file);
        await _uow.CompleteAsync();
    }
    public Task<int> CompleteAsync()
    {
        return _uow.CompleteAsync();
    }
    private async Task UpdateGenreAsync(BookDescriptor descriptor, Book book)
    {
        var entityGenreIds = book.Genres.Select(b => b.GenreId).ToArray();
        var descriptorGenreIds = descriptor.GenreIds;
        var idsToDelete = entityGenreIds.Except(descriptorGenreIds);
        var toDelete = await _uow.BookGenreRepository
            .FindAsync(item => item.BookId == book.Id && idsToDelete.Contains(item.GenreId));
        AddGenre(descriptorGenreIds.Except(entityGenreIds), book);
        _uow.BookGenreRepository.RemoveRange(toDelete);
    }
    private void AddGenre(IEnumerable<int> ids, Book book)
    {
        var toAdd = ids
            .Select(id => new BookGenre
            {
                Book = book,
                GenreId = id
            });
        _uow.BookGenreRepository.AddRange(toAdd);
    }
    private async Task UpdateAuthorAsync(BookDescriptor descriptor, Book book)
    {
        var entityAuthorIds = book.Authors.Select(b => b.AuthorId).ToArray();
        var descriptorAuthorIds = descriptor.AuthorIds;
        var idsToDelete = entityAuthorIds.Except(descriptorAuthorIds);
        var toDelete = await _uow.BookAuthorRepository
            .FindAsync(item => item.BookId == book.Id && idsToDelete.Contains(item.AuthorId));
        AddAuthor(descriptorAuthorIds.Except(entityAuthorIds), book);
        _uow.BookAuthorRepository.RemoveRange(toDelete);
    }
    private void AddAuthor(IEnumerable<int> ids, Book book)
    {
        var toAdd = ids
            .Select(id => new BookAuthor
            {
                Book = book,
                AuthorId = id
            });
        _uow.BookAuthorRepository.AddRange(toAdd);
```

```
        }
        private async Task UpdateCompilationAsync(BookDescriptor descriptor, Book book)
        {
            var entityCompilationIds = book.Compilations.Select(b => b.CompilationId).ToArray();
            var descriptorCompilationIds = descriptor.CompilationIds;
            var idsToDelete = entityCompilationIds.Except(descriptorCompilationIds);
            var toDelete = await _uow.BookCompilationRepository
                .FindAsync(item => item.BookId == book.Id && idsToDelete.Contains(item.CompilationId));
            AddCompilation(descriptorCompilationIds.Except(entityCompilationIds), book);
            _uow.BookCompilationRepository.RemoveRange(toDelete);
        }
        private void AddCompilation(IEnumerable<int> ids, Book book)
        {
            var toAdd = ids
                .Select(id => new BookCompilation
                {
                    Book = book,
                    CompilationId = id
                });
            _uow.BookCompilationRepository.AddRange(toAdd);
        }
    }
}
```

Файл CompilationService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BookGarden.Entities.Compilations;
using BookGarden.Service.Compilations.Descriptors;
using BookGarden.Service.Extensions;
using BookGarden.Shared;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Localization;
using BookGarden.Shared.Options;
using BookGarden.Shared.Pagination;
using BookGarden.Storage.Repositories.CompilationModule.Compilations.Filters;
using BookGarden.Storage.Repositories.CompilationModule.Compilations.Views;
using BookGarden.Storage.Uow;
using CloudinaryDotNet;
using Microsoft.Extensions.Options;
namespace BookGarden.Service.Compilations
{
    public class CompilationService : ICompilationService
    {
        private readonly IUnitOfWork _uow;
        private readonly IMapper _mapper;
        private readonly LocalizationService _localizationService;
        private readonly Cloudinary _cloudinary;
        public CompilationService(
            IUnitOfWork uow,
            IMapper mapper,
            IOptions<CloudinaryStorageOptions> options,
            LocalizationService localizationService)
        {
            _uow = uow;
            _mapper = mapper;
            _localizationService = localizationService;
            var account = new Account(
                options.Value.Cloud,
```

```
                options.Value.ApiKey,
                options.Value.ApiSecret);
            _cloudinary = new Cloudinary(account);
        }
        public async Task<PaginatedList<CompilationAdminView>>
GetAsAdminAsync(CompilationRequestAdminDescriptor descriptor)
        {
            var filter = _mapper.Map<CompilationRequestAdminFilter>(descriptor);
            var items = await _uow.CompilationRepository.GetViewsAsAdminAsync(filter);
            var count = await _uow.CompilationRepository.GetViewsCountAsAdminAsync(filter);
            return new PaginatedList<CompilationAdminView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public Task<Compilation> GetAsync(int compilationId)
        {
            return _uow.CompilationRepository.GetAsync(compilationId);
        }
        public Task<int> GetIdAsync(string compilationUrl)
        {
            return _uow.CompilationRepository.GetIdAsync(compilationUrl);
        }
        public Task<FullCompilationView> GetFullViewAsync(string compilationUrl, Language lang)
        {
            return _uow.CompilationRepository.GetFullViewAsync(compilationUrl, lang);
        }
        public async Task<PaginatedList<CompilationView>>
GetViewsAsync(CompilationRequestDescriptor descriptor)
        {
            var filter = _mapper.Map<CompilationRequestFilter>(descriptor);
            var items = await _uow.CompilationRepository.GetViewsAsync(filter);
            var count = await _uow.CompilationRepository.GetViewsCountAsync(filter);
            return new PaginatedList<CompilationView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public Task<List<CompilationLightView>> GetTopViewsAsync(Language lang)
        {
            return _uow.CompilationRepository.GetTopViewsAsync(lang);
        }
        public async Task<int> AddAsync(CompilationDescriptor descriptor)
        {
            if (descriptor.Image == null)
            {
                throw _localizationService.GetNotFoundException(
                    LocalizationConstants.ImageNotFound,
                    descriptor.RequestLanguage);
            }
            // todo check name and url
            var entity = _mapper.Map<Compilation>(descriptor);
            entity.StateStatus = StateStatus.New;
            var result = await _cloudinary.CustomUploadImage(
                descriptor.Image,
                descriptor.UrlName,
                CloudinaryImageType.Compilation);
            entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
            _uow.CompilationRepository.Add(entity);
            var localeList = new[]
            {
```

```
            descriptor.GetByLocale(entity, Language.Russian),
            descriptor.GetByLocale(entity, Language.English),
            descriptor.GetByLocale(entity, Language.Ukrainian)
        };
        _uow.CompilationLocaleRepository.AddRange(localeList);
        await _uow.CompleteAsync();
        return entity.Id;
    }
    public async Task UpdateAsync(CompilationDescriptor descriptor)
    {
        if (descriptor.Image == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.ImageNotFound,
                descriptor.RequestLanguage);
        }
        // todo check name and url
        if (!descriptor.Id.HasValue)
        {
            return;
        }
        var entity = await _uow.CompilationRepository.GetAsync(descriptor.Id.GetValueOrDefault());
        if (entity == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.CompilationNotFound,
                descriptor.RequestLanguage);
        }
        if (descriptor.Image.Path != entity.ImageLink)
        {
            await _cloudinary.DeleteResourcesAsync(entity.ImagePublicId);
            var result = await _cloudinary.CustomUploadImage(
                descriptor.Image,
                descriptor.UrlName,
                CloudinaryImageType.Compilation);
            entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
            entity.ImagePublicId = result.PublicId;
        }
        _mapper.Map(descriptor, entity);
        var languages = new[] { Language.English, Language.Russian, Language.Ukrainian };
        foreach (var language in languages)
        {
            var locale = entity.Locales.FirstOrDefault(locale => locale.Language == language);
            if (locale == null)
            {
                locale = descriptor.GetByLocale(entity, language);
                _uow.CompilationLocaleRepository.Add(locale);
            }
            else
            {
                locale.UpdateLocales(descriptor);
            }
        }
        await _uow.CompleteAsync();
    }
    public async Task ChangeActivityAsync(int compilationId)
    {
        var item = await _uow.CompilationRepository.FindAsync(compilationId);
        item.IsActive = !item.IsActive;
        await _uow.CompleteAsync();
    }
    public Task<bool> IsUrlExistAsync(string urlName, int compilationId)
    {
```

```
            return _uow.CompilationRepository.IsUrlExistAsync(urlName, compilationId);
        }
        public async Task DeleteAsync(int compilationId)
        {
            var item = await _uow.CompilationRepository.FindAsync(compilationId);
            await _cloudinary.DeleteResourcesAsync(item.ImagePublicId);
            _uow.CompilationRepository.Remove(item);
            await _uow.CompleteAsync();
        }
        public Task<List<ListItem<int>>> GetOptionsAsync(Language lang)
        {
            return _uow.CompilationRepository.GetOptionsAsync(lang);
        }
        public Task<List<Compilation>> GetNewToUpdateAsync(DateTime validDate)
        {
            return _uow.CompilationRepository.GetNewToUpdateAsync(validDate);
        }
        public Task<int> CompleteAsync()
        {
            return _uow.CompleteAsync();
        }
    }
}
```

Файл GenreExtension.cs

```
using System.Collections.Generic;
using System.Linq;
using BookGarden.Service.Genres.Views;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Storage.Repositories.GenreModule.Genres.Views;
namespace BookGarden.Service.Extensions
{
    public static class GenreExtension
    {
        public static List<ComplexGenre> CombineGenres(this ICollection<GenreStoredView> genres)
        {
            var complexGenres = new List<ComplexGenre>();
            complexGenres.AddRange(genres
                .Where(genre => !genre.ParentGenreId.HasValue)
                .Select(genre => new ComplexGenre(genre.Id, genre.NameEn, genre.NameRu, genre.NameUa,
genre.UrlName)));
            foreach (var genre in genres.Where(genre => genre.ParentGenreId.HasValue))
            {
                var complex = complexGenres.Find(item => item.Id == genre.ParentGenreId);
                var genreToAdd = new ComplexGenre(genre.Id, genre.NameEn, genre.NameRu, genre.NameUa,
genre.UrlName);
                if (complex == default)
                {
                    complexGenres.Add(genreToAdd);
                }
                else
                {
                    complex.SubGenres.Add(genreToAdd);
                }
            }
            return complexGenres;
        }
        public static List<ComplexLocaleGenre> ToComplexLocaleGenre(this ICollection<ComplexGenre>
genres, Language lang)
        {
            return genres
```

```
                .Select(genre => new ComplexLocaleGenre(genre.GetName(lang), genre.UrlName,
genre.SubGenres, lang))
                .OrderBy(item => item.Name)
                .ToList();
        }
        public static List<ListItem<int>> ToListItems(this IList<ComplexGenre> genres, Language lang)
        {
            var list = new List<ListItem<int>>();
            foreach (var genre in genres)
            {
                list.Add(new ListItem<int>
                {
                    Value = genre.Id,
                    Text = genre.GetName(lang)
                });
                list.AddRange(genre.SubGenres.Select(item => new ListItem<int>
                {
                    Value = item.Id,
                    Text = $"- {item.GetName(lang)}"
                }));
            }
            return list;
        }
    }
}
```

Файл LocaleExtension.cs

```
using System;
using BookGarden.Entities.Authors;
using BookGarden.Entities.Compilations;
using BookGarden.Entities.Genres;
using BookGarden.Service.Authors.Descriptors;
using BookGarden.Service.Compilations.Descriptors;
using BookGarden.Service.Genres.Descriptors;
using BookGarden.Shared.Enums;
namespace BookGarden.Service.Extensions
{
    public static class LocaleExtension
    {
        public static void UpdateLocales(this GenreLocale locale, GenreDescriptor descriptor)
        {
            switch (locale.Language)
            {
                case Language.English:
                    locale.Title = descriptor.TitleEn;
                    locale.Description = descriptor.DescriptionEn;
                    break;
                case Language.Russian:
                    locale.Title = descriptor.TitleRu;
                    locale.Description = descriptor.DescriptionRu;
                    break;
                case Language.Ukrainian:
                    locale.Title = descriptor.TitleUa;
                    locale.Description = descriptor.DescriptionUa;
                    break;
                default:
                    throw new ArgumentOutOfRangeException(locale.Language.ToString());
            }
        }
        public static void UpdateLocales(this AuthorLocale locale, AuthorDescriptor descriptor)
        {
            switch (locale.Language)
            {
```

```
                case Language.English:
                    locale.FullName = descriptor.FullNameEn;
                    locale.Description = descriptor.DescriptionEn;
                    break;
                case Language.Russian:
                    locale.FullName = descriptor.FullNameRu;
                    locale.Description = descriptor.DescriptionRu;
                    break;
                case Language.Ukrainian:
                    locale.FullName = descriptor.FullNameUa;
                    locale.Description = descriptor.DescriptionUa;
                    break;
                default:
                    throw new ArgumentOutOfRangeException(locale.Language.ToString());
            }
        }
        public static void UpdateLocales(this CompilationLocale locale, CompilationDescriptor descriptor)
        {
            switch (locale.Language)
            {
                case Language.English:
                    locale.Title = descriptor.TitleEn;
                    locale.Description = descriptor.DescriptionEn;
                    break;
                case Language.Russian:
                    locale.Title = descriptor.TitleRu;
                    locale.Description = descriptor.DescriptionRu;
                    break;
                case Language.Ukrainian:
                    locale.Title = descriptor.TitleUa;
                    locale.Description = descriptor.DescriptionUa;
                    break;
                default:
                    throw new ArgumentOutOfRangeException(locale.Language.ToString());
            }
        }
    }
}
```

Файл PaginationExtension.cs

```
using System.Collections.Generic;
using System.Linq;
using BookGarden.Shared.Pagination;
namespace BookGarden.Service.Extensions
{
    public static class PaginationExtension
    {
        public static PaginatedList<T> GetPaginationList<T>(
            this IEnumerable<T> list,
            PaginationModel model)
        {
            return new PaginatedList<T>
            {
                TotalCount = list.Count(),
                Items = list.Skip(--model.Page * model.PageSize)
                    .Take(model.PageSize)
                    .ToList()
            };
        }
    }
}
```

Файл ReviewExtension.cs

```
using System.Collections.Generic;
using System.Linq;
using BookGarden.Service.Reviews.Views;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Localization;
using BookGarden.Storage.Repositories.ReviewModule.Views;
namespace BookGarden.Service.Extensions
{
   public static class ReviewExtension
   {
      public static List<ReviewWithRepliesView> CombineReviews(this IEnumerable<ReviewView>
reviews, Language lang)
      {
         var complexReviews = new List<ReviewWithRepliesView>();
         complexReviews.AddRange(reviews
            .Where(review => !review.ReplyToId.HasValue)
            .Select(review => new ReviewWithRepliesView(review, lang)));
         foreach (var review in reviews.Where(genre => genre.ReplyToId.HasValue))
         {
            var complexReview = complexReviews.Find(item => item.Id == review.ReplyToId);
            var reviewToAdd = new ReviewWithRepliesView(review, lang);
            if (complexReview == default)
            {
               complexReviews.Add(reviewToAdd);
            }
            else
            {
               complexReview.Replies.Add(reviewToAdd);
            }
         }
         return complexReviews;
      }
      public static List<ReviewWithRepliesView> ToReviewWithRepliesView(
         this IEnumerable<ReviewView> reviews,
         Language lang,
         LocalizationService localizationService,
         bool isOneUser = false)
      {
         var complexReviews = new List<ReviewWithRepliesView>();
         complexReviews.AddRange(reviews
            .Where(review => !review.ReplyToId.HasValue)
            .Select(review => new ReviewWithRepliesView(review, lang,
localizationService.GetStringByReviewType(review.ReviewType, lang), isOneUser)));
         return complexReviews;
      }
   }
}
```

Файл GenreService.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BookGarden.Entities.Genres;
using BookGarden.Service.Extensions;
using BookGarden.Service.Genres.Descriptors;
using BookGarden.Service.Genres.Views;
using BookGarden.Shared;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Localization;
using BookGarden.Shared.Options;
using BookGarden.Shared.Pagination;
```

```
using BookGarden.Storage.Repositories.GenreModule.Genres.Filters;
using BookGarden.Storage.Repositories.GenreModule.Genres.Views;
using BookGarden.Storage.Uow;
using CloudinaryDotNet;
using Microsoft.Extensions.Options;
namespace BookGarden.Service.Genres
{
   public class GenreService : IGenreService
   {
      private readonly IUnitOfWork _uow;
      private readonly IMapper _mapper;
      private readonly LocalizationService _localizationService;
      private readonly Cloudinary _cloudinary;
      public GenreService(
         IUnitOfWork uow,
         IMapper mapper,
         IOptions<CloudinaryStorageOptions> options,
         LocalizationService localizationService)
      {
         _uow = uow;
         _mapper = mapper;
         _localizationService = localizationService;
         var account = new Account(
            options.Value.Cloud,
            options.Value.ApiKey,
            options.Value.ApiSecret);
         _cloudinary = new Cloudinary(account);
      }
      public async Task<PaginatedList<GenreAdminView>>
GetGenreViewsAsAdminAsync(GenreRequestAdminDescriptor descriptor)
      {
         var filter = _mapper.Map<GenreRequestAdminFilter>(descriptor);
         var items = await _uow.GenreRepository.GetViewsAsAdminAsync(filter);
         var count = await _uow.GenreRepository.GetViewsCountAsAdminAsync(filter);
         return new PaginatedList<GenreAdminView>
         {
            Items = items,
            TotalCount = count
         };
      }
      public Task<Genre> GetAsync(int genreId)
      {
         return _uow.GenreRepository.GetAsync(genreId);
      }
      public Task<FullGenreView> GetViewAsync(string genreUrl, Language lang)
      {
         return _uow.GenreRepository.GetViewAsync(genreUrl, lang);
      }
      public Task<int> GetIdAsync(string genreUrl)
      {
         return _uow.GenreRepository.GetIdAsync(genreUrl);
      }
      public Task<List<GenreLightView>> GetTopViewsAsync(Language lang)
      {
         return _uow.GenreRepository.GetTopViewsAsync(lang);
      }
      public async Task<PaginatedList<GenreView>> GetViewsAsync(GenreRequestDescriptor descriptor)
      {
         var filter = _mapper.Map<GenreRequestFilter>(descriptor);
         var items = await _uow.GenreRepository.GetViewsAsync(filter);
         var count = await _uow.GenreRepository.GetViewsCountAsync(filter);
         return new PaginatedList<GenreView>
         {
```

```
            Items = items,
            TotalCount = count
        };
    }
    public async Task<List<ListItem<int>>> GetAdminSelectListAsync(Language lang)
    {
        return (await _uow.GenreRepository.GetViewsAsync(true))
            .CombineGenres()
            .ToListItems(lang);
    }
    public async Task<List<ComplexLocaleGenre>> GetOptionsTreeAsync(Language lang)
    {
        return (await _uow.GenreRepository.GetViewsAsync(false))
            .CombineGenres()
            .ToComplexLocaleGenre(lang);
    }
    public Task<List<ListItem<string>>> GetParentOptionsAsync(Language lang)
    {
        return _uow.GenreRepository.GetParentOptionsAsync(lang);
    }
    public async Task<int> AddGenreAsync(GenreDescriptor descriptor)
    {
        if (descriptor.Image == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.ImageNotFound,
                descriptor.RequestLanguage);
        }
        // todo check name and link
        var entity = _mapper.Map<Genre>(descriptor);
        var result = await _cloudinary.CustomUploadImage(
            descriptor.Image,
            descriptor.UrlName,
            CloudinaryImageType.Genre);
        entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
        _uow.GenreRepository.Add(entity);
        var localeList = new[]
        {
            descriptor.GetByLocale(entity, Language.Russian),
            descriptor.GetByLocale(entity, Language.English),
            descriptor.GetByLocale(entity, Language.Ukrainian)
        };
        _uow.GenreLocaleRepository.AddRange(localeList);
        await _uow.CompleteAsync();
        return entity.Id;
    }
    public async Task UpdateGenreAsync(GenreDescriptor descriptor)
    {
        if (descriptor.Image == null)
        {
            throw _localizationService.GetNotFoundException(
                LocalizationConstants.ImageNotFound,
                descriptor.RequestLanguage);
        }
        // todo check name and link
        if (!descriptor.Id.HasValue)
        {
            return;
        }
        var entity = await _uow.GenreRepository.GetAsync(descriptor.Id.GetValueOrDefault());
        if (entity == null)
        {
            throw _localizationService.GetNotFoundException(
```

```
                LocalizationConstants.GenreNotFound,
                descriptor.RequestLanguage);
            }
            if (descriptor.Image.Path != entity.ImageLink)
            {
                await _cloudinary.DeleteResourcesAsync(entity.ImagePublicId);
                var result = await _cloudinary.CustomUploadImage(
                    descriptor.Image,
                    descriptor.UrlName,
                    CloudinaryImageType.Genre);
                entity.ImageLink = BookGardenConstants.Cloudinary.BlobUrl + result.Url.AbsolutePath;
                entity.ImagePublicId = result.PublicId;
            }
            _mapper.Map(descriptor, entity);
            var languages = new [] { Language.English, Language.Russian, Language.Ukrainian };
            foreach (var language in languages)
            {
                var locale = entity.Locales.FirstOrDefault(locale => locale.Language == language);
                if (locale == null)
                {
                    locale = descriptor.GetByLocale(entity, language);
                    _uow.GenreLocaleRepository.Add(locale);
                }
                else
                {
                    locale.UpdateLocales(descriptor);
                }
            }
            await _uow.CompleteAsync();
        }
        public async Task ChangeActivityAsync(int genreId)
        {
            var item = await _uow.GenreRepository.FindAsync(genreId);
            item.IsActive = !item.IsActive;
            await _uow.CompleteAsync();
        }
        public Task<bool> IsUrlExistAsync(string urlName, int genreId)
        {
            return _uow.GenreRepository.IsUrlExistAsync(urlName, genreId);
        }
        public async Task DeleteGenreAsync(int genreId)
        {
            var item = await _uow.GenreRepository.FindAsync(genreId);
            await _cloudinary.DeleteResourcesAsync(item.ImagePublicId);
            _uow.GenreRepository.Remove(item);
            await _uow.CompleteAsync();
        }
    }
}
```

Файл UserService.cs

```
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using BookGarden.Entities.Books;
using BookGarden.Service.Extensions;
using BookGarden.Service.Reviews.Views;
using BookGarden.Service.Users.Descriptors;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Localization;
using BookGarden.Shared.Options;
using BookGarden.Shared.Pagination;
using BookGarden.Storage.Repositories.BookModule.Books.Views;
```

```
using BookGarden.Storage.Repositories.UserModule.Filters;
using BookGarden.Storage.Repositories.UserModule.Views;
using BookGarden.Storage.Uow;
using CloudinaryDotNet;
using Microsoft.Extensions.Options;
namespace BookGarden.Service.Users
{
   public class UserService : IUserService
   {
      private readonly IUnitOfWork _uow;
      private readonly LocalizationService _localizationService;
      private readonly IMapper _mapper;
      private readonly Cloudinary _cloudinary;
      public UserService(
         IUnitOfWork uow,
         IMapper mapper,
         IOptions<CloudinaryStorageOptions> options,
         LocalizationService localizationService)
      {
         _uow = uow;
         _mapper = mapper;
         _localizationService = localizationService;
         var account = new Account(
            options.Value.Cloud,
            options.Value.ApiKey,
            options.Value.ApiSecret);
         _cloudinary = new Cloudinary(account);
      }
      public async Task UpdateAsAdminAsync(UserAdminDescriptor descriptor)
      {
         var entity = await _uow.UserRepository.GetAsync(descriptor.Id);
         if (entity == null)
         {
            throw _localizationService.GetNotFoundException(
               LocalizationConstants.UserNotFound,
               descriptor.RequestLanguage);
         }
         if (descriptor.PicturePublicId != entity.PicturePublicId)
         {
            await _cloudinary.DeleteResourcesAsync(entity.PicturePublicId);
         }
         _mapper.Map(descriptor, entity);
         await _uow.CompleteAsync();
      }
      public async Task<PaginatedList<BookViewWithAuthors>> GetSavedBooksAsync(string userHash,
BookStatus status, PaginationModel model)
      {
         var books = await _uow.BookRepository.GetSavedBooksAsync(userHash, status,
model.RequestLanguage);
         return books.GetPaginationList(model);
      }
      public async Task<BookStatus? > GetBookStatusAsync(string bookUrl, string userHash)
      {
         var entity = await _uow.BookUserRepository.GetAsync(bookUrl, userHash);
         return entity?.BookStatus;
      }
      public async Task SetBookStatusAsync(string bookUrl, string userHash, BookStatus status)
      {
         var bookId = await _uow.BookRepository.GetIdAsync(bookUrl);
         var entity = await _uow.BookUserRepository
            .FirstOrDefaultAsync(bu => bu.UserHash == userHash && bu.BookId == bookId);
         if (entity == null)
         {
```

```
                entity = new BookUser
                {
                    BookId = bookId,
                    UserHash = userHash,
                    BookStatus = status
                };
                _uow.BookUserRepository.Add(entity);
                await _uow.CompleteAsync();
            }
            else if (entity.BookStatus != status)
            {
                entity.BookStatus = status;
                await _uow.CompleteAsync();
            }
        }
        public async Task UpdateAvatarAsync(UserAvatarDescriptor descriptor)
        {
            var entity = await _uow.UserRepository.GetAsync(descriptor.UserId);
            if (entity == null)
            {
                throw _localizationService.GetNotFoundException(
                    LocalizationConstants.UserNotFound,
                    descriptor.RequestLanguage);
            }
            if (descriptor.PicturePublicId != entity.PicturePublicId)
            {
                await _cloudinary.DeleteResourcesAsync(entity.PicturePublicId);
            }
            _mapper.Map(descriptor, entity);
            await _uow.CompleteAsync();
        }
        public async Task<PaginatedList<UserAdminView>>
GetViewsAsAdminAsync(UserRequestAdminDescriptor descriptor)
        {
            var filter = _mapper.Map<UserRequestAdminFilter>(descriptor);
            var items = await _uow.UserRepository.GetViewsAsAdminAsync(filter);
            var count = await _uow.UserRepository.GetViewsCountAsAdminAsync(filter);
            return new PaginatedList<UserAdminView>
            {
                Items = items,
                TotalCount = count
            };
        }
        public Task<FullUserAdminView> GetFullUserAdminViewAsync(int userId)
        {
            return _uow.UserRepository.GetFullUserAdminViewAsync(userId);
        }
        public Task<FullUserView> GetFullUserViewAsync(string email)
        {
            return _uow.UserRepository.GetFullUserViewAsync(email);
        }
        public async Task<PaginatedList<ReviewWithRepliesView>> GetLastReviewsAsync(string
userHash, PaginationModel model)
        {
            var bookReviews = await _uow.BookReviewRepository.GetLastAsync(userHash);
            var authorReviews = await _uow.AuthorReviewRepository.GetLastAsync(userHash,
model.RequestLanguage);
            var compilationReviews = await _uow.CompilationReviewRepository.GetLastAsync(userHash,
model.RequestLanguage);
            var reviews = bookReviews.Union(authorReviews).Union(compilationReviews)
                .OrderByDescending(review => review.CreatedOnUtc);
            return reviews
                .ToReviewWithRepliesView(model.RequestLanguage, _localizationService, true)
```

```
                .GetPaginationList(model);
        }
        public async Task DeleteAsync(int userId, Language lang)
        {
            var user = await _uow.UserRepository.FindAsync(userId);
            if (user == null)
            {
                throw _localizationService.GetNotFoundException(
                    LocalizationConstants.UserNotFound,
                    lang);
            }
            await _cloudinary.DeleteResourcesAsync(user.PicturePublicId);
            _uow.UserRepository.Remove(user);
            await _uow.CompleteAsync();
        }
        public async Task DeleteBookStatusAsync(string bookUrl, string userHash)
        {
            var entity = await _uow.BookUserRepository.GetAsync(bookUrl, userHash);
            if (entity != null)
            {
                _uow.BookUserRepository.Remove(entity);
                await _uow.CompleteAsync();
            }
        }
    }
}
```

Файл BookExtension.cs

```
using BookGarden.Storage.Repositories.BookModule.Books.Views;
using System.Linq;
using BookGarden.Entities.Books;
using BookGarden.Shared.Enums;
namespace BookGarden.Storage.Extensions
{
    internal static class BookExtension
    {
        public static IQueryable<BookViewWithAuthors> SelectBookViewWithAuthors(this
IQueryable<Book> query, Language language)
        {
            return query.Select(item => new BookViewWithAuthors
            {
                Id = item.Id,
                UrlName = item.UrlName,
                Rating = item.Rating,
                Title = item.Title,
                ImageLink = item.ImageLink,
                Authors = item.Authors.Select(author =>
                    author.Author.Locales.FirstOrDefault(l => l.Language == language) != null
                        ? author.Author.Locales.FirstOrDefault(l => l.Language == language).FullName
                        : "").ToArray(),
                StateStatus = item.StateStatus,
            });
        }
    }
}
```

Файл LocaleExtension.cs

```
using System.Collections.Generic;
using System.Linq;
using BookGarden.Entities.Authors;
using BookGarden.Entities.Genres;
using BookGarden.Shared.Enums;
using BookGarden.Entities.Compilations;
namespace BookGarden.Storage.Extensions
```

```
{
    internal static class LocaleExtension
    {
        public static string GetAuthorFullName(this ICollection<AuthorLocale> locales, Language language)
        {
            return locales.FirstOrDefault(l => l.Language == language)?.FullName;
        }
        public static string GetAuthorDescription(this ICollection<AuthorLocale> locales, Language language)
        {
            return locales.FirstOrDefault(l => l.Language == language)?.Description;
        }
        public static string GetGenreTitle(this ICollection<GenreLocale> locales, Language language)
        {
            return locales.FirstOrDefault(l => l.Language == language)?.Title;
        }
        public static string GetGenreDescription(this ICollection<GenreLocale> locales, Language language)
        {
            return locales.FirstOrDefault(l => l.Language == language)?.Description;
        }
        public static string GetCompilationTitle(this ICollection<CompilationLocale> locales, Language language)
        {
            return locales.FirstOrDefault(l => l.Language == language)?.Title;
        }
        public static string GetCompilationDescription(this ICollection<CompilationLocale> locales, Language language)
        {
            return locales.FirstOrDefault(l => l.Language == language)?.Description;
        }
    }
}
```

Файл UnitOfWork.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Shared.Entities;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Repositories.AuthorModule.AuthorLocales;
using BookGarden.Storage.Repositories.AuthorModule.Authors;
using BookGarden.Storage.Repositories.BookModule.BookAuthors;
using BookGarden.Storage.Repositories.BookModule.BookCompilations;
using BookGarden.Storage.Repositories.BookModule.BookGenres;
using BookGarden.Storage.Repositories.BookModule.Books;
using BookGarden.Storage.Repositories.BookModule.BooksFiles;
using BookGarden.Storage.Repositories.BookModule.BooksUser;
using BookGarden.Storage.Repositories.CompilationModule.CompilationLocales;
using BookGarden.Storage.Repositories.CompilationModule.Compilations;
using BookGarden.Storage.Repositories.GenreModule.GenreLocales;
using BookGarden.Storage.Repositories.GenreModule.Genres;
using BookGarden.Storage.Repositories.ReviewModule.AuthorReviews;
using BookGarden.Storage.Repositories.ReviewModule.BookReviews;
using BookGarden.Storage.Repositories.ReviewModule.CompilationReviews;
using BookGarden.Storage.Repositories.RoleModule;
using BookGarden.Storage.Repositories.UserModule;
using Dapper;
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Storage;
namespace BookGarden.Storage.Uow
```

```
{
    public sealed class UnitOfWork : IUnitOfWork
    {
        private readonly DatabaseContext _context;
        private readonly IServiceProvider _serviceProvider;
        public UnitOfWork(
            DatabaseContext context,
            IServiceProvider serviceProvider)
        {
            _context = context;
            _serviceProvider = serviceProvider;
        }
        #region Repositories
        private IAuthorLocaleRepository _authorLocaleRepository;
        public IAuthorLocaleRepository AuthorLocaleRepository =>
            _authorLocaleRepository ??= Resolve<IAuthorLocaleRepository>();
        private IAuthorRepository _authorRepository;
        public IAuthorRepository AuthorRepository =>
            _authorRepository ??= Resolve<IAuthorRepository>();
        private IBookRepository _bookRepository;
        public IBookRepository BookRepository =>
            _bookRepository ??= Resolve<IBookRepository>();
        private IBookUserRepository _bookUserRepository;
        public IBookUserRepository BookUserRepository =>
            _bookUserRepository ??= Resolve<IBookUserRepository>();
        private IBookAuthorRepository _bookAuthorRepository;
        public IBookAuthorRepository BookAuthorRepository =>
            _bookAuthorRepository ??= Resolve<IBookAuthorRepository>();
        private IBookCompilationRepository _bookCompilationRepository;
        public IBookCompilationRepository BookCompilationRepository =>
            _bookCompilationRepository ??= Resolve<IBookCompilationRepository>();
        private IBookGenreRepository _bookGenreRepository;
        public IBookGenreRepository BookGenreRepository =>
            _bookGenreRepository ??= Resolve<IBookGenreRepository>();
        private IBookFileRepository _bookFileRepository;
        public IBookFileRepository BookFileRepository =>
            _bookFileRepository ??= Resolve<IBookFileRepository>();
        private ICompilationLocaleRepository _compilationLocaleRepository;
        public ICompilationLocaleRepository CompilationLocaleRepository =>
            _compilationLocaleRepository ??= Resolve<ICompilationLocaleRepository>();
        private ICompilationRepository _compilationRepository;
        public ICompilationRepository CompilationRepository =>
            _compilationRepository ??= Resolve<ICompilationRepository>();
        private IGenreLocaleRepository _genreLocaleRepository;
        public IGenreLocaleRepository GenreLocaleRepository =>
            _genreLocaleRepository ??= Resolve<IGenreLocaleRepository>();
        private IGenreRepository _genreRepository;
        public IGenreRepository GenreRepository =>
            _genreRepository ??= Resolve<IGenreRepository>();
        private IBookReviewRepository _bookReviewRepository;
        public IBookReviewRepository BookReviewRepository =>
            _bookReviewRepository ??= Resolve<IBookReviewRepository>();
        private ICompilationReviewRepository _compilationReviewRepository;
        public ICompilationReviewRepository CompilationReviewRepository =>
            _compilationReviewRepository ??= Resolve<ICompilationReviewRepository>();
        private IAuthorReviewRepository _authorReviewRepository;
        public IAuthorReviewRepository AuthorReviewRepository =>
            _authorReviewRepository ??= Resolve<IAuthorReviewRepository>();
        private IRoleRepository _roleRepository;
        public IRoleRepository RoleRepository =>
            _roleRepository ??= Resolve<IRoleRepository>();
        private IUserRepository _userRepository;
        public IUserRepository UserRepository =>
```

```
            _userRepository ??= Resolve<IUserRepository>();
        #endregion
        #region Methods
        private TRepository Resolve<TRepository>() where TRepository : class
        {
            return _serviceProvider.GetService(typeof(TRepository)) as TRepository;
        }
        public DbSet<TEntity> GenericSource<TEntity>() where TEntity : BaseEntity<int>
        {
            return _context.Set<TEntity>();
        }
        public Task<int> CompleteAsync()
        {
            return _context.SaveChangesAsync();
        }
        public T QueryFirst<T>(string sql, object param = null)
        {
            return _context.Connection.QueryFirst<T>(sql, param);
        }
        public T QueryFirstOrDefault<T>(string sql, object param = null)
        {
            return _context.Connection.QueryFirstOrDefault<T>(sql, param);
        }
        public T StoredProcedureFirst<T>(string name, object param = null)
        {
            return _context.Connection.QueryFirst<T>(name, param, commandType:
CommandType.StoredProcedure);
        }
        public async Task<List<T>> QueryAsync<T>(string sql, object param = null)
        {
            var transaction = _context.Database.CurrentTransaction?.GetDbTransaction() as SqlTransaction;
            return (await _context.Connection.QueryAsync<T>(sql, param, transaction)).ToList();
        }
        public async Task<T> QueryFirstAsync<T>(string sql, object param = null)
        {
            var transaction = _context.Database.CurrentTransaction?.GetDbTransaction() as SqlTransaction;
            return await _context.Connection.QueryFirstAsync<T>(sql, param, transaction);
        }
        public async Task<T> QueryFirstOrDefaultAsync<T>(string sql, object param = null)
        {
            var transaction = _context.Database.CurrentTransaction?.GetDbTransaction() as SqlTransaction;
            return await _context.Connection.QueryFirstOrDefaultAsync<T>(sql, param, transaction);
        }
        public async Task<T> StoredProcedureFirstAsync<T>(string name, object param = null)
        {
            var transaction = _context.Database.CurrentTransaction?.GetDbTransaction() as SqlTransaction;
            return await _context.Connection.QueryFirstAsync<T>(name, param, transaction, commandType:
CommandType.StoredProcedure);
        }
        public async Task<List<T>> StoredProcedureAsync<T>(string name, object param = null)
        {
            var transaction = _context.Database.CurrentTransaction?.GetDbTransaction() as SqlTransaction;
            return (await _context.Connection.QueryAsync<T>(name, param, transaction, commandType:
CommandType.StoredProcedure)).ToList();
        }
        public Task<IDbContextTransaction> BeginTransactionAsync(IsolationLevel isolationLevel =
IsolationLevel.ReadCommitted)
        {
            return _context.Database.BeginTransactionAsync(isolationLevel);
        }
        public void Dispose()
        {
            _context?.Dispose();
```

```
        }
        #endregion
    }
}
```

Файл RoleRepository.cs

```
using System.Threading.Tasks;
using BookGarden.Entities.Users;
using BookGarden.Shared;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.RoleModule
{
    public class RoleRepository : BaseRepository<Role>, IRoleRepository
    {
        public RoleRepository(DatabaseContext context) : base(context)
        { }
        public async Task<Role> GetUserRoleAsync()
        {
            return await Source.FirstOrDefaultAsync(r => r.Name == BookGardenConstants.Roles.User);
        }
        public async Task<Role> GetAdminRoleAsync()
        {
            return await Source.FirstOrDefaultAsync(r => r.Name == BookGardenConstants.Roles.Admin);
        }
    }
}
```

Файл UserRepository.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Entities.Users;
using BookGarden.Shared;
using BookGarden.Shared.Enums;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using BookGarden.Storage.Repositories.UserModule.Filters;
using BookGarden.Storage.Repositories.UserModule.Views;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.UserModule
{
    public class UserRepository : BaseRepository<User>, IUserRepository
    {
        public UserRepository(DatabaseContext context) : base(context)
        { }
        public async Task<User> FindByEmailAsync(string email)
        {
            return await Source
                .Include(d => d.Role)
                .FirstOrDefaultAsync(user => user.Email == email);
        }
        public Task<string> GetUserHashAsync(string email)
        {
            return Source
                .Where(u => u.Email == email)
                .Select(u => u.UserHash)
                .OrderByDescending(u => u)
                .FirstOrDefaultAsync();
        }
        public async Task<bool> UserWithThisEmailExists(string email)
        {
            email = email.ToLower();
```

```
        return await Source.FirstOrDefaultAsync(user => user.Email.ToLower() == email) != null;
    }
    public async Task<int> GetUserNextIdAsync()
    {
        return await Source
            .Select(u => u.Id)
            .OrderByDescending(u => u)
            .FirstOrDefaultAsync();
    }
    public Task<User> GetAsync(int userId)
    {
        return Source
            .FirstOrDefaultAsync(user => user.Id == userId);
    }
    public Task<User> GetAsync(string userHash)
    {
        return Source
            .FirstOrDefaultAsync(user => user.UserHash == userHash);
    }
    public async Task<bool> CheckUsernameAsync(string username)
    {
        username = username.ToLower();
        return !await Source.AllAsync(user => user.UserName.ToLower() != username);
    }
    public Task<List<UserAdminView>> GetViewsAsAdminAsync(UserRequestAdminFilter filter)
    {
        return GetViewsAsAdmin(filter)
            .Skip(--filter.Page * filter.PageSize)
            .Take(filter.PageSize)
            .ToListAsync();
    }
    public Task<int> GetViewsCountAsAdminAsync(UserRequestAdminFilter filter)
    {
        return GetViewsAsAdmin(filter)
            .CountAsync();
    }
    public Task<FullUserAdminView> GetFullUserAdminViewAsync(int userId)
    {
        return Source
            .Where(d => d.Id == userId)
            .Select(item => new FullUserAdminView
            {
                Id = item.Id,
                UserName = item.UserName,
                PictureLink = item.PictureLink,
                PicturePublicId = item.PicturePublicId,
                IsAdmin = item.RoleId == BookGardenConstants.Roles.AdminId,
                Email = item.Email
            }).FirstOrDefaultAsync();
    }
    public Task<FullUserView> GetFullUserViewAsync(string email)
    {
        return Source
            .Where(d => d.Email.ToLower() == email.ToLower())
            .Include(d => d.AuthorReviews)
            .Include(d => d.BookReviews)
            .Include(d => d.CompilationReviews)
            .Include(d => d.SavedBooks)
            .Select(item => new FullUserView
            {
                UserName = item.UserName,
                PictureLink = item.PictureLink,
                PicturePublicId = item.PicturePublicId,
```

```
                    Email = item.Email,
                    FinishedCount = item.SavedBooks.Count(sb => sb.BookStatus == BookStatus.Finished),
                    ReadingCount = item.SavedBooks.Count(sb => sb.BookStatus == BookStatus.Reading),
                    WantToReadCount = item.SavedBooks.Count(sb => sb.BookStatus ==
BookStatus.WantToRead),
                    AuthorReviewCount = item.AuthorReviews.Count,
                    BookReviewCount = item.BookReviews.Count,
                    CompilationReviewCount = item.CompilationReviews.Count,
                }).FirstOrDefaultAsync();
            }
            public Task<UserView> GetUserViewAsync(string username)
            {
                return Source
                    .Where(d => d.UserName.ToLower() == username.ToLower())
                    .Select(item => new UserView
                    {
                        UserName = item.UserName,
                        PictureLink = item.PictureLink,
                        PicturePublicId = item.PicturePublicId,
                        Email = item.Email
                    }).FirstOrDefaultAsync();
            }
            private IQueryable<UserAdminView> GetViewsAsAdmin(UserRequestAdminFilter filter)
            {
                return Source
                    .Where(item => item.Id != filter.RequestAdminId)
                    .Select(item => new UserAdminView
                    {
                        Id = item.Id,
                        UserName = item.UserName,
                        PictureLink = item.PictureLink,
                        Role = item.RoleId,
                        Email = item.Email
                    });
            }
        }
    }
```

Файл AuthorLocaleRepository.cs

```
using BookGarden.Entities.Authors;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
namespace BookGarden.Storage.Repositories.AuthorModule.AuthorLocales
{
    public class AuthorLocaleRepository : BaseRepository<AuthorLocale>, IAuthorLocaleRepository
    {
        public AuthorLocaleRepository(DatabaseContext context) : base(context)
        { }
    }
}
```

Файл AuthorRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Entities.Authors;
using BookGarden.Shared;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Extensions;
using BookGarden.Storage.Extensions;
using BookGarden.Storage.Interceptors;
using BookGarden.Storage.Persistence.Context;
```

```
using BookGarden.Storage.Persistence.Repository;
using BookGarden.Storage.Repositories.AuthorModule.Authors.Filters;
using BookGarden.Storage.Repositories.AuthorModule.Authors.Views;
using BookGarden.Storage.Repositories.BookModule.Books.Views;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.AuthorModule.Authors
{
   public class AuthorRepository : BaseRepository<Author>, IAuthorRepository
   {
      public AuthorRepository(DatabaseContext context) : base(context)
      { }
      public Task<Author> GetAsync(int itemId)
      {
         return Source
            .Include(d => d.Locales)
            .FirstOrDefaultAsync(item => item.Id == itemId);
      }
      public Task<Author> GetAsync(string authorUrl)
      {
         return Source
            .Include(d => d.Locales)
            .FirstOrDefaultAsync(item => item.UrlName == authorUrl);
      }
      public Task<FullAuthorView> GetAsync(string authorUrl, Language lang)
      {
         return Source
            .Include(d => d.Locales)
            .Include(d => d.Books).ThenInclude(d => d.Book).ThenInclude(d => d.Genres).ThenInclude(d
=> d.Genre).ThenInclude(d => d.Locales)
            .Where(item => item.UrlName.ToLower() == authorUrl.ToLower())
            .Select(item => new FullAuthorView
            {
               ImageLink = item.ImageLink,
               FullName = item.Locales.GetAuthorFullName(lang),
               Description = item.Locales.GetAuthorDescription(lang),
               UrlName = item.UrlName,
               RatesCount = item.RatesCount,
               Rating = item.Rating,
               BooksCount = item.Books.Count,
               YearsOfLife = item.YearsOfLife,
               StateStatus = item.StateStatus,
               LinkedInLink = item.LinkedInLink,
               TwitterLink = item.TwitterLink,
               FacebookLink = item.FacebookLink,
               GoodReadsLink = item.GoodReadsLink,
               AmazonLink = item.AmazonLink,
               YakabooLink = item.YakabooLink,
               Genres = item.Books.SelectMany(b => b.Book.Genres).Select(genre => new BookGenreView
               {
                  GenreUrlName = genre.Genre.UrlName,
                  Title = genre.Genre.Locales.GetGenreTitle(lang)
               }).ToArray(),
            }).FirstOrDefaultAsync();
      }
      public Task<int> GetIdAsync(string authorUrl)
      {
         return Source
            .Where(item => item.UrlName == authorUrl)
            .Select(item => item.Id)
            .FirstOrDefaultAsync();
      }
      public Task<List<Author>> GetNewToUpdateAsync(DateTime validDate)
      {
```

```
        return Source
            .Where(item => item.StateStatus == StateStatus.New)
            .Where(book => book.CreatedOnUtc <= validDate)
            .ToListAsync();
    }
    public Task<List<ListItem<int>>> GetOptionsAsAdminAsync(Language lang)
    {
        return Source
            .Include(d => d.Locales)
            .Select(item => new ListItem<int>
            {
                Value = item.Id,
                Text = item.Locales.FirstOrDefault(l => l.Language == lang) != null
                    ? item.Locales.FirstOrDefault(l => l.Language == lang).FullName
                    : "",
            })
            .OrderBy(item => item.Text)
            .ToListAsync();
    }
    public Task<List<ListItem<string>>> GetOptionsAsync(Language lang)
    {
        return Source
            .Include(d => d.Locales)
            .Select(item => new ListItem<string>
            {
                Value = item.UrlName,
                Text = item.Locales.FirstOrDefault(l => l.Language == lang) != null
                    ? item.Locales.FirstOrDefault(l => l.Language == lang).FullName
                    : "",
            })
            .OrderBy(item => item.Text)
            .ToListAsync();
    }
    public Task<List<AuthorLightView>> GetTopViewsAsync(Language language)
    {
        return Source
            .Where(item => item.IsActive)
            .Select(item => new AuthorLightView
            {
                ImageLink = item.ImageLink,
                Name = item.Locales.GetAuthorFullName(language),
                UrlName = item.UrlName,
                StateStatus = item.StateStatus,
                BooksCount = item.Books.Count
            })
            .Take(10)
            .ToListAsync();
    }
    public Task<List<AuthorView>> GetViewsAsync(AuthorRequestFilter filter)
    {
        var query = GetQuery(filter);
        query = filter.SortType switch
        {
            SortType.Popularity => query.OrderByDescending(item => item.Books.Count),
            SortType.Date => query.OrderByDescending(item => item.CreatedOnUtc),
            // todo SortType.Rating => query.OrderByDescending(item => item.Rating)
            _ => query,
        };
        var selectedQuery = query
            .Skip(--filter.Page * filter.PageSize)
            .Take(filter.PageSize)
            .Include(item => item.Locales)
            .Select(item => new AuthorView
```

```
                {
                    ImageLink = item.ImageLink,
                    Name = item.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage) != null
                        ? item.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage).FullName
                        : "",
                    UrlName = item.UrlName,
                    StateStatus = item.StateStatus,
                    BooksCount = item.Books.Count,
                    Books = item.Books.Select(b => b.Book)
                        .OrderByDescending(item => item.PagesCount)
                        .Take(BookGardenConstants.DescriptionBookCount)
                        .Select(ba => new BookViewWithAuthors
                        {
                            Title = ba.Title,
                            StateStatus = ba.StateStatus,
                            UrlName = ba.UrlName,
                            Rating = ba.Rating,
                            ImageLink = ba.ImageLink,
                            Authors = ba.Authors.Select(author => author.Author.Locales.GetAuthorFullName(filter.RequestLanguage)).ToArray(),
                        })
                        .ToList(),
                });
            selectedQuery = filter.SortType switch
            {
                SortType.Title => selectedQuery.OrderBy(item => item.Name),
                SortType.BookCount => selectedQuery.OrderByDescending(item => item.BooksCount),
                _ => selectedQuery,
            };
            return selectedQuery.ToListAsync();
        }
        public Task<int> GetViewsCountAsync(AuthorRequestFilter filter)
        {
            return GetQuery(filter)
                .CountAsync();
        }
        public Task<List<AuthorAdminView>> GetViewsAsAdminAsync(AuthorRequestAdminFilter filter)
        {
            return GetAdminViewsQuery(filter)
                .Skip(--filter.Page * filter.PageSize)
                .Take(filter.PageSize)
                .ToListAsync();
        }
        public Task<int> GetViewsCountAsAdminAsync(AuthorRequestAdminFilter filter)
        {
            return GetAdminViewsQuery(filter)
                .CountAsync();
        }
        public async Task<bool> IsUrlExistAsync(string urlName, int authorId)
        {
            return await Source.AnyAsync(item => item.UrlName == urlName && item.Id != authorId);
        }
        private IQueryable<AuthorAdminView> GetAdminViewsQuery(AuthorRequestAdminFilter filter)
        {
            return Source
                .Select(item => new AuthorAdminView
                {
                    Id = item.Id,
                    FullName = item.Locales.GetAuthorFullName(filter.RequestLanguage),
                    Description = item.Locales.GetAuthorDescription(filter.RequestLanguage),
                    UrlName = item.UrlName,
                    YearsOfLife = item.YearsOfLife,
                    ImageLink = item.ImageLink,
```

```
                BooksCount = item.Books.Count,
                IsActive = item.IsActive
            });
        }
        private IQueryable<Author> GetQuery(AuthorRequestFilter filter)
        {
            var query = Source.Where(book => book.IsActive);
            if (filter.Search.IsNotNullOrWhiteSpace())
            {
                filter.Search = FtsInterceptor.Fts(filter.Search, FtsConditionType.And, true);
                query = query.Where(item => item.SearchKeywords.Contains(filter.Search));
            }
            if (filter.GenreUrls.Any())
            {
                query = query
                    .Where(item => item.Books.SelectMany(b => b.Book.Genres)
                        .Select(a => a.Genre.UrlName)
                        .Any(a => filter.GenreUrls.Contains(a)));
            }
            if (filter.Languages.Any())
            {
                query = query
                    .Where(item => item.Books.Select(b => b.Book.Language)
                        .Any(lang => filter.Languages.Contains(lang)));
            }
            if (filter.Rating > 1)
            {
                query = query.Where(item => item.Rating >= filter.Rating);
            }
            return query;
        }
    }
}
```

Файл BookAuthorRepository.cs

```
using System.Threading.Tasks;
using BookGarden.Entities.Books;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.BookModule.BookAuthors
{
    public class BookAuthorRepository : BaseRepository<BookAuthor>, IBookAuthorRepository
    {
        public BookAuthorRepository(DatabaseContext context) : base(context)
        { }
        public Task<bool> CheckAsync(int authorId, int bookId)
        {
            return Source.AnyAsync(item => item.AuthorId == authorId && item.BookId == bookId);
        }
    }
}
```

Файл BookCompilationRepository.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Entities.Books;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Pagination;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using BookGarden.Storage.Repositories.BookModule.Books.Views;
using Microsoft.EntityFrameworkCore;
```

```
namespace BookGarden.Storage.Repositories.BookModule.BookCompilations
{
    public class BookCompilationRepository : BaseRepository<BookCompilation>, IBookCompilationRepository
    {
        public BookCompilationRepository(DatabaseContext context) : base(context)
        { }
        public Task<bool> CheckAsync(int compilationId, int bookId)
        {
            return Source.AnyAsync(item => item.CompilationId == compilationId && item.BookId == bookId);
        }
        public Task<List<BookGridView>> GetViewsByCompilationAsync(int compilationId, SortPaginationModel model)
        {
            var query = Source
                .Where(item => item.CompilationId == compilationId)
                .Include(item => item.Book)
                    .ThenInclude(d => d.Authors)
                    .ThenInclude(d => d.Author)
                    .ThenInclude(d => d.Locales)
                .Select(item => item.Book);
            query = model.SortType switch
            {
                SortType.Popularity => query.OrderByDescending(item => item.PagesCount), // todo https://www.notion.so/4c6c6a3a69bf40cd911996bcaa59a458?v=ae1b7cb62af548179087da8c88ace7f7&p=499c2f51bcd74ba09eec8e3831152d6e&pm=s
                SortType.Date => query.OrderByDescending(item => item.CreatedOnUtc),
                SortType.Title => query.OrderBy(item => item.Title),
                SortType.Rating => query.OrderByDescending(item => item.Rating),
                _ => query,
            };
            return query
                .Skip(--model.Page * model.PageSize)
                .Take(model.PageSize)
                .Select(item => new BookGridView
                {
                    Title = item.Title,
                    Subtitle = item.Subtitle,
                    Description = item.Description,
                    StateStatus = item.StateStatus,
                    UrlName = item.UrlName,
                    Rating = item.Rating,
                    ImageLink = item.ImageLink,
                    Authors = item.Authors.Select(author =>
                        author.Author.Locales.FirstOrDefault(l => l.Language == model.RequestLanguage) != null
                            ? author.Author.Locales.FirstOrDefault(l => l.Language == model.RequestLanguage).FullName
                            : "").ToArray(),
                }).ToListAsync();
        }
        public Task<int> GetViewsCountByCompilationAsync(int compilationId, SortPaginationModel model)
        {
            return Source
                .Where(item => item.CompilationId == compilationId)
                .Include(item => item.Book)
                .Select(item => item.Book)
                .CountAsync();
        }
    }
}
```

Файл BookGenreRepository.cs

```
using BookGarden.Entities.Books;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
namespace BookGarden.Storage.Repositories.BookModule.BookGenres
{
   public class BookGenreRepository : BaseRepository<BookGenre>, IBookGenreRepository
   {
      public BookGenreRepository(DatabaseContext context) : base(context)
      { }
   }
}
```

Файл BookRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using BookGarden.Entities.Books;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Extensions;
using BookGarden.Storage.Extensions;
using BookGarden.Storage.Interceptors;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using BookGarden.Storage.Repositories.BookModule.Books.Filters;
using BookGarden.Storage.Repositories.BookModule.Books.Views;
using BookGarden.Storage.Repositories.CompilationModule.Compilations.Views;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.BookModule.Books
{
   public class BookRepository : BaseRepository<Book>, IBookRepository
   {
      public BookRepository(DatabaseContext context) : base(context)
      {
      }
      public Task<Book> GetAsync(int itemId)
      {
         return Source
            .Include(d => d.Authors)
            .Include(d => d.OriginalBook)
            .Include(d => d.Genres)
            .Include(d => d.TranslatedBooks)
            .Include(d => d.SavedBooks)
            .Include(d => d.Files)
            .Include(d => d.Compilations)
            .FirstOrDefaultAsync(item => item.Id == itemId);
      }
      public Task<Book> GetAsync(string bookUrl)
      {
         return Source
            .Include(d => d.Authors)
            .Include(d => d.OriginalBook)
            .Include(d => d.Genres)
            .Include(d => d.TranslatedBooks)
            .Include(d => d.Files)
            .Include(d => d.Compilations)
            .FirstOrDefaultAsync(item => item.UrlName == bookUrl);
      }
      public Task<FullBookView> GetAsync(string bookUrl, Language lang)
      {
```

```
return Source
    .Include(d => d.Authors)
    .Include(d => d.OriginalBook)
    .Include(d => d.Genres)
    .Include(d => d.TranslatedBooks)
    .Include(d => d.Files)
    .Include(d => d.Compilations)
    .Where(item => item.UrlName.ToLower() == bookUrl.ToLower())
    .Select(item => new FullBookView
    {
        Title = item.Title,
        Subtitle = item.Subtitle,
        Description = item.Description,
        UrlName = item.UrlName,
        Rating = item.Rating,
        ImageLink = item.ImageLink,
        PublishedYear = item.PublishedYear,
        PagesCount = item.PagesCount,
        Language = item.Language,
        StateStatus = item.StateStatus,
        ViewsCount = item.ViewsCount,
        DownloadsCount = item.DownloadsCount,
        RatesCount = item.RatesCount,
        OriginalBook = item.OriginalBookId != null
            ? new BookViewWithAuthors
            {
                Title = item.OriginalBook.Title,
                StateStatus = item.OriginalBook.StateStatus,
                UrlName = item.OriginalBook.UrlName,
                Rating = item.OriginalBook.Rating,
                ImageLink = item.OriginalBook.ImageLink,
                Authors = item.OriginalBook.Authors.Select(author =>
                    author.Author.Locales.FirstOrDefault(l => l.Language == lang) != null
                        ? author.Author.Locales.FirstOrDefault(l => l.Language == lang).FullName
                        : "").ToArray(),
            }
            : null,
        Authors = item.Authors.Select(author => new BookAuthorView
        {
            AuthorUrlName = author.Author.UrlName,
            FullName = author.Author.Locales.FirstOrDefault(l => l.Language == lang) != null
                ? author.Author.Locales.FirstOrDefault(l => l.Language == lang).FullName
                : ""
        }).ToArray(),
        Compilations = item.Compilations.Select(compilation => new CompilationLightView
        {
            ImageLink = compilation.Compilation.ImageLink,
            Title = compilation.Compilation.Locales.FirstOrDefault(l => l.Language == lang) != null
                ? compilation.Compilation.Locales.FirstOrDefault(l => l.Language == lang).Title
                : "",
            BooksCount = compilation.Compilation.Books.Count,
            Language = compilation.Compilation.Language,
            StateStatus = compilation.Compilation.StateStatus,
            UrlName = compilation.Compilation.UrlName
        }).ToArray(),
        Genres = item.Genres.Select(genre => new BookGenreView
        {
            GenreUrlName = genre.Genre.UrlName,
            Title = genre.Genre.Locales.FirstOrDefault(l => l.Language == lang) != null
                ? genre.Genre.Locales.FirstOrDefault(l => l.Language == lang).Title
                : ""
        }).ToArray(),
        Files = item.Files.Select(file => file.FileType).ToArray(),
```

```
            }).FirstOrDefaultAsync();
        }
        public Task<BookReadView> GetReadViewAsync(string bookUrl)
        {
            return Source
                .Where(item => item.UrlName == bookUrl)
                .Include(d => d.Files)
                .Select(item => new BookReadView
                {
                    Title = item.Title,
                    PageCount = item.PagesCount,
                    BlobUri = item.Files.First(f => f.FileType == FileType.Epub).BlobUrl
                })
                .FirstOrDefaultAsync();
        }
        public Task<int> GetIdAsync(string bookUrl)
        {
            return Source
                .Where(item => item.UrlName == bookUrl)
                .Select(item => item.Id)
                .FirstOrDefaultAsync();
        }
        public Task<List<BookViewWithAuthors>> GetMoreByGenre(int genreId, string bookUrl, Language
lang)
        {
            return Source
                .Where(book => book.IsActive)
                .Where(book => book.UrlName != bookUrl)
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.Genres.Any(genre => genre.GenreId == genreId))
                .OrderByDescending(item => item.CreatedOnUtc)
                .SelectBookViewWithAuthors(lang)
                .Take(10)
                .ToListAsync();
        }
        public Task<List<BookViewWithAuthors>> GetMoreByGenre(int[] genreIds, string bookUrl,
Language lang)
        {
            return Source
                .Where(book => book.IsActive)
                .Where(book => book.UrlName != bookUrl)
                .Include(d => d.Genres)
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.Genres.Any(g => genreIds.Contains(g.GenreId)))
                .OrderByDescending(item => item.CreatedOnUtc)
                .SelectBookViewWithAuthors(lang)
                .Take(10)
                .ToListAsync();
        }
        public Task<List<BookViewWithAuthors>> GetMoreByAuthor(int authorId, string bookUrl,
Language lang)
        {
            return Source
                .Where(book => book.IsActive)
                .Where(book => book.UrlName != bookUrl)
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.Authors.Any(a=> a.AuthorId == authorId))
                .OrderByDescending(item => item.CreatedOnUtc)
                .SelectBookViewWithAuthors(lang)
                .Take(10)
                .ToListAsync();
        }
```

```
        public Task<List<BookViewWithAuthors>> GetSavedBooksAsync(string userHash, BookStatus status, Language lang)
        {
            return Source
                .Where(book => book.IsActive)
                .Include(d=>d.SavedBooks)
                .Where(item => item.SavedBooks.Any(sb => sb.UserHash == userHash && sb.BookStatus == status))
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .OrderByDescending(item => item.SavedBooks.First(sb => sb.UserHash == userHash && sb.BookStatus == status).UpdatedOnUtc)
                .SelectBookViewWithAuthors(lang)
                .ToListAsync();
        }
        public Task<List<BookView>> GetViewsAsAdminAsync(BookRequestAdminFilter filter)
        {
            return GetViewsAsAdminQuery(filter)
                .Skip(--filter.Page * filter.PageSize)
                .Take(filter.PageSize)
                .ToListAsync();
        }
        public Task<int> GetViewsCountAsAdminAsync(BookRequestAdminFilter filter)
        {
            return GetViewsAsAdminQuery(filter)
                .CountAsync();
        }
        public Task<List<BookGridView>> GetViewsAsync(BookRequestFilter filter)
        {
            var query = GetQuery(filter);
            query = filter.SortType switch
            {
                SortType.Popularity => query.OrderByDescending(item => item.PagesCount), // todo https://www.notion.so/4c6c6a3a69bf40cd911996bcaa59a458?v=ae1b7cb62af548179087da8c88ace7f7&p=499c2f51bcd74ba09eec8e3831152d6e&pm=s
                SortType.Date => query.OrderByDescending(item => item.CreatedOnUtc),
                SortType.Title => query.OrderBy(item => item.Title),
                SortType.Rating => query.OrderByDescending(item => item.Rating),
                _ => query,
            };
            return query
                .Skip(--filter.Page * filter.PageSize)
                .Take(filter.PageSize)
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Select(item => new BookGridView
                {
                    Title = item.Title,
                    Subtitle = item.Subtitle,
                    Description = item.Description,
                    StateStatus = item.StateStatus,
                    UrlName = item.UrlName,
                    Rating = item.Rating,
                    ImageLink = item.ImageLink,
                    Authors = item.Authors.Select(author =>
                        author.Author.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage) != null
                            ? author.Author.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage).FullName
                            : "").ToArray(),
                }).ToListAsync();
        }
        public Task<int> GetViewsCountAsync(BookRequestFilter filter)
        {
            return GetQuery(filter)
                .CountAsync();
```

```
        }
        public Task<List<Book>> GetNewToUpdateAsync(DateTime validDate)
        {
            return Source
                .Include(d => d.Files)
                .Where(item => item.StateStatus == StateStatus.New)
                .Where(book => book.CreatedOnUtc <= validDate
                        && book.Files.All(file => file.CreatedOnUtc <= validDate))
                .ToListAsync();
        }
        public Task<int[]> GetGenresAsync(string bookUrl)
        {
            return Source
                .Include(item => item.Genres)
                .Where(item => item.UrlName == bookUrl)
                .SelectMany(item => item.Genres)
                .Select(genre => genre.GenreId)
                .ToArrayAsync();
        }
        public async Task<List<BookViewWithAuthors>> GetNewViewsAsync(Language language)
        {
            var tops = await Source
                .Where(book => book.IsActive)
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.StateStatus == StateStatus.New)
                .OrderByDescending(item => item.CreatedOnUtc)
                .SelectBookViewWithAuthors(language)
                .Take(10)
                .ToListAsync();
            if (tops.Count == 10)
            {
                return tops;
            }
            var additional = GetViewsQuery(
                item => item.StateStatus != StateStatus.New,
                item => item.CreatedOnUtc,
                language,
                10 - tops.Count);
            return tops.Union(additional).ToList();
        }
        public async Task<List<BookViewWithAuthors>> GetTopViewsAsync(Language language)
        {
            var tops = await Source
                .Where(book => book.IsActive)
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.StateStatus == StateStatus.Top)
                .OrderByDescending(item => item.ViewsCount)
                .SelectBookViewWithAuthors(language)
                .Take(10)
                .ToListAsync();
            if (tops.Count == 10)
            {
                return tops;
            }
            var additional = GetViewsQuery(
                item => item.StateStatus != StateStatus.Top,
                item => item.ViewsCount,
                language,
                10 - tops.Count);
            return tops.Union(additional).ToList();
        }
        public    Task<List<BookViewWithAuthors>>    SearchBookAsAdminAsync(BookSearchAdminFilter
filter)
```

```
        {
            var query = Source
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => !filter.IsWithoutAuthor || item.Authors.Count == 0);
            if (filter.Search.IsNotNullOrWhiteSpace())
            {
                filter.Search = FtsInterceptor.Fts(filter.Search, FtsConditionType.And, true);
                query = query.Where(item => item.SearchKeywords.Contains(filter.Search));
            }
            return query
                .SelectBookViewWithAuthors(filter.RequestLanguage)
                .Take(20)
                .ToListAsync();
        }
        public Task<List<ListItem<int>>> SearchOptionAsAdminAsync(string search, Language lang)
        {
            IQueryable<Book> query = Source;
            if (search.IsNotNullOrWhiteSpace())
            {
                search = FtsInterceptor.Fts(search, FtsConditionType.And, true);
                query = query.Where(item => item.SearchKeywords.Contains(search));
            }
            return query
                .Select(item => new ListItem<int>
                {
                    Value = item.Id,
                    Text = item.Title,
                })
                .Take(20)
                .ToListAsync();
        }
        public Task<List<BookViewWithAuthors>> GetViewsByAuthorAsAdminAsync(int authorId,
Language lang)
        {
            return Source
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.Authors.Select(a => a.AuthorId).Contains(authorId))
                .SelectBookViewWithAuthors(lang)
                .ToListAsync();
        }
        public Task<List<BookViewWithAuthors>> GetViewsByCompilationAsAdminAsync(int
compilationId, Language lang)
        {
            return Source
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Include(d => d.Compilations)
                .Where(item => item.Compilations.Select(a => a.CompilationId).Contains(compilationId))
                .SelectBookViewWithAuthors(lang)
                .ToListAsync();
        }
        public Task<List<BookViewWithAuthors>> GetTranslatedViewsAsAdminAsync(int bookId,
Language lang)
        {
            return Source
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Where(item => item.OriginalBookId == bookId)
                .SelectBookViewWithAuthors(lang)
                .ToListAsync();
        }
        public async Task<bool> CheckUrlAsync(string urlName, int bookId)
        {
            return await Source.AnyAsync(item => item.UrlName == urlName && item.Id != bookId);
        }
```

```
        private IQueryable<BookView> GetViewsAsAdminQuery(BookRequestAdminFilter filter)
        {
            return Source
                .Include(d => d.Authors).ThenInclude(d => d.Author).ThenInclude(d => d.Locales)
                .Select(item => new BookView
                {
                    Id = item.Id,
                    Title = item.Title,
                    Subtitle = item.Subtitle,
                    Description = item.Description,
                    UrlName = item.UrlName,
                    Authors = item.Authors.Select(author =>
                        author.Author.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage) != null
                            ? author.Author.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage).FullName
                            : "").ToArray(),
                    ImageLink = item.ImageLink,
                    Rating = item.Rating,
                    IsActive = item.IsActive
                });
        }
        private IQueryable<Book> GetQuery(BookRequestFilter filter)
        {
            IQueryable<Book> query = Source
                .Where(book => book.IsActive)
                .Include(item => item.Authors).ThenInclude(a=> a.Author)
                .Include(item => item.Genres).ThenInclude(a=> a.Genre);
            if (filter.Search.IsNotNullOrWhiteSpace())
            {
                filter.Search = FtsInterceptor.Fts(filter.Search, FtsConditionType.And, true);
                query = query.Where(item => item.SearchKeywords.Contains(filter.Search));
            }
            if (filter.AuthorUrls.Any())
            {
                query = query.Where(item => item.Authors.Select(a => a.Author.UrlName).Any(a => filter.AuthorUrls.Contains(a)));
            }
            if (filter.GenreUrls.Any())
            {
                query = query.Where(item => item.Genres.Select(a => a.Genre.UrlName).Any(a => filter.GenreUrls.Contains(a)));
            }
            if (filter.Languages.Any())
            {
                query = query.Where(item => filter.Languages.Contains(item.Language));
            }
            if (filter.Rating > 1)
            {
                query = query.Where(item => item.Rating >= filter.Rating);
            }
            return query;
        }
        private IQueryable<BookViewWithAuthors> GetViewsQuery<TKey>(
            Expression<Func<Book, bool>> predicate,
            Expression<Func<Book, TKey>> keySelector,
            Language language,
            int count)
        {
            return Source
                .Where(book => book.IsActive)
                .Where(predicate)
                .OrderByDescending(keySelector)
                .SelectBookViewWithAuthors(language)
```

```
            .Take(count);
        }
    }
}
```

Файл BookFileRepository.cs

```
using BookGarden.Entities.Books;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Shared.Enums;
using Microsoft.EntityFrameworkCore;
using System;
namespace BookGarden.Storage.Repositories.BookModule.BooksFiles
{
    public class BookFileRepository : BaseRepository<BookFile>, IBookFileRepository
    {
        public BookFileRepository(DatabaseContext context) : base(context)
        { }
        public Task<List<FileType>> GetFilesAsync(int bookId)
        {
            return Source
                .Where(item => item.BookId == bookId)
                .Select(file => file.FileType)
                .ToListAsync();
        }
        public Task<bool> IsFileFormatFreeAsync(int bookId, FileType format)
        {
            return Source
                .Where(item => item.BookId == bookId)
                .AllAsync(item => item.FileType != format);
        }
        public Task<Tuple<string, string>> GetFileLinkAsync(string bookUrl, FileType format)
        {
            return Source
                .Include(d => d.Book)
                .Where(item => item.FileType == format)
                .Where(item => item.Book.UrlName == bookUrl)
                .Select(item => new Tuple<string, string>(item.Book.Title, item.BlobFileName))
                .FirstOrDefaultAsync();
        }
    }
}
```

Файл BookUserRepository.cs

```
using BookGarden.Entities.Books;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;
namespace BookGarden.Storage.Repositories.BookModule.BooksUser
{
    public class BookUserRepository : BaseRepository<BookUser>, IBookUserRepository
    {
        public BookUserRepository(DatabaseContext context) : base(context)
        { }
        public Task<BookUser> GetAsync(string bookUrl, string userHash)
        {
            return Source
                .Include(d => d.Book)
                .FirstOrDefaultAsync(bu => bu.UserHash == userHash && bu.Book.UrlName == bookUrl);
        }
```

```
    }
}
```

Файл CompilationLocaleRepository.cs

```
using BookGarden.Entities.Compilations;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
namespace BookGarden.Storage.Repositories.CompilationModule.CompilationLocales
{
    public class CompilationLocaleRepository : BaseRepository<CompilationLocale>,
ICompilationLocaleRepository
    {
        public CompilationLocaleRepository(DatabaseContext context) : base(context)
        { }
    }
}
```

Файл CompilationRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Entities.Compilations;
using BookGarden.Shared;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Extensions;
using BookGarden.Storage.Extensions;
using BookGarden.Storage.Interceptors;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
using BookGarden.Storage.Repositories.BookModule.Books.Views;
using BookGarden.Storage.Repositories.CompilationModule.Compilations.Filters;
using BookGarden.Storage.Repositories.CompilationModule.Compilations.Views;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.CompilationModule.Compilations
{
    public class CompilationRepository : BaseRepository<Compilation>, ICompilationRepository
    {
        public CompilationRepository(DatabaseContext context) : base(context)
        { }
        public Task<Compilation> GetAsync(int itemId)
        {
            return Source
                .Include(item => item.Locales)
                .FirstOrDefaultAsync(item => item.Id == itemId);
        }
        public Task<Compilation> GetAsync(string compilationUrl)
        {
            return Source
                .Include(item => item.Locales)
                .FirstOrDefaultAsync(item => item.UrlName == compilationUrl);
        }
        public Task<List<Compilation>> GetNewToUpdateAsync(DateTime validDate)
        {
            return Source
                .Where(item => item.StateStatus == StateStatus.New)
                .Where(book => book.CreatedOnUtc <= validDate)
                .ToListAsync();
        }
        public Task<int> GetIdAsync(string compilationUrl)
        {
            return Source
                .Where(item => item.UrlName == compilationUrl)
```

```
            .Select(item => item.Id)
            .FirstOrDefaultAsync();
    }
    public Task<FullCompilationView> GetFullViewAsync(string compilationUrl, Language lang)
    {
        return Source
            .Include(d => d.Locales)
            .Include(d => d.Books).ThenInclude(d => d.Book)
            .Where(item => item.UrlName.ToLower() == compilationUrl.ToLower())
            .Select(item => new FullCompilationView
            {
                UrlName = item.UrlName,
                Title = item.Locales.GetCompilationTitle(lang),
                Description = item.Locales.GetCompilationDescription(lang),
                ImageLink = item.ImageLink,
                Language = item.Language,
                StateStatus = item.StateStatus,
                CreatedOnUtc = item.CreatedOnUtc,
                RatesCount = item.RatesCount,
                Rating = item.Rating,
                BooksCount = item.Books.Count
            }).FirstOrDefaultAsync();
    }
    public Task<List<CompilationLightView>> GetTopViewsAsync(Language language)
    {
        return Source
            .Where(item => item.IsActive)
            .Select(item => new CompilationLightView
            {
                ImageLink = item.ImageLink,
                Title = item.Locales.GetCompilationTitle(language),
                BooksCount = item.Books.Count,
                Language = item.Language,
                StateStatus = item.StateStatus,
                UrlName = item.UrlName
            })
            .Take(10)
            .ToListAsync();
    }
    public Task<List<CompilationView>> GetViewsAsync(CompilationRequestFilter filter)
    {
        var query = GetQuery(filter);
        query = filter.SortType switch
        {
            SortType.Popularity => query.OrderByDescending(item => item.Books.Count),
            SortType.Date => query.OrderByDescending(item => item.CreatedOnUtc),
            // todo SortType.Rating => query.OrderByDescending(item => item.Rating)
            _ => query,
        };
        var selectedQuery = query
            .Skip(--filter.Page * filter.PageSize)
            .Take(filter.PageSize)
            .Include(item => item.Locales)
            .Select(item => new CompilationView
            {
                ImageLink = item.ImageLink,
                Title = item.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage) != null
                    ? item.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage).Title
                    : "",
                UrlName = item.UrlName,
                BooksCount = item.Books.Count,
                Language = item.Language,
                StateStatus = item.StateStatus,
```

```
                Books = item.Books.Select(b => b.Book)
                    .OrderByDescending(item => item.PagesCount)
                    .Take(BookGardenConstants.DescriptionBookCount)
                    .Select(item => new BookViewWithAuthors
                    {
                        Title = item.Title,
                        StateStatus = item.StateStatus,
                        UrlName = item.UrlName,
                        Rating = item.Rating,
                        ImageLink = item.ImageLink,
                        Authors = item.Authors.Select(author =>
                            author.Author.Locales.GetAuthorFullName(filter.RequestLanguage)).ToArray(),
                    })
                    .ToList(),
            });
        selectedQuery = filter.SortType switch
        {
            SortType.Title => selectedQuery.OrderBy(item => item.Title),
            SortType.BookCount => selectedQuery.OrderByDescending(item => item.BooksCount),
            _ => selectedQuery,
        };
        return selectedQuery.ToListAsync();
    }
    public Task<int> GetViewsCountAsync(CompilationRequestFilter filter)
    {
        return GetQuery(filter)
            .CountAsync();
    }
    public Task<List<ListItem<int>>> GetOptionsAsync(Language lang)
    {
        return Source
            .Select(item => new ListItem<int>
            {
                Value = item.Id,
                Text = item.Locales.FirstOrDefault(l => l.Language == lang) != null
                    ? item.Locales.FirstOrDefault(l => l.Language == lang).Title
                    : "",
            })
            .OrderBy(item => item.Text)
            .ToListAsync();
    }
    public                                                    Task<List<CompilationAdminView>>
GetViewsAsAdminAsync(CompilationRequestAdminFilter filter)
    {
        return GetViewsAsAdminQuery(filter)
            .Skip(--filter.Page * filter.PageSize)
            .Take(filter.PageSize)
            .ToListAsync();
    }
    public Task<int> GetViewsCountAsAdminAsync(CompilationRequestAdminFilter filter)
    {
        return GetViewsAsAdminQuery(filter)
            .CountAsync();
    }
    public async Task<bool> IsUrlExistAsync(string urlName, int compilationId)
    {
        return await Source.AnyAsync(item => item.UrlName == urlName && item.Id != compilationId);
    }
    private                                                   IQueryable<CompilationAdminView>
GetViewsAsAdminQuery(CompilationRequestAdminFilter filter)
    {
        return Source
            .Select(item => new CompilationAdminView
```

```
            {
                Id = item.Id,
                Title = item.Locales.GetCompilationTitle(filter.RequestLanguage),
                Description = item.Locales.GetCompilationDescription(filter.RequestLanguage),
                Language = item.Language,
                UrlName = item.UrlName,
                ImageLink = item.ImageLink,
                BooksCount = item.Books.Count,
                IsActive = item.IsActive
            });
        }
        private IQueryable<Compilation> GetQuery(CompilationRequestFilter filter)
        {
            var query = Source.Where(book => book.IsActive);
            if (filter.Search.IsNotNullOrWhiteSpace())
            {
                filter.Search = FtsInterceptor.Fts(filter.Search, FtsConditionType.And, true);
                query = query.Where(item => item.SearchKeywords.Contains(filter.Search));
            }
            if (filter.GenreUrls.Any())
            {
                query = query
                    .Where(item => item.Books.SelectMany(b=> b.Book.Genres)
                        .Select(a => a.Genre.UrlName)
                        .Any(a => filter.GenreUrls.Contains(a)));
            }
            if (filter.Languages.Any())
            {
                query = query.Where(item => filter.Languages.Contains(item.Language));
            }
            if (filter.Rating > 1)
            {
                query = query.Where(item => item.Rating >= filter.Rating);
            }
            return query;
        }
    }
}
```

Файл GenreLocaleRepository.cs

```
using BookGarden.Entities.Genres;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
namespace BookGarden.Storage.Repositories.GenreModule.GenreLocales
{
    public class GenreLocaleRepository : BaseRepository<GenreLocale>, IGenreLocaleRepository
    {
        public GenreLocaleRepository(DatabaseContext context) : base(context)
        { }
    }
}
```

Файл GenreRepository.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookGarden.Entities.Genres;
using BookGarden.Shared.Entities;
using BookGarden.Shared.Enums;
using BookGarden.Shared.Extensions;
using BookGarden.Storage.Extensions;
using BookGarden.Storage.Interceptors;
using BookGarden.Storage.Persistence.Context;
using BookGarden.Storage.Persistence.Repository;
```

```
using BookGarden.Storage.Repositories.GenreModule.Genres.Filters;
using BookGarden.Storage.Repositories.GenreModule.Genres.Views;
using Microsoft.EntityFrameworkCore;
namespace BookGarden.Storage.Repositories.GenreModule.Genres
{
   public class GenreRepository : BaseRepository<Genre>, IGenreRepository
   {
      public GenreRepository(DatabaseContext context) : base(context)
      { }
      public Task<Genre> GetAsync(int itemId)
      {
         return Source
            .Include(item => item.Locales)
            .FirstOrDefaultAsync(item => item.Id == itemId);
      }
      public Task<List<GenreLightView>> GetTopViewsAsync(Language language)
      {
         return Source
            .Where(item => item.IsActive)
            .Select(item => new GenreLightView
            {
               ImageLink = item.ImageLink,
               Name = item.Locales.GetGenreTitle(language),
               UrlName = item.UrlName
            })
            .Take(10)
            .ToListAsync();
      }
      public Task<int> GetViewsCountAsync(GenreRequestFilter filter)
      {
         return GetQuery(filter)
            .CountAsync();
      }
      public Task<List<ListItem<int>>> GetMainAsync(Language lang)
      {
         return Source
            .Include(item => item.Locales)
            .Where(item => item.ParentGenreId == null)
            .Select(item => new ListItem<int>
            {
               Value = item.Id,
               Text = item.Locales.GetGenreTitle(lang),
            })
            .ToListAsync();
      }
      public Task<int> GetIdAsync(string genreUrl)
      {
         return Source
            .Where(item => item.UrlName == genreUrl)
            .Select(item => item.Id)
            .FirstOrDefaultAsync();
      }
      public Task<List<ListItem<int>>> GetChildAsync(int parentGenreId, Language lang)
      {
         return Source
            .Include(item => item.Locales)
            .Where(item => item.ParentGenreId == parentGenreId)
            .Select(item => new ListItem<int>
            {
               Value = item.Id,
               Text = item.Locales.GetGenreTitle(lang),
            })
            .ToListAsync();
```

```
}
public Task<FullGenreView> GetViewAsync(string genreUrl, Language lang)
{
    return Source
        .Where(item => item.IsActive)
        .Where(item => item.UrlName == genreUrl)
        .Include(d => d.ParentGenre).ThenInclude(d => d.Locales)
        .Select(item => new FullGenreView
        {
            ImageLink = item.ImageLink,
            Name = item.Locales.GetGenreTitle(lang),
            Description = item.Locales.GetGenreDescription(lang),
            UrlName = item.UrlName,
            ParentGenre = item.ParentGenreId != null
                ? new GenreLightView
                {
                    ImageLink = item.ParentGenre.ImageLink,
                    Name = item.ParentGenre.Locales.GetGenreTitle(lang),
                    UrlName = item.ParentGenre.UrlName
                }
                : null
        })
        .FirstOrDefaultAsync();
}
public Task<List<GenreView>> GetViewsAsync(GenreRequestFilter filter)
{
    var query = GetQuery(filter);
    query = filter.SortType switch
    {
        SortType.Popularity => query.OrderByDescending(item => item.BookGenres.Count),
        SortType.Date => query.OrderByDescending(item => item.CreatedOnUtc),
        SortType.BookCount => query.OrderByDescending(item => item.BookGenres.Count),
        // todo SortType.Rating => query.OrderByDescending(item => item.Rating)
        _ => query,
    };
    var selectedQuery = query
        .Skip(--filter.Page * filter.PageSize)
        .Take(filter.PageSize)
        .Include(item => item.Locales)
        .Select(item => new GenreView
        {
            ImageLink = item.ImageLink,
            Name = item.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage) != null
                ? item.Locales.FirstOrDefault(l => l.Language == filter.RequestLanguage).Title
                : "",
            UrlName = item.UrlName
        });
    selectedQuery = filter.SortType switch
    {
        SortType.Title => selectedQuery.OrderBy(item => item.Name),
        _ => selectedQuery,
    };
    return selectedQuery.ToListAsync();
}
public Task<List<GenreStoredView>> GetViewsAsync(bool isAdmin)
{
    var genres = Source
        .Include(item => item.Locales)
        .Where(item => isAdmin || item.IsActive)
        .Select(item => new
        {
            item.Id,
            item.ParentGenreId,
```

```
            item.UrlName,
            NameEn = item.Locales.GetGenreTitle(Language.English),
            NameRu = item.Locales.GetGenreTitle(Language.Russian),
            NameUa = item.Locales.GetGenreTitle(Language.Ukrainian),
        });
    var query = from item in genres
                join pGenre in genres
                    on item.ParentGenreId equals pGenre.Id
                    into pGenreJoin
                from pGenre in pGenreJoin.DefaultIfEmpty()
                select new GenreStoredView
                {
                    Id = item.Id,
                    UrlName = item.UrlName,
                    NameEn = item.NameEn,
                    NameRu = item.NameRu,
                    NameUa = item.NameUa,
                    ParentGenreId = item.ParentGenreId,
                    ParentGenreNameEn = pGenre.NameEn,
                    ParentGenreNameRu = pGenre.NameRu,
                    ParentGenreNameUa = pGenre.NameUa
                };
    return query.ToListAsync();
}
public Task<List<ListItem<string>>> GetParentOptionsAsync(Language lang)
{
    return Source
        .Include(item => item.Locales)
        .Include(item => item.ChildGenres)
        .Where(item => item.IsActive && item.ChildGenres.Count > 0)
        .Select(item => new ListItem<string>
        {
            Value = item.UrlName,
            Text = item.Locales.GetGenreTitle(lang)
        })
        .ToListAsync();
}
public Task<List<GenreAdminView>> GetViewsAsAdminAsync(GenreRequestAdminFilter filter)
{
    return GetViews(filter)
        .Skip(--filter.Page * filter.PageSize)
        .Take(filter.PageSize)
        .ToListAsync();
}
public Task<int> GetViewsCountAsAdminAsync(GenreRequestAdminFilter filter)
{
    return GetViews(filter)
        .CountAsync();
}
public async Task<bool> IsUrlExistAsync(string urlName, int genreId)
{
    return await Source.AnyAsync(item => item.UrlName == urlName && item.Id != genreId);
}
private IQueryable<GenreAdminView> GetViews(GenreRequestAdminFilter filter)
{
    return Source.Select(item => new GenreAdminView
    {
        Id = item.Id,
        Title = item.Locales.GetGenreTitle(filter.RequestLanguage),
        Description = item.Locales.GetGenreDescription(filter.RequestLanguage),
        UrlName = item.UrlName,
        ImageLink = item.ImageLink,
        ParentId = item.ParentGenreId,
```

```
                IsActive = item.IsActive
            });
        }
        private IQueryable<Genre> GetQuery(GenreRequestFilter filter)
        {
            var query = Source.Where(book => book.IsActive);
            if (filter.Search.IsNotNullOrWhiteSpace())
            {
                filter.Search = FtsInterceptor.Fts(filter.Search, FtsConditionType.And, true);
                query = query.Where(item => item.SearchKeywords.Contains(filter.Search));
            }
            if (filter.GenreUrls.Any())
            {
                query = query
                    .Include(d => d.ParentGenre)
                    .Where(item => item.ParentGenre != null &&
filter.GenreUrls.Contains(item.ParentGenre.UrlName));
            }
            return query;
        }
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ОНЛАЙН БІБЛІОТЕКИ

Програма та методика тестування

КПІ.ІТ-9409.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
Катерина ЛІЩУК

Нормоконтроль:
Ірина ВІТКОВСЬКА

Виконавець:
Даніл КРАВЧЕНКО

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб застосунок, що виконує основні функції онлайн бібліотеки, а саме 2 його частини: серверна на клієнтська. Серверна частина розроблена на платформі .Net з мовою програмування C#. А клієнтська частина розроблена за допомогою фреймоворку Nuxt.js.

2 МЕТА ТЕСТУВАННЯ

Було обрано для тестування з метою перевірки відповідний функціонал:

– правильна робота веб-застосунку відповідно до функціональних вимог;

– валідація ввідних даних користувача;

– збереження фотографій користувачів та контенту;

– збереження та завантаження файлів;

– збереження оновлених даних;

– сумісність веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox, Microsoft Edge);

– зрозумілість та зручність графічного інтерфейсу;

– швидкість навігації та відображення контенту.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються два методи:

– мануальне тестування;

– тестування «сірої скриньки».

Це надає можливість перевірити не тільки взаємодію клієнтської частини з серверною, а й протестувати певні частини тільки серверної частини.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується у два етапи, перший етап це автоматизоване тестування за допомогою бібліотеки xUnit, що допомагає у написанні unit-тестів. Другий етап – ручне тестування, коли ми впевнені, що серверна частина правильно виконує свої функції, можемо приступити до мануального тестування, чим самим підсумувати працездатність та відповідність за допомогою:

– тестування підтримки сервером мов локалізації;

– тестування CRUD операцій над контентом;

– тестування працездатності автоматичного копіювання даних між шарами;

– мануального тестування основних процесів створенню та перегляду контенту;

– мануального тестування процесу публікації відгуку;

– тестування локалізації клієнтської частини.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ОНЛАЙН БІБЛІОТЕКИ

Керівництво користувача

КПІ.ІТ-9409.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
Катерина ЛІЩУК

Нормоконтроль:
Ірина ВІТКОВСЬКА

Виконавець:
Даніл КРАВЧЕНКО

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

BookGarden – це веб-застосунок для онлайн бібліотеки, який призначений для забезпечення зручного та доступного способу отримання та користування книжками у електронному форматі через Інтернет.

Деякі з основних функцій та користувацьких можливостей, які реалізовані веб-застосунком для онлайн бібліотеки, включають:

– перегляд каталогу книжок та контенту: Користувачі можуть переглядати список доступних книжок, авторів, жанрів та збірок у бібліотеці, включаючи фільтрацію та сортування за різними параметрами, такими як автор, жанр, рейтинг тощо;

– пошук книжок: Користувачі можуть шукати книжки за назвою, автором або ключовими словами;

– завантаження книжки або читання онлайн: користувачі можуть завантажити електронний формат книжки у наявному форматі або почитати онлайн за допомогою epub-reader;

– управління користувачами: Адміністратори системи можуть мати можливість керувати користувачами, зокрема реєструвати нових користувачів, надавати права доступу або видаляти;

– рейтинг та відгуки: Користувачі можуть залишати свої оцінки та відгуки про книжки, що допомагає іншим користувачам при виборі книжок для читання;

– особистий кабінет: Кожен користувач може мати свій особистий кабінет, де можна переглянути власні колекції книжок позичені книжки, історія читання, відгуки, налаштування профілю тощо.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Мінімальна конфігурація технічних засобів:

– будь-яка операційна система: Linux, Windows, MacOS;

– підключення до інтернету;

– будь-який веб-браузер;

– підключення до інтернету.

Рекомендована конфігурація технічних засобів:

– процесор: Intel Pentium III, Celeron або AMD Athlon, з тактовою частотою 500 МГц;

– оперативна память: 2 Гб;

– відеокарта: с 1 Гб відеопам'яті;

– підключення до інтернету.

2.2 Завантаження застосунку

Програмне забезпечення є веб-застосунком, а отже воно розгорнуте та є доступне цілодобово.

2.3 Перевірка коректної роботи

У відкритій вкладці застосунку перейти на головну сторінку і переглянути чи наявні дані про авторів, книжки та колекції. Також можна перевірити взаємодію між клієнтом та сервером за допомогою створення будь-якого повідомлення це може бути публікація відгук, реєстрація нового користувача або авторизація існуючого.

3 ВИКОНАННЯ ПРОГРАМИ

При першому відкритті застосунку користувачу буде відображено головну сторінку застосунку, де знаходяться популярні книжки, автори та жанри (рисунок 3.1)

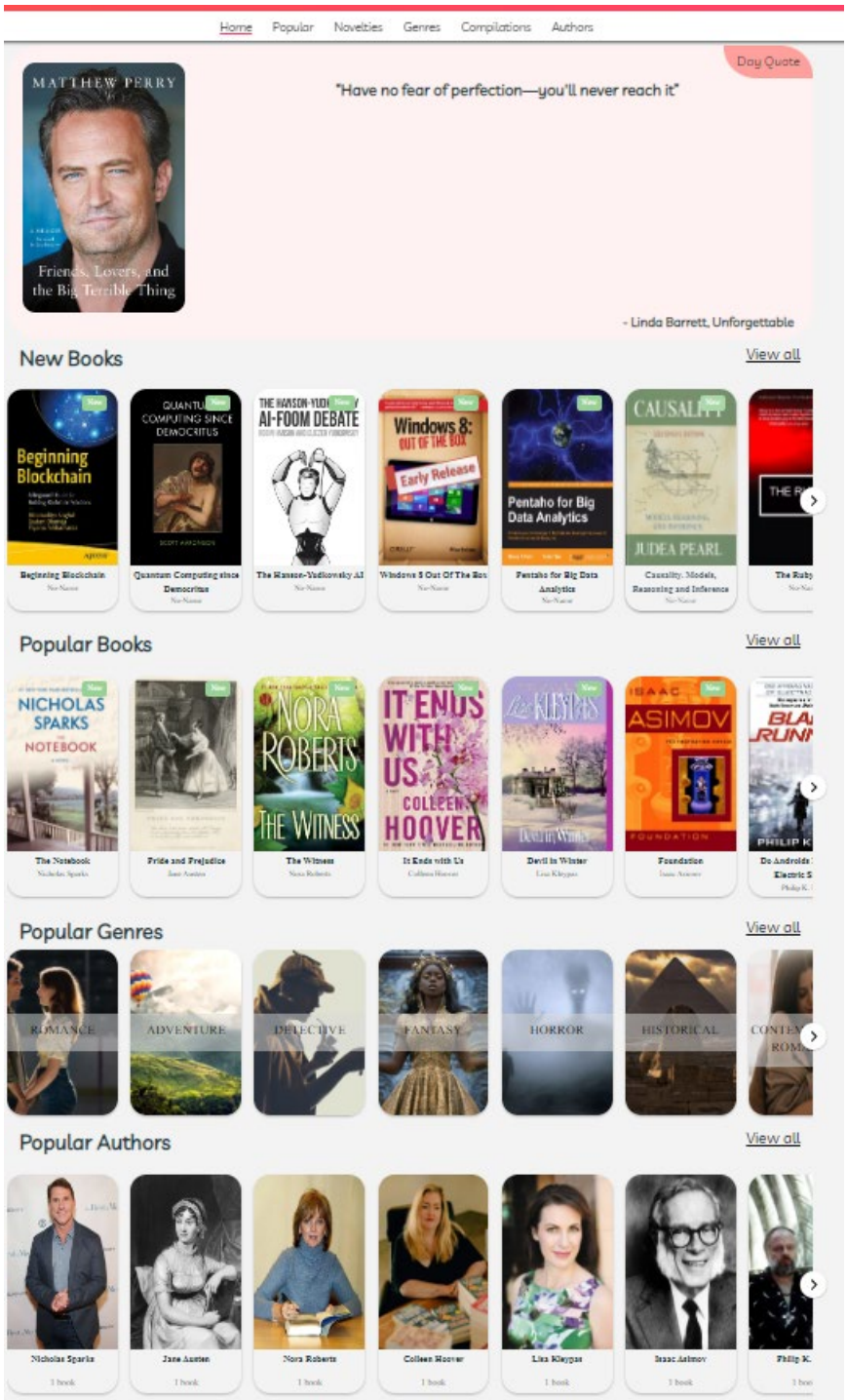


Рисунок 3.1 – Стартова сторінка застосунку

Далі користувач має змогу перейти до бажаного автора (рисунок 3.2) або переглянути книжки відповідного жанру (рисунок 3.3).

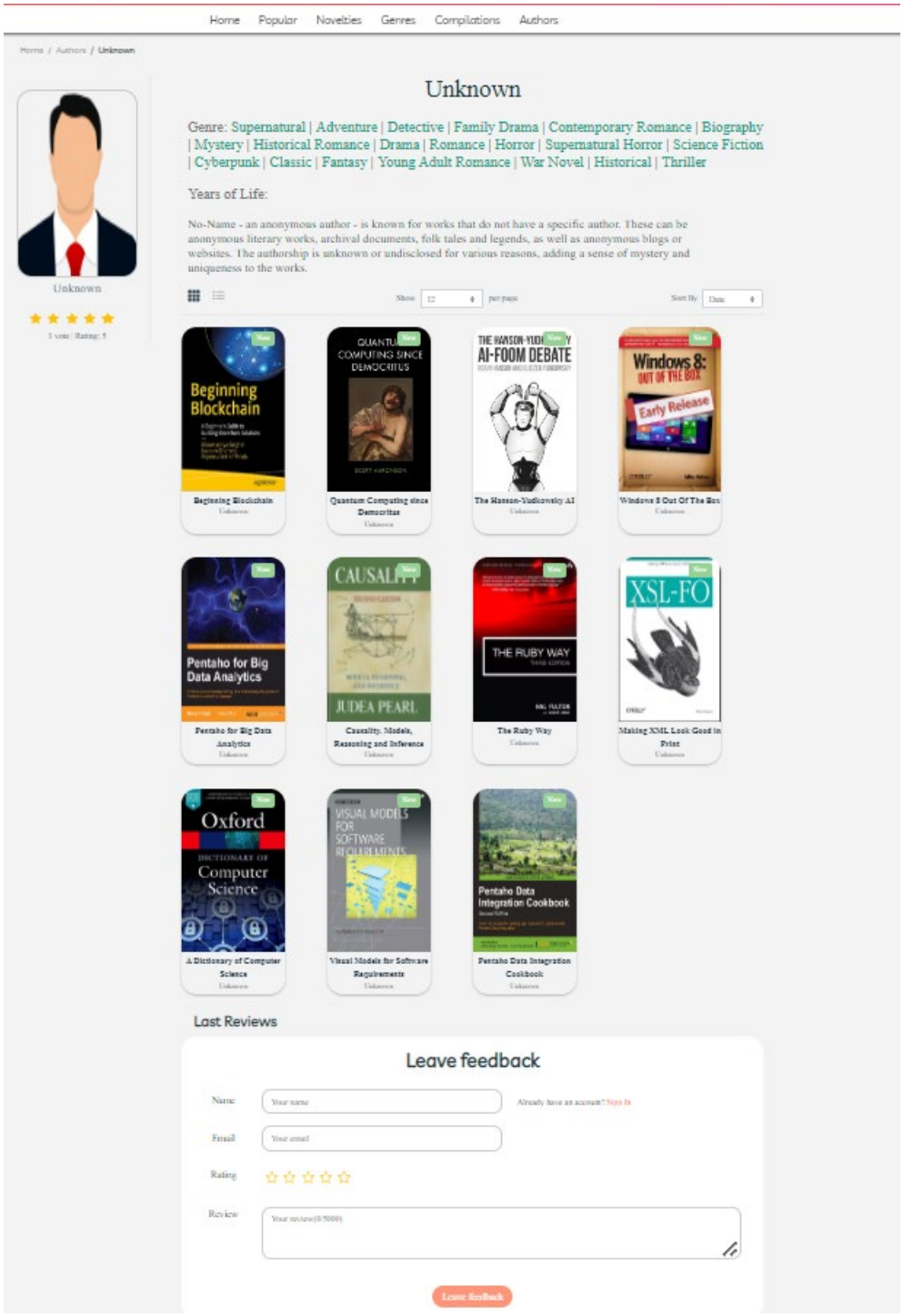


Рисунок 3.2 – Сторінка автора книжок

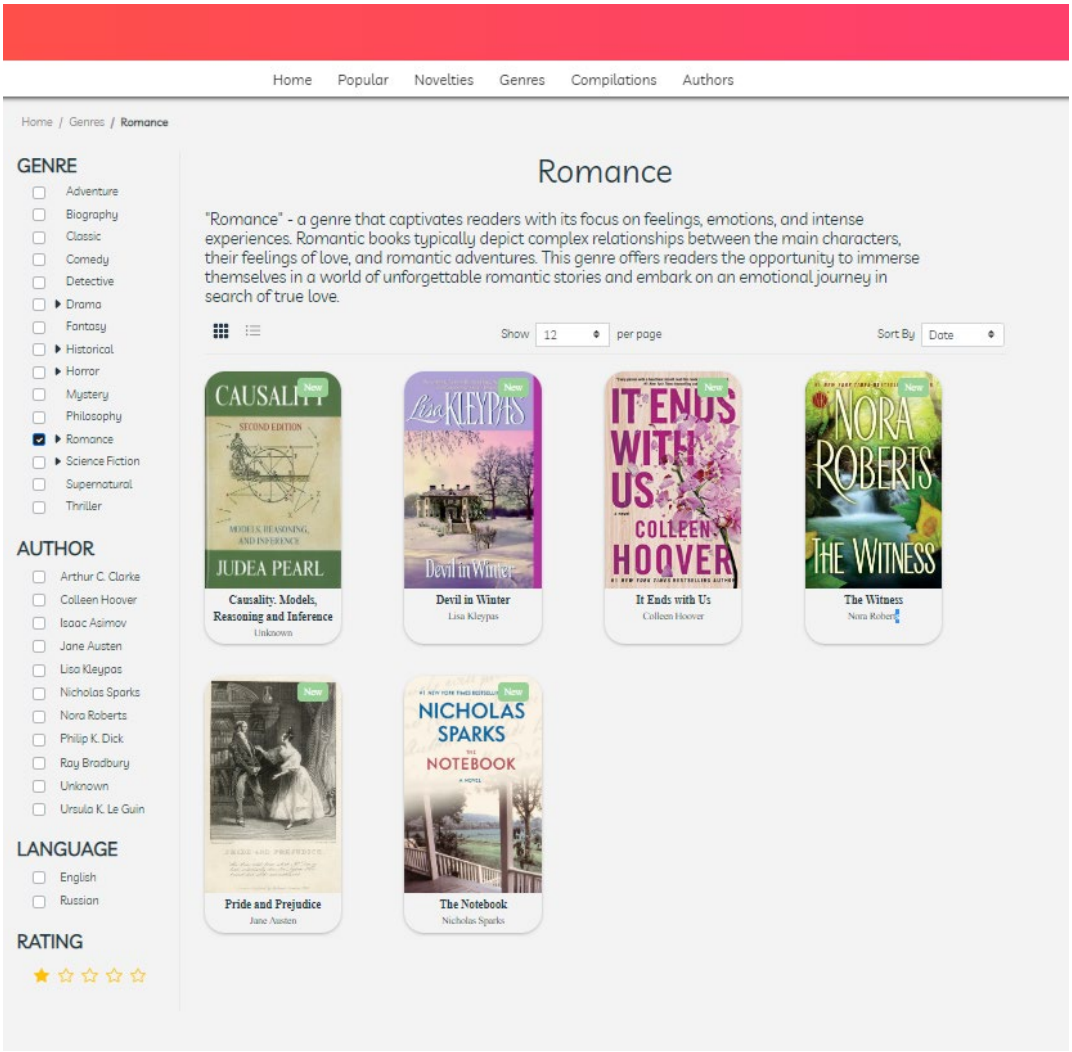


Рисунок 3.3 – Сторінка обраного жанру

Далі користувач може здійснити пошук чи просто перейти до бажаної книжки, де буде зображено деталі книжка як на рисунку 3.4

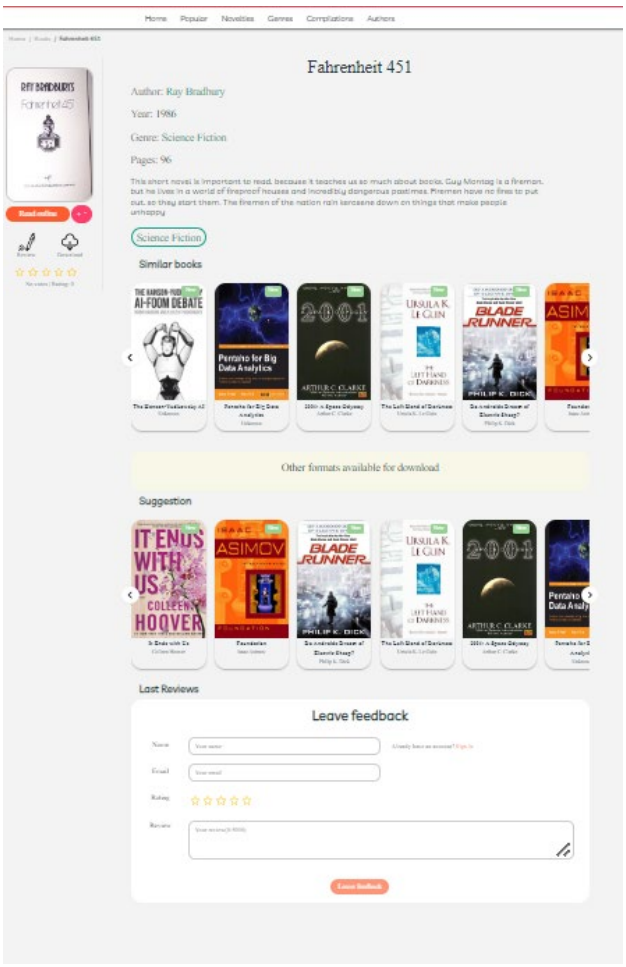


Рисунок 3.4 – Сторінка обраної книжки

Тепер користувач може здійснити такі операції:

– завантажити книжку у необхідному форматі (рисунок 3.5);

– читання книжки онлайн (рисунок 3.6);

– публікація відгуку (рисунок 3.7).

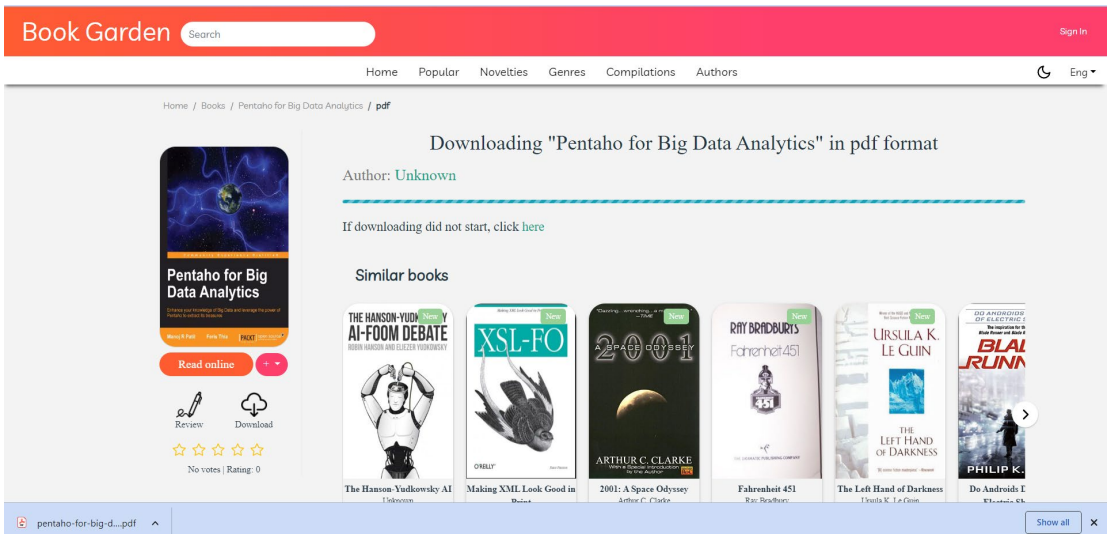


Рисунок 3.5 – Завантаження електронний формат книжки

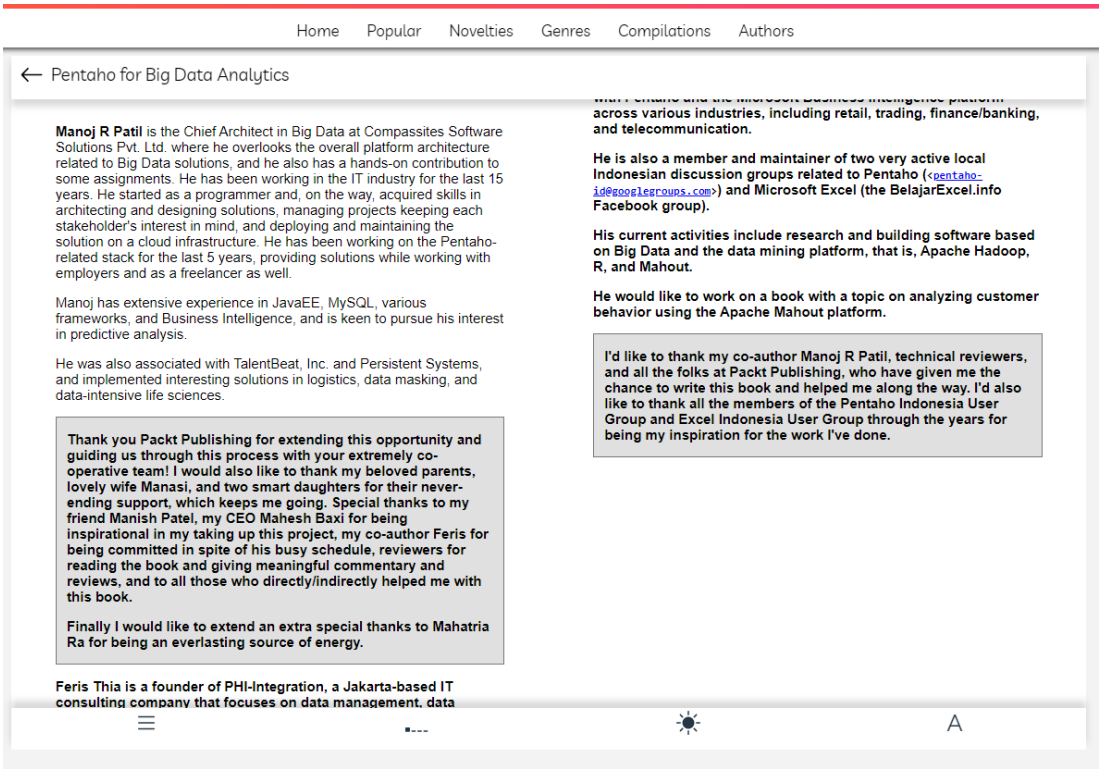


Рисунок 3.6 – Читання онлайн цифрового варіанту

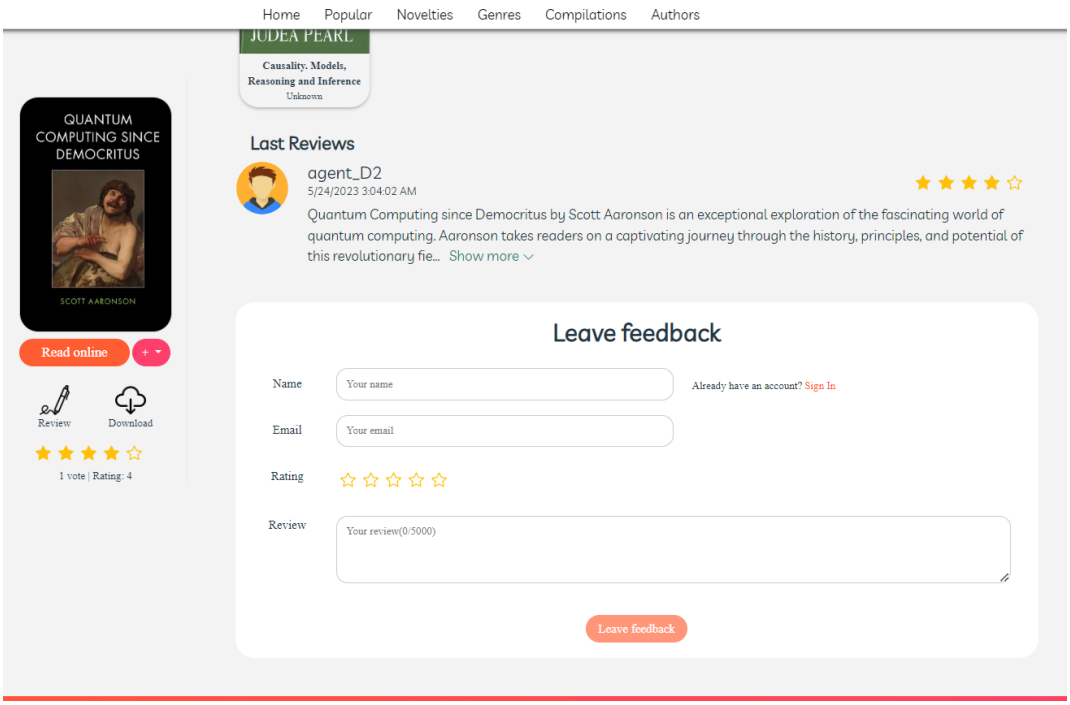


Рисунок 3.7 – Успішна публікація відгуку

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ВЕБ-ЗАСТОСУНОК ДЛЯ ОНЛАЙН БІБЛІОТЕКИ

Графічний матеріал

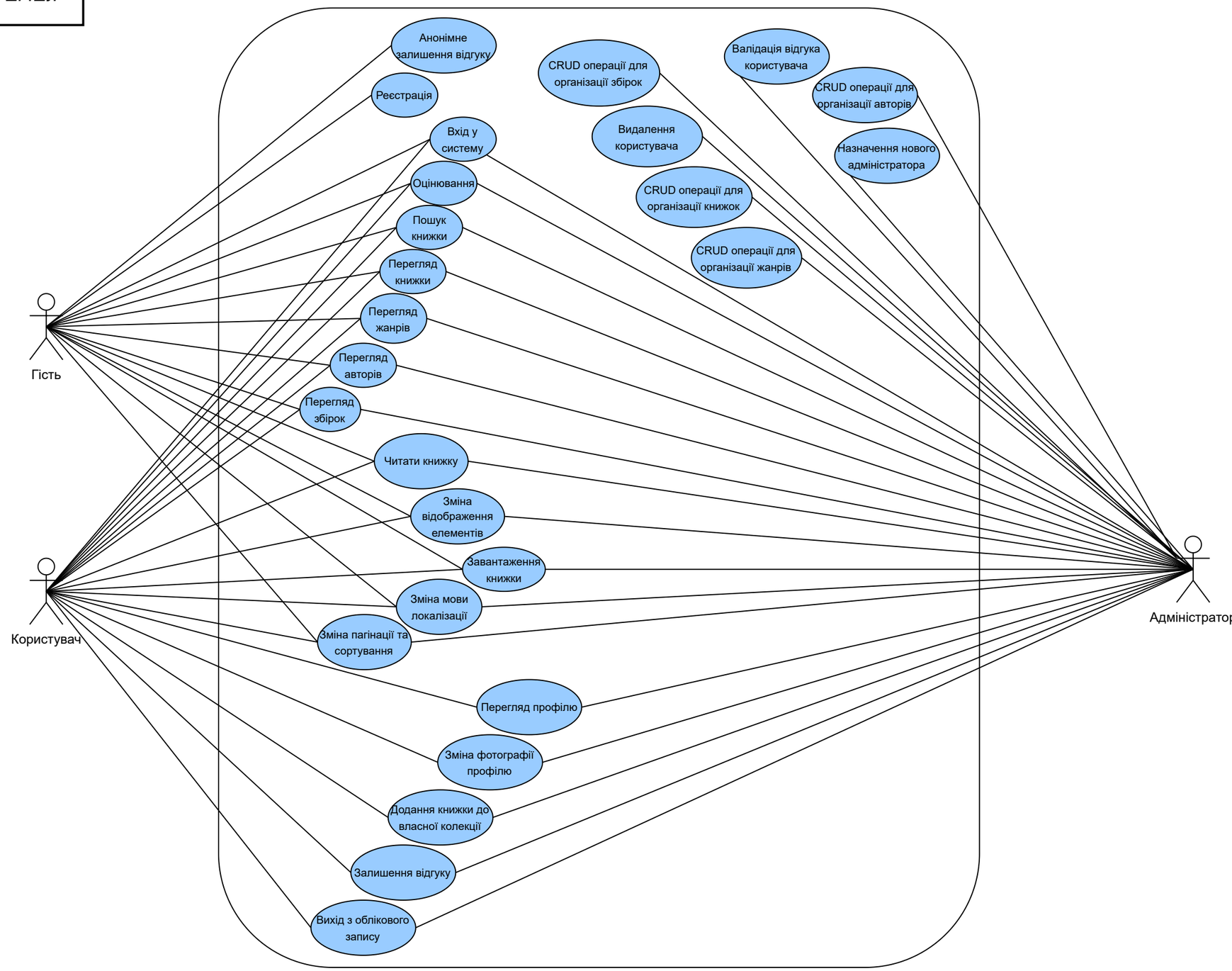
КПІ.ІТ-9409.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
Катерина ЛІЩУК

Нормоконтроль:
Ірина ВІТКОВСЬКА

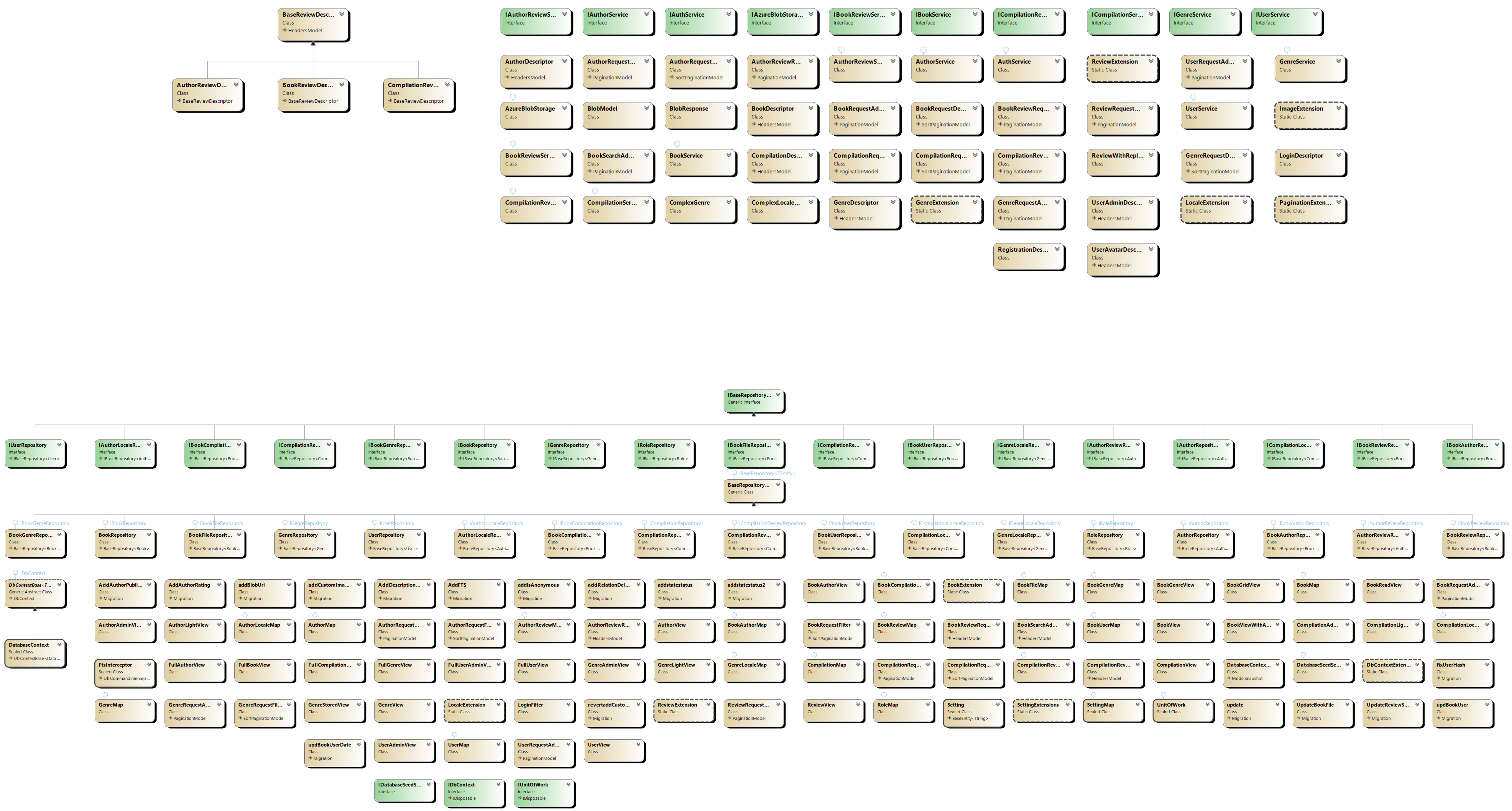
Виконавець:
Даніл КРАВЧЕНКО

Київ – 2023

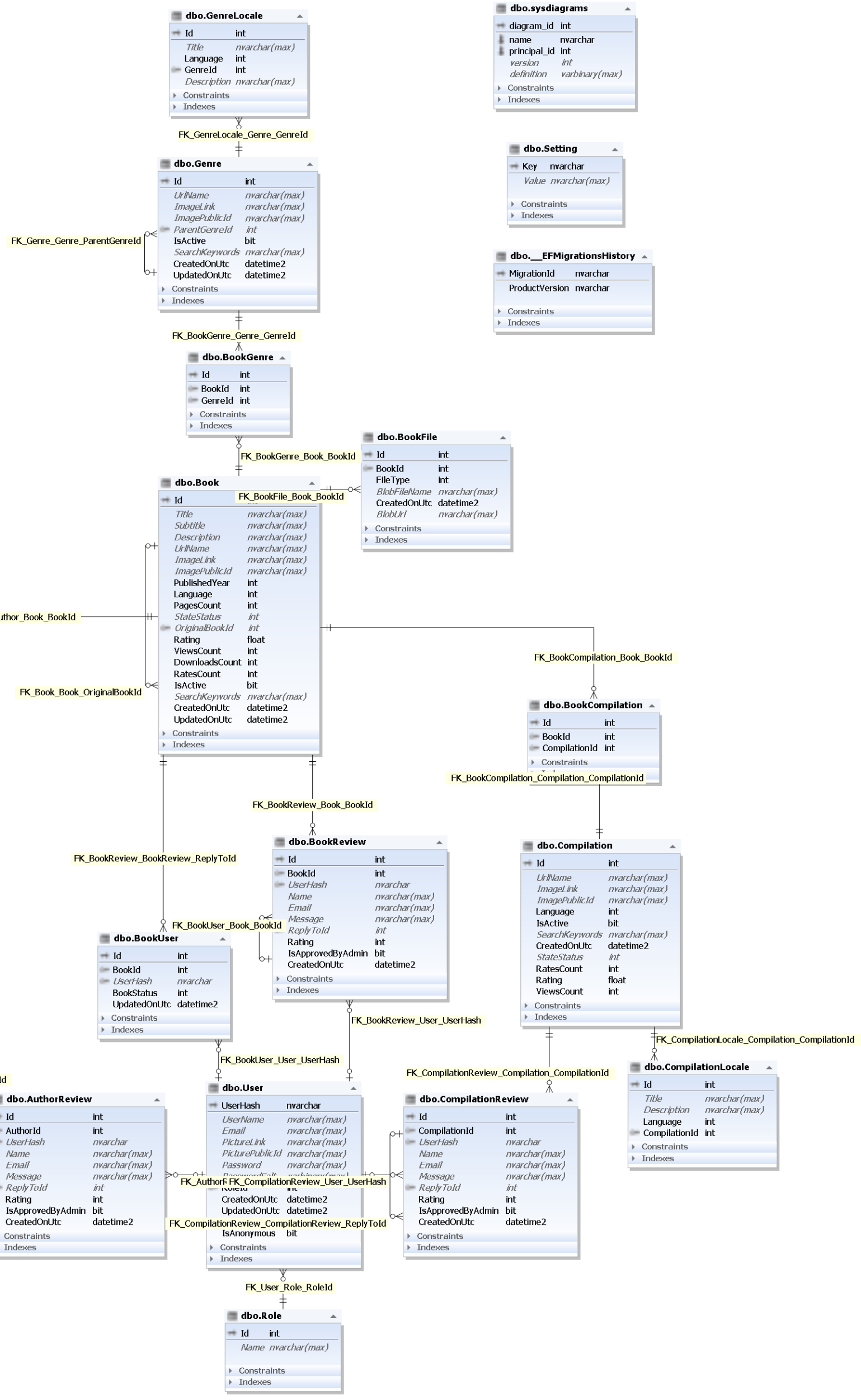


					КПІ.ІТ-9409.045440.06.99.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера / Маса / Масштаб
Розробив		Кравченко Д.О.				
Перевірив		Ліщук К.І.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Вітковська І.І.			Веб-застосунок для онлайн бібліотеки	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-94
Затвердив		Ліщук К.І.				

КПІ.ІТ-9409.045440.06.99.СКК



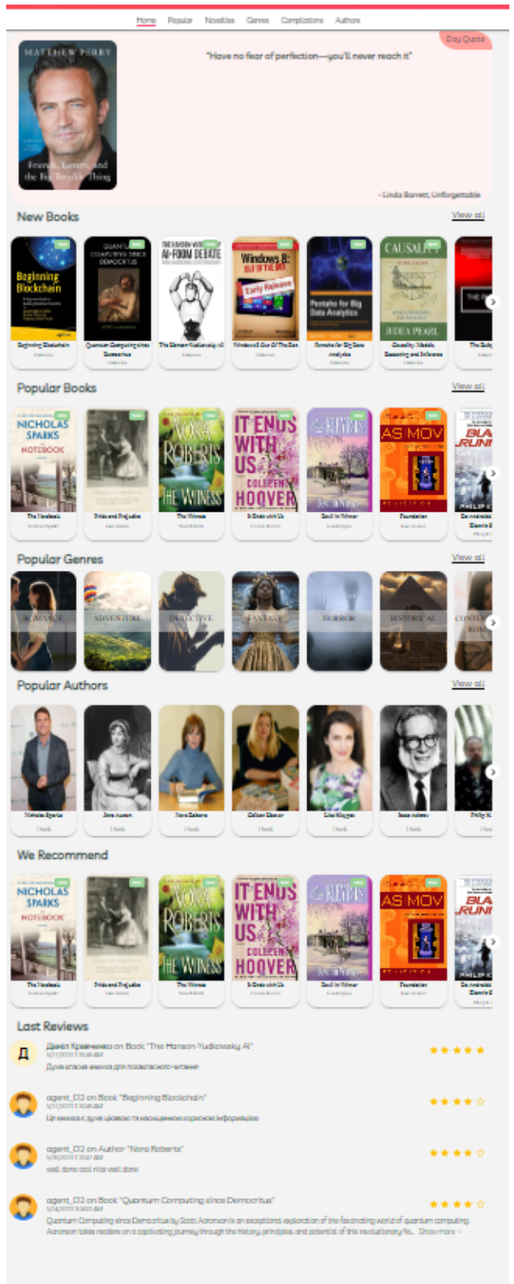
Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІТ-9409.045440.06.99.СКК		
Розробив		Кравченко Д.О.			Схема структурна класів програмного забезпечення	Літера	Маса	Масштаб
Перевірив		Ліщук К.І.					
Т. кон.						Аркуш	Аркушів
Н. кон.		Вітковська І.І.			Веб-застосунок для онлайн бібліотеки	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-94	
Затвердив		Ліщук К.І.					

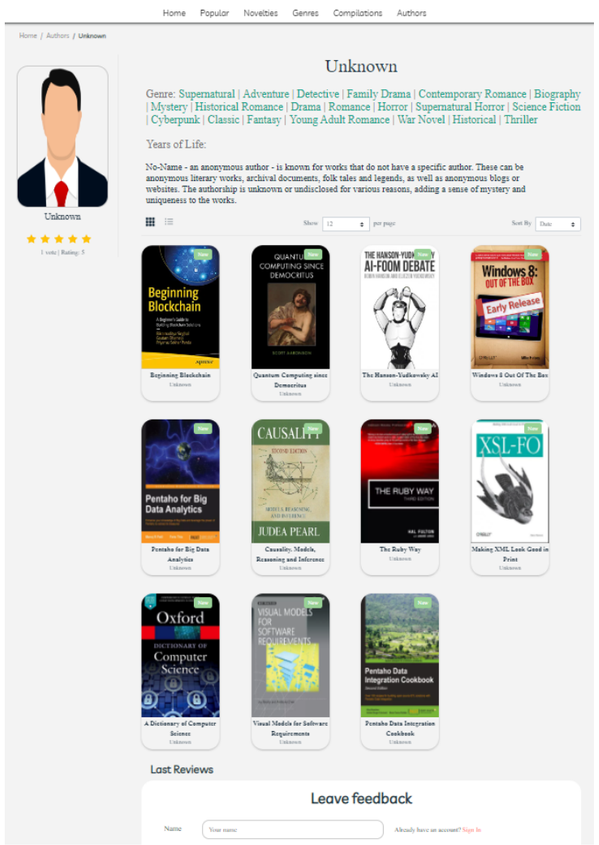


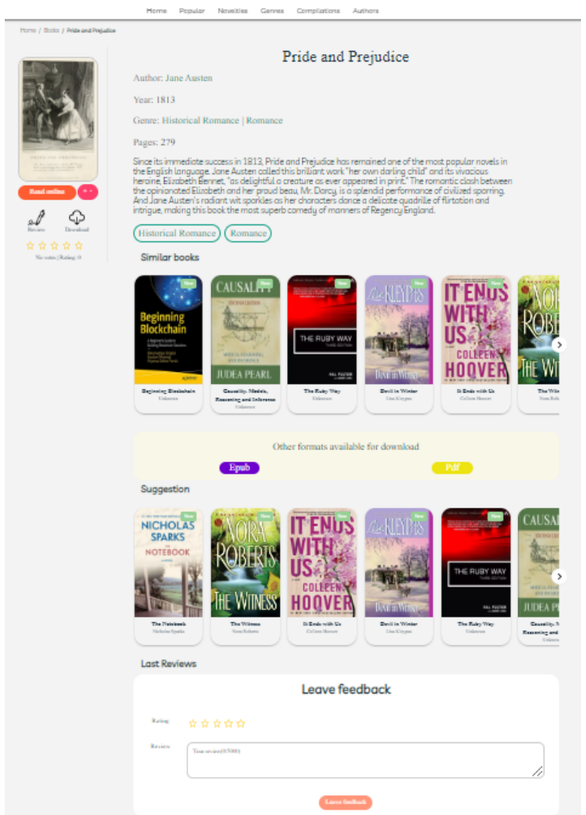
					КПІ.ІТ-9409.045440.06.99.СБД
Зм.	Арк.	№ докум.	Підп.	Дата	
Розроб.		Кравченко Д.О.			Схема бази даних
Перев.		Ліщук К.І.			
Т. Кон.					
Н. Кон.		Вітковська І.І.			Веб-застосунок для онлайн бібліотеки
Затв.		Ліщук К.І.			

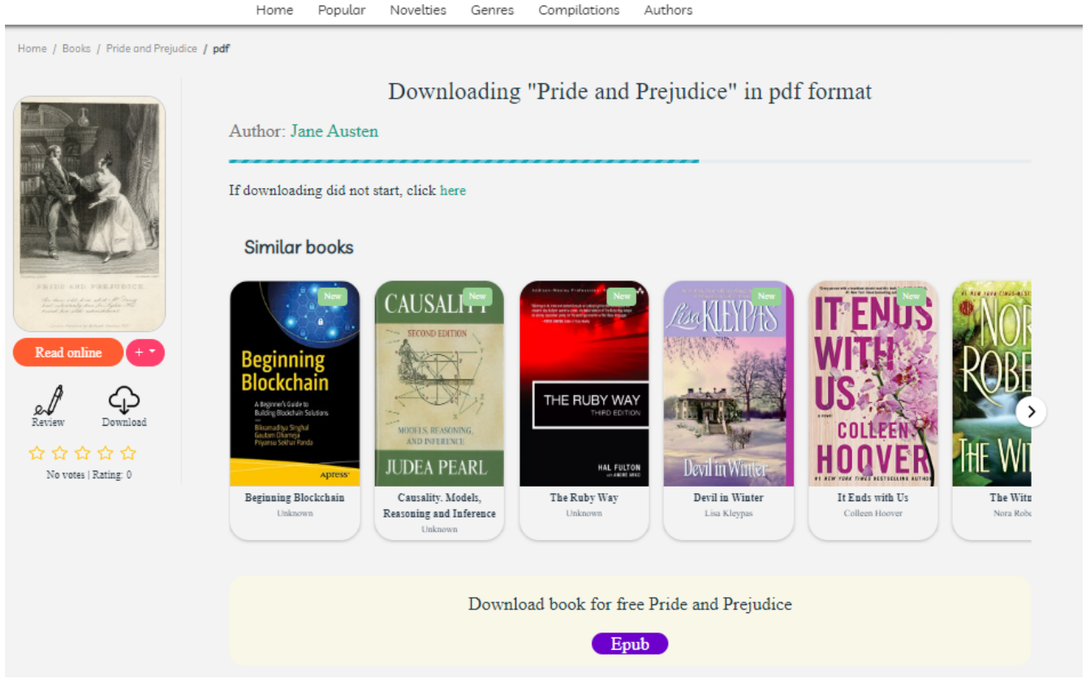
Лит.	Арк.	Аркушів
		1
Аркуш	Аркушів	

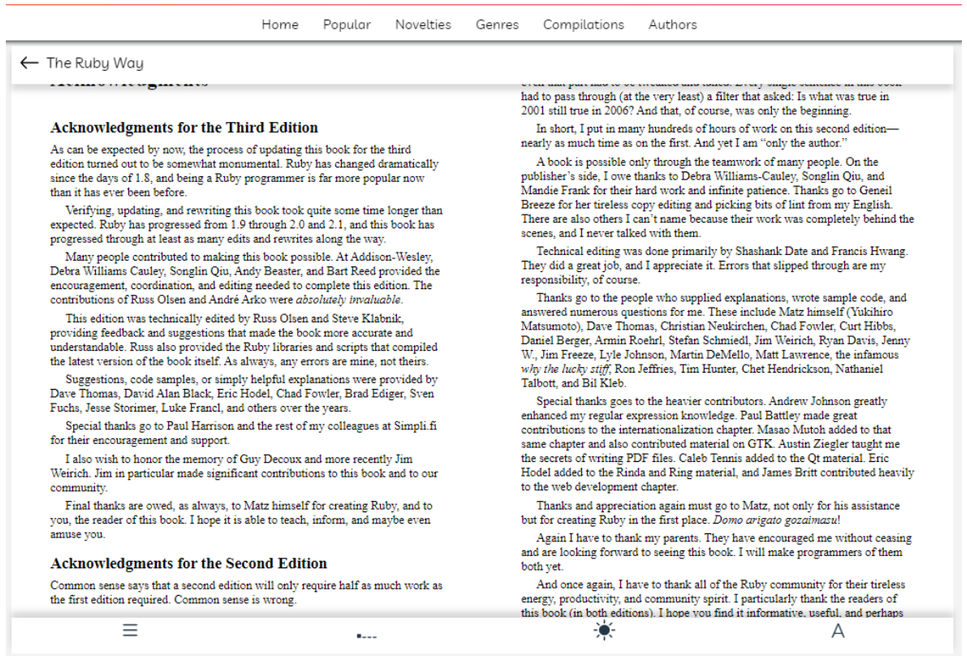
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-94





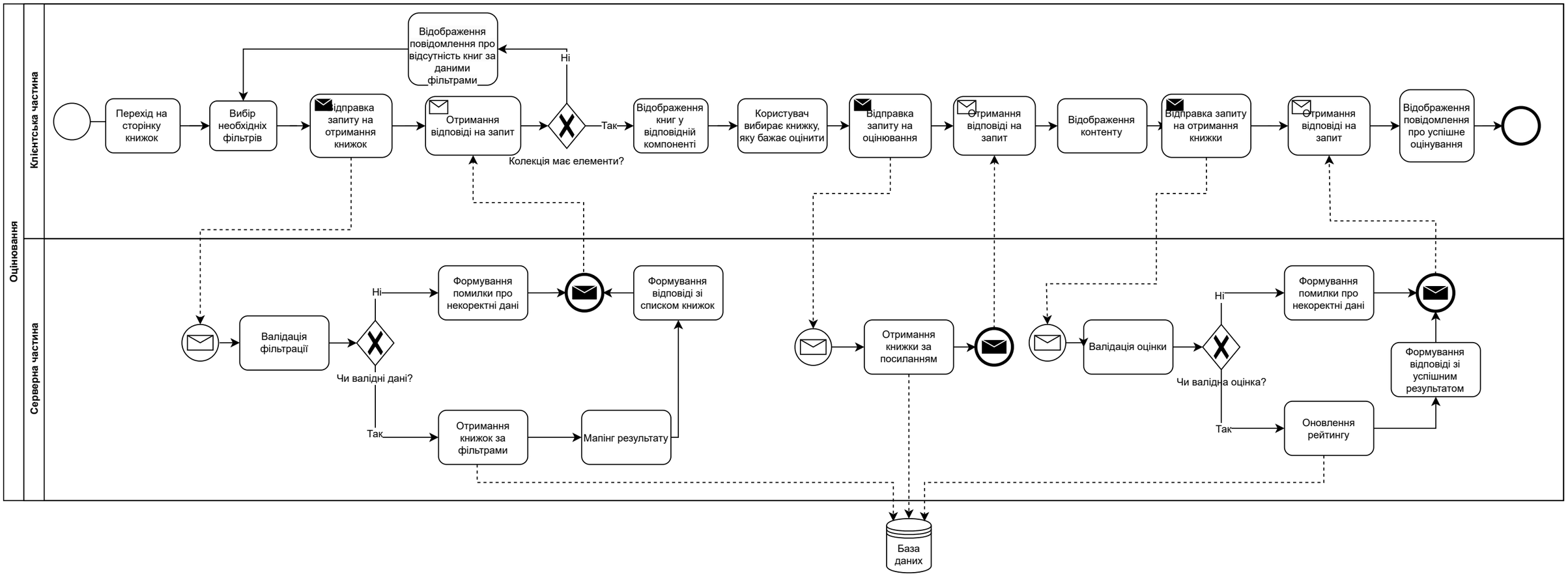






					КПІ.ІТ-9409.045440.06.99.КЕ	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	
Розробив		Кравченко Д.О.				Аркуш / Аркушів
Перевірив		Ліщук К.І.				
Т. кон.						
Н. кон.		Вітковська І.І.			Веб-застосунок для онлайн бібліотеки	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-94
Затвердив		Ліщук К.І.				

Схема бізнес процесу "Оцінювання книги"



Демонстраційний плакат до дипломного проекту

Веб-застосунок для онлайн бібліотеки

Виконав студент гр. ІТ-94 Кравченко Д.О.

Керівник Ліщук К.І.