НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія ”

на тему: *Веб-застосунок “Онлайн бібліотека”*

Виконав : студент __4__ курсу, групи _ІВ-91_
(шифр групи)

_____Дерачиц Віталій Віталійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Пустовіт О.М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) старший викладач, Виноградов Ю. М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Дерачица Віталія Віталійовича

1. Тема проєкту *Веб-застосунок “Онлайн бібліотека”*
керівник проєкту Пустовіт Олександр Михайлович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2023 року №1226-с

2. Термін здачі студентом закінченого проєкту *22 травня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області.

Розділ 2. Огляд технологій для розробки ойлайн бібліотеки.

Розділ 3. Деталі розробки системи.

Розділ 4. Огляд та тестування розробленого веб-застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, схема бази даних, схема роботи системи.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю. М.		

7. Дата видачі завдання «30» серпня 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2022-15.12.2022*	
2.	*Вивчення та аналіз завдання*	*15.12.2022-15.03.2023*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2023-25.03.2023*	
4.	*Розробка структур окремих підсистем*	*25.03.2023-5.04.2023*	
5.	*Програмна реалізація системи*	*05.04.2023-15.04.2023*	
6.	*Оформлення пояснювальної записки*	*15.04.2023-20.05.2023*	
7.	*Захист програмного продукту*	*25.05.2023*	
8.	*Передзахист*	*08.06.2023*	
9.	*Захист*	*22.06.2023*	

Студент-дипломник _____ Віталій ДЕРАЧИЦ
(підпис)

Керівник проєкту _____ Олександр ПУСТОВІТ
(підпис)

АНОТАЦІЯ

У даній роботі було детально розглянуто наявні онлайн бібліотеки та реалізовано новий веб-застосунок, який є кращою їм альтернативою. Застосунок розроблено відповідно сучасним тенденціям розробки. Реалізована онлайн бібліотека надаватиме користувачам зручний та візуально привабливий інтерфейс, можливість реєструватись і авторизовуватись для керування колекціями та перегляду статистики, а також функціонал пошуку, фільтрації та оцінювання книг.

Серверна частина проєкту розроблена на мові TypeScript з середовищем виконання Node.js та використанням прогресивного фреймворка NestJS, користувацький інтерфейс було реалізовано мовою JavaScript з використанням бібліотеки ReactJS. Було розгорнуто базу даних з використанням СУБД PostgreSQL.

Ключові слова: онлайн бібліотека, TypeScript, Node.js, NestJS, JavaScript, ReactJs, PostgreSQL.

ANNOTATION

This project for a Bachelor's Degree thoroughly examined existing online libraries and implemented a new web application that serves as a better alternative to them. The application was developed in line with modern trends. The implemented online library provides users with a convenient and visually appealing interface, the ability to register and authenticate for managing collections and viewing statistics, as well as functionality for searching, filtering, and rating books.

The server-side of the project was developed using TypeScript with the Node.js runtime environment and the progressive NestJS framework. The user interface was created with JavaScript using ReactJS library. And a database was deployed using the PostgreSQL Relational Database Management System.

Key words: online library, TypeScript, Node.js, NestJS, JavaScript, ReactJs, PostgreSQL.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Розроблена по-новому			
	А4	***ІАЛЦ.467200.002 ТЗ***	Веб-застосунок	4		
			“Онлайн бібліотека”			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Веб-застосунок	63		
			“Онлайн бібліотека”			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Веб-застосунок	1		
			“Онлайн бібліотека”			
			Структурна схема			
			системи (структурна схема)			
	А4	***ІАЛЦ.467200.005 Д2***	Веб-застосунок	1		
			“Онлайн бібліотека”			
			Схема бази даних			
			(функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Веб-застосунок	1		
			“Онлайн бібліотека”			
			Схема роботи системи			
			(принципова схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Веб-застосунок	41		
			“Онлайн бібліотека”			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	
Розроб		Дерачиц В.В.			*Веб-застосунок “Онлайн бібліотека”* Опис альбому
Перев		Волокита А.М.			
Н-контр.		Виноградов Ю.М.			
Утверд.					

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського ІВ-91

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок “Онлайн бібліотека”*»

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку веб-застосунку онлайн бібліотеки для перегляду, пошуку та читання книг.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом "Інформатики та обчислювальної техніки" кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка веб-застосунку онлайн бібліотеки зі зручним, привабливим інтерфейсом та корисною функціональністю.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Зрозумілий і привабливий інтерфейс.
- Надати можливість користувачам читати книги онлайн та завантажувати на локальний диск.

- Надати можливість користувача здійснювати пошук та фільтрування книжок за різними критеріями, наприклад, за автором, жанром чи назвою.
- Надати користувачам можливість реєстрації та авторизації в системі.
- Забезпечити можливість користувачам залишати свої оцінки книгам.
- Надати можливість користувачам створювати власні колекції та зберігати книги для подальшого перегляду та використання.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac, Linux.
- WebStorm.
- Node.js

5.3. Вимоги до апаратної частини

- ЦП з 2 або більше ядрами.
- ROM від 100 ГБ.
- RAM від 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2022-15.12.2022
Вивчення та аналіз завдання	15.12.2022-15.03.2023
Розробка архітектури та загальної структури системи	15.03.2023-25.03.2023
Розробка структур окремих частин системи	25.03.2023-05.04.2023
Програмна реалізація системи	05.04.2023-15.04.2023
Виправлення помилок	15.04.2023-20.05.2023
Оформлення пояснювальної записки	25.05.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб застосунок "Онлайн бібліотека"*»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СУБД	(Система управління базами даних)
HTTP	(HyperText Transfer Protocol) Протокол передачі даних
CLI	(Command-line interface) Інтерфейс командного рядка
ORM	(Object Relational Mapping) Об'єктно-реляційна проєкція
JSON	(JavaScript Object Notation) Запис об'єктів JavaScript
API	(Application programming interface) Прикладний програ́мний інтерфейс
HTTPS	(Hypertext Transfer Protocol Secure) Захищений протокол передачі гіпертексту
URL	(Uniform Resource Locator) Адреса ресурсу
JWT	(JSON Web Token) Токен доступу на основі JSON
DOM	(Document Object Model) Об'єктна модель документа

ВСТУП

У сучасному інформаційному суспільстві доступ до знань та літератури є ключовим фактором розвитку та саморозвитку. Завдяки швидкому розвитку технологій Інтернету та цифрової сфери, онлайн бібліотеки стали важливим інструментом для забезпечення широкого доступу до книжок та інших навчальних матеріалів.

Метою даної дипломної роботи є розробка та реалізація веб-застосунку для онлайн бібліотеки, який надасть користувачам зручний та ефективний спосіб пошуку, перегляду онлайн та можливість завантаження вмісту книжок в локальне сховище в зручному форматі.

У роботі буде використано сучасні технології розробки веб-додатків, таких як ReactJS для фронтенду та NestJS для бекенду, що дозволить забезпечити швидку та масштабовану роботу системи. Окрім цього, у роботі буде проведено дослідження сучасних підходів до створення онлайн бібліотек, аналіз існуючих рішень та побудова оптимальної архітектури бази даних.

Веб-застосунок онлайн бібліотеки, що буде розроблено буде включати функціональність для реєстрації та аутентифікації користувачів, пошуку та фільтрації книжок, перегляду деталей книги, оцінки користувачів, можливість зберігання обраного матеріалу у власній колекції, а також отримувати статистику по прочитаним книгам.

Важливим аспектом розробки буде забезпечення зручного та інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам максимально комфортно взаємодіяти з онлайн бібліотекою та отримувати задоволення від користування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасні технології та винаходи є невід'ємною частиною повсякденного життя людей, сприяючи постійному вдосконаленню засобів виробництва, предметів праці та ефективності виробництва. Вони надають нам чимало переваг, допомагають у різних аспектах роботи, починаючи від елементарних задач, таких як виконання простих розрахунків, завдяки калькулятору, закінчуючи складними "розумними" автомобілями, які самостійно керуються та взаємодіють з дорогами, світлофорами і навіть погодними умовами.

Однак, такий прогрес не став можливим одразу. Це було результатом багатовікової еволюції та вдосконалення простих речей, починаючи з часів, коли людство не мало жодного уявлення про те, що очікує на нього у майбутньому. З плином часу люди все більше усвідомлювали цінність знань, а також їх передачу майбутнім поколінням. Так народилася писемність, яка дозволяла переказувати мову за допомогою ієрогліфів, завдяки якій згодом з'явилися перші книги.

Це призвело до появи газет, рукописів, хронік, літописів, ну і звичайно ж книг. Але було необхідно місце, де ці цінності можна було б зберігати та надавати доступ до них іншим людям, які бажають розвитку. Так народилися бібліотеки - місця, де знання з усього світу зберігалися у належному стані, а кожен мав можливість користуватися ними.

Бібліотеки завжди були важливими установами, які зберігають, організовують та забезпечують доступ до різноманітної інформації та ресурсів для громади. Вони завжди були зручними, адже створювали систему організації та класифікації, щоб користувачі могли швидко знаходити необхідну інформацію, оскільки інформація в них часто посортована за жанрами, авторами, роками видання та іншими категоріями.

Багато людей в наш час не мають можливості відвідувати бібліотеки через ряд чинників: не підходящий графік роботи, відсутність часу на відвідування, не зручне розташування, а також такі риси характеру як сором'язливість та відсутність бажання комунікувати з іншими людьми. Всі ці недоліки вирішують онйлан бібліотеки.

Сьогодні з'явилися електронні онлайн бібліотеки, які надають доступ до інформації практично з будь-якого куточка світу. Електронна бібліотека - інформаційна система, що складається з впорядкованого фонду електронних ресурсів, каталогу для цього фонду та комплексу засобів, що підтримують функціонування пошукової системи і надає доступ до нього через мережу Інтернет [1]. Вам потрібен лише доступ до Інтернету та пристрій, що підключається до нього для доступу до інформації онлайн бібліотек.

За інформацією від Google, популярнісь запиту "online library" за останні два роки знаходить на значно вищій позначці ніж раніше. Це вказує на актуальність створення та вдосконалення онлайн бібліотек.

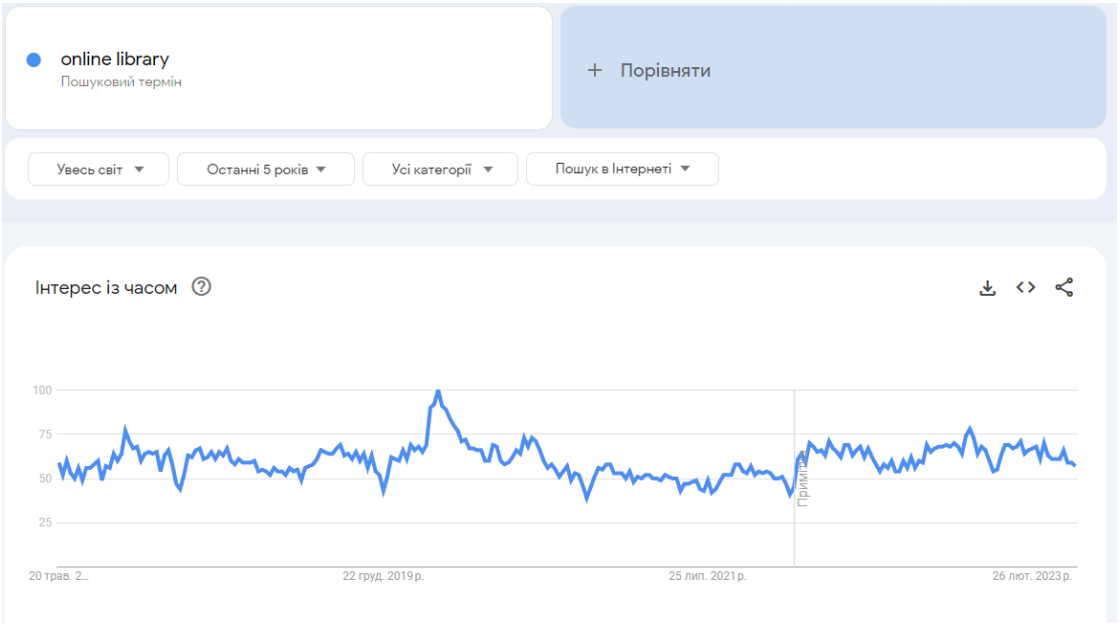


Рисунок 1.1 – Графік популярності пошукового запиту "online library"[2]

Для реалізації проекту онлайн бібліотеки необхідні наступні компоненти:

- Структурована база даних для зберігання та організації інформації про матеріали, які доступні в бібліотеці.
- Клієнтський інтерфейс, що забезпечуватиме взаємодію користувача з сервером та забезпечуватиме йому потрібні функції через веб-браузер.
- Сервер, для комунікації веб-інтерфейсу з базою даних для виконання необхідних дій.

Таким чином, бакалаврська робота спрямовуватиметься на проектування та реалізацію онлайн бібліотеки, для надання зручного веб-інтерфейсу, широкого вибору цифрових матеріалів, отримання структурованої інформації та її пошуку, інтерактивності для користувачів з усього світу.

1.2 Огляд існуючих рішень

Перші онлайн бібліотеки з'явилися у 1990-х роках і вони мають цікаву історію розвитку. Ось початкові кроки розвитку цифрових бібліотек:

- В 1971 році було запущено проект Gutenberg, котрий вважається одним з перших проектів електронних книг. Він дозволяв завантажувати книги з інтернету в форматі ASCII [3].
- Компанія Online Computer Library Center у 1971 створила свою першу онлайн систему каталогізації та пошуку, відому як Online Union Catalog і запровадила WorldCat, світову бібліографічну базу даних, яка зберігає записи про книги [4].
- Проект Digital Library Initiative 1994 року, що був запущений в США мав на меті створення цифрових бібліотек на основі новий технологій та метою зберігання великих обсягів інформації [5].
- У 2004 році компанія Google оголосила про проект Google Books з оцифрування мільйонів книг і створення електронних версій для

онлайн-доступу. Завдяки співпраці з багатьма бібліотеками Google отримав велику кількість літературних ресурсів.[6]

Всі ці розробки стали визначними для розвитку цифрових онлайн бібліотек і завдяки ним з часом тільки нарощували популярність. Також повпливали чинники такі як: більша кількість доступних матеріалів ні у звичайних бібліотеках, розширені функції та можливості (пошук за ключовими словами, ведення обліку прочитаних книг та інші), зручність та швидкість доступу, особливо за часів Covid-19, коли люди могли користуватись привілегіями бібліотек тільки з дому.

1.2.1 The Online Books Page

З першої ж миті перебування на сайті можна зробити висновок, що ним не зручно користуватись, а також орієнтуватись. Дизайн веб-інтерфейсу відсутній, а отже рішення не приваблює користувачів, це відлякує потенційних відвідувачів. Він представляє собою просту сторінку з впорядкованими посиланнями, котрі зливаються одне з одним, що ускладнує роботу з сайтом.

Незважаючи на те, що The Online Books Page надає доступ до багатьох електронних книг, він може бути обмежений порівняно з іншими бібліотеками або комерційними платформами. Це може обмежувати користувачів в їх виборі та доступі до певних книжок.

The Online Books Page може мати обмежені можливості пошуку і фільтрації, що ускладнює знаходження конкретних книг або навігацію серед великого обсягу доступних матеріалів. Це може викликати неефективне використання часу та ускладнити пошук потрібної літератури.

Так як на сайті немає авторизації, книги не мають рейтингу, відгуків, а отже й рекомендацій чи списків популярних книг. З цього слідує і проблема відсутності інтерактивності та соціальної взаємодії.

Важливим негативним фактором для користування є відсутність картинок палітурок і описів книг, що ускладнює вибір бажаного для прочитання, все на що можна орієнтуватись – назва. Тому якщо потрібно дізнатись більше інформації, потрібно обов'язково переходити на посилання конкретної книги. Деякі книги в хорошому стані, а інші є сканованими копіями, які важко читати.

Окрім мінусів можна виділити позитивні сторони сервісу, такі як:

- повністю безкоштовний та відкритий для усіх користувачів;
- не потребує стабільного якісного інтернет з'єднання, адже сторінки містять тільки текст, а будь-які елементи, що могли б сповілнювати його роботу відсутні.

Посилання: https://onlinebooks.library.upenn.edu/

Рисунок 1.2 – Головна сторінка The Online Books Page

1.2.2 OpenLibrary

Веб-інтерфейс OpenLibrary не має сучасного та привабливого дизайну. Він виглядає застарілим та наврядчи може залучати користувачів своїм зовнішнім виглядом. Не всі книги присутні на сайті можна завантажити на пристрій для читання його в зручний час за відсутності інтернет доступу. Використання стороннього сервісу InternetArchive для читання книг онлайн створює залежність від його статусу, за умов проблем з ним, користувачі будуть позбавлені можливостей перегляду вмісту книг.

Незважаючи на вказані вище недоліки, застосунок надає безкоштовний доступ до вражаючої колекції книг, яка включає як сучасні, так і класичні твори з різних жанрів та мов, з перевагами для зареєстрованих користувачів, котрі можуть додавати нові книги, вносити правки до наявних записів, а також ділитися власними відгуками та оцінками, що створює активну спільноту любителів книг, які діляться своїми знаннями та враженнями. Окрім цього, сервіс адаптований під девайси з різним розміром екрану (смартфони, планшети, ноутбуки, комп'ютери).

Також застосунок містить ряд особливо корисних функціональних рис: розширений пошук книжок за різними критеріями, пошук слів чи фраз всередині книг, а також можливість прослуховування частини книжок в аудіо форматі.

Посилання: https://openlibrary.org/

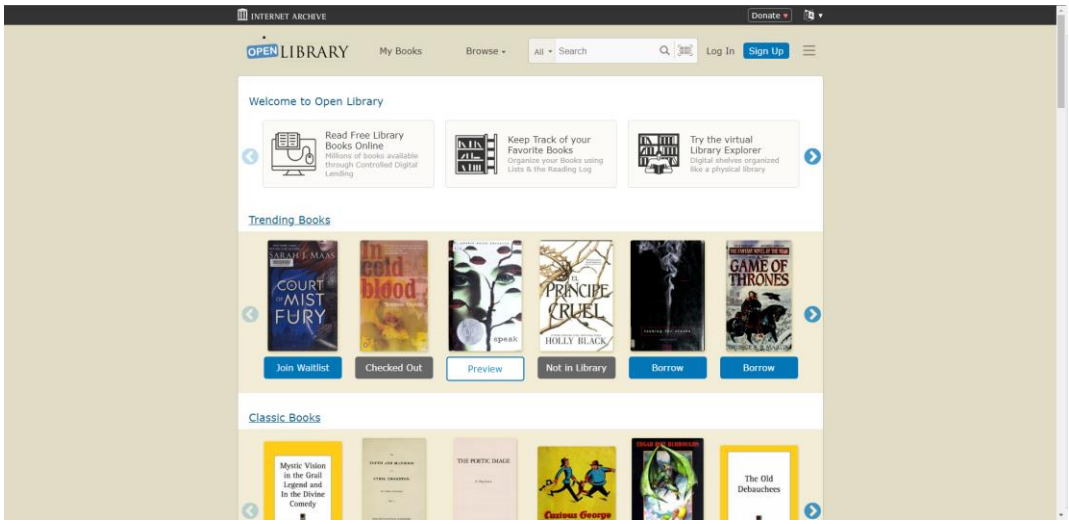


Рисунок 1.3 – Головна сторінка OpenLibrary

1.2.3 Project Gutenberg

Project Gutenberg було запущено у 1971 році, завдяки цьому за такий довгий період його база наповнилась більше ніж 70000 книг – одна з найбільших у світі колекцій безкоштовних електронних книг. Значною перевагою є те що всі ці книги доступні для безкоштовного завантаження в

різних форматах, таких як EPUB, MOBI, HTML, PDF та інші. Project Gutenberg пропонує книги в різних мовах, що дозволяє користувачам з різних країн отримувати доступ до літератури у своїй рідній мові. Це робить проект доступним та корисним для широкої аудиторії. А також застосунок базується на принципах відкритого джерела, що означає, що кожен може використовувати, змінювати та поширювати книги з цього ресурсу з відповідною ліцензією.

Наперекір позитивним рисам Project Gutenberg має обмежену візуальну привабливість, не озброєним оком видно, що над зовнішнім виглядом сайту не проводились та не проводяться ніякі роботи. Інтерфейс занадто простий і непропрацьований, що в свою чергу має негативний вплив на бажання його використання. Окрім цього його веб-інтерфейс не підтримує адаптивність для його використання за допомогою смартфона чи планшета.

Сучасні онлайн бібліотеки пропонують багато інтерактивних функцій для побудови живої спільноти книголюбів , які обмінюються своїми знаннями та враженнями, а даний застосунок нажаль не має таких переваг, так само як не дає змогу організовувати книги в колекції чи відслідковувати прочитані книги. Project Gutenberg не може суперничати з конкурентами у пошукових можливостях, адже не має розширених фільтрів, сортування або додаткових параметрів пошуку.

Також, способом перегляду вдалося визначити, що не всі книги мають високоякісні сканування або добре оформлені електронні версії, оскільки проект розробляється на основі внеску волонтерів.

Посилання: https://www.gutenberg.org/

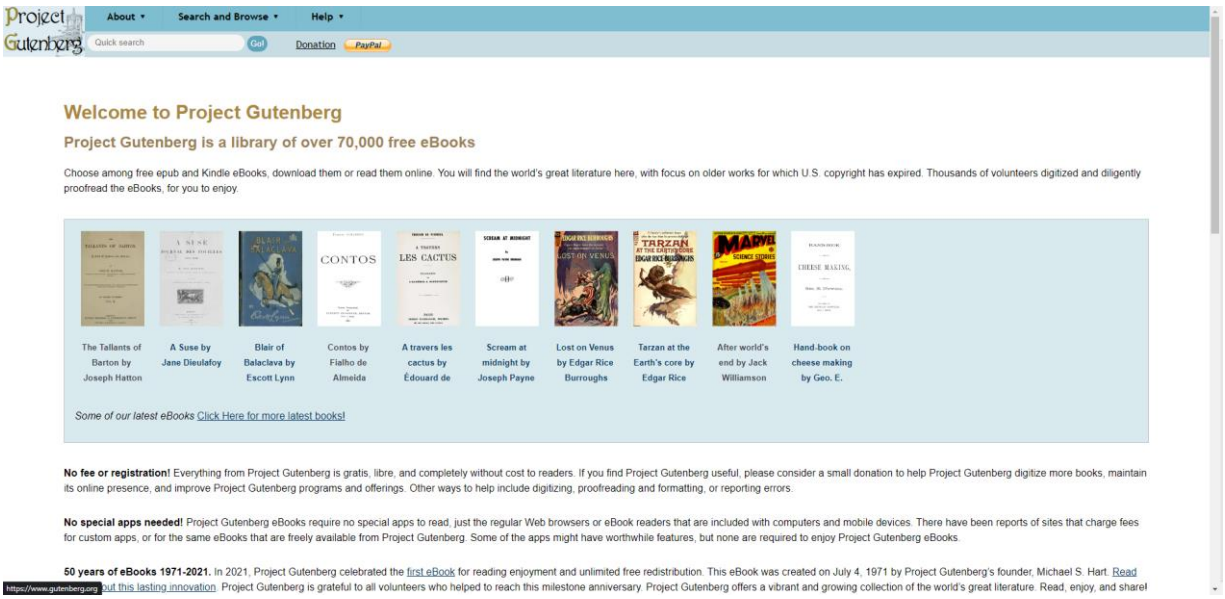


Рисунок 1.4 – Головна сторінка Project Gutenberg

1.2.4 Manybooks

Позитивні аспекти застосунку:

- Приємний на вигляд і простий у використанні інтерфейс: забезпечує інтуїтивно зрозумілу навігація по сайту і приваблює нових користувачів.
- Сторінка пошуку має значну кількість можливостей, зручна у використанні та надає широкий спектр фільтрів.
- Можливість читати всі книги онлайн і завантажувати на свої девайси в pdf форматі.
- Підтримка кількох мов, включаючи англійську, французьку, німецьку, іспанську та інші.
- Наявність авторизації та можливостей залишати відгуки, оцінювати і додавати книги до вибраного.
- Адаптивність для екранів будь-якого розміру.

Негативні аспекти:

- Головним недоліком є дуже велика кількість реклами. В деяких випадках рекламні банери займають більше ніж 50% наповнення сторінки, що ускладнює навігацію, отримання інформації і читання електронних книг.
- При виборі читання книг онлайн, сторінка з книгою може завантажуватися кілька хвилин або й не завантажитись взагалі.
- Сторінка особистого профілю не має функціоналу для перегляду прочитаних книг, роботи з користувацькими колекціями, ведення статистики, тощо.

Посилання: https://manybooks.net/

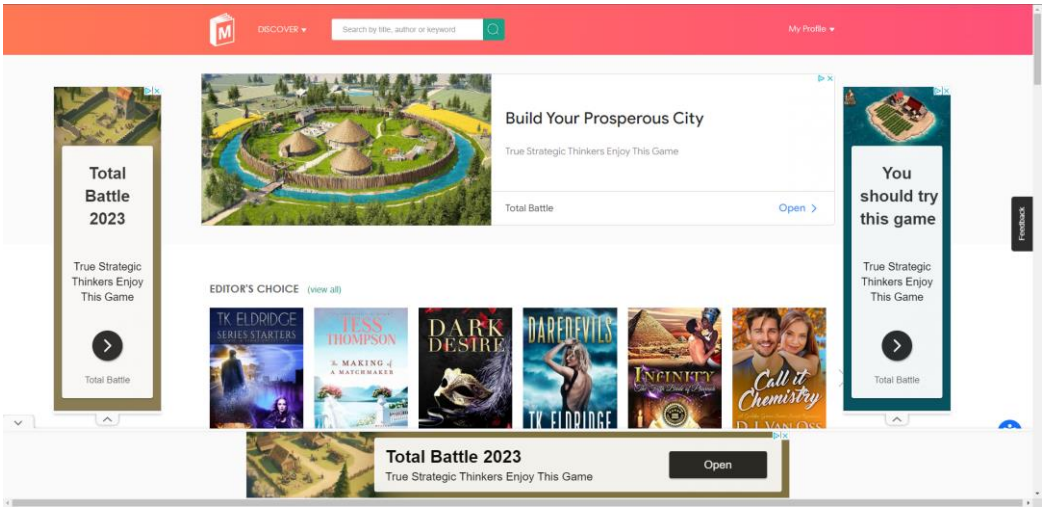


Рисунок 1.5 – Головна сторінка Manybooks

1.2.5 UABook

UABook — український сайт, який пропонує безкоштовний доступ до великої колекції книг українською мовою. Сайт був заснований у 2014 році групою українських ентузіастів, які хотіли зробити українську літературу доступнішою для суспільства. Він має велику колекцію книг, включаючи класичну літературу, документальну прозу і дитячі книги. Книги доступні в різних форматах, включаючи PDF, EPUB і MOBI. Окрім цього надає можливість читати онлайн на сайті.

Переваги використання UABook:

- книги UABook можна читати безкоштовно;
- усі книги UABook є суспільним надбанням, що означає, що їх можна вільно використовувати та поширювати;
- книги UABook доступні в широкому спектрі форматів (PDF, EPUB, TXT, FB2, RTF і MOBI), з яких користувач може обрати потрібний та читати офлайн;
- UABook – це спільнота українських ентузіастів, які захоплюються українською літературою. Вони постійно працюють над поповненням колекції новими книгами;
- можливість додавати книги до списку вибраного, щоб потім легко їх знаходити.

Недоліки UABook:

- Застосунок має обмежену пошукову систему, яка не надає додаткових можливостей пошуку та фільтрації окрім.
- Наявність великої кількості реклами, яка може дратувати.
- Застарілий вигляд веб-інтерфейсу.
- Відсутність картинок обкладинок книг та інших вкладень.
- Відсутність авторизації та персоналізації читацького середовища.

1.3 Порівняльний аналіз існуючих рішень

Розглянуті бібліотеки мають значну кількість корисних характеристик та надають доступ до великої кількості інформації. Кожна з них задовільняє подібні функціональні та візуальні потреби користувачів. Але всі вони переважають один одного якимись можливостями, а чимось поступаються. Ні одна з них не може повністю задовільнити усі сучасні побажання та вимоги потенційних клієнтів.

Таблиця 1.1 – Порівняння готових рішень

Застосунок	The Online Books Page	OpenLibrary	Project Gutenberg	Manybooks	UABook
Добре виконаний веб-інтерфейс	-	+	-	+	-
Наявність особистого кабінету	-	+	-	+	-
Швидкість роботи	+	-	+	-	+
Відсутність реклами	+	+	+	-	-
Надання великої кількісті книг	+	+	+	+	+/-
Адаптивність	-	+	-	+	+
Завантаження книг	-	+	-	-	+

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі було проаналізовано актуальність бібліотек у світовій спільноті, а з розвитком технологій та подій навколо і зріст популярності цифрових онлайн бібліотек. Проведене дослідження, що представлене в даному розділі, розглядає сутність онлайн бібліотек і вказує їх роль та значення, які вони виконують у сучасному світі, а також причини, чому вони вважаються кращими та які переваги мають над традиційними паперовими книгами. Також була опрацьована та описана інформація про історію виникнення самого концепту онлайн бібліотеки.

Було оглянуто та надано аналіз наявних рішень, як достатньо нових (“Manybooks”, “OpenLibrary”, “UABook”), так і тих, що працюють багато років (“Project Gutenberg”, “The Online Books Page”). Кожен з них було співставлено та описано відповідностям до сучасних вимог та зі сторони зручного користувацького використання. Всі вони мають свої переваги, але водночас поступаються чимось конкурентам або й взагалі не мають важливих аспектів. Було виділено найважливіші риси для аналізу, на основі яких було створено таблицю ключових факторів в якій порівняно розглянуті існуючі рішення.

З проаналізованих факторів та деталей цього розділу, можна зробити висновок, що люди потребують онлайн бібліотеки, але існуючі рішення не завжди можуть вдовольнити їх бажання. Тому створення веб-застосунку онлайн бібліотеки є актуальним. Вирішено зробити наголос на зручний та красивий веб-інтерфейс, надання розгорнутого пошуку та фільтрів, можливість авторизації, соціальної взаємодії та стабільного загального функціонування застосунку. При виборі технологій розробки всі вище згадані аспекти буде враховано.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ОНЛАЙН БІБЛІОТЕКИ

2.1 Архітектура

Архітектура веб-застосунків - це структура та організація компонентів, що використовуються для реалізації функціональності веб-застосунків чи систем. Вона визначає, які компоненти взаємодіють між собою, як передаються дані та як забезпечується функціональність та доступ до ресурсів.

Базова типова архітектура веб-застосунок включає наступні компоненти:

- Клієнтська частина (Client): Це інтерфейс, через який користувач взаємодіє з веб-сервісом. Клієнт може бути реалізований у вигляді веб-браузера, мобільного додатка або іншого програмного забезпечення.
- Серверна частина (Server): Це програмний компонент, який надає функціональність веб-застосунок. Він обробляє запити від клієнтів, взаємодіє з базою даних та іншими ресурсами, та повертає результати клієнту.
- Бази даних (Databases): Веб-застосунки можуть використовувати бази даних для зберігання та управління даними. Бази даних забезпечують постійне зберігання і доступ до інформації, необхідної для роботи веб-застосунку.
- Безпека (Security): Веб-застосунки повинні забезпечувати захист даних та перешкоджати несанкціонованому доступу. Це може включати шифрування передачі даних, аутентифікацію користувачів, контроль доступу та інші заходи безпеки.

Архітектура дозволяє організувати та координувати роботу різних компонентів, забезпечувати функціональність та надійність веб-застосунку, а

також забезпечувати ефективну взаємодію з користувачем. Наряду з цими компонентами, вона може включати й інші складові в залежності від конкретного проекту та потреб користувачів.

Для майбутнього проекту було обрано один з архітектурних патернів – клієнт-серверна архітектура, одна з найпоширеніших та базових архітектур для веб-застосунків. Клієнт-серверні програми можна створювати незалежно від платформи чи стеку технологій. У цій архітектурі веб-додаток розподіляється на дві основні складові: клієнтську частину і серверну частину. Клієнт та сервер взаємодіють між собою за допомогою мережевих протоколів, зазвичай це HTTP [7].

Основні компоненти клієнт-серверної архітектури включають:

- Клієнт: Це програмне забезпечення або пристрій, що надає інтерфейс користувача для взаємодії з веб-додатком. Клієнт може бути веб-браузером, мобільним додатком або іншою програмою. Він надсилає запити серверу та отримує відповіді, які можуть включати веб-сторінки, дані або іншу інформацію.
- Сервер: Це програмне забезпечення або фізичний комп'ютер, який обробляє запити клієнтів і надає їм відповіді. Сервер зберігає дані, обробляє логіку додатка та надає необхідну функціональність клієнту.
- Комунікація: Клієнт та сервер взаємодіють за допомогою мережевих протоколів, таких як HTTP. Клієнт надсилає запити на сервер, які містять інформацію про те, що він хоче отримати або виконати. Сервер обробляє ці запити та надсилає назад відповіді, які можуть містити веб-сторінки, дані або повідомлення.
- Розподілені обчислення: Клієнт-серверна архітектура дозволяє розподілити завдання між клієнтами та серверами. Клієнт відповідає за відображення інтерфейсу користувача та деякі

обчислення на локальному пристрої, тоді як сервер забезпечує бізнес-логіку, зберігання даних та інші обчислення.

- Масштабованість: Клієнт-серверна архітектура дозволяє масштабувати систему шляхом додавання додаткових серверів або розподілених компонентів. Це дозволяє обробляти більше запитів та забезпечувати високу доступність додатка.

Плюси клієнт-серверної архітектури:

- Розділення обов'язків між клієнтом та сервером дозволяє покращити модульність та масштабування додатка.
- Дозволяє розробляти різні клієнти для різних пристроїв, що дозволяє забезпечити гнучкість та доступність.
- Забезпечує більшу безпеку, оскільки сервер може контролювати доступ до ресурсів і обмежити можливості клієнта.
- Клієнти отримують доступ до даних із сервера за допомогою авторизованого доступу, що покращує обмін даними.

Мінуси клієнт-серверної архітектури:

- Основним недоліком мережі клієнт-сервер є перевантаженість трафіком, який вона зазнає. Велика кількість запитів від користувачів призводить до збоїв або уповільнення з’єднання. Перевантажений сервер створює багато проблем із доступом до інформації.
- Залежність від мережевого з'єднання, оскільки без доступу до сервера клієнт не може взаємодіяти з додатком.
- Синхронний характер взаємодії, що означає, що клієнт очікує відповіді від сервера перед тим, як продовжити роботу.

Підсумувавши, клієнт-серверна архітектура є ефективним рішенням для багатьох веб-застосунків, які потребують розподіленого обчислення, керування даними та взаємодії з багатьма користувачами. Правильне використання та

налагодження цієї архітектури дозволяє побудувати потужні, масштабовані та безпечні веб-застосунки.

2.2 Технології для серверної та клієнтської частин

Як для серверної, так і для клієнтської частини в основі використане середовище виконання Node.js. Node.js – це середовище виконання JavaScript, яке використовується для розробки веб-застосунків. Воно дозволяє використовувати JavaScript як на серверній, так і на клієнтській стороні, що спрощує розробку повностекових веб-додатків. Node.js має відкритий вихідний код і є кросплатформним і використовується для створення широкого спектру веб-застосунків [8].

Однією з основних переваг Node.js є його використання однієї мови програмування (JavaScript або її типізований супер-набір - TypeScript) як на фронтенді, так і на бекенді, що спрощує комунікацію між клієнтом та сервером і дозволяє використовувати загальний код і модулі на обох сторонах.

Node.js також славиться своєю швидкодією та масштабованістю. Воно використовує однопотокову архітектуру, де кожен запит обробляється асинхронно, що дозволяє ефективно використовувати ресурси сервера та обробляти багато запитів одночасно.

Отже, Node.js є потужним інструментом для розробки веб-застосунків, який дозволяє розробникам будувати швидкі, ефективні та масштабовані додатки з використанням однієї мови програмування на всіх рівнях стеку.

2.2.1 Серверна частина

Для реалізації серверної частини застосунку було обрано прогресивний бекенд фреймворк NestJS. Це back-end середовище розробки Node.js, побудоване на Express, що використовує можливості TypeScript. NestJS

користується неймовірною популярністю та надійністю JavaScript як мови та Node.js як технології. Він натхненний спільними бібліотеками та фреймворками, такими як Angular, React та Vue, які покращують продуктивність та досвід розробників.

Навіть з урахуванням кількості чудових бібліотек, помічників та інструментів, які існують для серверного Node.js, жодна з них не вирішує ефективно головну проблему – архітектуру програми.

NestJS надає готову архітектуру додатків, яка дозволяє розробникам та командам створювати тестовані, масштабовані, слабозв'язані та легкі в обслуговуванні додатки.

Основні характеристики та можливості NestJS:

- Використовує TypeScript - строго типізована мова, яка є супер-набором JavaScript;
- Має модульну архітектуру, яка дозволяє організовувати код в окремі логічні блоки. Це сприяє зручному розподілу завдань і полегшує розширення функціональності. Фреймворк використовує підхід контролерів та провайдерів, які схожі на підхід, використаний в Angular. Контролери відповідають за обробку HTTP-запитів, а провайдери надають необхідні сервіси.
- Надає потужний механізм інжектування залежностей (Dependency injection), який полегшує роботу з залежностями і тестування коду. Крім того має вбудовану підтримку WebSocket, що дозволяє легко реалізувати багаторівневі веб-додатки, які працюють в режимі реального часу.
- Має підтримку middleware, завдяки чому дає можливість обробляти HTTP-запити перед тим, як вони потрапляють до контролера. Це дозволяє реалізувати різноманітну функціональність, таку як автентифікація або логування.

- Надає розширену систему фільтрів та перехоплювачів виключень, а тобто дозволяє елегатно обробляти помилки і додавати кастомні фільтри для обробки виключень.

Всі ці можливості, а в додаток деталізована документація роблять NestJS потужним і гнучким фреймворком для розробки веб-застосунків з якісним кодом і легко масштабувати проекти [9].

2.2.2 Клієнтська частина

Для розробки клієнтської частини застосунку було розглянуто два варіанти: JavaScript бібліотека React і JavaScript-фреймворк Vue.js.

React - це бібліотека, розроблена компанією Facebook. Вона базується на концепції компонентів і використовує віртуальний DOM (Document Object Model), що дозволяє ефективно маніпулювати і оновлювати відображення на стороні клієнта. React є декларативним, що означає, що програміст описує, як має виглядати інтерфейс, і React забезпечує оновлення відображення при зміні стану. Він має широку спільноту розробників і багато готових компонентів та бібліотек [10].

Vue - це прогресивний фреймворк JavaScript для розробки користувацького інтерфейсу. Він пропонує інкрементальний підхід до розробки, де розробники можуть поступово впроваджувати його у свої проекти. Vue має простий синтаксис і легко вивчається. Він також надає можливості для організації компонентів, маршрутизації, керування станом за допомогою вбудованої системи управління станом і багато інших функцій [11].

React і Vue схожі в плані продуктивності, обидва підходять для створення проектів різних масштабів. Потрібно відзначити, що екосистема React набагато більша, ніж у Vue, що цілком може сприяти успішності та швидкості розробки. Якщо для React-додатку потрібен якийсь важливий функціонал, великі шанси на те, що щось подібне вже реалізовано. Насправді, ймовірно те, що для

багатьох проблем у сфері React можна знайти навіть кілька варіантів рішень. Також можливість багаторазового використання компонентів підвищує масштабованість програм на React, що є вагомою перевагою для його вибору.

React базується на компонентній архітектурі, що сприяє поділу інтерфейсу на малий, перевикористовуваний та легко керований код. Це дозволяє швидко розробляти нові функції, покращувати і підтримувати код, а також сприяє спільній роботі в команді розробників. Також він пропонує однозв'язковий потік даних (unidirectional data flow) та підхід до управління станом за допомогою компонентів, що надає можливість дієво керувати станом додатка та забезпечує простоту відслідковування та оновлення даних. Використання Virtual DOM дозволяє оновлювати тільки змінені елементи інтерфейсу, що зменшує навантаження на браузер та поліпшує продуктивність додатка.

Ці аргументи демонструють, що використання React для розробки клієнтської частини застосунку має серйозне підґрунтя з точки зору керування станом, використання компонентної архітектури, активної спільноти розробників, забезпечення продуктивності, можливості розширення та підтримки кросс-платформових рішень.

2.4 Система управління базами даних

Вибір СУБД для веб-застосунку було зроблено на користь PostgreSQL, також відома як Postgres, котра позиціонує себе як «найдосконалішу реляційну базу даних з відкритим кодом у світі» [12]. Він був створений, щоб бути багатофункціональним, розширюваним і відповідати стандартам. У минулому продуктивність Postgres була більш збалансованою – читання, як правило, було повільніше, ніж MySQL, але він був здатний записувати великі обсяги даних ефективніше, і він краще обробляв паралельність.

Відмінності в продуктивності між MySQL і Postgres були значною мірою стерті в останніх версіях. Вибираючи між MySQL і PostgreSQL, продуктивність не повинна бути фактором для більшості стандартних додатків – вона буде достатньо хорошою в обох випадках, навіть якщо врахувати очікуване зростання в майбутньому. Обидві платформи ідеально здатні до реплікації, і багато постачальників хмарних послуг пропонують керовані масштабовані версії будь-якої бази даних. Тому варто розглянути інші переваги Postgres перед MySQL.

Postgres – це об’єктно-реляційна база даних, а MySQL – чисто реляційна база даних. Це означає, що Postgres включає такі функції, як успадкування таблиць і перевантаження функцій. Окрім цього Postgres дуже розширюваний. Він підтримує ряд розширених типів даних, недоступних в MySQL (JSONB, який можна індексувати, власний UUID, часові позначки з урахуванням часового поясу). Є можливість додати власні типи даних, оператори та типи індексів. А також, Postgres більш точно дотримується стандартів SQL. Він відомий захистом цілісності даних на рівні транзакції, що робить його менш вразливим до пошкодження даних.

Описані вище переваги обґрунтовують вибір саме PostgreSQL для реалізації веб-застосунку.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі детально описано архітектуру і технології, обрані для розробки майбутнього веб-застосунку. Всі рішення щодо вибору обґрунтовано і їх актуальність доведено. Також вказано суть та призначення даних технології у проекті та які переваги вони принесуть з собою при їх застосуванні.

За результатами проведених досліджень та порівнять стек технологій для реалізації веб-застосунку онлайн бібліотеки буде включати такі елементи:

- клієнт-серверна архітектура;
- об’єктно-реляційна база даних PostgreSQL;
- прогресивний бекенд фреймворк NestJS, що працює на NodeJS з підтримкою TypeScript;
- JavaScript бібліотека для створення інтерфейсів користувача – React;

Відбір цих технологій допоможе нам забезпечити швидку та ефективну розробку, забезпечити стабільну та масштабовану архітектуру і надати користувачам надійний та зручний веб-застосунок.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Розробка та проектування бази даних

Першим етапом проектування бузи даних є аналіз вимог, а саме: виділення інформації, потрібної веб-застосунку, сутності для відстеження, атрибути, наявні у сутностях та які зв'язки існують між ними. Це допоможе зрозуміти, які таблиці потрібно створити та які поля ці таблиці мають мати.

Для застосунку онлайн бібліотеки можна виокремити дві важливі та основоположні групи, а саме: користувачі та книги. Перша група буде містити інформацію про зареєстрованих відвідувачів, їх вподобання, цілі, дружні зв’язки та колекції. Друга в свою чергу буде складатися з обліку книг наявних у застосунку, їх детальної інформації, жанрів, авторів, рейтингів та файлів для читання онлайн та завантаження.

Сутність користувачів буде включати поля:

- id – унікальний ідентифікатор
- username – псевдонім вигаданий користувачем, що буде відображатись в особистому кабінеті та для інших користувачів
- email – поштова скринька, закріплена за користувачем
- password – пароль для входу в особистий кабінет (вказується при реєстрації)
- createdAt – дата реєстрації акаунту
- emailConfirmationCode – код, що буде згенеровано при реєстрації та відправлено для підтвердження на електронну скриньку
- isEmailConfirmed – поле, що вказує на те, що користувач підтвердив власність електронної скриньки

Поля сутності книг:

- id – унікальний ідентиікатор

- title – назва
- description – детальний або короткий опис
- pagesNumber – кількість сторінок
- author – автор
- genre – жанр
- attachment – файл для читання і завантаження

Наступним етапом є фізичне проектування – створення фізичної моделі бази даних. На цьому етапі визначаються конкретні деталі реалізації бази даних, такі як структура таблиць, типи даних, індекси, обмеження та інші параметри. Основні складові фізичної моделі бази даних включають наступне:

- таблиці – набори елементів даних, що містять назву та стовпці. Таблиці здатні утворювати зв’язки
- стовпці – визначаються їх іменем, типом даних та можливими обмеженнями. Типи даних вказують на природу даних, що зберігаються в стовпці, такі як цілі числа, рядки, дата і час тощо.
- первинні ключі – ідентифікатори унікальних записів в таблиці і забезпечення унікальності та ідентифікації кожного рядка даних
- вторинні ключі - зовнішні ключі, що визначають зв'язки між таблицями (вказівники залежностей між даними). Використовуються для створення зв'язків між різними таблицями та підтримки цілісності даних.

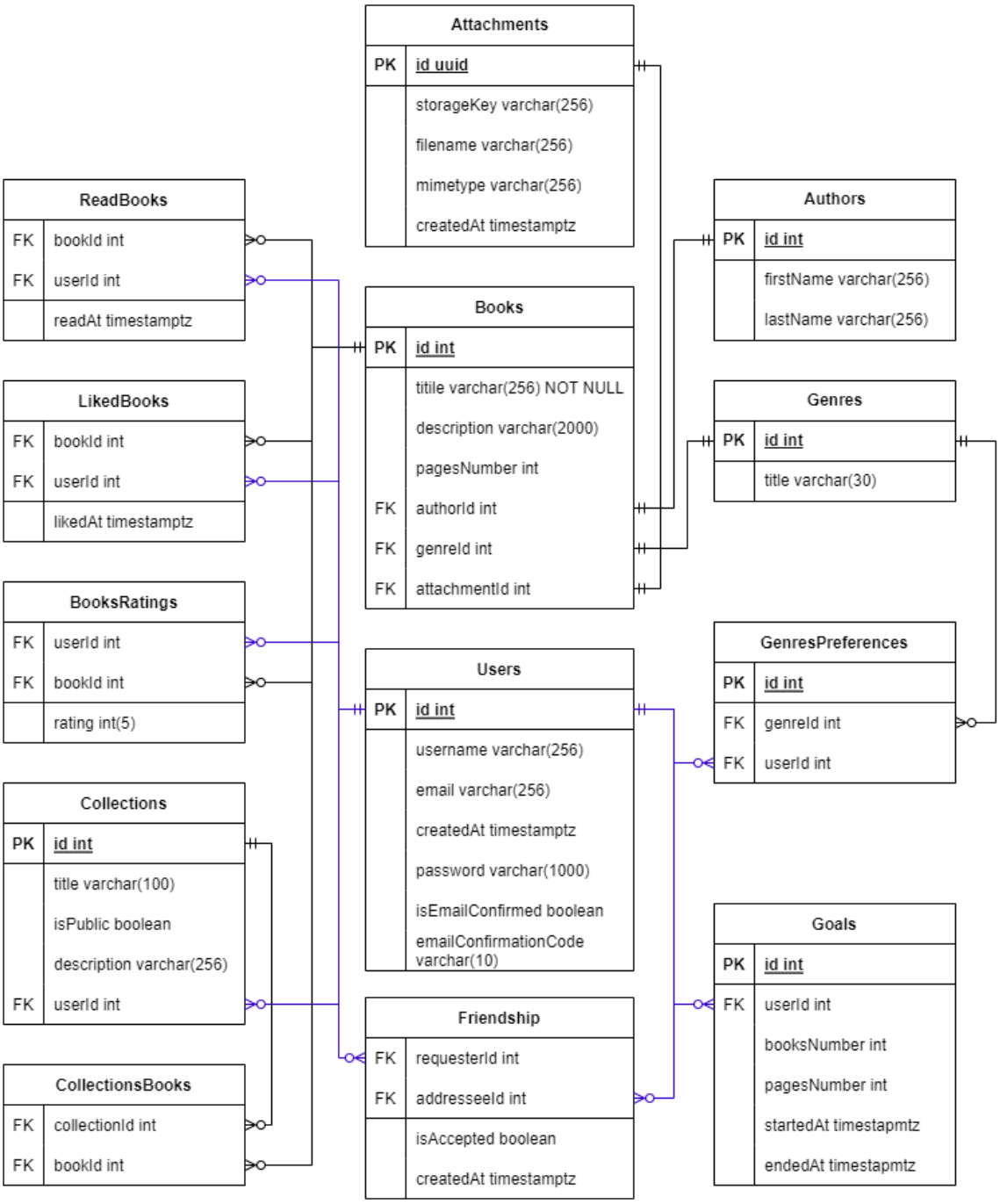


Рисунок 3.1 – Entity-relationship схема бази даних

При побудові таблиць було дотримано правил, для забезпечення ефективної організації та оптимальної продуктивності бази даних. Основні з них:

- Кожна таблиця має представляти лише одну сутність або концепцію. Це допомагає уникнути дублювання даних та забезпечує цілісність бази даних.
- Кожна таблиця повинна мати унікальний ідентифікатор, який використовується для однозначної ідентифікації записів в таблиці (зазвичай досягається за допомогою первинного ключа).
- Нормалізація даних – принцип, що допомагає уникнути дублювання даних та забезпечує ефективну організацію бази даних. Вона використовується для розбиття таблиць на менші, пов'язані між собою, щоб забезпечити ефективну, зручну роботу з даними та уникнути проблем.
- Використання зовнішніх ключів для встановлення зв'язків між таблицями. Це допомагає забезпечити цілісність даних та дозволяє виконувати запити, які об'єднують дані з різних таблиць.
- Визначення та дотримання типів даних стовпців таблиць. Це допомагає забезпечити правильне збереження, обробку та роботу з даними.
- Встановлення зрозумілих та описових назв для таблиць та стовпців, щоб полегшити розуміння структури бази даних. Уникнення використання скорочених, загадкових назв.

Таблиця 3.1 – Опис таблиці авторів книг Authors

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
id	int (ціле число)	Так	унікальний ідентифікатор (первинний ключ)

Продовження таблиці 3.1

firstName	varchar(256)	Так	Ім’я автора
lastName	varchar(256)	Так	Прізвище автора

Таблиця 3.2 – Опис таблиці жанрів книг Genres

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
id	int (ціле число)	Так	унікальний ідентифікатор (первинний ключ)
title	varchar(256)	Так	Назва жанру

Таблиця 3.3 – Опис таблиці вкладень Attachments

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
id	uuid	Так	унікальний ідентифікатор
storageKey	varchar(256)	Так	унікальна назва файлу на сервері
filename	varchar(256)	Так	назва файлу для відображення
mimetype	varchar(256)	Так	код ідентифікації формату файлів або тип контенту, переданих мережею Інтернет
createdAt	timestamptz	так	дата створення файлу з типом (завантаження його в систему).

Таблиця 3.4 – Опис таблиці користувацьких колекцій книг Collections

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
id	int	Так	унікальний ідентифікатор
title	varchar(100)	Так	title
isPublic	boolean	Так	вказує на те чи доступна ця колекція для перегляду іншим користувачам (true – доступна, false – недоступна)
description	varchar(256)	Ні	короткий опис колекції
userId	int	Так	вторинний ключ – посилання на таблицю користувачів, позначає власника колекції

Таблиця 3.5 – Опис таблиці користувацьких цілей по читанню книг Goals

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
id	int	Так	унікальний ідентифікатор
booksNumber	int	Ні	число книг до прочитання
pagesNumber	int	Ні	число сторінок до прочитання
startedAt	timestamptz	Так	дата початку виконання цілі
endedAt	timestamptz	так	дата завершення цілі

Таблиця 3.6 – Опис таблиці дружніх взаємозв’язків Friendship

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
requesterId	int	Так	ідентифікатор користувача, який надіслав запит на дружбу (вторинний ключ)
addresseeId	int	Так	ідентифікатор користувача, який отримає запит і може його прийняти (вторинний ключ)
isAccepted	boolean	Так	несе інформацію про те чи прийнятий запит
createdAt	timestamptz	Так	дата створення запиту дружби

Поля requesterId і addresseeId є вторинним ключами і посилаються на таблицю користувачів і в той же час разом (парою) становлять первинний ключ.

Таблиця 3.7 – Опис таблиці прочитаних книг ReadBooks

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
bookId	int	Так	Ідентифікатор прочитаної книги (вторинний ключ)
userId	int	Так	ідентифікатор користувача, що прочитав книгу (вторинний ключ)
readAt	timestamptz	Так	дата прочитання книги

Поля bookId і userId є вторинним ключами і в той же час разом (парою) становлять первинний ключ.

Таблиця 3.8 – Опис таблиці вподобаних книг LikedBooks

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
bookId	int	Так	Ідентифікатор прочитаної книги (вторинний ключ)
userId	int	Так	ідентифікатор користувача, що вподобав книгу (вторинний ключ)
likedAt	timestamptz	Так	дата вподобання

Поля bookId і userId є вторинним ключами і в той же час разом (парою) становлять первинний ключ.

Таблиця 3.9 – Опис таблиці користувацьких рейтингів книг BooksRatings

Назва поля	Тип даних	Обов’язкове до заповнення	Опис
bookId	int	Так	Ідентифікатор прочитаної книги (вторинний ключ)
userId	int	Так	ідентифікатор користувача, що встановив рейтингову оцінку книги (вторинний ключ)
rating	int(5)	Так	Рейтингова оцінка від 1 до 5

Поля bookId і userId є вторинним ключами і в той же час разом (парою) становлять первинний ключ.

Інші таблиці є допоміжними та несуть якусь додаткову інформацію для основних або встановлюють зв’язки між головними сутностями. Таблиця CollectionsBooks є зв’язною для колекцій та книг. Таблиця GenresPreferences встановлює зв’язок між користувачем жанрами і вказує на вподобання жанрів користувачами.

Опис сутностей завершено. Наступною задачею є створення та налаштування серверної частини веб-застосунку.

3.2 Розробка серверної частини

Так як для серверної частини було обрано платформу Node.js і фреймворк NestJS першим кроком є встановлення NestJS CLI – інструмент інтерфейсу командного рядка, для розробки, ініціалізації та підтримування застосунків на Nest. Він включає створення базової структури проекту, роботи з ним в режимі розробки, а також збірки програми для розгортання на продакшн сервері. Встановлення виконується командою npm install -g @nestjs/cli.

Наступним кроком є створення проекту за допомогою CLI командою nest new project-name, де project-name – library в моєму випадку. Після виконання команди NestJS CLI створює новий каталог з базовою структурою проекту.

Перейшовши в робочий каталог проекту його можна запустити в режимі розробки та в режимі реального часу переглядати результати.

Для написання коду було обрано інтегроване середовище розробки (IDE), розробленою компанією JetBrains – WebStorm. Воно спеціалізується на розробці проектів з використанням мов програмування JavaScript, TypeScript, HTML та CSS. WebStorm надає потужні можливості редагування коду з підсвічуванням синтаксису, автодоповненням коду, перевіркою правильності коду та багатьма іншими корисними функціями, що значно пришвидшує та полегшує розробку. Він має потужні можливості для розробки на мові TypeScript, надає інтелектуальні функції, такі як рефакторинг, відлагодження,

підказки типів та багато іншого, що є вагомою причиною для вибору. В цілому, WebStorm - це потужне інструмент для розробки веб-додатків, який пропонує багато корисних функцій та спрощує роботу розробника [13].

Для розорбки було використано Sequelize - це об'єктно-реляційний мапер (ORM) для мови програмування JavaScript, який дозволяє розробникам легко взаємодіяти з базами даних, використовуючи об'єктно-орієнтований підхід. Він підтримує різні системи управління базами даних, включаючи PostgreSQL. Sequelize дозволяє визначати моделі, що представляють таблиці бази даних, використовуючи класи або схеми, встановлювати зв'язки між моделями, такі як один до одного, один до багатьох або багато до багатьох. Крім того, надає можливість використовувати міграції для автоматичного створення та оновлення схеми бази даних. Отже, мною були створені міграції з використанням Sequelize для спрощення роботи з ними та забезбечення зручного контролю версій [14].

Було створено дві міграції: create-users – для створення таблиці користувачів, похідних та пов’язаних з нею таблиць і create-books – для створення таблиць авторів, жанрів, файлів, книг і тд (таблиці пов’язані з сутністю книги). З використанням Sequelize CLI – ще одного корисного інструменту командного рядка було запущено міграції до бази, де створились всі таблиці описані в міграціях, а також таблиці обліку версійності міграцій та версійності сідерів (використання сідерів дозволяє автоматизувати процес створення початкових записів у таблицях бази даних).

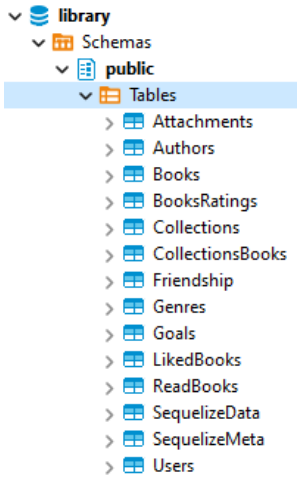


Рисунок 3.2 – Створені міграціями таблиці в базі даних

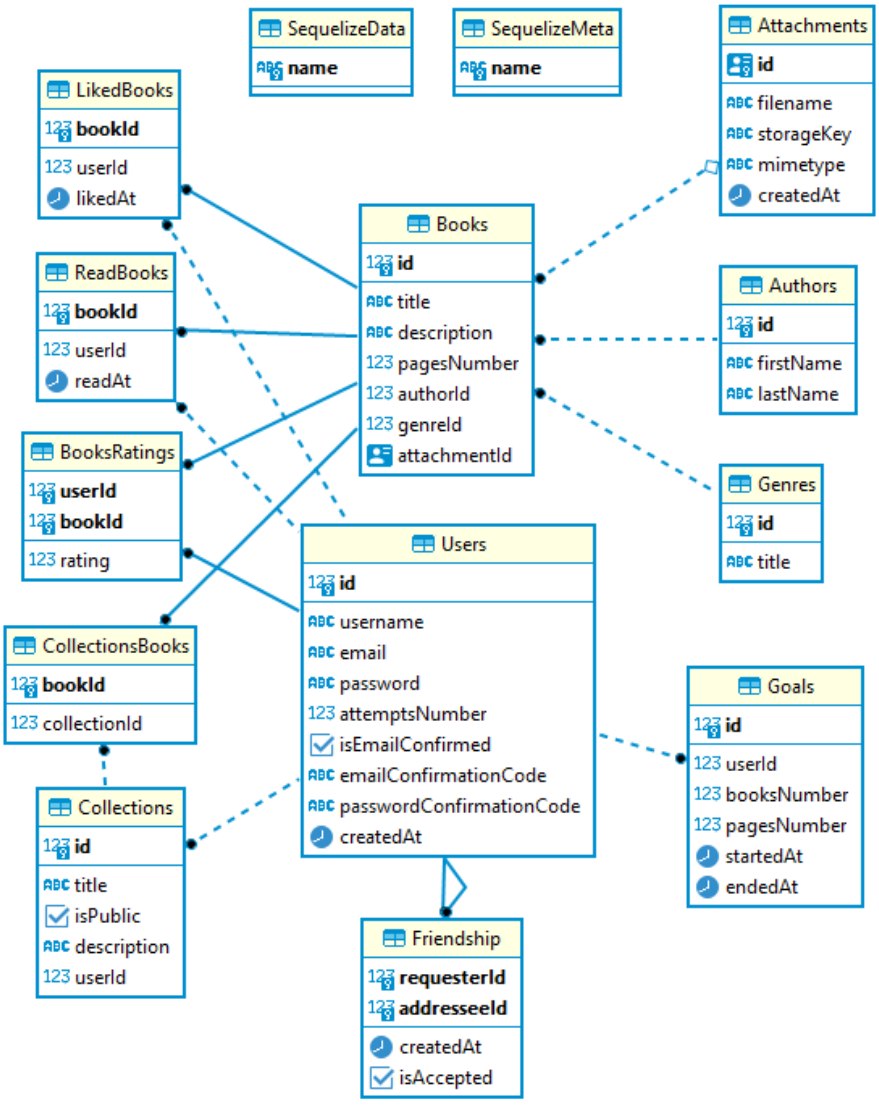


Рисунок 3.3 – Діаграма таблиць створених міграціями

Наступним кроком було підключено служба конфігурації NestJS – вбудований модуль, наданий NestJS, який полегшує керування та отримання значень конфігурації програми. Він надає можливість відокремити логіку конфігурації від решти коду програми та забезпечує централізоване місце для зберігання та доступу до змінних конфігурації. У Node.js програмах часто використовуються файли з розширенням .env, в яких зберігаються пари ключ-значення для представлення різних середовищ. Цей підхід дозволяє легко змінювати конфігурацію програми для різних середовищ шляхом простої заміни відповідного .env файлу. Але в NestJS цей підхід можна реалізувати за допомогою ConfigModule, який надає ConfigService для завантаження значень з відповідного .env файлу, що я і зробив.

А також було конфігуровано Swagger для документації та тестування API.

```
const app = await NestFactory.create(AppModule);

const configService = app.get(ConfigService);

app.useGlobalPipes(new ValidationPipe( options: { whitelist: true, transform: true }));
app.setGlobalPrefix( prefix: 'api');

const swaggerConfig = new DocumentBuilder()
  .setTitle('Library')
  .setDescription('Backend for Library project')
  .setVersion(process.env.npm_package_version)
  .addBearerAuth()
  .build();
const swaggerDocument = SwaggerModule.createDocument(app, swaggerConfig);
SwaggerModule.setup( path: 'api/docs', app, swaggerDocument);
```

Рисунок 3.4 – Код налаштування документації swagger та служби конфігурації

Після налаштувань та конфігурації проєкту, було розроблено моделі для кожної таблиці бази даних та встановлено зв’язки між ними. Всі ці можливості надає sequelize, для кожної таблиці створюється клас з назвою таблиці, який розширяє клас Model, імпортований з sequelize-typescript.

Далі було ключові компоненти архітектури: контролери, сервіси і модулі. Контролери в NestJS відповідають за обробку запитів та відправку відповідей клієнту, описуються за допомогою класів та декораторів, які анотують методи та шляхи (роути) контролера. Сервіси виконують бізнес-логіку та забезпечують розділення логіки обробки даних від логіки контролерів, а саме виконують

операції над даними, співпрацюючи з базою даних, зовнішніми API або іншими сервісами. Модулі в NestJS використовуються для організації компонентів додатків. Вони групують контролери, сервіси та інші компоненти, що відносяться до певної функціональності чи сутності та використовуються для реалізації модульної архітектури, забезпечення модульності та перевикористання коду.

Ключовим і початковим модулем було розроблено модуль авторизації. Було реалізовано дві стратегії авторизації: локальна – за логіном та паролем (для отримання токена) і jwt – авторизація запитів за допомогою jwt токена. Після входу на сайт за допомогою логіну та паролю користувач отримує jwt токен і в усі наступні запити, що потребують авторизації його буде передано в заголовок авторизації за схемою “Bearer Token” – схема автентифікації HTTP, яка включає токени безпеки, які називаються bearer tokens [15].

```
1 usage  new *
async validateUser(email: string, password: string): Promise<any> {
  const user = await this.usersService.findOne(email);

  if (!user || !(await bcrypt.compare(password, user.password))) {
    throw new UnauthorizedException();
  } else if (user && !user.isEmailConfirmed) {
    throw new UnauthorizedException(
      EXCEPTION_MESSAGES.EMAIL_IS_NOT_CONFIRMED
    );
  }

  delete user.password;
  return user;
}

1 usage  new *
async signIn(user: any): Promise<{ token: string }> {
  const payload = { sub: user.id, email: user.email };

  return {
    token: this.jwtService.sign(payload)
  };
}
```

Рисунок 3.5 – Функції basic авторизації (за логіном та паролем) і створення jwt токена

Таблиця 3.10 – Опис контролерів веб-застосунку

Назва контролеру	Опис
AttachmentsController	Містить роути для створення, видалення та отримання по id вкладень

Продовження таблиці 3.10

BooksController	Містить роути для отримання усіх книг з приміненням фільтрів, книги по id, створенн, видалення, оцінення книг
AuthorsController	Містить роути для отримання усіх авторів та автора по ідентифікатору
GenresController	Містить роути для отримання усіх жанрів та жанру по ідентифікатору
CollectionsController	Містить роути для отримання усіх колекцій її власником, отримання публічних колекцій іншого користувача, створення колекцій, додання книги в колекцію, видалення колекції
GoalsController	Містить роути для отримання користувачем його цілей, створення нових цілей та їх видалення
UsersController	Містить роути для реєстрації нових користувачів, завершення реєстрації шляхом підтвердження електронної скриньки, зміни паролю, оновлення імені користувача, отримання списку друзів та запитів дружби

Функцінал для вище описаних контролерів описаний в сервісах з відповідними назвами.

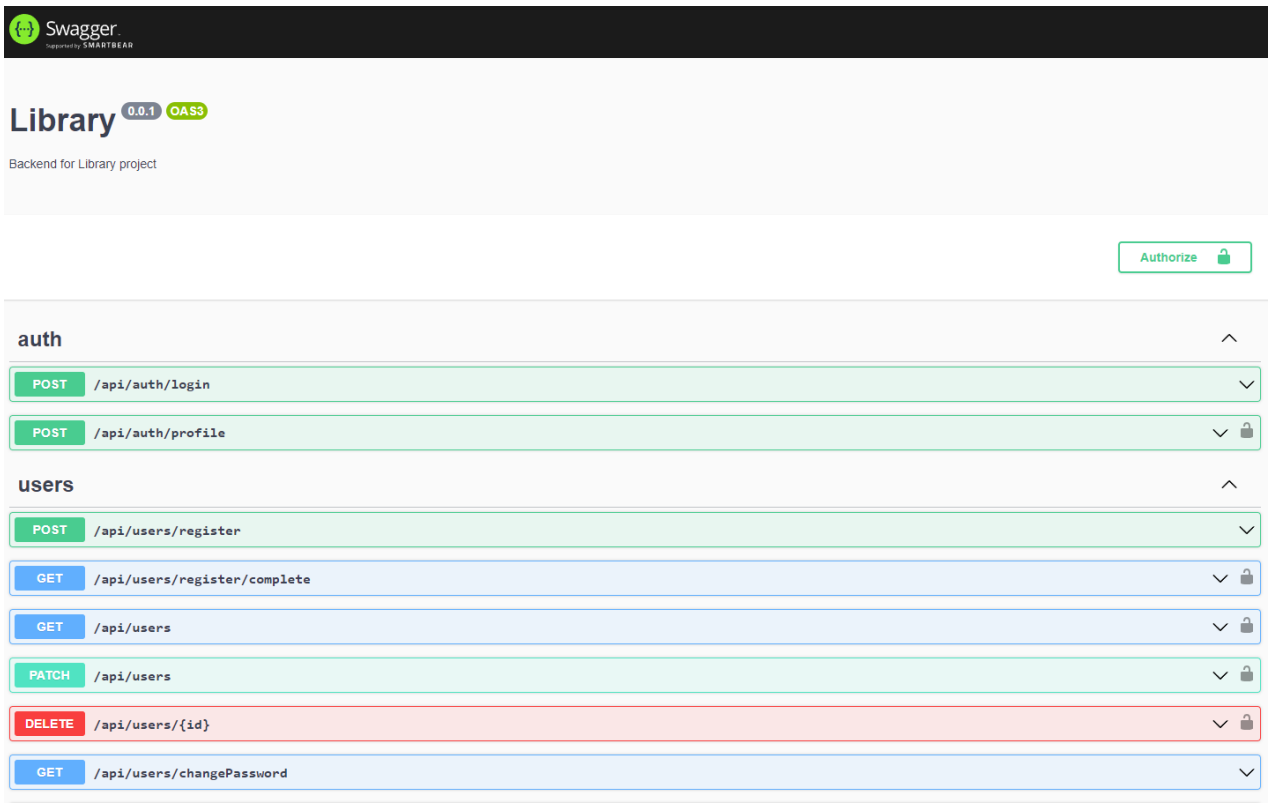


Рисунок 3.5 – приклад описаних в контролерах роутів в документації swagger

За допомогою свагеру було виконано перевірку правильності функціонування API. Це вказує на те, що можна виконувати розробку користувацького інтерфейсу.

3.3 Розробка користувацького інтерфейсу

Так як для реалізації користувацького інтерфейсу було обрано бібліотеку React в середовищі виконання Node.js, ознайомившись з документацією було покроково створено та налаштовано проект.

3.3.1 Створення та налаштування проекту

Першим кроком було налаштування проекту, а саме створення нового проекту React за допомогою стартового шаблону Create React App.

Наступним кроком було встановлено допоміжні бібліотеки з менеджеру пакунків для мови програмування JavaScript - npm (Node Package Manager):

- react-icons - бібліотека іконок для веб- застосунків, розроблених з використанням ReactJS, що надає широкий набір готових векторних іконок, які можуть бути легко використані в будь-якому проекті. Має велику колекцію готових іконок різних стилів та наборів, таких як Material Design, Font Awesome, Ionicons, Feather і багато інших.;
- react-router-dom – бібліотека для маршрутизації в React-застосунках. Дана бібліотека надає можливість створювати динамічні маршрути і переходи між різними компонентами на основі URL-адреси. React Router DOM побудована на основі React і дозволяє створювати односторінкові додатки з безперервними змінами вмісту на сторінці без перезавантаження;
- react-toastify – бібліотека для створення повідомлень (тостів) у веб-застосунках, розроблених з React. Надає простий спосіб показу повідомлень користувачам у вигляді спливаючих елементів (тостів) з різними типами повідомлень, такими як успіх, помилка, попередження і т.д. Вигляд тостів можна кастомізувати та налштувати під конкретні вимоги: змінювати стилі, час зникнення, позицію, анімацію та інші параметри тостів відповідно до дизайну;
- swiper – потужна бібліотека для створення різноманітних слайдерів, каруселей та галерей у веб-застосунках. Вона дозволяє створювати мобільно-дружні та динамічні компоненти, які можуть бути прокручуваними за допомогою жестів на сенсорних пристроях або кнопок у веб-браузері. Для коректного відображення каруселів та слайдерів у проекті потрібно правильно налаштувати його параметри та стилі.

3.3.2 Розробка компонентів для сторінок

Далі було розроблено загальні React компоненти спільні для всіх сторінок: Header – контейнер для вступного вмісту, такого як: логотип бібліотеки, навігаційних кнопок основних сторінок та кнопки авторизації (для не авторизавоних користувачів) або кнопки профілю (для авторизованих), Footer – “підвал” (блок в нижній частині сторінки) сайту, що вміщає корисну, але не головну інформацію: посилання на сторінки контакти зв’язку, допомоги, інформацію про копірайт та міні лого, Layout – компонент, що являється макетом сторінки та включає в себе хедер, футер та основний (центральний) вміст сторінки.

А також інші компоненти, наведені у таблиці 3.11.

Таблиця 3.11 – Компоненти сторінок та допомоміжні (перевикористовувані) компоненти

Назва	Опис
GenresSwiper	Слайдер для популярних жанрів бібліотеки
StarRating	Компонент який отримує книгу і рейтинг встановлений нинішнім авторизованим користувачем та відображає цей рейтинг у вигляді зірочок (якщо рейтинг не встановлено зірки сірі, якщо встановлено - жовті)
BookCard	Картка книги для списку книг. Використовується компонентами BooksList та Books
BooksList	Простий список карток книг з рамкою для використання на різних сторінках
Books	Компонент для сторінки /books з фільтрацією книг. Використовує компонент BooksList для відображення відфільтрованих книг
BookPage	Компонент вмісту сторінки конкретно обраної книги зі всьою інформацією про неї, можливістю додавання її в обрані чи інші колекції, оцінки. Також містить блок з рекомендаціями
About	Компонент вмісту інформаційної сторінки про онлайн бібліотеку
Main	Містить наповнення головної сторінки сайту
SignInForm	Форма входу користувача на сайт
AuthModal	Модальне вікно авторизації та реєстрації

3.3.3 Авторизація

Для зберігання токену авторизації було обрано варіант, досить поширений для збереження тимчасових даних в браузері, збереження в локальній пам’яті застосунку (Local Storage), звідки до нього можна отримати доступ з усіх сторінок, щоб забезпечити авторизований доступ до захищених ресурсів. Також для безпеки було встановлено обмеження терміну життя токену

– 24 год, а також для передачі даних буде використано HTTPS, адже local storage є уразливим до атак типу Cross-Site Scripting [16].

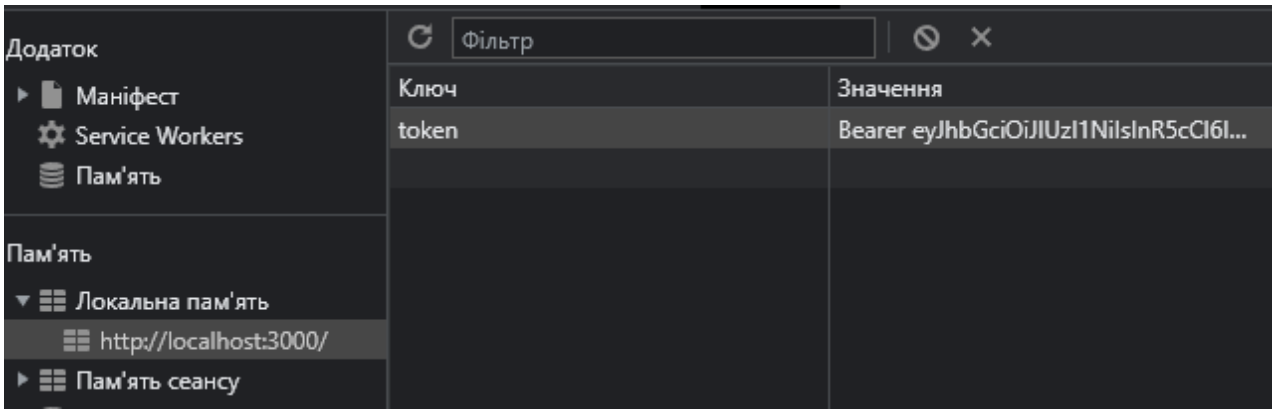


Рисунок 3.6 – Приклад збереження токену авторизації

3.3.4 Маршрутизація

З використанням бібліотеки react-router-dom налаштування роутингу веб-застосунку є досить простим завданням. В головному компоненті App застосунку потрібно імпортувати BrowserRouter – компонент, який надає контекст маршрутизації і обгорнути ним даний компонент. Далі було налаштовано всі маршрути (сторінки/компоненти, до яких ми хочемо переходити). Для кожного компоненту для якого повинен бути маршрут (окрема сторінка) потрібно використати компонент Route – він визначає співставлення між URL та відповідним компонентом і їх всі обгорнуто тегом Routes. Кожен компонент Route приймає пропси – значення асоційовані з компонентом (параметри) “path” – (шлях URL), “exact” – (чи повинен співпадати введений користувачем URL точно) та “component” – (компонент, який буде відображений за цим URL) [17].

Рисунок 3.6 – Код з налаштованою маршрутизацією

3.3.5 Взаємодія з бекендом

Для відправки запитів до сервера було розроблено файл api-helpers.js, в якому створено метод getResourceFromApi для відправки запитів, який приймає параметрами:

- url – відносне посилання на ендпоінт
- query – об’єкт з query параметрами (допоміжними параметрами запита, що додаються до URL після знаку "?" та мають вигляд “ключ=значення”)
- method –тип запита (GET, POST, PATCH, DELETE)
- body – тіло запита
- headers – додаткові http заголовки в разі необхідності

Сам же метод використовує JavaScript функцію fetch для виконання HTTP-запитів до серверу і отримання даних з нього. Також використовуючи створену функцію було розроблено ще п’ять методів для зручної відправки запитів, а саме:

- getAllItems – для відправки GET запитів до API
- getItemById – для відправки GET запитів з переданим в параметри ідентифікатором
- createItem – для відправки POST запитів
- updateItem – для відправки PATCH запитів

- deleteItem – для відправки DELETE запитів з переданим в параметри ідентифікатором

Ще було реалізовано фукції для створення та видалення файлів з серверу postFile і deleteFile відповідно.

За допомогою описаних методів, а також двох основних хуків в React – useState і useEffect було реалізовано отримання даних в компонентах, їх актуалізацію, керування їх станами та виконання інших дій, які необхідно виконати після рендеру компонента.

Хук useState використовується для створення та оновлення стану в компонентах. Даний хук повертає два значення: змінну, що містить поточне значення стану і функцію для оновлення стану цієї змінної. За допомогою стану можна оновлювати компонент відповідно до нинішнього значення стану.

Хук useEffect використовується для виконання побічних ефектів в компонентах, таких як завантаження даних, підписка на події або виконання інших дій, які необхідно виконати після рендеру компонента. Він приймає два аргументи: функцію ефекту та масив залежностей. Функція ефекту виконується після кожного рендеру компонента, якщо один з елементів залежностей змінився. Якщо потрібно виконати функцію тільки при першому рендері компонента – можна передати порожній масив залежностей.

```
const [genres, setGenres] = useState( initialState: null);
const [authors, setAuthors] = useState( initialState: null);

useEffect( effect: () => {
  genresApi
    .getAllItems( query: { sortField: 'title', sortDirection: 'asc' }) Promise<any>
    .then((res) => setGenres(res.rows)) Promise<void>
    .catch((err) => {});

  authorsApi
    .getAllItems( query: { sortField: 'firstName', sortDirection: 'asc' }) Promise<any>
    .then((res) => setAuthors(res.rows)) Promise<void>
    .catch((err) => {});
}, deps: []);
```

Рисунок 3.7 – Приклад використання хуків useState та useEffect

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі було оглянуто та охарактеризовано основні функціональні частини розробленого веб-застосунку, а саме: базу даних, серверну та клієнтську частини. Всі етапи розробки кожної з частин було детально описано, продемонстровано та обґрунтовано використані технології. Під час розробки і опису бази даних було побудовано схему для неї.

РОЗДІЛ 4. ОГЛЯД ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ

4.1 Розгортання застосунку

Для розгортання веб-застосунку було використано docker – відкрите програмне забезпечення, що надає можливість упаковувати, розгортати та управляти програмними додатками у віртуальних контейнерах [18]. Контейнери в Docker дозволяють ізолювати додатки та їх залежності, забезпечуючи їх портативність та скорочуючи ризик конфліктів у середовищі розгортання. Docker використано для упаковування та розгортання компонентів, а саме бази даних, серверу та фронтенду у віртуальні контейнери. На рисунку 4.1 наведено конфінурацію для контейнери, що потрібно розгорнути, з файлу docker-compose.yml.

```
services:
  library-db:
    image: postgres:14
    container_name: library-db
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=cH2DP42mB2xy
      - POSTGRES_DB=library
    volumes:
      - ./postgres/library:/var/lib/postgresql/data
    ports:
      - 5433:5432
    restart: always

  library-backend:
    container_name: library-backend
    image: library/backend
    depends_on:
      - library-db
    environment:
      - NODE_ENV=development
      - PORT=3000
      - DB_HOST=library-db
      - DB_USER=postgres
      - DB_PASSWORD=cH2DP42mB2xy
      - DB_NAME=library
      - DB_PORT=5432
      - JWT_SECRET=PjnSLXITYpmI
      - ADMIN_PASSWORD=String-8
    volumes:
      - ./uploads:/app/uploads
    command: "sh -c 'npm run migrate && npm run start'"
    restart: always

  library-public:
    container_name: library-public
    image: library/public
    environment:
      - NODE_ENV=development
      - REACT_APP_PUBLIC_HOST=http://localhost
      - REACT_APP_API_HOST=http://localhost/api
    working_dir: /app
    command: "sh -c 'npm run start'"
```

Рисунок 4.1 – Опис конфігурації контейнерів бази даних, серверу, фронтенду

Також окремим контейнером було розгорнуто NGINX. NGINX є веб-сервером з відкритим кодом та проксі-сервером, який широко використовується для розгортання веб-застосунків. Він працює як проксі, що приймає запити від клієнтів та передає їх до відповідних серверів, а також забезпечує обробку статичних файлів та балансування навантаження.

```yaml
version: "3.8"
x-logging: &default-logging
  options:
    max-size: "50m"
    max-file: "6"
  driver: "json-file"
services:
  nginx:
    image: nginx:latest
    container_name: nginx
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    # start sed
      - ./library/uploads:/app/uploads
      - ./library/nginx.conf:/etc/nginx/library/nginx.conf

    ports:
      - "80:80"
    restart: always
networks:
  default:
    external:
      name: projects_network
```

Рисунок 4.2 – Docker-compose.yml файл конфігурації nginx

За допомогою docker-compose up команди можна розгорнути всі сервіси, описані у файлі docker-compose.yml. Docker Compose дозволяє легко управляти багатоконтейнерними додатками та їх залежностями, а також полегшує розгортання додатків у різних середовищах.

4.2 Тестування веб-застосунку

Для тестування було прийнято рішення використати підхід мануального тестування. Воно являє собою процес при якому тестувальник вручну перевіряє функціональність веб-застосунку, спираючись на вимоги, специфікації та орієнтацію на користувача. Опис протестованих кейсів наведено у таблицях 4.1 – 4.8.

Таблиця 4.1 – Тест 1

Тест	Реєстрація користувача
Номер	1

Продовження таблиці 4.1

Початковий стан системи	Користувач в модульному вікні може ввести дані та надіслати запит на сервер
Вхідні дані	Дані про користувача (юзернейм, поштова скринька, пароль)
Опис проведення тесту	Користувач заповнює форму необхідними даними в потрібному форматі, натискає “Sign up” для відправки запиту на сервер і очікує на відповідь
Очікуваний результат	Користувач отримує повідомлення по успішну реєстрацію
Фактичний результат	Користувач отримує повідомлення по успішну реєстрацію

Таблиця 4.2 – Тест 2

Тест	Авторизація користувача
Номер	2
Початковий стан системи	Користувач в модульному вікні може ввести дані та надіслати запит на сервер
Вхідні дані	Дані для входу – email – поштова скринька і password – пароль до акаунту
Опис проведення тесту	Користувач заповнює форму необхідними даними в потрібному форматі, натискає “Authorize” для відправки запиту на сервер і очікує на відповідь
Очікуваний результат	Користувач отримує повідомлення по успішну авторизацію, токен авторизації додається в локальне сховище браузера
Фактичний результат	Користувач отримує повідомлення по успішну авторизацію, токен авторизації додається в локальне сховище браузера

Таблиця 4.3 – Тест 3

Тест	Відправка запиту дружби користувачу
Номер	3
Початковий стан системи	Користувач знайшов іншого користувача за юзернеймом
Вхідні дані	-
Опис проведення тесту	Користувач натискає на кнопку “Add”
Очікуваний результат	Користувач отримує повідомлення про успішну відправку запиту, кнопка змінюється на “Remove”
Фактичний результат	Користувач отримує повідомлення про успішну відправку запиту, кнопка змінюється на “Remove”

Таблиця 4.4 – Тест 4

Тест	Отримання усіх книг
Номер	4
Початковий стан системи	Користувач має можливість перейти на сторінку усіх книг
Вхідні дані	-
Опис проведення тесту	Користувач переходить на сторінку усіх книг
Очікуваний результат	Користувач отримує список книг з бази даних з пагінацією
Фактичний результат	Користувач отримує список книг з бази даних з пагінацією

Таблиця 4.5 – Тест 5

Тест	Фільтрація книг за параметрами
Номер	5
Початковий стан системи	Користувач на сторінці книг може обрати фільтри
Вхідні дані	Ідентифікатори жанрів, авторів, кількість зірок рейтингу і текст для пошуку у назві
Опис проведення тесту	Користувач обирає потрібні фільтри та вводить текст в поле пошуку та натискає “Apply filters”
Очікуваний результат	Користувач отримує усі книги, що відповідають обраним параметрам (фільтрам)
Фактичний результат	Користувач отримує усі книги, що відповідають обраним параметрам (фільтрам)

Таблиця 4.6 – Тест 6

Тест	Отримання детальної інформації про конкретну книгу на її сторінці
Номер	6
Початковий стан системи	Користувач може перейти на її сторінку натиснувши на неї
Вхідні дані	Ідентифікатор книги
Опис проведення тесту	Користувач обирає книгу зі списку і натискає на неї або переходить за посиланням і потрапляє на сторінку з її детальною інформацією
Очікуваний результат	Користувач переходить на сторінку з детальною інформацією про книгу (назвою, описом, жанром, автором, кількістю сторінок, рейтингом)

Продовження таблиці 4.6

Фактичний результат	Користувач переходить на сторінку з детальною інформацією про книгу

Таблиця 4.7 – Тест 7

Тест	Оцінка книги за рейтингом від 1 до 5
Номер	7
Початковий стан системи	Користувач на сторінці книги може встановити свою оцінку
Вхідні дані	Ідентифікатор книги і рейтинг від 1 до 5
Опис проведення тесту	Користувач натискає на потрібну кількість зірок на сторінці книги
Очікуваний результат	Користувач бачить тост про успішне встановлення рейтингу і на сторінці обрана кількість зірок змінюють колір на жовтий
Фактичний результат	Користувач бачить тост про успішне встановлення рейтингу і на сторінці обрана кількість зірок змінюють колір на жовтий

Таблиця 4.8 – Тест 8

Тест	Отримання користувачем колекцій книг
Номер	8
Початковий стан системи	Користувач зареєстрований у веб-застосунку
Вхідні дані	-
Опис проведення тесту	Користувач у своєму профілі переходить на сторінку “Collections”

Продовження таблиці 4.8

Очікуваний результат	Користувач потрапляє на сторінку з переліком його колекцій
Фактичний результат	Користувач потрапляє на сторінку з переліком його колекцій

Таблиця 4.9 – Тест 9

Тест	Створення користувацької колекції книг
Номер	9
Початковий стан системи	Зареєстрований у веб-застосунку користувач перебуває на сторінці колекцій і може натиснути на кнопку “Create”
Вхідні дані	Назва, опис та позначка публічного доступу до колекції
Опис проведення тесту	Користувач заповнює форму створення і натискає “Create”
Очікуваний результат	Успішно створена колекція з’являється у списку колекцій
Фактичний результат	Успішно створена колекція з’являється у списку колекцій

Таблиця 4.10 – Тест 10

Тест	Додання книги до користувацької колекції
Номер	10
Початковий стан системи	Зареєстрований у веб-застосунку користувач перебуває на сторінці книги
Вхідні дані	-

Продовження таблиці 4.10

Опис проведення тесту	Користувач натискає на кнопку “Add to collection” і обирає колекцію в яку хоче додати книгу
Очікуваний результат	Книгу додано в колекцію
Фактичний результат	Книгу додано в колекцію

Було проведено та успішно завершено тестування основних користувацьких сценаріїв.

4.3 Демонстрація роботи веб-застосунку

Після розгортання усіх компонентів веб-застосунку за допомогою docker, можна отримати доступ до користувацького інтерфейсу перейшовши за посиланням http://localhost:3000/. За посиланням користувач потрапляє на головну сторінку з хедером, футером та основною частиною, що містить список наявних жанрів і шість книг з найвищим рейтингом (рис. 4.3).

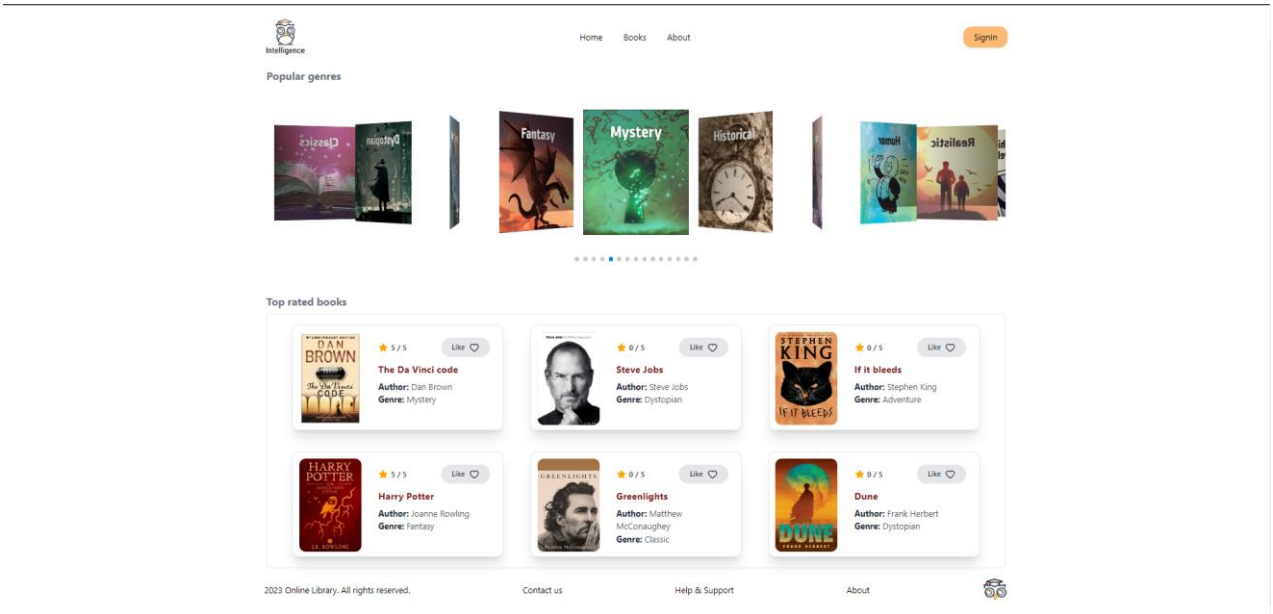


Рисунок 4.3 – Головна сторінка онлайн бібліотеки

Так як для користувачів, що не авторизувались функціонал доволі обмежений, потрібно увійти в акаунт для отримання усіх переваг веб-застосунку. Для цього потрібно натиснути кнопку “SignIn” в правому верхньому куті, після чого відкриється модальне вікно для вводу даних для входу (рис. 4.4). Якщо користувач ще не має зареєстрованого акаунту можна натиснути на “Sign up” в нижньому правому куті поточного вікна (рис. 4.4).

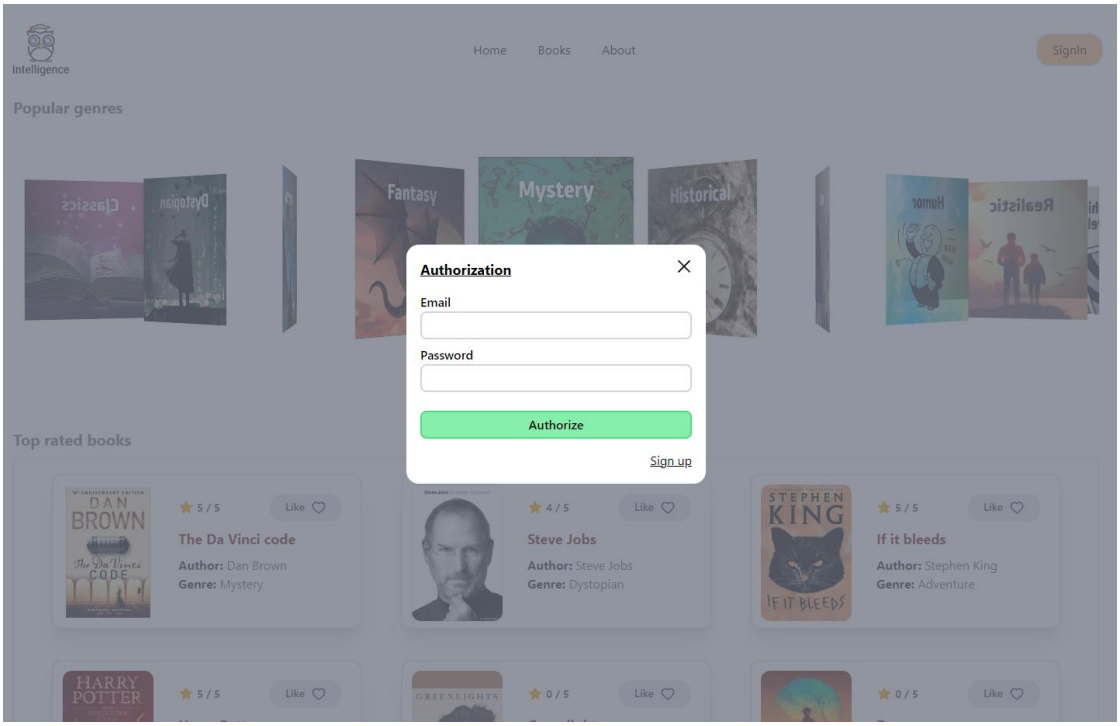


Рисунок 4.4 – Модальне вікно авторизації

Для реєстрації достатньо натиснути на “Sign up” та заповнити усі поля у формі реєстрації (рис. 4.5).

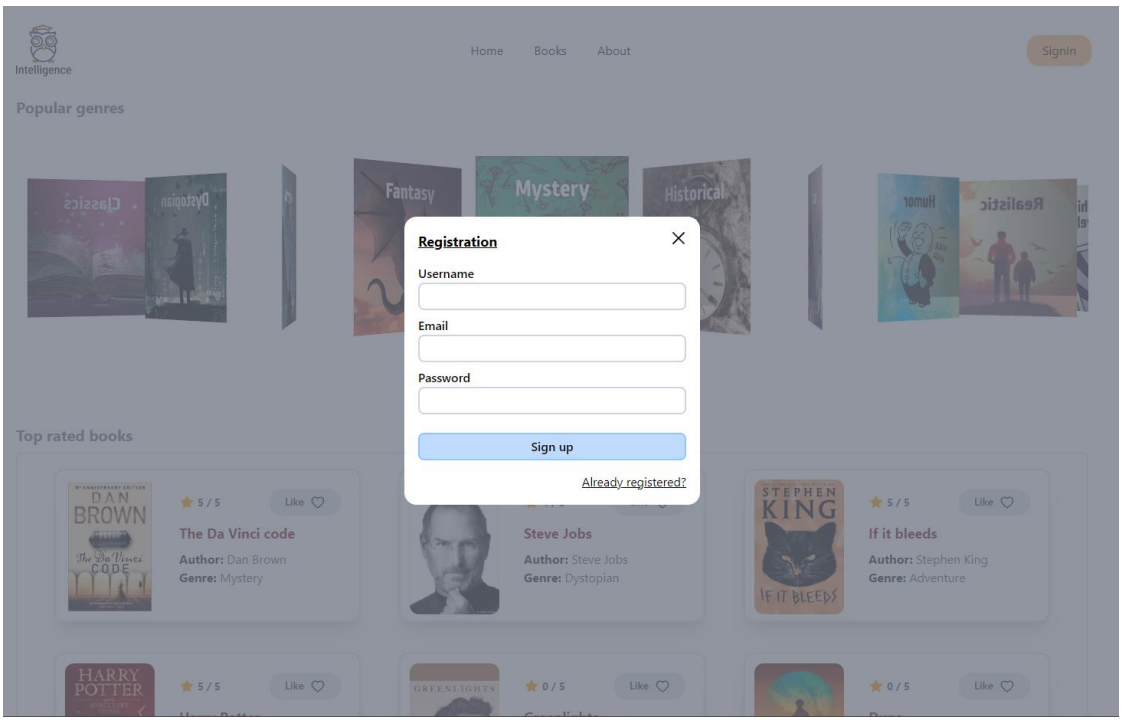


Рисунок 4.5 – Модальне вікно реєстрації

Після успішної реєстрації форма зміниться на форму авторизації, а за потреби з вікна реєстрації можна з легкістю повернутись на вікно авторизації, натиснувши на “Already registered?” (див. рис. 4.5), наприклад якщо зареєстрований користувач випадково потрапив на форму реєстрації.

Якщо користувач успішн зареєструвався та авторизувався у веб-застосунку він отримує можливість перейти на сторінку власного профілю, де відображено його статистику.

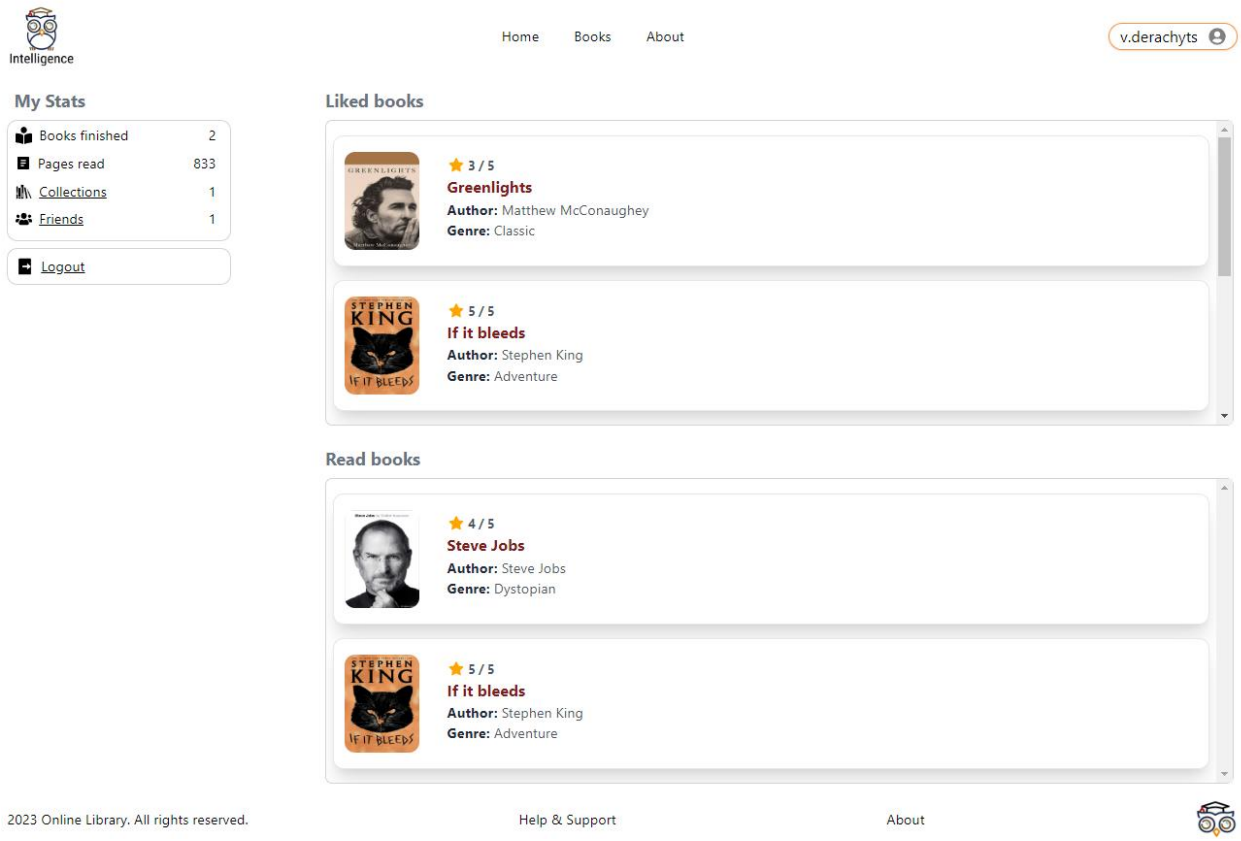


Рисунок 4.6 – Сторінка профілю користувача

Важливим функціоналом, що доступний усім відвідувачам є фільтрація та пошук книг в загальному списку (рис. 4.7). На сторінці можна обрати фільтри за авторами, жанрами, рейтингом та шукати книги по назві. Сторінка містить серверну пагінацію каталогу книг, що робить завантаження сторінки швидким.

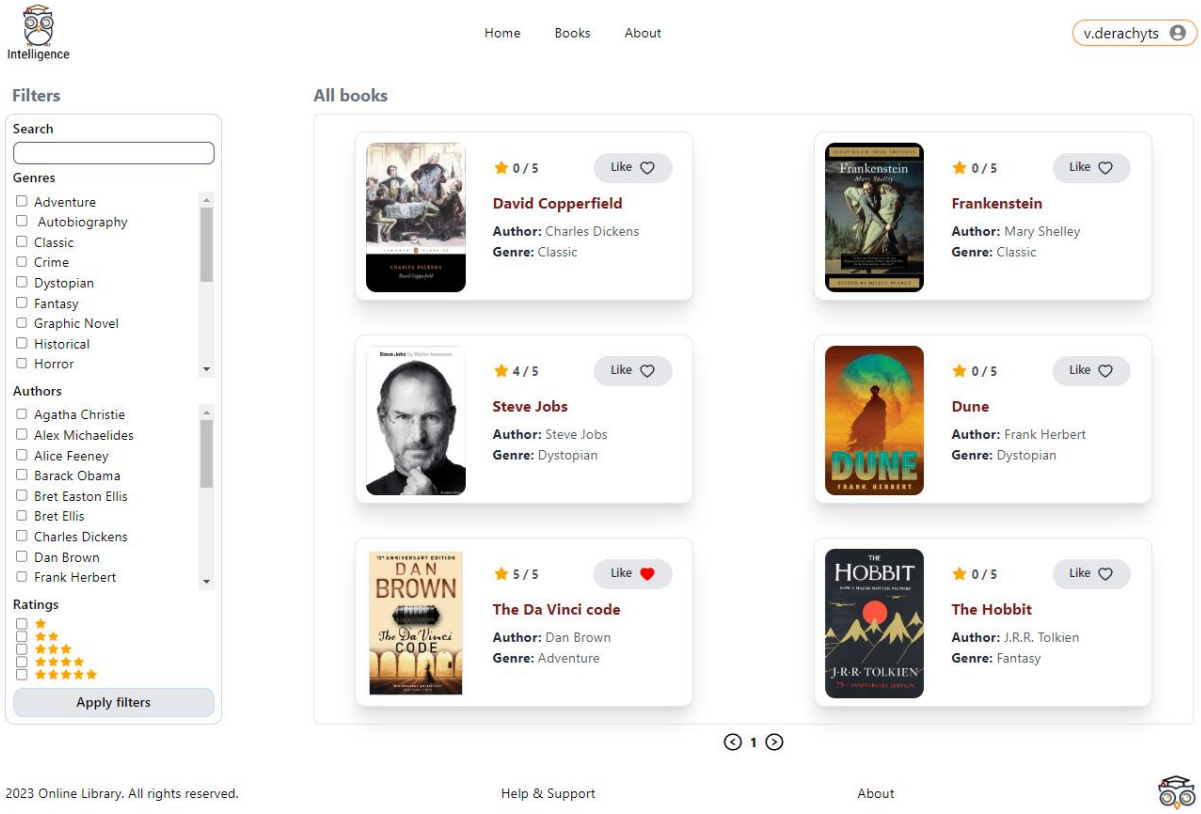


Рисунок 4.7 – Сторінка книг з фільтрацією та пошуком

Демонстрацію роботи фільтрів та пошуку наведено на рис. 4.8.

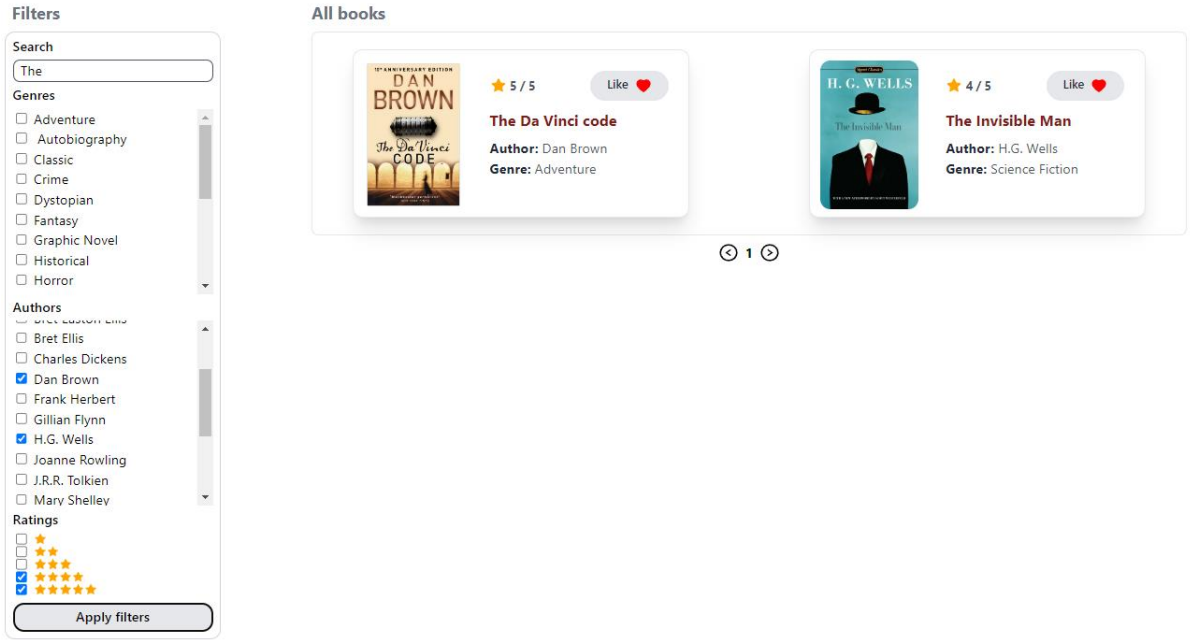


Рисунок 4.8 – Результат примінення фільтрів

Після натиснення на кнопку “Read online” в браузері користувача створюється нова вкладка зі вмістом pdf файлу книги (рис. 4.9). Користувач одразу потраляє на неї та може розпочати читання.

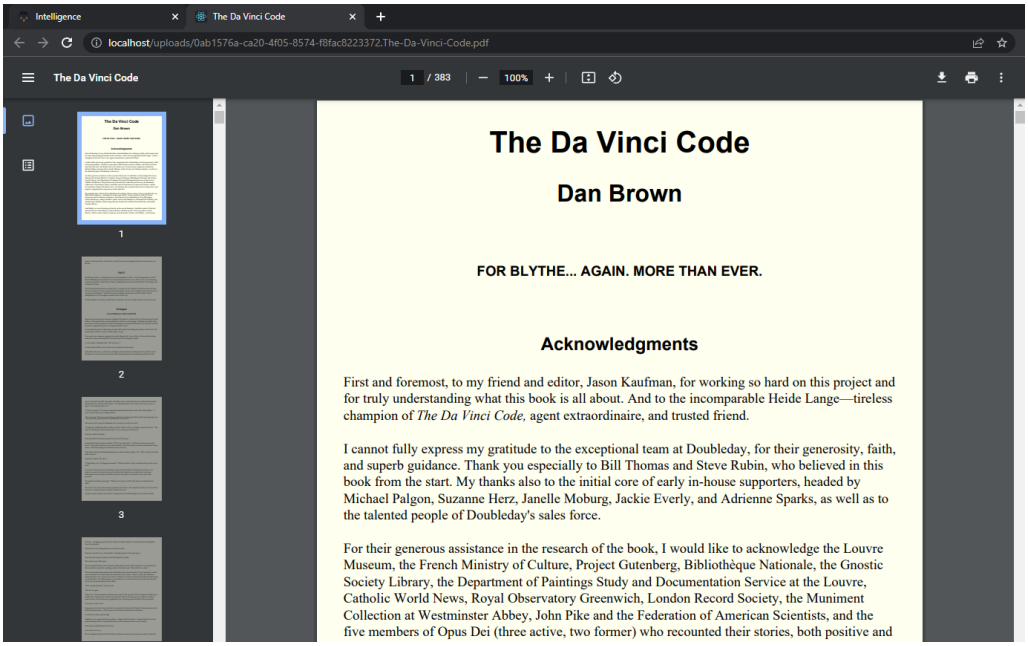


Рисунок 4.9 – Нова вкладка з вмістом книги в форматі pdf

Якщо відвідувач хоче завантажити книгу на свій девайс для прочитання її в зручний час офлайн він може натиснути на текст “Download” під кнопкою для читання онлайн. Для нього буде відкрите модальне вікно в якому можна обрати місце для збереження та назву (рис 4.10).

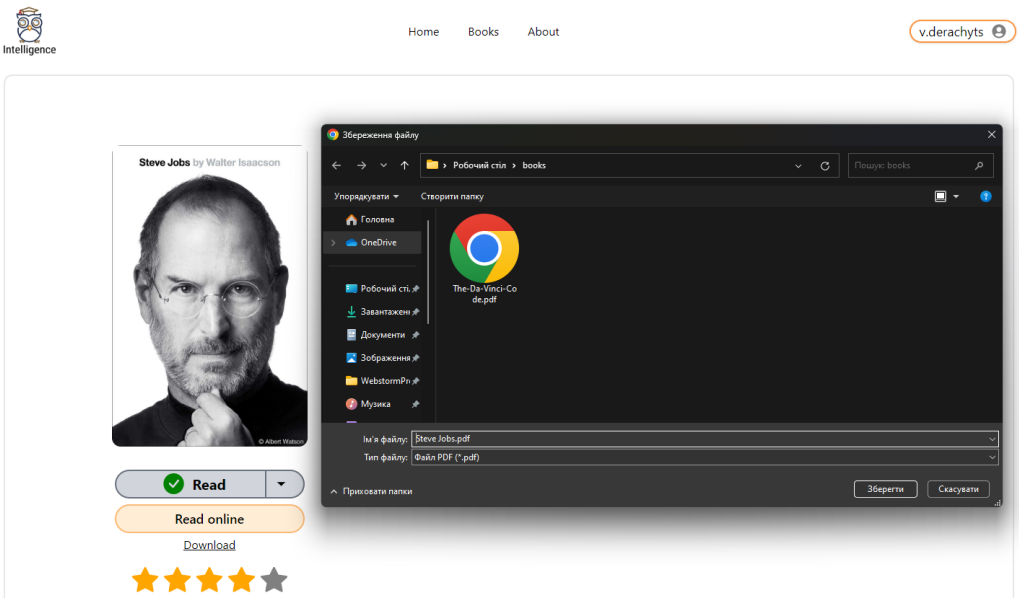


Рисунок 4.10 – Модальне вікно завантаження файлу

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі було проведено та описано тестування основних сценаріїв використання веб-застосунку мануальним методом. По завершенню всіх тестів було визначено, що застосунок працює коректно. Наступним кроком онлайн бібліотеку було розгорнуто за допомогою програми розгортання та управління контейнерами – docker. Фінальним етапом було продемонстровано роботу та перевірено функціонал застосунку з підтвердженням цьому – прикріпленням знімків екрану.

ВИСНОВКИ

Під час виконання даної дипломної роботи був розроблений веб-застосунок для онлайн бібліотеки, який надає зручний та ефективний спосіб пошуку, перегляду та взаємодії з навчальними матеріалами. Робота виконана з використанням сучасних технологій розробки веб-додатків, таких як ReactJS для фронтенду та NestJS для бекенду.

В першому розділі було опрацьовано матеріали і проаналізовано предметну облать, а також проведено аналіз існуючих рішень та їх порівняння.

В другому розділі було розроблено архітектуру майбутього рішення з урахуванням вимог і використанням сучасних технологій та інструментів розробки для можливого подальшого масштабування веб-застосунку. Було обрано СУБД – PostgreSQL, NestJS для зручної та швидкої рзробки бекенду і ReactJS для розробки користувацького інтерфейсу.

У третьому розділі було виконано та описано основні етапи розробки, а саме – розробку кожної частини веб-застосунку.

У процесі розробки було реалізовано функціональність для реєстрації та аутентифікації користувачів, пошуку та фільтрації книжок, перегляду деталей книги, оцінки користувачів, а також можливість читання онлайн і завантаження контенту книг. Було забезпечено зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам максимально комфортно використовувати усі можливості онлайн бібліотеки.

У четвертому розділі було проведено розгортання застосунку, тестування вручну та продемонстровано це за допомогою знімків екрану та їх опису.

Поставлена задача була виконана, а мета дипломного проєкту досягнута, про що свідчать вище наведені аргументи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна бібліотека [Електронний ресурс]. – 2205. – Режим доступу до ресурсу: https://ube.nlu.org.ua/article/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%20%D0%B1%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0.
2. Google Trends. Графік популярності пошукового запиту “online library” [Електронний ресурс] / Google Trends – Режим доступу до ресурсу: https://trends.google.com.ua/trends/explore?date=today%205-y&q=online%20library&hl=uk.
3. About Project Gutenberg [Електронний ресурс] – Режим доступу до ресурсу: https://www.gutenberg.org/about/.
4. Smarter Libraries Through Technology: 50 Years of Technology at OCLC [Електронний ресурс] – Режим доступу до ресурсу: https://librarytechnology.org/document/23317.
5. Digital Libraries Initiative Projects [Електронний ресурс] – Режим доступу до ресурсу: https://paginaspersonales.deusto.es/abaitua/konzeptu/bbdd/dli_home.html.
6. Arimetrics. What is Google Books [Електронний ресурс] / Arimetrics – Режим доступу до ресурсу: https://www.arimetrics.com/en/digital-glossary/google-books.
7. Oluwatosin H. S. Client-Server Model / Haroon Shakirat Oluwatosin. // 1. – 2014. – С. 67–71.
8. https://nodejs.org/en/about
9. NestJS [Електронний ресурс] – Режим доступу до ресурсу: https://nestjs.com/.

10. React A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/.
11. Vue.js - The Progressive JavaScript Framework [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/.
12. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.
13. WebStorm: The Smartest JavaScript IDE, by JetBrains [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/webstorm/.
14. Sequelize [Електронний ресурс] – Режим доступу до ресурсу: https://sequelize.org/.
15. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction.
16. LocalStorage vs Cookies: All You Need To Know About Storing JWT Tokens Securely in The Front-End [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/cotter/localstorage-vs-cookies-all-you-need-to-know-about-storing-jwt-tokens-securely-in-the-front-end-15id.
17. React Router. Main concepts [Електронний ресурс] – Режим доступу до ресурсу: https://reactrouter.com/en/main/start/concepts.
18. Docker: Accelerated, Containerized Application Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/.

ДОДАТОК 1

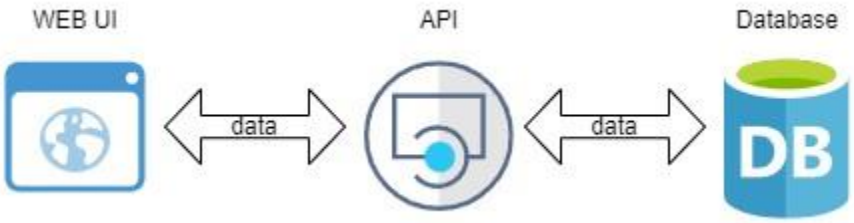
Веб-застосунок “Онлайн бібліотека”

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Дерачиц В.В.			**Веб-застосунок “Онлайн бібліотека”** Структурна схема системи (структурна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91	
Н. Контр.		Виноградов Ю. М.					
Затвердив							

ДОДАТОК 2

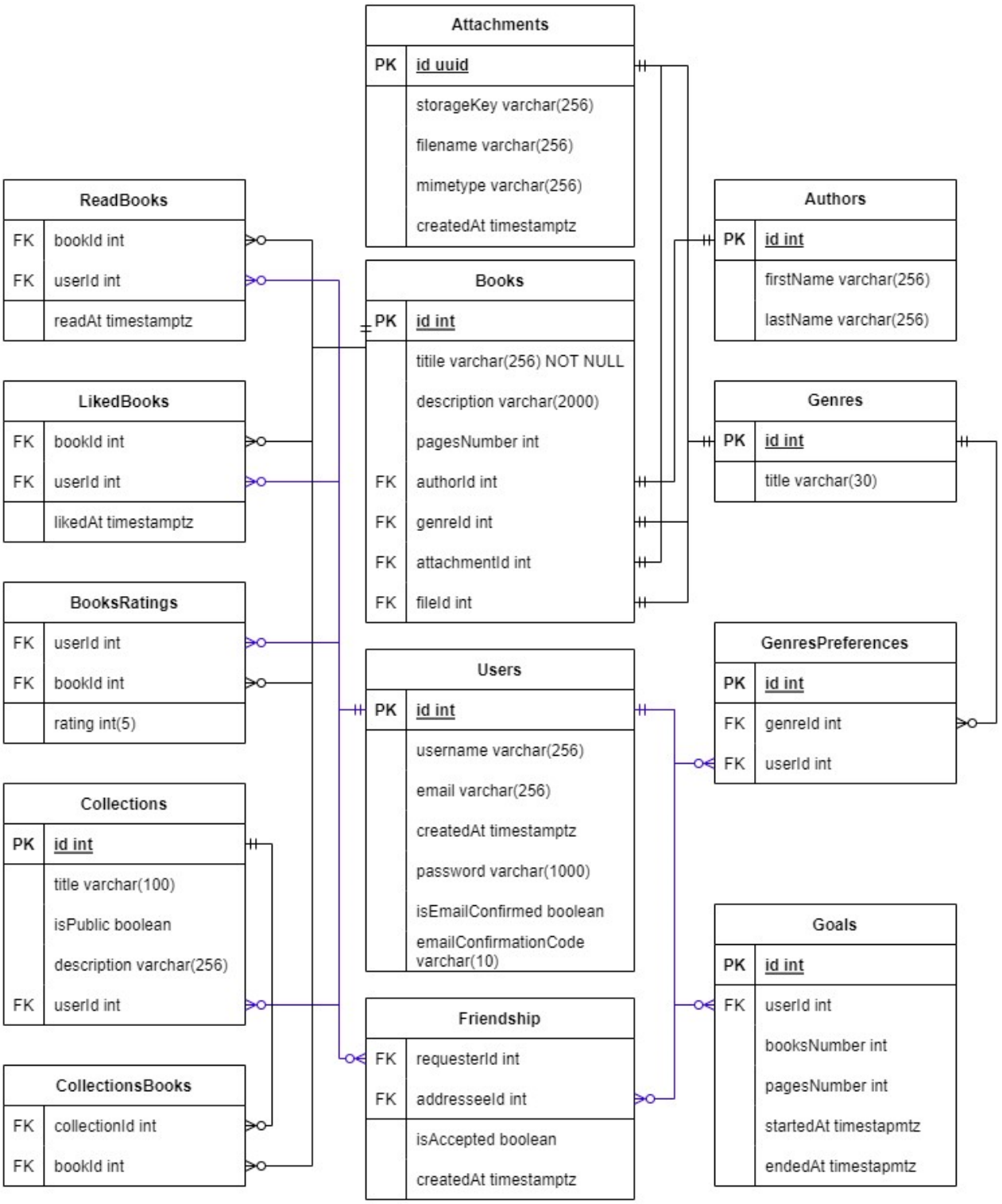
Веб-застосунок “Онлайн бібліотека”

Схема бази даних

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д2
		№ докум.	Підпис	Дата	
Розробив		Дерачиц В.В.			**Веб-застосунок “Онлайн бібліотека”** Схема бази даних (функціональна схема)
Перевірив		Волокита А. М.			
Н. Контр.		Виноградов Ю. М.			
Затвердив					

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91

ДОДАТОК 3

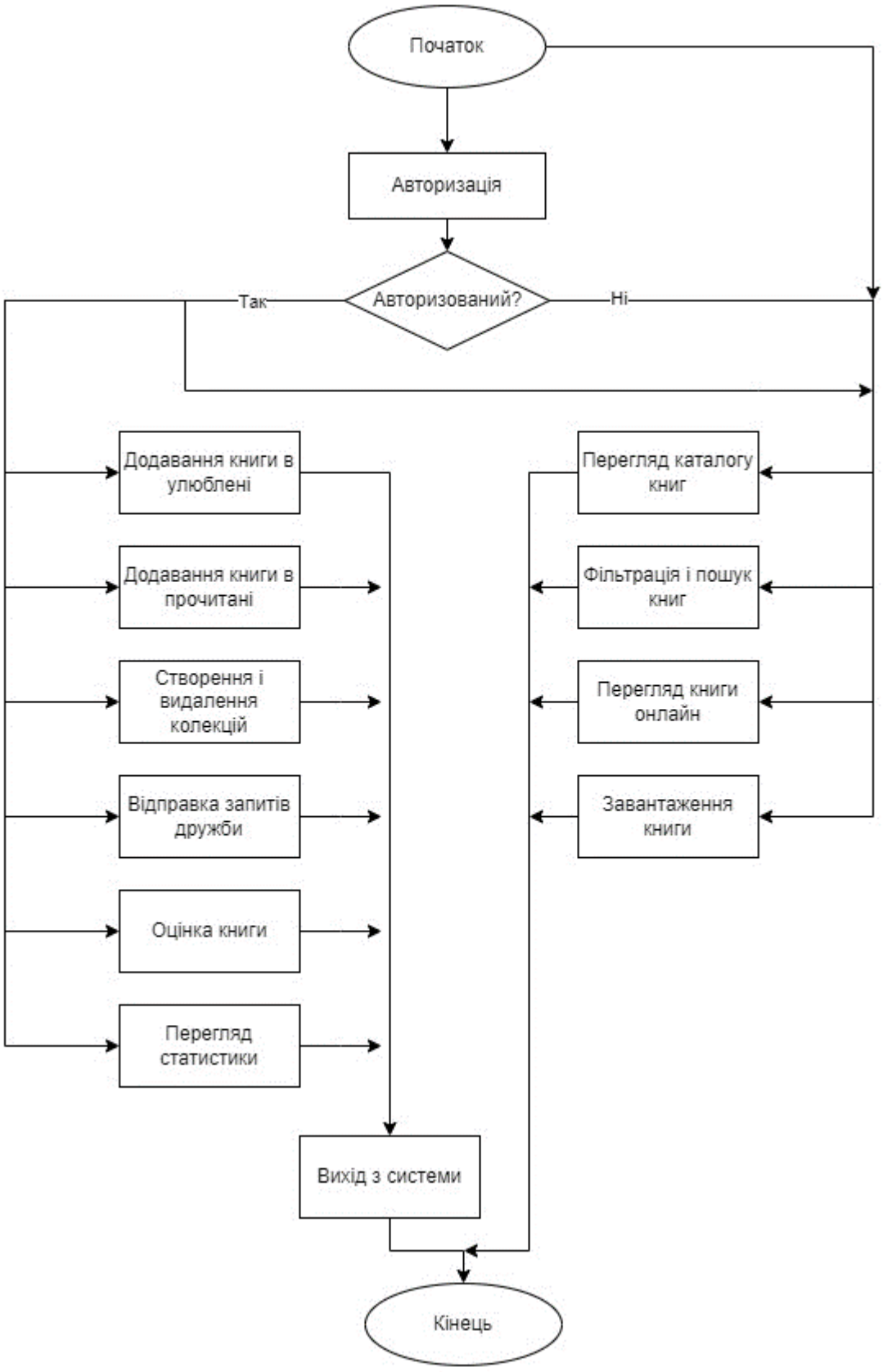
Веб-застосунок “Онлайн бібліотека”

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.006 Д3
		№ докум.	Підпис	Дата	
Розробив		Дерачиц В.В.			**Веб-застосунок “Онлайн бібліотека”** Схема роботи системи (принципова схема)
Перевірив		Волокита А. М.			
Н. Контр.		Виноградов Ю. М.			
Затвердив					

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91

ДОДАТОК 4

Веб-застосунок “Онлайн бібліотека”

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 41

Київ 2023 р

auth.controller.ts

```
import {
  Body,
  Controller,
  Post,
  Request,
  UnauthorizedException,
  UseGuards
} from '@nestjs/common';
import { AuthService } from './auth.service';
import { LocalAuthGuard } from './guards/local-auth.guard';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { AuthUserDto } from './dto/auth-user.gto';
import { OptionalJwtAuthGuard } from './guards/optional-auth.guard';

@Controller('auth')
@ApiTags('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @UseGuards(LocalAuthGuard)
  @Post('login')
  login(@Body() authUserDto: AuthUserDto, @Request() req) {
    return this.authService.signIn(req.user);
  }
  @ApiBearerAuth()
  @UseGuards(OptionalJwtAuthGuard)
  @Post('profile')
  getUserFromToken(@Request() req) {
    if (!req.user) {
      throw new UnauthorizedException();
    }
       return {
      id: req.user.id,
      username: req.user.username,
```

```
      email: req.user.email
    };
  }
}
```

auth.service.ts

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { UsersService } from '../users/users.service';
import { JwtService } from '@nestjs/jwt';
import { EXCEPTION_MESSAGES } from '../constants';
import * as bcrypt from 'bcrypt';

@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService
  ) {}

  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.usersService.findOne(email);

    if (!user || !(await bcrypt.compare(password, user.password))) {
      throw new UnauthorizedException();
    } else if (user && !user.isEmailConfirmed) {
      throw new UnauthorizedException(
        EXCEPTION_MESSAGES.EMAIL_IS_NOT_CONFIRMED
      );
    }

    delete user.password;
    return user;
  }

  async signIn(user: any): Promise<{ token: string }> {
    const payload = { sub: user.id, email: user.email };

    return {
      token: this.jwtService.sign(payload)
    };
  }
}
```

books.controller.ts

```
import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Patch,
  Post,
  Query,
  UseGuards
} from '@nestjs/common';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { BooksService } from './books.service';
import { CreateBookDto } from './dto/create-book.dto';
import { Book } from './models/book.model';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { User } from '../auth/decorators/user.decorator';
import { GetItemDto } from '../common/dto/get-item.dto';
import { OptionalJwtAuthGuard } from '../auth/guards/optional-auth.guard';
import { GetBooksDto } from './dto/get-books.dto';
import { RateBookDto } from './dto/rate-book.dto';
import { GetItemsDto } from '../common/dto/get-items.dto';

@Controller('books')
@ApiTags('books')
export class BooksController {
  constructor(private readonly booksService: BooksService) {}

  @Post()
  async create(@Body() createBookDto: CreateBookDto): Promise<Book> {
    return this.booksService.create(createBookDto);
  }

  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Get('liked')
  findLiked(
    @Query() query: GetItemsDto,
    @User() user: any
  ): Promise<{ rows: Book[]; count: number }> {
    return this.booksService.findLiked(query, user.id);
  }
```

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get('read')
findRead(
  @Query() query: GetItemsDto,
  @User() user: any
): Promise<{ rows: Book[]; count: number }> {
  return this.booksService.findRead(query, user.id);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get('pages/read')
pagesRead(@User() user: any): Promise<{ pagesRead: number }> {
  return this.booksService.pagesRead(user.id);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post('liked')
async addToLiked(@Body('id') id: number, @User() user: any): Promise<void> {
  return this.booksService.addToLiked(id, user.id);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Post('read')
async addToRead(@Body('id') id: number, @User() user: any): Promise<void> {
  return this.booksService.addToRead(id, user.id);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('liked/:id')
async removeFromLiked(
  @Param('id') id: number,
  @User() user: any
): Promise<void> {
  return this.booksService.removeFromLiked(id, user.id);
}
```

```
@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete('read/:id')
async removeFromRead(
  @Param('id') id: number,
  @User() user: any
): Promise<void> {
  return this.booksService.removeFromRead(id, user.id);
}

@ApiBearerAuth()
@UseGuards(OptionalJwtAuthGuard)
@Get(':id')
findById(
  @Param('id') id: number,
  @Query() query: GetItemDto,
  @User() user: any
): Promise<Book> {
  return this.booksService.findById(id, query, user.id);
}

@ApiBearerAuth()
@UseGuards(OptionalJwtAuthGuard)
@Get()
findAll(
  @Query() query: GetBooksDto,
  @User() user: any
): Promise<{ rows: Book[]; count: number }> {
  return this.booksService.findAll(query, user.id);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Patch(':id/rate')
async rate(
  @Param('id') id: number,
  @Body() rateBookDto: RateBookDto,
  @User() user: any
): Promise<Book> {
  return this.booksService.rate(id, rateBookDto, user.id);
}
```

```
  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Delete(':id')
  remove(@Param('id') id: number) {
    return this.booksService.remove(id);
  }
}
```

books.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import { Sequelize } from 'sequelize-typescript';
import { FindAndCountOptions, FindOptions, Op } from 'sequelize';
import { Book } from './models/book.model';
import { CreateBookDto } from './dto/create-book.dto';
import { GetItemDto } from '../common/dto/get-item.dto';
import { Attachment } from '../attachments/models/attachment.model';
import { BooksRating } from './models/books-rating.model';
import { Genre } from './genres/models/genre.model';
import { Author } from './authors/models/author.model';
import { GetBooksDto } from './dto/get-books.dto';
import { RateBookDto } from './dto/rate-book.dto';
import { LikedBooks } from './models/liked-books.model';
import { GetItemsDto } from '../common/dto/get-items.dto';
import { ReadBooks } from './models/read-books.model';

@Injectable()
export class BooksService {
  constructor(
    @InjectModel(Book)
    private bookModel: typeof Book,
    @InjectModel(LikedBooks)
    private likedBooksModel: typeof LikedBooks,
    @InjectModel(ReadBooks)
    private readBooksModel: typeof ReadBooks,
    @InjectModel(BooksRating)
    private booksRatingModel: typeof BooksRating
  ) {}

  async create(createBookDto: CreateBookDto): Promise<Book> {
    return this.bookModel.create({ ...createBookDto });
  }
```

```
async addToLiked(id: number, userId: number): Promise<void> {
  const book = await this.bookModel.findByPk(id);

  if (book) {
    await this.likedBooksModel.create({
      bookId: id,
      userId: userId,
      likedAt: new Date(Date.now())
    });
  }
}

async addToRead(id: number, userId: number): Promise<void> {
  const book = await this.bookModel.findByPk(id);

  if (book) {
    await this.readBooksModel.create({
      bookId: id,
      userId: userId,
      readAt: new Date(Date.now())
    });
  }
}

async removeFromLiked(id: number, userId: number): Promise<void> {
  const book = await this.likedBooksModel.findOne({
    where: { bookId: id, userId: userId }
  });

  if (book) {
    await book.destroy();
  }
}

async removeFromRead(id: number, userId: number): Promise<void> {
  const book = await this.readBooksModel.findOne({
    where: { bookId: id, userId: userId }
  });

  if (book) {
    await book.destroy();
  }
```

```
}

async findById(id: number, query: GetItemDto, userId: number): Promise<Book> {
  const options: FindOptions = {};

  const includes = this.createIncludes(query.include);

  options.attributes = [
    ...(query.fields
      ? query.fields
      : [
          'id',
          'title',
          'description',
          'pagesNumber',
          'genreId',
          'authorId',
          'attachmentId'
        ].filter((item) => !includes.attributes.exclude.includes(item))),
    ...includes.attributes.include
  ];

  if (userId) {
    includes.include.push({
      model: BooksRating,
      as: 'userRating',
      required: false,
      attributes: ['rating'],
      where: {
        userId: userId
      }
    });
    options.attributes.push(
      [
        Sequelize.literal(
          '(SELECT true FROM "ReadBooks" rb WHERE rb."bookId" = "Book"."id" LIMIT 1)'
        ),
        'isRead'
      ],
      [
        Sequelize.literal(
          '(SELECT true FROM "LikedBooks" lb WHERE lb."bookId" = "Book"."id" LIMIT 1)'
```

```
        ),
        'isLiked'
      ]
    );
  }
  options.include = includes.include;

  return this.bookModel.findByPk(id, options);
}

async findAll(
  query: GetBooksDto,
  userId: number
): Promise<{ rows: Book[]; count: number }> {
  const options: FindAndCountOptions = {
    subQuery: false,
    distinct: true,
    offset: query.start,
    limit: query.count
  };

  const includes = this.createIncludes(query.include);

  options.attributes = [
    ...(query.fields
      ? query.fields
      : [
          'id',
          'title',
          'description',
          'pagesNumber',
          'genreId',
          'authorId',
          'attachmentId'
        ].filter((item) => !includes.attributes.exclude.includes(item))),
    ...includes.attributes.include
  ];

  if (userId) {
    includes.include.push({
      model: BooksRating,
      as: 'userRating',
```

```
    required: false,
    attributes: ['rating'],
    where: {
      userId: userId
    }
  });
  options.attributes.push(
    [
      Sequelize.literal(
        `(SELECT true FROM "ReadBooks" rb WHERE rb."bookId" = "Book"."id" AND rb."userId" = ${userId} LIMIT 1)`
      ),
      'isRead'
    ],
    [
      Sequelize.literal(
        `(SELECT true FROM "LikedBooks" lb WHERE lb."bookId" = "Book"."id" AND lb."userId" = ${userId} LIMIT 1)`
      ),
      'isLiked'
    ]
  );
}

options.include = includes.include;

if (query.sortField)
  options.order = [[query.sortField, query.sortDirection]];

if (query.search && query.searchKeys?.length) {
  options.where = {
    [Op.or]: query.searchKeys.map((key) => {
      return {
        [key]: { [Op.iLike]: '%' + query.search + '%' }
      };
    })
  };
}

const search = [];
if (query.search && query.searchKeys?.length) {
  search.push({
    [Op.or]: query.searchKeys.map((key) => {
      return {
```

```
        [key]: { [Op.iLike]: '%' + query.search + '%' }
      };
    })
  });
 }

 options.where = {
  [Op.and]: search
 };

 if (query.authors) {
  options.where['authorId'] = { [Op.in]: query.authors };
 }

 if (query.genres) {
  options.where['genreId'] = { [Op.in]: query.genres };
 }

 if (query.ratings && query.include && query.include.includes('ratings')) {
  options.where[Op.and].push(
   Sequelize.literal(
    '(SELECT AVG(br."rating") FROM "BooksRatings" br WHERE br."bookId" = "Book"."id" GROUP BY br."bookId") IN (:ratings)'
   )
  );
  options.replacements = { ratings: query.ratings };
 }

 return this.bookModel.findAndCountAll(options);
}

async findLiked(
 query: GetItemsDto,
 userId: number
): Promise<{ rows: Book[]; count: number }> {
 const options: FindAndCountOptions = {
  subQuery: false,
  distinct: true,
  offset: query.start,
  limit: query.count
 };

 const includes = this.createIncludes(query.include);
```

```
options.attributes = [
  ...(query.fields
    ? query.fields
    : [
        'id',
        'title',
        'description',
        'pagesNumber',
        'genreId',
        'authorId',
        'attachmentId'
      ].filter((item) => !includes.attributes.exclude.includes(item))),
  ...includes.attributes.include
];

if (userId) {
  includes.include.push({
    model: BooksRating,
    as: 'userRating',
    required: false,
    attributes: ['rating'],
    where: {
      userId: userId
    }
  });
  options.attributes.push(
    [
      Sequelize.literal(
        `(SELECT true FROM "ReadBooks" rb WHERE rb."bookId" = "Book"."id" AND rb."userId" = ${userId} LIMIT 1)`
      ),
      'isRead'
    ],
    [
      Sequelize.literal(
        `(SELECT true FROM "LikedBooks" lb WHERE lb."bookId" = "Book"."id" AND lb."userId" = ${userId} LIMIT 1)`
      ),
      'isLiked'
    ]
  );
}
```

```
  options.include = includes.include;

  options.where = {
    [Op.and]: [
      Sequelize.literal(
        `(SELECT true FROM "LikedBooks" lb WHERE lb."bookId" = "Book"."id" AND lb."userId" = ${userId} LIMIT 1) = true`
      )
    ]
  };

  return this.bookModel.findAndCountAll(options);
}

async findRead(
  query: GetItemsDto,
  userId: number
): Promise<{ rows: Book[]; count: number }> {
  const options: FindAndCountOptions = {
    subQuery: false,
    distinct: true,
    offset: query.start,
    limit: query.count
  };

  const includes = this.createIncludes(query.include);

  options.attributes = [
    ...(query.fields
      ? query.fields
      : [
          'id',
          'title',
          'description',
          'pagesNumber',
          'genreId',
          'authorId',
          'attachmentId'
        ].filter((item) => !includes.attributes.exclude.includes(item))),
    ...includes.attributes.include
  ];

  if (userId) {
```

```
    includes.include.push({
      model: BooksRating,
      as: 'userRating',
      required: false,
      attributes: ['rating'],
      where: {
        userId: userId
      }
    });
    options.attributes.push(
      [
        Sequelize.literal(
          `(SELECT true FROM "ReadBooks" rb WHERE rb."bookId" = "Book"."id" AND rb."userId" = ${userId} LIMIT 1)`
        ),
        'isRead'
      ],
      [
        Sequelize.literal(
          `(SELECT true FROM "LikedBooks" lb WHERE lb."bookId" = "Book"."id" AND lb."userId" = ${userId} LIMIT 1)`
        ),
        'isLiked'
      ]
    );
  }

  options.include = includes.include;

  options.where = {
    [Op.and]: [
      Sequelize.literal(
        `(SELECT true FROM "ReadBooks" rb WHERE rb."bookId" = "Book"."id" AND rb."userId" = ${userId} LIMIT 1) = true`
      )
    ]
  };

  return this.bookModel.findAndCountAll(options);
}

createIncludes(includes: string[]): {
  include: any[];
  attributes: { include: any[]; exclude: any[] };
} {
```

```
const include = [];
const attributesInclude = [];
const attributesExclude = [];
if (includes) {
  if (includes.includes('ratings')) {
    attributesInclude.push([
      Sequelize.literal(
        '(SELECT AVG(br."rating") FROM "BooksRatings" br WHERE br."bookId" = "Book"."id" GROUP BY br."bookId")'
      ),
      'rating'
    ]);
  }

  if (includes.includes('genres')) {
    include.push({
      model: Genre,
      as: 'genre'
    });
    attributesExclude.push('genreId');
  }

  if (includes.includes('authors')) {
    include.push({
      model: Author,
      as: 'author'
    });
    attributesExclude.push('authorId');
  }

  if (includes.includes('attachments')) {
    include.push({
      model: Attachment,
      as: 'attachment'
    });
    attributesExclude.push('attachmentId');
  }

  if (includes.includes('files')) {
    include.push({
      model: Attachment,
      as: 'file'
    });
```

```
      attributesExclude.push('fileId');
    }
  }

  return {
    include,
    attributes: { include: attributesInclude, exclude: attributesExclude }
  };
}

async rate(id: number, rateBookDto: RateBookDto, userId): Promise<Book> {
  const bookRating = await this.booksRatingModel.findOne({
    where: { userId: userId, bookId: id }
  });

  if (bookRating) {
    await bookRating.update({
      userId: userId,
      rating: rateBookDto.rating,
      bookId: id
    });
  } else {
    await this.booksRatingModel.create({
      userId: userId,
      rating: rateBookDto.rating,
      bookId: id
    });
  }

  return this.bookModel.findByPk(id);
}

async remove(id: number): Promise<void> {
  const book = await this.bookModel.findByPk(id);

  return book.destroy();
}

async pagesRead(userId: number): Promise<{ pagesRead: number }> {
  const books = await this.readBooksModel.findAll({
    attributes: [
      [Sequelize.fn('sum', Sequelize.col('book.pagesNumber')), 'pagesRead']
```

```
      ],
      include: [{ model: Book, as: 'book', attributes: [] }],
      where: { userId: userId },
      raw: true
    });

    if (books && books[0]['pagesRead']) {
      return { pagesRead: books[0]['pagesRead'] };
    } else {
      return { pagesRead: 0 };
    }
  }
}
```

users.controller.ts

```
import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Post,
  Query,
  UseGuards,
  Patch,
  Response
} from '@nestjs/common';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { UsersService } from './users.service';
import { CreateUserDto } from './dto/create-user.dto';
import { User } from './models/user.model';
import { GetUsersDto } from './dto/get-users.dto';
import { EmailDto } from './dto/email.dto';
import { ConfigService } from '@nestjs/config';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { ChangePasswordDto } from './dto/change-password.dto';
import * as UserDecorator from '../auth/decorators/user.decorator';
import { CompleteRegistrationDto } from './dto/complete-registration.dto';
import { Friendship } from './models/friendship.model';

@Controller('users')
@ApiTags('users')
export class UsersController {
```

```
constructor(
  private readonly usersService: UsersService,
  private readonly configService: ConfigService
) {}

@Post('register')
async create(@Body() createUserDto: CreateUserDto): Promise<User> {
  return this.usersService.create(createUserDto);
}

@ApiBearerAuth()
@Get('register/complete')
async completeRegistration(
  @Query() completeRegistrationDto: CompleteRegistrationDto,
  @Response() res
): Promise<void> {
  await this.usersService.completeRegistration(completeRegistrationDto);
  res.redirect(301, this.configService.get('PUBLIC_SITE_URL'));
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Get()
findAll(
  @Query() query: GetUsersDto
): Promise<{ rows: User[]; count: number }> {
  return this.usersService.findAll(query);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Patch()
update(
  @UserDecorator.User() user: any,
  @Body('username') username: string
): Promise<number> {
  return this.usersService.update(user.id, username);
}

@ApiBearerAuth()
@UseGuards(JwtAuthGuard)
@Delete(':id')
```

```
  remove(@Param('id') id: number, @UserDecorator.User() req) {
    return this.usersService.remove(id, req.user.id);
  }

  @Get('changePassword')
  async changePassword(
    @Body() changePasswordDto: ChangePasswordDto
  ): Promise<void> {
    await this.usersService.changePassword(changePasswordDto);
  }

  @Post('emailConfirmation')
  sendConfirmation(@Body() emailDto: EmailDto): Promise<void> {
    return this.usersService.sendPasswordConfirmation(emailDto);
  }

  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Get('friends/list')
  async getFriendsList(@UserDecorator.User() user: any): Promise<Friendship[]> {
    return this.usersService.getFriendsList(user.id);
  }

  @ApiBearerAuth()
  @UseGuards(JwtAuthGuard)
  @Get('friends/requests')
  async getFriendsRequestsList(
    @UserDecorator.User() user: any
  ): Promise<{ forMe: Friendship[]; my: Friendship[] }> {
    return this.usersService.getFriendsRequestsList(user.id);
  }
}
```

users.service.ts

```
import { BadRequestException, HttpException, Injectable } from '@nestjs/common';
import { User } from './models/user.model';
import { InjectModel } from '@nestjs/sequelize';
import { Sequelize } from 'sequelize-typescript';
import { CreateUserDto } from './dto/create-user.dto';
import { MailsService } from '../mails/mails.service';
import { GetUsersDto } from './dto/get-users.dto';
import { FindAndCountOptions, Op } from 'sequelize';
import { EmailDto } from './dto/email.dto';
```

```
import { EXCEPTION_MESSAGES } from '../constants';
import * as randomize from 'randomatic';
import * as bcrypt from 'bcrypt';
import * as process from 'process';
import { ChangePasswordDto } from './dto/change-password.dto';
import { CompleteRegistrationDto } from './dto/complete-registration.dto';
import { ConfigService } from '@nestjs/config';
import { Friendship } from './models/friendship.model';

@Injectable()
export class UsersService {
  constructor(
    @InjectModel(User)
    private userModel: typeof User,
    @InjectModel(Friendship)
    private friendshipModel: typeof Friendship,
    private readonly configService: ConfigService,
    private readonly mailsService: MailsService,
    private sequelize: Sequelize
  ) {}

  async create(createUserDto: CreateUserDto): Promise<User> {
    return this.sequelize.transaction(async (transaction) => {
      const nowDate = new Date(Date.now());
      const confirmationCode = randomize('0', 10);
      const passwordHash = await bcrypt.hash(createUserDto.password, 12);

      const existedUser = await this.findOne(createUserDto.email);

      if (existedUser) {
        throw new BadRequestException(
          EXCEPTION_MESSAGES.USER_WITH_EMAIL_ALREADY_EXISTS
        );
      }

      const user = await this.userModel.create(
        {
          ...createUserDto,
          isEmailConfirmed: false,
          emailConfirmationCode: confirmationCode,
          createdAt: nowDate,
          password: passwordHash
```

```
      },
      { transaction }
    );

    await this.mailsService.sendEmailByTemplate({
      recipients: [createUserDto.email],
      subject: 'Підтвердження реєстрації',
      template: 'email-confirmation.hbs',
      context: {
        siteUrl: this.configService.get('host'),
        confirmationUrl:
          this.configService.get('host') +
          `/api/users/register/complete?email=${createUserDto.email}&code=${confirmationCode}`
      }
    });

    const userData = user.toJSON();

    delete userData.emailConfirmationCode;

    return userData;
  });
}

async findOne(email: string): Promise<User> {
  return this.userModel.findOne({
    where: Sequelize.where(
      Sequelize.fn('LOWER', Sequelize.col('email')),
      '=',
      email.toLowerCase()
    )
  });
}

async findById(id: number): Promise<User> {
  return this.userModel.findByPk(id);
}

async findAll(query: GetUsersDto): Promise<{ rows: User[]; count: number }> {
  const options: FindAndCountOptions = {
    offset: query.start,
    limit: query.count,
```

```
    attributes: query.fields,
    where: {}
  };

  if (query.sortField)
    options.order = [[query.sortField, query.sortDirection]];

  if (query.search && query.searchKeys?.length) {
    options.where = {
      [Op.or]: query.searchKeys.map((key) => {
        return {
          [key]: { [Op.iLike]: '%' + query.search + '%' }
        };
      })
    };
  }

  return this.userModel.findAndCountAll(options);
}

async update(id: number, username: string): Promise<number> {
  const user = await this.userModel.findByPk(id);

  if (user) {
    await user.update({ username: username });
  }

  return user.id;
}

async remove(id: number, userId: number): Promise<void> {
  if (id !== userId) {
    throw new HttpException('Forbidden resource', 403);
  }
  const user = await this.userModel.findByPk(id, { rejectOnEmpty: true });

  await user.destroy();
}

async changePassword(changePasswordDto: ChangePasswordDto): Promise<void> {
  const user = await this.userModel.findOne({
    where: {
```

```
      email: changePasswordDto.email,
      passwordConfirmationCode: changePasswordDto.passwordConfirmationCode
    }
  });

  if (user) {
    const passwordHash = await bcrypt.hash(changePasswordDto.password, 12);
    await user.update({
      password: passwordHash,
      passwordConfirmationCode: null
    });
  } else {
    throw new BadRequestException(EXCEPTION_MESSAGES.WRONG_CONFIRMATION_CODE);
  }
}

async sendPasswordConfirmation(
  emailConfirmationDto: EmailDto
): Promise<void> {
  return this.sequelize.transaction(async (transaction) => {
    const confirmation = randomize('0', 10);

    const user = await this.userModel.findOne({
      where: { email: emailConfirmationDto.email }
    });

    if (user) {
      await user.update(
        { emailConfirmationCode: confirmation },
        { transaction }
      );
      await this.mailsService.sendEmailByTemplate({
        recipients: [emailConfirmationDto.email],
        subject: 'Підтвердження зміни пароля',
        template: 'password-change-confirmation.hbs',
        context: {
          confirmationUrl: `${process.env.SITE_URL}/api/users/changePassword?email=${emailConfirmationDto.email}&code=${confirmation}`
        }
      });
    } else {
      throw new BadRequestException(EXCEPTION_MESSAGES.USER_NOT_FOUND);
    }
```

```
  });
}

async completeRegistration(
  completeRegistrationDto: CompleteRegistrationDto
): Promise<void> {
  const user = await this.findOne(completeRegistrationDto.email);

  if (user && !user.isEmailConfirmed) {
    if (user.emailConfirmationCode === completeRegistrationDto.code) {
      await user.update({
        isEmailConfirmed: true
      });
    } else {
      throw new BadRequestException(
        EXCEPTION_MESSAGES.WRONG_CONFIRMATION_CODE
      );
    }
  }
}

async getFriendsList(userId: number) {
  return this.friendshipModel.findAll({
    where: {
      isAccepted: true,
      [Op.or]: [{ requesterId: userId }, { addresseeId: userId }]
    },
    include: [
      {
        model: User,
        as: 'requester',
        attributes: ['id', 'username']
      },
      {
        model: User,
        as: 'addressee',
        attributes: ['id', 'username']
      }
    ]
  });
}
```

```
async getFriendsRequestsList(userId: number) {
  const myRequests = await this.friendshipModel.findAll({
    attributes: ['requesterId'],
    where: {
      isAccepted: false,
      requesterId: userId
    },
    include: [
      {
        model: User,
        as: 'requester',
        attributes: ['id', 'username']
      },
      {
        model: User,
        as: 'addressee',
        attributes: ['id', 'username']
      }
    ]
  });

  const requestsToMe = await this.friendshipModel.findAll({
    attributes: ['requesterId'],
    where: {
      isAccepted: false,
      addresseeId: userId
    },
    include: [
      {
        model: User,
        as: 'requester',
        attributes: ['id', 'username']
      },
      {
        model: User,
        as: 'addressee',
        attributes: ['id', 'username']
      }
    ]
  });

  return { my: myRequests, forMe: requestsToMe };
```

```
  }
}
```

app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
import configuration from './config/configuration';
import { AuthModule } from './auth/auth.module';
import { SequelizeModule } from '@nestjs/sequelize';
import { UsersModule } from './users/users.module';
import { BooksModule } from './books/books.module';
import { CollectionsModule } from './books/collections/collections.module';
import { AttachmentsModule } from './attachments/attachments.module';
import { GoalsModule } from './goals/goals.module';

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      load: [configuration],
      envFilePath: '.env'
    }),
    SequelizeModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => ({
        dialect: 'postgres',
        host: configService.get('database.host'),
        port: configService.get<number>('database.port'),
        username: configService.get('database.user'),
        password: configService.get('database.password'),
        database: configService.get('database.name'),
        autoLoadModels: true,
        synchronize: false,
        define: {
          timestamps: false
        }
      }),
      inject: [ConfigService]
    }),
    AuthModule,
    UsersModule,
```

```
    BooksModule,
    CollectionsModule,
    AttachmentsModule,
    GoalsModule
  ],
  controllers: [AppController],
  providers: [AppService]
})
export class AppModule {}
```

main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ConfigService } from '@nestjs/config';
import { ValidationPipe } from '@nestjs/common';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const configService = app.get(ConfigService);

  app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true }));
  app.setGlobalPrefix('api');

  const swaggerConfig = new DocumentBuilder()
    .setTitle('Library')
    .setDescription('Backend for Library project')
    .setVersion(process.env.npm_package_version)
    .addBearerAuth()
    .build();
  const swaggerDocument = SwaggerModule.createDocument(app, swaggerConfig);
  SwaggerModule.setup('api/docs', app, swaggerDocument);

  app.enableCors({
    origin:
      configService.get('nodeEnv') == 'development'
        ? true
        : [configService.get('host')],
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS',
    credentials: true
  });
```

```
  await app.listen(configService.get<number>('port'));
}
bootstrap();
```

AuthModal.jsx

```
import React, { useState } from 'react';
import SignInForm from '../SignInForm';
import SignUpForm from '../SignUpForm';

const AuthModal = ({ onClose }) => {
  const [isRegistered, setIsRegistered] = useState(true);

  return (
    <div
      onClick={onClose}
      className="fixed inset-0 z-10 bg-gray-500 bg-opacity-75 transition-opacity before:h-full before:inline-block before:align-middle text-[0] text-center"
    >
      <div className="inline-block align-middle text-left text-base">
        <div onClick={(e) => e.stopPropagation()} className="p-4 border-white border-2 rounded-2xl bg-white">
          {isRegistered ? (
            <SignInForm handleClose={onClose} setIsRegistered={setIsRegistered} />
          ) : (
            <SignUpForm handleClose={onClose} setIsRegistered={setIsRegistered} />
          )}
        </div>
      </div>
    </div>
  );
};

export default AuthModal;
```

bookCard.jsx

```
import React, { useState } from 'react';
import { AiFillStar, AiOutlineHeart, AiFillHeart } from 'react-icons/ai';
import { toast } from 'react-toastify';
import ApiService from '../../helpers/api-helpers';
const booksApi = new ApiService('books/liked');

const BookCard = ({ book }) => {
  const [bookIsLiked, setBookIsLiked] = useState(book?.isLiked);
  const addToLiked = () => {
    const token = localStorage.getItem('token');
```

```
  if (token) {
    if (bookIsLiked) {
      booksApi
        .deleteItem(book.id)
        .then(() => {
          setBookIsLiked(false);
          toast.success('Book removed from liked!');
        })
        .catch();
    } else {
      booksApi
        .createItem({ id: book.id })
        .then(() => {
          setBookIsLiked(true);
          toast.success('Book added to liked!');
        })
        .catch();
    }
  }
};


 return (
   <div className="flex flex-col justify-center h-[90%] my-5">
     <div className="relative flex flex-col w-[250px] lg:w-[400px] lg:flex-row h-[400px] lg:h-[200px] lg:space-x-5 space-y-3 lg:space-y-0 rounded-xl shadow-xl p-3 max-w-xs lg:max-w-[400px] mx-auto border border-2 bg-white">
       <div className="w-full lg:w-1/3 bg-white grid place-items-center">
         <img src={`http://localhost/uploads/${book.attachment.storageKey}`} alt="" className="h-44 w-32 rounded-xl" />
       </div>
       <div className="w-full lg:w-2/3 bg-white flex flex-col space-y-2 p-3">
         <div className="flex justify-between item-center">
           <div className="flex items-center">
             <AiFillStar color="orange" size={20} />
             <p className="text-gray-600 font-bold text-sm ml-1">{parseInt(book.rating) || 0} / 5</p>
           </div>
           <button
             onClick={addToLiked}
             className="relative z-10 bg-gray-200 px-3 py-1 rounded-full text-sm font-medium text-gray-800"
           >
             <div className="inline-block ps-2 py-auto">Like</div>
             {bookIsLiked ? (
               <AiFillHeart size={20} color="red" className="inline-block my-1 mx-2" />
             ) : (
               <AiOutlineHeart size={20} className="inline-block my-1 mx-2" />
```

```
          )}
        </button>
      </div>
      <a href={`books/${book.id}`}>
        <div className="text-black text-lg text-red-900 font-bold">
          {book.title}
          <span className="absolute w-full h-full top-0 left-0"></span>
        </div>
      </a>
      <div className="flex-col justify-between">
        <p className="font-bold text-gray-800 text-base">
          Author:{' '}
          <span className="font-normal text-gray-600 text-base">
            {`${book.author.firstName} ${book.author.lastName}`}
          </span>
        </p>
        <p className="font-bold text-gray-800 text-base">
          Genre: <span className="font-normal text-gray-600 text-base">{book.genre?.title}</span>
        </p>
      </div>
    </div>
  </div>
 );
};

export default BookCard;
```

BooksList.jsx

```
import React from 'react';
import BookCard from './BookCard';

const BooksList = ({ title, books }) => {
 return (
  <div className="px-5">
   <h2 className="flex font-bold text-gray-500 text-xl pb-2">{title}</h2>
   <div className="flex flex-wrap justify-center sm:justify-between px-auto md:px-14 xl:px-2 2xl:px-12 sm:px-5 border border-2 rounded rounded-lg">
    {books && books.length > 0 ? (
     books.map((book) => <BookCard key={book.id} book={book} />)
    ) : (
     <div className="w-full text-2xl text-center py-2">No books found</div>
    )}
   </div>
```

```
    </div>
  );
};

export default BooksList;
```

StarRating.jsx

```
import React, { useState } from 'react';
import { AiFillStar } from 'react-icons/ai';
import { toast } from 'react-toastify';
import { getResourceFromApi, methods } from '../../helpers/api-helpers';

const StarRating = ({ bookId, userRating }) => {
  const [rating, setRating] = useState(userRating ? userRating.rating : null);
  const rateBook = ({ id, ...other }) =>
    getResourceFromApi('books/:id/rate'.replace(':id', id), null, methods.patch, other);

  const onClick = (e, newRating) => {
    e.preventDefault();
    rateBook({ id: bookId, rating: newRating }).then(() => {
      setRating(newRating);
      toast.success('Rating has been updated!');
    });
  };

  return (
    <div className="flex items-center">
      {[1, 2, 3, 4, 5].map((item) => (
        <button>
          <AiFillStar
            onClick={(e) => onClick(e, item)}
            key={item}
            fill={!rating || item > rating ? 'gray' : 'orange'}
            size="100%"
          />
        </button>
      ))}
    </div>
  );
};

export default StarRating;
```

GenresSwiper.jsx

```
import React, { useEffect, useState } from 'react';
import { Swiper, SwiperSlide } from 'swiper/react';
import { EffectCoverflow, Pagination } from 'swiper';
import { useNavigate } from 'react-router-dom';

import 'swiper/css';
import 'swiper/css/effect-coverflow';
import 'swiper/css/pagination';
import './style.css';
import ApiService from '../../helpers/api-helpers';

const genresApi = new ApiService('books/genres');

const GenresSwiper = () => {
  const [genres, setGenres] = useState([]);
  const navigate = useNavigate();

  useEffect(() => {
    genresApi
      .getAllItems({
        count: 10,
        include: 'attachments'
      })
      .then((res) => {
        setGenres(res.rows);
      });
  }, []);

  return (
    genres && (
      <div className="block px-5">
        <h2 className="w-full text-left text-xl text-gray-500 font-bold">Popular genres</h2>
        <Swiper
          effect={'coverflow'}
          grabCursor={true}
          centeredSlides={true}
          slidesPerView={'auto'}
          initialSlide={4}
          coverflowEffect={{
            rotate: 50,
            stretch: 0,
            depth: 100,
```

```
          modifier: 1,
          slideShadows: true
        }}
        pagination={true}
        modules={[EffectCoverflow, Pagination]}
        className="mySwiper"
      >
        {genres.map((genre) => (
          <SwiperSlide onClick={() => navigate(`/genres/${genre.id}`)}>
            <img src={`http://localhost/uploads/${genre.attachment.storageKey}`} alt={genre.attachment.filename} />
          </SwiperSlide>
        ))}
      </Swiper>
    </div>
  )
 );
};

export default GenresSwiper;
```

Layout.jsx

```
import React, { useState } from 'react';
import Footer from './Footer';
import Header from './Header';

const Layout = ({ children }) => {
 const [isModalOpen, setIsModalOpen] = useState(false);

 const setModalState = (value) => {
  setIsModalOpen(value);
 };

 return (
  <>
   <Header setModalOpen={setModalState} />
   <div className="min-h-[750px]">{children}</div>
   <Footer />
  </>
 );
};

export default Layout;
```

Header.jsx

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import SignInButton from './SignInButton';
import MobileMenu from './MobileMenu';
import { GiHamburgerMenu } from 'react-icons/gi';

const Header = () => {
  const [nav, setNav] = useState(true);

  const handleNav = () => {
    setNav(!nav);
  };

  return (
    <header className="w-full container px-4">
      <div className="flex justify-between items-center h-full my-5">
        <a href="/">
          <img src="/logo.png" alt="logo" className={'w-20'} />
        </a>
        <ul className="hidden px-5 md:flex">
          <li className="px-5 py-2">
            <Link to="/">Home</Link>
          </li>
          <li className="px-5 py-2">
            <Link to="/books">Books</Link>
          </li>
          <li className="px-5 py-2">
            <Link to="/about">About</Link>
          </li>
        </ul>
        <div className="hidden md:flex">
          <SignInButton />
        </div>

        <div onClick={handleNav} className="fixed right-5 top-12 md:hidden">
          {nav && <GiHamburgerMenu size={20} />}
        </div>
      </div>
      <MobileMenu nav={{ nav, handleNav }} />
    </header>
  );
};
```

```
export default Header;
```

Footer.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';

const Footer = () => {
  return (
    <footer className="h-15 container px-4 mt-4">
      <ul className="block sm:flex justify-between items-center">
        <li className="py-2">2023 Online Library. All rights reserved.</li>
        <li className="py-2">
          <Link to="/help">Help & Support</Link>
        </li>
        <li className="py-2">
          <Link to="/about">About</Link>
        </li>
        <li>
          <Link to="/">
            <img src="/footer-logo.png" alt="footer-logo" className="w-12" />
          </Link>
        </li>
      </ul>
    </footer>
  );
};

export default Footer;
```

api-helpers.js

```
export const methods = {
  get: 'GET',
  post: 'POST',
  patch: 'PATCH',
  delete: 'DELETE'
};

export const getResourceFromApi = async (url, query, method = methods.get, body, headers = {}) => {
  const token = localStorage.getItem('token');
  const absUrl = new URL(`${process.env.REACT_APP_API_HOST}/${url}`);

  if (query) {
    Object.keys(query).forEach((key) => absUrl.searchParams.append(key, query[key]));
```

```
  }

  if (body) {
    body = JSON.stringify(body);
    headers['Content-Type'] = 'application/json';
  }

  if (token) {
    headers.Authorization = `${token}`;
  }

  const res = await fetch(absUrl, {
    method,
    body,
    headers: {
      ...headers
    }
  });

  const contentType = res.headers.get('Content-Type');
  const contentLength = res.headers.get('Content-Length');

  const checkRes = async (type) => {
    const resData = type.includes('application/json') ? await res.json() : await res.blob();

    if (res.ok) return resData?.data ? resData.data : resData;

    if (!res.ok) {
      throw resData?.message;
    }
  };

  return +contentLength > 0 ? checkRes(contentType) : null;
};

export default class ApiService {
  constructor(apiBase) {
    this.apiBase = apiBase;
  }

  getResource = getResourceFromApi;
```

```
  getAllItems = async (query) => this.getResource(this.apiBase, query);

  getItemById = async (id, query) => this.getResource(`${this.apiBase}/${id}`, query);

  createItem = async (body) => this.getResource(this.apiBase, null, methods.post, body);

  updateItem = async ({ id, ...other }) => this.getResource(`${this.apiBase}/${id}`, null, methods.patch, other);

  deleteItem = async (id) => this.getResource(`${this.apiBase}/${id}`, null, methods.delete);
}

export const postFiles = async (files) => {
  return await Promise.all(files.map((f) => postFile(f)));
};

export const postFile = async (file) => {
  const formData = new FormData();
  formData.append('files', file);

  const token = localStorage.getItem('token');

  const options = {
    method: methods.post,
    body: formData
  };

  if (token) {
    options.header = {
      Authorization: token
    };
  }

  const response = fetch(`${process.env.REACT_APP_API_HOST}/attachments/upload`, options).then((res) => res.json());
  const res = await response;
  return res[0];
};

export const deleteFiles = async (ids = []) => {
  return await getResourceFromApi('attachments', null, methods.delete, ids);
};

export const deleteFile = (id) => deleteFiles([id]);
```

```
export const getImageSrc = (id) => `${process.env.REACT_APP_API_HOST}/attachments/${id}`;

export const getAttachmentUrl = (storageKey) => `${process.env.REACT_APP_PUBLIC_HOST}/uploads/${storageKey}`;
```

Main.jsx

```
import React, { useEffect, useState } from 'react';
import BooksList from '../../components/BooksList';
import GenresSwiper from '../../components/GenresSwiper';
import ApiService from '../../helpers/api-helpers';

const booksApi = new ApiService('books');

const Main = () => {
  const [books, setBooks] = useState([]);

  useEffect(() => {
    const payload = {
      include: 'genres,authors,ratings,attachments,files',
      count: 6,
      sortField: 'id',
      sortDirection: 'asc',
      start: 0
    };
    booksApi.getAllItems(payload).then((res) => {
      setBooks(res.rows);
    });
  }, []);

  return (
    <div className="container">
      <GenresSwiper />
      <BooksList title={'Top rated books'} books={books} />
    </div>
  );
};

export default Main;
```

Profile.jsx

```
import React, { useEffect, useState } from 'react';
import { GrClose } from 'react-icons/gr';
import { AiOutlineMenuUnfold } from 'react-icons/ai';
import { GiBookshelf } from 'react-icons/gi';
```

```
import { FaBookReader, FaUsers } from 'react-icons/fa';
import { RiFileList2Fill, RiLogoutBoxRFill } from 'react-icons/ri';
import Card from './Card';
import ApiService from '../../helpers/api-helpers';
import { useNavigate } from 'react-router-dom';
const likedBooksApi = new ApiService('books/liked');
const readBooksApi = new ApiService('books/read');
const pagesApi = new ApiService('books/pages/read');

const Profile = () => {
  const [likedBooks, setLikedBooks] = useState([]);
  const [readBooks, setReadBooks] = useState([]);
  const [readBooksNumber, setReadBooksNumber] = useState(0);
  const [pagesRead, setPagesRead] = useState(0);
  const [isFiltersOpen, setIsFiltersOpen] = useState(false);
  const navigate = useNavigate();

  useEffect(() => {
    likedBooksApi
      .getAllItems({ include: 'ratings,authors,genres,attachments' })
      .then((res) => {
        setLikedBooks(res.rows);
      })
      .catch();
    readBooksApi
      .getAllItems({ include: 'ratings,authors,genres,attachments' })
      .then((res) => {
        setReadBooks(res.rows);
        setReadBooksNumber(res.count);
      })
      .catch();
    pagesApi
      .getAllItems({})
      .then((res) => {
        setPagesRead(res.pagesRead);
      })
      .catch();
  }, []);

  const onLogoutClick = () => {
    localStorage.removeItem('token');
    navigate('/');
```

```
  window.location.reload();
};

return (
  <div className="container">
    <div className="flex justify-between">
      <div className="xl:w-[20%] xl:px-4">
        <div
          className={`fixed xl:static left-0 inset-y-0 z-20 p-6 xl:p-0 bg-zinc-300 xl:bg-transparent ${
            isFiltersOpen ? '' : 'xl:translate-x-0 -translate-x-full'
          } transition-transform`}
        >
          <div className="flex justify-between items-center">
            <h2 className="flex font-bold text-gray-500 text-xl pb-2 pl-2">My Stats</h2>
            <div className="xl:hidden px-5">
              <button onClick={() => setIsFiltersOpen(false)}>
                <GrClose />
              </button>
            </div>
          </div>
          <div className="px-2 pb-2 border-2 border-gray-300 rounded-xl">
            <div className="flex py-1">
              <FaBookReader size={24} />
              <div className="w-full pl-2">Books finished</div>
              <div className="px-2">{readBooksNumber}</div>
            </div>
            <div className="flex py-1">
              <RiFileList2Fill size={24} />
              <div className="w-full pl-2">Pages read</div>
              <div className="px-2">{pagesRead}</div>
            </div>
            <div className="flex py-1 cursor-pointer">
              <GiBookshelf size={24} />
              <div className="w-full pl-2 underline">Collections</div>
              <div className="px-2">1</div>
            </div>
            <div className="flex py-1 cursor-pointer">
              <FaUsers size={24} />
              <div className="w-full pl-2 underline">Friends</div>
              <div className="px-2">1</div>
            </div>
          </div>
```

```
          <div
            onClick={onLogoutClick}
            className="flex px-2 py-2 my-2 border-2 border-gray-300 rounded-xl cursor-pointer"
          >
            <RiLogoutBoxRFill size={24} />
            <div className="w-full pl-2 underline">Logout</div>
          </div>
        </div>
      </div>
      <div className="xl:w-[75%] w-full">
        <div className="xl:hidden px-5">
          <button onClick={() => setIsFiltersOpen(true)}>
            <AiOutlineMenuUnfold size={20} />
          </button>
        </div>
        <div className="px-5">
          <h2 className="flex font-bold text-gray-500 text-xl pb-2">Liked books</h2>
          <div className="flex flex-col max-h-[350px] overflow-y-scroll scroll py-2 px-2 border border-2 border-gray-300 rounded rounded-lg">
            {likedBooks && likedBooks.length > 0 ? (
              likedBooks.map((book) => <Card key={book.id} book={book} />)
            ) : (
              <div className="w-full text-2xl text-center py-2">No books found</div>
            )}
          </div>

          <h2 className="flex font-bold text-gray-500 text-xl pt-6 pb-2">Read books</h2>
          <div className="flex flex-col max-h-[350px] overflow-y-scroll scroll py-2 px-2 border border-2 border-gray-300 rounded rounded-lg">
            {readBooks && readBooks.length > 0 ? (
              readBooks.map((book) => <Card key={book.id} book={book} />)
            ) : (
              <div className="w-full text-2xl text-center py-2">No books found</div>
            )}
          </div>
        </div>
      </div>
    </div>
  );
};

export default Profile;
```

App.jsx

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import Layout from './components/Layout';
import React, { useEffect, useState } from 'react';
import { ToastContainer } from 'react-toastify';
import { UserContext } from './UserContext';
import Main from './pages/Main';
import BookPage from './pages/BookPage';
import Books from './pages/Books';
import About from './pages/About';
import HelpAndSupport from './pages/HelpAndSupport';
import Profile from './pages/Profile';
import 'react-toastify/dist/ReactToastify.css';
import ApiService from './helpers/api-helpers';
const authApi = new ApiService('auth/profile');

function App() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    const token = localStorage.getItem('token');

    if (token) {
      authApi
        .createItem()
        .then((res) => setUser(res))
        .catch(() => {});
    }
  }, []);

  return (
    <div className="App">
      <BrowserRouter>
        <UserContext.Provider value={{ user, setUser }}>
          <Layout>
            <Routes>
              <Route path="/" element={<Main />} />
              <Route path="/books" element={<Books />} />
              <Route path="/books/:id" element={<BookPage />} />
              <Route path="/profile" element={<Profile />} />
              <Route path="/about" element={<About />} />
              <Route path="/help" element={<HelpAndSupport />} />
            </Routes>
```

```
          </Layout>
        </UserContext.Provider>
      </BrowserRouter>
      <ToastContainer position="top-right" autoClose={3000} closeOnClick pauseOnHover theme="light" />
    </div>
  );
}

export default App;
```