МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

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

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система для створення, публікації та поширення електронних книг»
здобувача (ки) групи ІН - 91 Короті Микити Івановича

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

_____ Микита КОРОТЯ
(підпис)

Керівник,
старший викладач, кандидат
технічних наук Борис Кузіков _____
(підпис)

Суми – 2023

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-ХХ Прізвище, Ім'я, по Батькові

1. Тема роботи: «Інформаційна система для створення, публікації та поширення електронних книг»
затверджую наказом по СумДУ від

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для прогнозування курсу валют. 3) Розробка інтелектуальної системи з прогнозування курсу валют. 4) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

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

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		

2	*Огляд технологій, що використовуються для прогнозування курсу валют*		
3	*Розробка інтелектуальної системи з прогнозування курсу валют*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 80 стр., 20 рис., 2 додатки, 20 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена створенню інформаційної системи для публікації книг. Така система буде надавати користувачам можливість публікації власних творів без контакту з видавництвами, що в час перенасичення контентом і пониженого інтересу до книжок є досить актуальним для нових авторів, які не є цікавими для видавництв, бо у них з'являється шанс поширити свій твір в маси.

Об'єкт дослідження — інформаційна система для створення, публікації та поширення електронних книг.

Мета роботи — розробка інформаційної системи для створення, публікації та поширення електронних книг.

Результати — розроблено інформаційну систему, яка дозволяє публікувати книги. Розроблено прототип дизайну веб-сайту.

ІНФОРМАЦІЙНА СИСТЕМА, LARAVEL, REACT, PHP, NODE.JS, JAVASCRIPT, MYSQL.

ЗМІСТ

ВСТУП

Електронні книги вже давно замінили паперові, а отже з’явися попит на зручні сервіси, які надають можливість шукати потрібну книгу, купувати книги, створювати власну бібліотеку книг. Зазвичай на таких ресурсах публікуються книги популярних видань.

Молодому автору дуже складно вмовити видання видати його книгу, бо видання має бути впевненим, що книгу будуть купувати, тому неохоче співпрацюють з невідомими письменниками. Частіше за все такі автори пишуть свої твори для близького оточення або для себе.

Тому ресурс, який надає можливість самостійно видавати книжки нині є доволі актуальним.

Отже, задачі роботи:

- Огляд існуючих інструментів для створення веб-додатків;
- Літературний огляд сучасних джерел за тематикою розробки веб-додатків;
- Формулювання вимог до веб-додатку;
- Вибір оптимальних інструментів.
- Розробка інформаційної системи для створення, публікації та поширення електронних книг.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

Нижче будуть розглянуті сайти, які реалізують найтиповіший функціонал для рішення вищеописаних задач.

1.1 Yakaboo

На рисунку 1.1 зображено головну сторінку сайту Yakaboo.

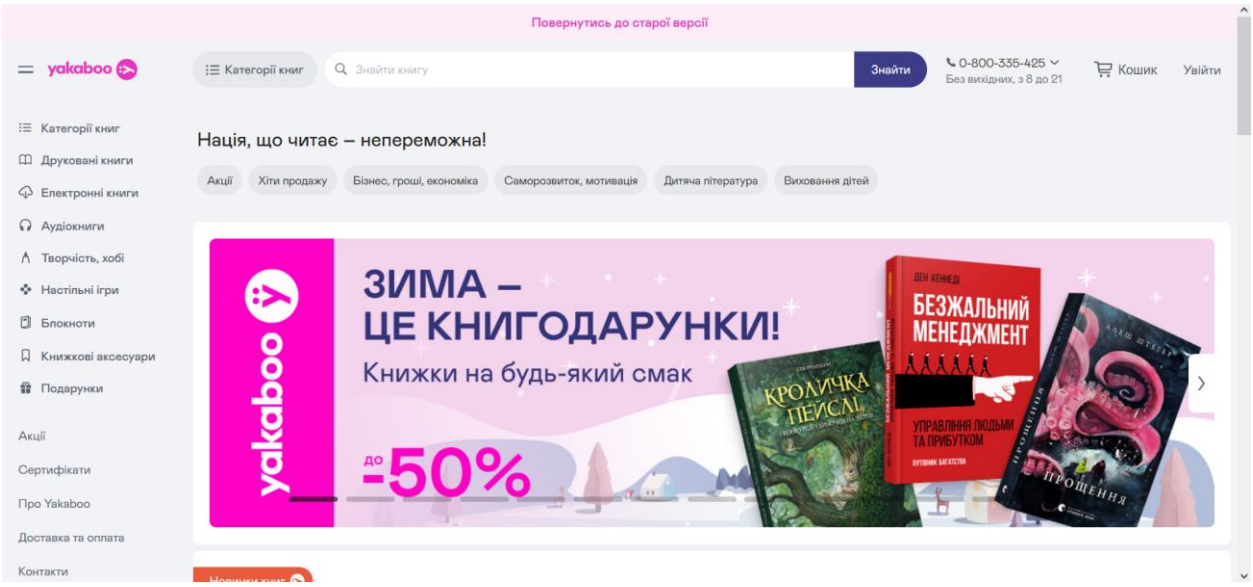

Рисунок 1.1. Головна сторінка сайту Yakaboo

Yakaboo – це інтернет ресурс, який надає користувачеві сервіс для пошуку і купівлі потрібної книги. Книги згруповані по категоріям і по жанрам. На головній сторінці можна переглянути останні надходження книг і отримати персональні рекомендації.

При пошуку книги можна фільтрувати за автором, за видавництвом і за ціною. Також є можливість створювати свою персональну бібліотеку. Кожну книгу можна оцінити і написати до неї рецензію. Перед тим як купити є можливість прочитати уривок книги.

Сайт створений як Single Page Application, тобто сторінка не оновлюється при переході по вкладкам.

1.2. Букнет

На рисунку 1.2 зображено головну сторінку сайту Букнет.

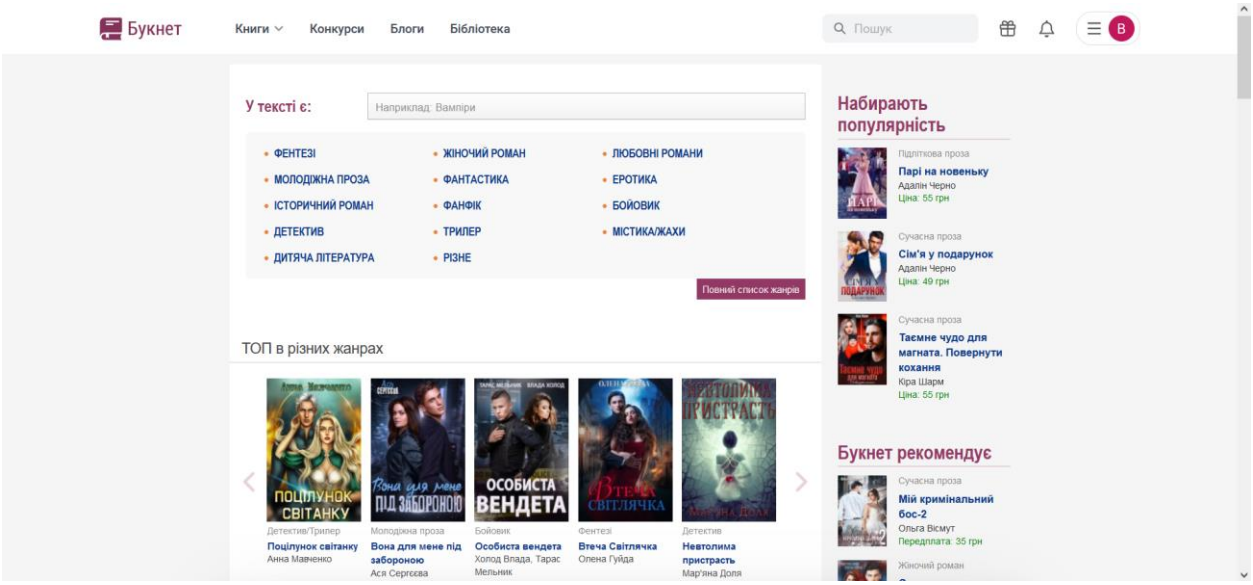

Рисунок 1.2. Головна сторінка сайту Букнет

Букнет надає майже такий самий сервіс, що і вищерозглянутий сайт, але додатково користувачі мають можливість стати авторами, тобто почати публікувати свої власні книги, які можна розповсюджувати безкоштовно або за гроші.

Цей сайт поєднує в собі сервіс для розповсюдження книг і щось на зразок соцмережі, де у кожного користувача є своя особиста сторінка, яку можуть переглядати інші користувачі і коментувати. Користувач може слідкувати за цікавими йому авторами використовуючи функцію підписки. Кожну книгу можна коментувати і лайкати.

Також користувачі мають можливість створювати блоги, де можуть писати про будь-що, але це доступно тільки для користувачів, які є авторами, тобто опублікували хоча б одну книгу.

Сайт має стандартну структуру і складається з багатьох документів, які підвантажуються при переході на відповідну сторінку.

1.3. Книгарня «Є»

На рисунку 1.3 зображено головну сторінку сайту Книгарня «Є».

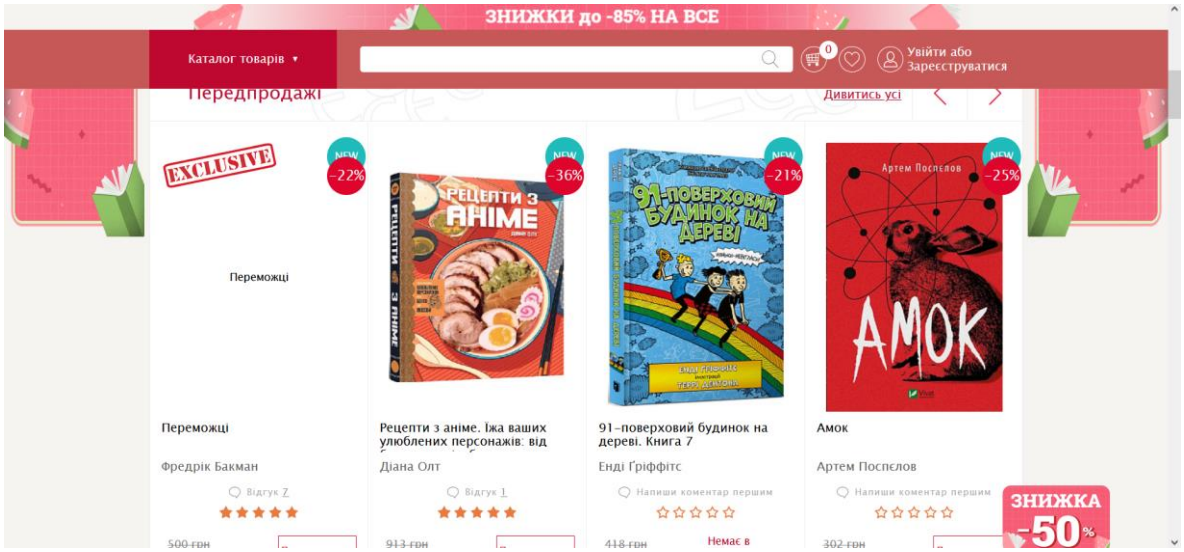

Рисунок 1.3 Головна сторінка сайту Книгарня «Є»

На відміну від інших ресурсів, цей сайт є сайтом фізичного магазину книжок, який надає функціонал для пошуку книг, а також замовлення їх фізичних копій. Тут нема можливості читати книгу онлайн, але можна прочитати уривок із книги і вирішити чи варто її купувати. У своєму профілю можна відслідковувати замовлення.

1.4. Amazon KDP

На рисунку 1.4 зображено сторінку сайту Amazon KDP.

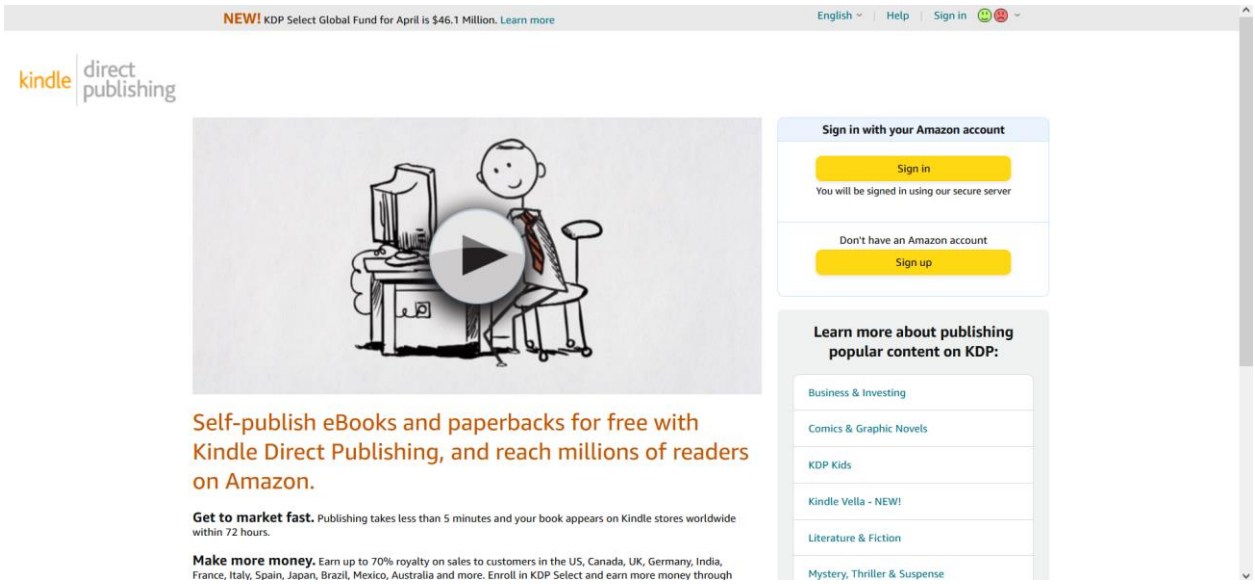

Рисунок 1.4 Головна сторінка сайту Amazon KDP

Цей ресурс є англомовним і надає можливість публікувати власноруч написані книги. На відміну від Букнет тут не можна вести свій блог, а також потрібен час на публікацію книги. Зазвичай це займає 72 години. Після публікації книга з’являється в інтернет-магазині і автор отримує можливість знайти своїх читачів. Читачі ж мають можливість коментувати книгу і ставити їй оцінку.

1.5. Огляд існуючих рішень

Отже, інтернет-ресурси для самостійної публікації книжок надають авторам ряд можливостей для публікації та поширення їх творів. Ось основні можливості, які можна знайти в багатьох інтернет-ресурсах для самостійної публікації книжок:

1. Завантаження та форматування контенту: Користувачам надається можливість завантажувати свої книжки у різних форматах, таких як

текстові файли (наприклад, PDF, DOC, TXT) або електронні книги (наприклад, ePub, MOBI). Ресурси також можуть надавати можливості форматування тексту, включаючи стилізацію заголовків, списків, розміщення зображень та інше.

2. Редактор контенту: Ресурси можуть мати вбудований редактор, який дозволяє авторам редагувати свої тексти, додавати зображення, використовувати форматування, перевіряти правопис тощо. Це дозволяє авторам створювати та вдосконалювати свої книжки прямо на платформі.
3. Керування метаданими: Авторам надається можливість встановлювати метадані для своїх книжок, такі як назва, автор, жанр, опис, обкладинка тощо. Це допомагає у пошуку та ідентифікації книжок на ресурсі.
4. Публікація та дистрибуція: Ресурси надають авторам можливість публікувати їх книжки та забезпечують механізми для їх поширення. Це може включати розміщення книжок у каталогу або магазині, де користувачі можуть знайти та завантажити їх, або надання прямих посилань для поширення творів через інші канали.

1.6 Постановка задачі

Букнет на даний момент єдиний помітний україномовний ресурс, який надає сервіс для самостійного видання книжок. При цьому існують фічі, які цей сайт не реалізує, але які будуть реалізовані у цьому проекті.

Букнет надає можливість публікувати блоги тільки користувачам, які є авторами, а ресурс, який розроблюється в ході виконання цієї роботи дозволить публікувати блоги всім користувачам, що перетворить блоги в щось на зразок форуму, де користувачі зможуть створювати теми і влаштовувати дискусії.

На кожну книгу можна поставити реакцію, що даю можливість користувачам швидко і лаконічно висловлювати думки стосовно книги.

До того ж, даний сайт буде реалізований як single page application.

Отже, необхідно реалізувати наступні функціональні вимоги:

1. Виконати огляд інструментів для розробки веб-сайтів і обрати оптимальні;
2. Розробити інтуїтивно зрозумілий інтерфейс для веб-сайту;
3. Виконати проектування архітектури веб-сайту;
4. Розробити базу даних, у якій зберігаються відомості про користувачів сайту і про книги.
5. Надати можливість користувачу переглядати наявні в базі даних книги, шукати потрібну книгу, сортувати книги по відповідним ключам.
6. Надати можливість користувачам публікувати власні книги.
7. Користувач може додавати книги до бібліотеки.
8. Реалізувати можливість читати книгу прямо на сайті використовуючи зручний інтерфейс.
9. Завантажити книгу можна тільки з дозволу автора.
10. Користувачі можуть додавати реакції до книги.
11. Реалізувати можливість ділитися думками про прочитану книгу.
12. Користувачі можуть створювати блоги.

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧ

На рисунку 2.1 представлено схему сторінок інтернет-ресурсу

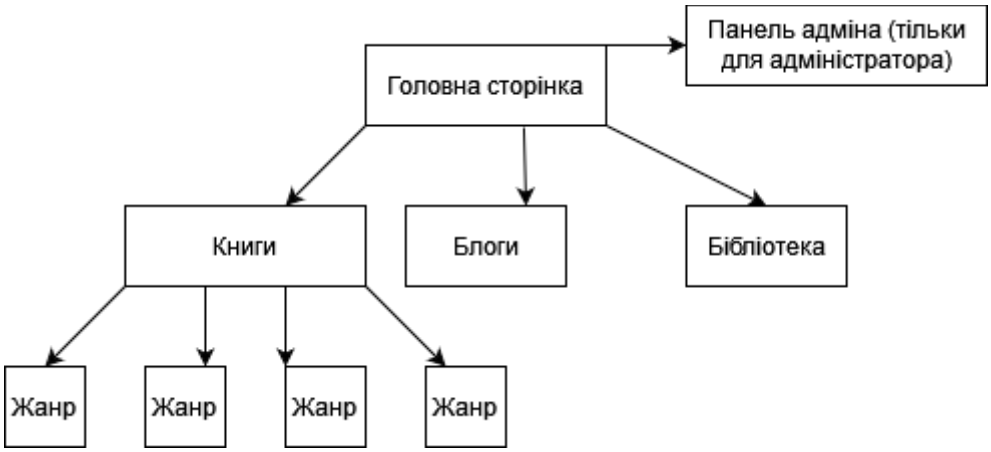

Рисунок 2.1. – Основні сторінки інтернет-ресурсу

Сайт буде спроектований так, що з кожної сторінки можна потрапити на будь-яку іншу сторінку.

2.1 Проектування бази даних

Для вирішення поставлених задач має бути реалізована база даних з такими таблицями:

- Books – таблиця, що зберігає відомості про книги.
- Book_genres – таблиця, що зберігає жанри книг. Жанри до бази даних може додавати лише адміністратор.
- Content – таблиця, що зберігає назви глав книги, а також вміст глави
- Tags – таблиця, що зберігає теги, які можна додати до книги. Це слова або словосполучення, які певним чином описують те, що очікує читача в книзі. Нові теги додаються до таблиці, коли користувач створює книгу з тегом, якого ще немає в базі даних.

- Book_tags – таблиця, що показує, які теги додані до книги. Потрібна для реалізації відношення many-to-many між таблицями books і tags.
- Users – таблиця в якій зберігаються користувачі. Інформація про нових користувачів додається до таблиці після їх реєстрації.
- Roles – таблиця, яка зберігає ролі користувачів. Кожен новий користувач отримує роль user за замовчуванням. Нові ролі може створювати адміністратор.
- Book_comments – таблиця, що зберігає коментарі до книг.
- Reactions – таблиця, що зберігає реакції, які можна поставити на книгу. Нові реакції може додавати до бази даних адміністратор.
- Book_user_reactions – таблиця, що зберігає реакції, які користувачі ставили на книги.
- Library – таблиця, у якій зберігаються бібліотеки користувачів.
- Posts – таблиця, у якій зберігаються записи до блогу користувачів.
- Post_comments – таблиця, у якій зберігаються коментарі до постів користувачів.
- Baskets – таблиця, у якій зберігаються корзини з товарами для оплати
- Basket_books – таблиця, у якій показано, які товари належать до певної корзини.

У таблиці 2.1 показана логічна реалізація бази даних.

Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
tags	id	Ідентифікатор тегу	BIGINT	PK	Not null, Unsigned
	title	Номер тегу	VARCHAR(45)		
book_tags	id	Ідентифікатор	BIGINT	PK	Not null, Unsigned
	book_id	Ідентифікатор книги	BIGINT	FK (зв’язок з books)	
	tag_id	Ідентифікатор тегу	BIGINT	FK (зв’язок з tags)	
books	id	Ідентифікатор книги	BIGINT	PK	Not null, unsigned
	name	Назва книги	VARCHAR(45)		
	Annotation	Короткий опис книги	TEXT		
	cover	Обкладинка книги	VARCHAR(255)		
	price	Ціна книги	INT		
	public_start	Початок публікації книги	DATETIME		
	public_end	Кінець публікації книги	DATETIME		
	book_genre_id	Жанр книги	BIGINT	FK (зв’язок з book_genres)	

	author_id	Автор книги	BIGINT	FK (зв’язок з users)	
book_genres	Id	Ідентифікатор жанру	BIGINT	PK	Not null, unsigned
	genre	Назва жанру	VARCHAR(45)		
content	Id	Ідентифікатор глави	BIGINT	PK	Not null, unsigned
	Title	Назва глави	VARCHAR(45)		
	Text	Вміст глави	LONGTEXT		
	Book_id	Ідентифікатор книги	BIGINT	FK (зв’язок з books)	Unsigned
users	id	Ідентифікатор користувача	BIGINT	PK	Not null, unsigned
	Name	Ім’я користувача	VARCHAR(45)		
	Email	Електронна скринька користувача	VARCHAR(255)		
	Role	Роль користувача	SMALLINT	FK (зв’язок з roles)	
	Email_verified_at	Дата верифікації елетронної скриньки	DATETIME		
	Password	Пароль користувача	VARCHAR(45)		Not null

	Remember_token	Унікальний ідентифікатор для сеансу	VARCHAR(255)		
roles	id	Ідентифікатор ролі	BIGINT	PK	Not null, unsigned
	Title	Назва ролі	VARCHAR(45)		
Book_comments	id	Ідентифікатор коментаря	BIGINT	PK	Not null, unsigned
	Message	Вміст коментаря	TEXT		
	User_id	Користувач, який прокоментував	BIGINT	FK (зв’язок з users)	Unsigned
	Book_id	Книга, яку прокоментували	BIGINT	FK (зв’язок з books)	Unsigned
Book_user_reactions	Id	Ідентифікатор	BIGINT	PK	Not null, unsigned
	User_id	Користувач, що відреагував	BIGINT	FK (зв’язок з users)	Unsigned
	Book_id	Книга, на яку відреагували	BIGINT	FK (зв’язок з books)	Unsigned
	Reaction_id	Тип реакції	SMALLINT	FK (зв’язок з reactions)	Unsigned
reactions	Id	Ідентифікатор реакції	BIGINT	PK	Not null, unsigned

	icon	Іконка рефакції	VARCHAR(255)		
libary	Id	Ідентифікатор	BIGINT	PK	Not null, unsigned
	Book_id	Ідентифікатор книги	BIGINT	FK (зв’язок з book)	Unsigned
	User_id	Ідентифікатор користувача	BIGINT	FK (зв’язок з users)	Unsigned
baskets	Id	Ідентифікатор корзини	BIGINT	PK	Unsigned
	User_id	Ідентифікатор користувача	BIGINT	FK (зв’язок з users)	Unsigned
Basket_books	Id	Ідентифікатор	BIGINT	PK	Not null, unsigned
	Book_id	Ідентифікатор книги	BIGINT	FK (зв’язок з books)	Unsigned
	Basket_id	Ідентифікатор корзини	BIGINT	FK (зв’язок з baskets)	unsigned
posts	id	Ідентифікатор посту	BIGINT	PK	Not null, unsigned
	Title	Назва посту	VARCHAR(45)		
	Content	Вміст посту	TEXT		

	Author_id	Ідентифікатор автора	BIGINT	FK (зв’язок з users)	Unsigned
Post_comments	Id	Ідентифікатор коментаря	BIGINT	PK	Not null, unsigned
	Message	Вміст коментаря	TEXT		
	User_id	Ідентифікатор користувача	BIGINT	FK	Unsigned
	Post_id	Ідентифікатор посту	BIGINT	FK	Unsigned

Таблиця 2.1 – Логічна реалізація бази даних

На рисунку 2.2 можна побачити ER-діаграму бази даних ресурсу.

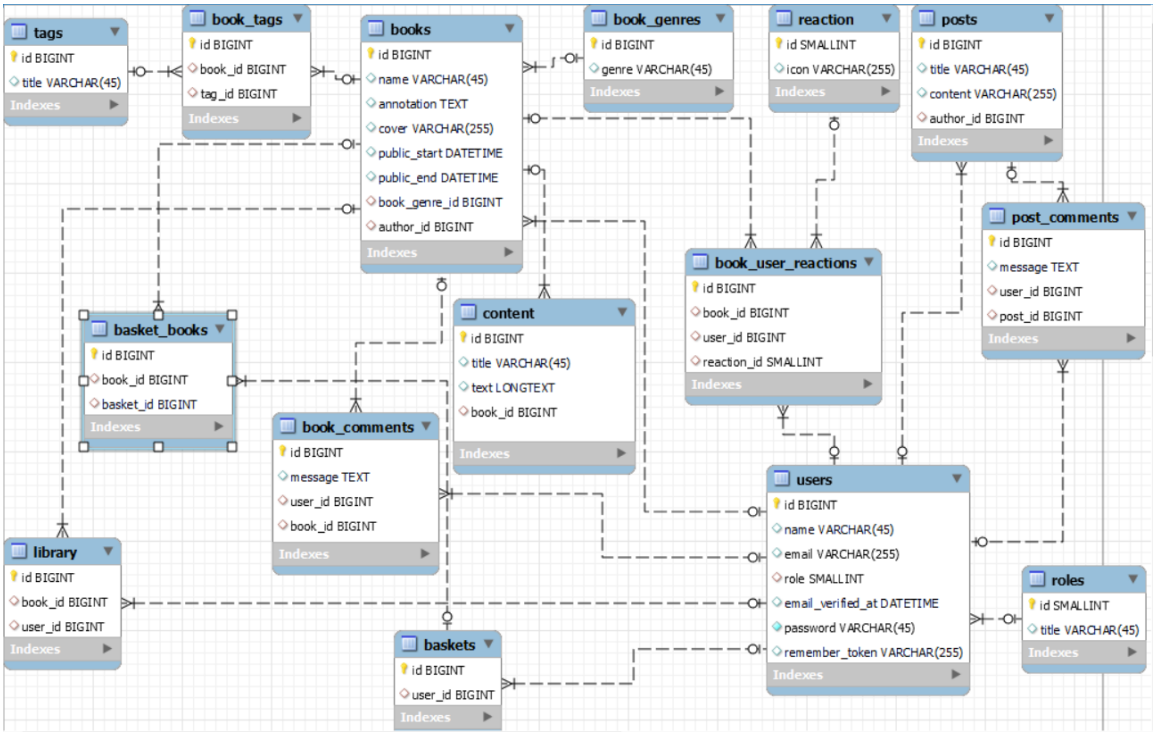

Рисунок 2.2 – ER-діаграма

Для розробки бази використовується СУБД Maria DB.

2.2. Фреймворк Laravel

Цей фреймворк використовує патерн mvc (Model View Controller). Model взаємодіє з базою даних, view відповідає за зовнішній вигляд сайту (html верстка сайту), controller виконує взаємодію між model і view. Це означає, що при виконанні http-запиту він ініціює взаємодію з controller, який за допомогою model отримує дані з бази даних і передає їх у view для обробки. Схематично вищеописаний процес можна зобразити так як на рисунку 2.3.

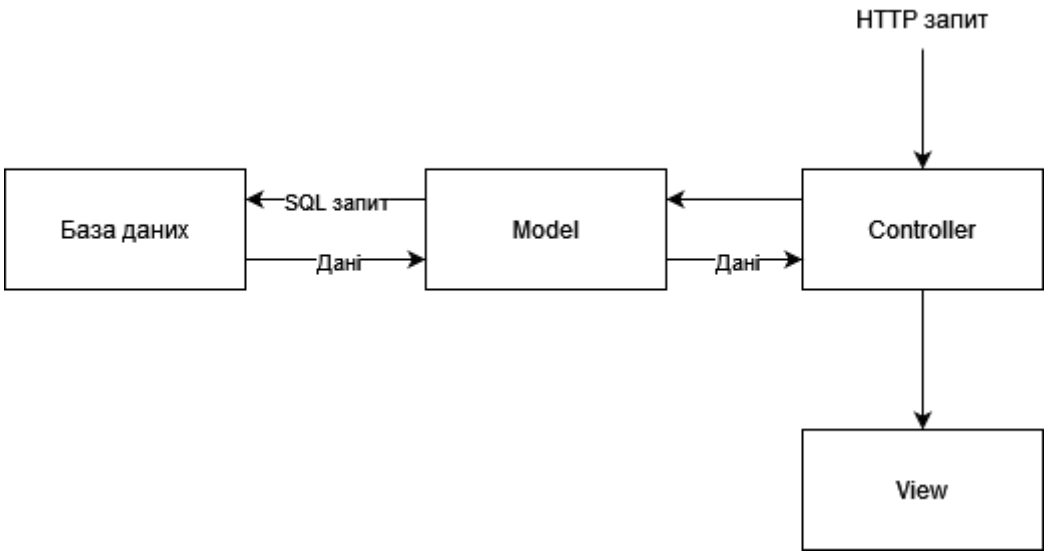

Рисунок 2.3. Модель MVC у фреймворку Laravel

Route це клас, який містить методи для формування http-запиту і зв'язування його з контролером. Зареєструвати маршрути можна у файлі web.php. При взаємодії з моделлю передбачається, що користувач може додавати, оновлювати, видаляти дані або читати їх з бази даних. Це означає, що потрібно створювати окремий маршрут для кожної з дії концепції CRUD. CRUD – це 4 основні функції управління даними: створення (create), читання (read), оновлення (update) і видалення (delete). При переході на URL маршруту відбуваються певні дії з базою даних відповідно до того, яка саме функція повинна виконуватись на цій сторінці. У таблиці 2.1. представлений рекомендований метод іменування маршрутів у фреймворку Laravel.

Таблиця 2.1. CRUD у фреймворку Laravel

Verb	URI	Action	Route Name
GET	/books	index	books.index
GET	/books/create	create	books.create
POST	/books	store	books.store
GET	/books/{book}	show	books.show
GET	/books/{book}/edit	edit	books.edit

PUT/PATCH	/books/{book}	update	books.update
DELETE	/books/{book}	destroy	books.destroy

Маршрути можна об’єднувати у групи, у якій можна вказати простір імен до якого відносяться контролери, префікс url, а також middleware, до якого відносяться маршрути.

Controllers це класи в яких відбувається взаємодія model з view. У контролері можна отримати дані з моделі і передати до view, використовуючи метод compact. У даній роботі використовуються однометодні контролери, що означає, що для кожного маршруту, зареєстрованого у відповідній групі, створюється окремий контролер. Пакет, у якому зберігаються контролери носить назву моделі, з якою вони пов’язані. Для того, щоб створити контролер у терміналі потрібно запустити команду:

php artisan make:controller <name>

У класі Request можна визначити правила, за якими буде відбуватися вставка у базу даних або її оновлення. Також можна визначити яке повідомлення буде виводитися при порушенні відповідного правила і отриманні помилки.

Клас Model містить методи для взаємодії з таблицею бази даних. Зазвичай клас має таку ж назву, що і відповідна таблиця, тільки в однині. Зв’язок таблиці з моделлю відбувається через властивість $table. Також тут можна визначити зв’язки між таблицями. Для того, що створити модель у терміналі потрібно запустити команду:

php artisan make:model <name>

За допомогою класу Migration створюється таблиця бази даних. Для того, що створити міграції у терміналі потрібно запустити команду:

php artisan make:mirgation <name>

Також міграції можна зв’язати з моделлю відразу при створенні:

php artisan make:model <name> -m

Для запуску міграції використовується команда:

php artisan migrate

Для скасуванні міграції використовується команда:

php artisan migrate:rollback

Для того, щоб laravel розумів у якому порядку відбувались міграції, у базі даних створюється таблиця migration, в якій вказується назва міграції і її порядковий номер. Ті міграції, що відбувались одночасно при запуску команди migrate мають один порядковий номер. Скасування міграції відбувається у зворотному порядку.

Черги у Laravel потрібні для того, щоб виконувати деякі дії визначені в методі handle jobs-класу у фоновому режимі. Для цього потрібно створити таблицю jobs використовуючи команду

php artisan queue:table

Для того, щоб черга запрацювала потрібно виконати команду:

php artisan queue:table

2.3. Html

HTML використовується для опису структури веб-сторінок. Іншими словами це «розмітка» документу, для якої використовуються різні теги. Ці елементи утворюють структуру веб-сторінки та описують, що відображати веб-браузеру.

Окрім розмітки HTML не надає можливості взаємодіяти з елементами сторінки. Для цього використовується JavaScript. Документ HTML має бути представлений таким чином, щоб код JavaScript міг запитувати та оновлювати його. Цю функцію виконує модель DOM.

При використанні JavaScript розробники взаємодіють з DOM, щоб проводити різні операції з об’єктами HTML.

2.4. React.js

React – це JavaScript бібліотека для побудови інтерфейсів користувача (UI). При роботі з бібліотекою рекомендується використовувати JSX синтаксис.

JSX – це синтаксичне розширення стандартного JavaScript, яке дозволяє використовувати інструменти для верстки сайтів всередині JavaScript файлів. Це означає, що при використанні JSX логіка рендерингу пов'язана з іншою логікою UI: обробкою подій і станів.

Основна концепція React.js полягає в компонуванні складних інтерфейсів з невеликих частин коду, які називаються компонентами. В React компоненти це JavaScript функції. Приклад компоненту зображено на рисунку 2.4. Для того, щоб зробити компонент доступним для використання його потрібно експортувати.

```
const Navbar = () => {
    return (
        <div className={classes.nav}>
            <div className={classes.nav__item}>
                <Link to="/">Головна</Link>
            </div>
            <div className={classes.nav__item}>
                <Link to="/books">Книги</Link>
            </div>
            <div className={classes.nav__item}>
                <Link to="/blogs">Блоги</Link>
            </div>
            <div className={classes.nav__item}>
                <Link to="/library">Бібліотека</Link>
            </div>
        </div>
    );
}
```

Рисунок 2.4. Приклад компоненту в React

React використовує так званий shadow dom. Це технологія, яка дозволяє рендерити об'єкти спочатку в віртуальний dom і лише потім застосовувати для реального dom і рендерити в браузері. Через це React доволі швидкий.

React дозволяє створювати так звані Single Page Applications, тобто веб-додатки, в яких є лише одна сторінка, а для навігації використовуються компоненти.

2.5. Клієнтська частина додатку

Даний ресурс буде створюватись як Single Page Application використовуючи фреймворк React.js. На рисунку 2.5 зображений макет головної сторінки сайту.

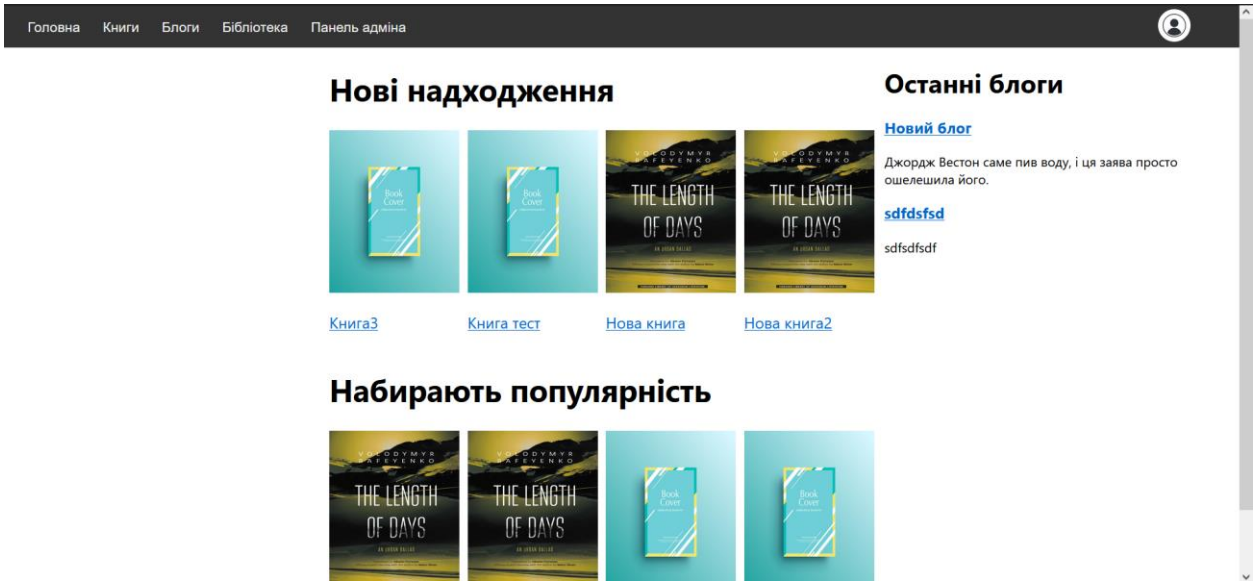

Рисунок 2.5. Головна сторінка сайту

На головній сторінці будуть відображені найбільш популярні книги, нові надходження книг, останні оновлення вже існуючих книг, рандомна добірка тегів, відомості про сайт, а також останні опубліковані блоги.

При натисканні на кнопку «Книги» користувачеві відобразиться список доступних жанрів.

На вкладці «Блоги» користувач може переглянути список опублікованих блогів.

Бібліотека доступна лише для зареєстрованих користувачів. Незареєстрований користувач отримає відповідне повідомлення, якщо намагатиметься переглянути свою бібліотеку. Також незареєстрований

користувач не матиме можливості публікувати книги, публікувати блоги, залишати коментарі, лайкати книги.

У кожної книги є своя сторінка, де можна переглянути відомості про книгу, залишити коментар, лайкнути книгу. Приклад зображено на рисунку 2.6.

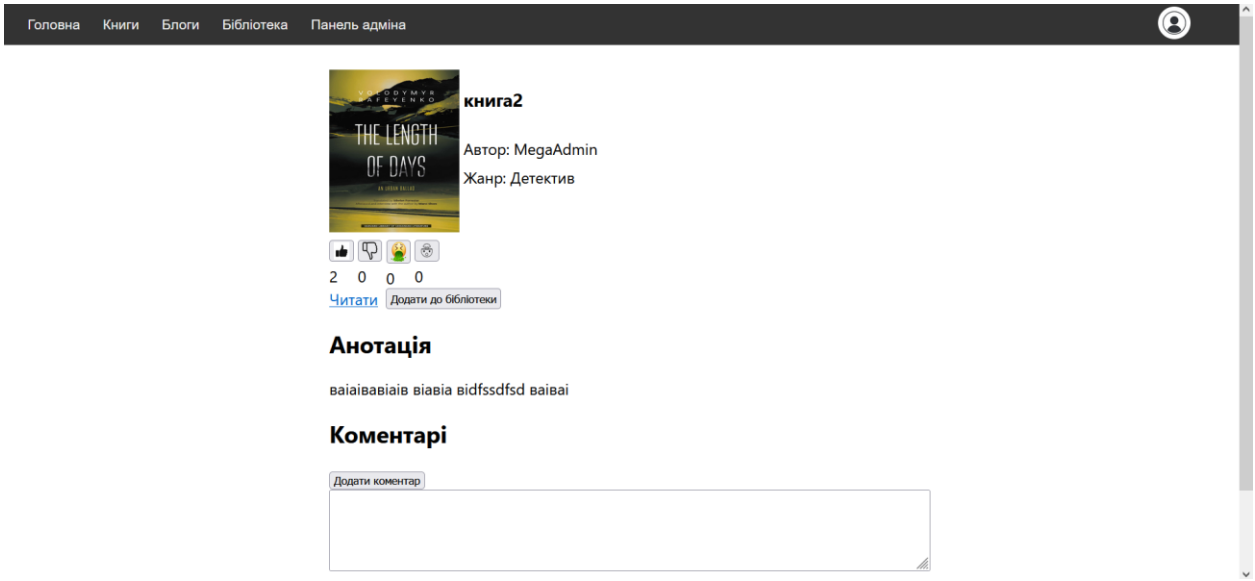

Рисунок 2.6. Сторінка книги

Для публікації книги у користувача має бути інтерфейс, за допомогою якого можна редагувати і стилізувати текст. Для цього використовуватиметься готове рішення у вигляді пакету tinymce-react. Приклад зображено на рисунку 2.7.

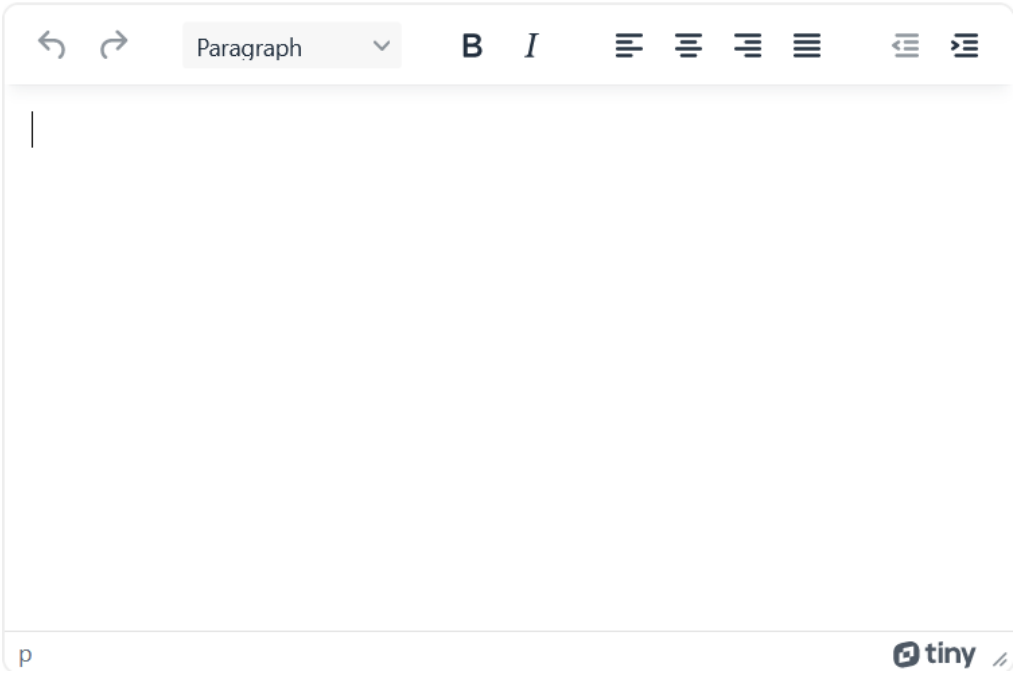

Рисунок 2.7. Tinymce [6]

Існуватиме також користувач з особливими правами, який може блокувати інших користувачів, а також приховати книги, тобто займатися модерацією сайту.

Для стилізації проекту використовуватиметься каскадна таблиця стилів CSS.

2.6 Rest API

Серверна частина веб-додатку буде представлена у вигляді окремого Laravel-проекту, який розташований на сервері і використовується як API для взаємодії клієнту і серверу.

Клієнт буде використовувати API-запити, щоб отримувати записи із таблиць бази даних або передавати дані на сервер. Laravel в свою чергу буде взаємодіяти з таблицями бази даних через відповідні моделі і передавати дані або отримувати їх і оновлювати таблиці.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

На рисунку 3.1 зображена головна сторінка:

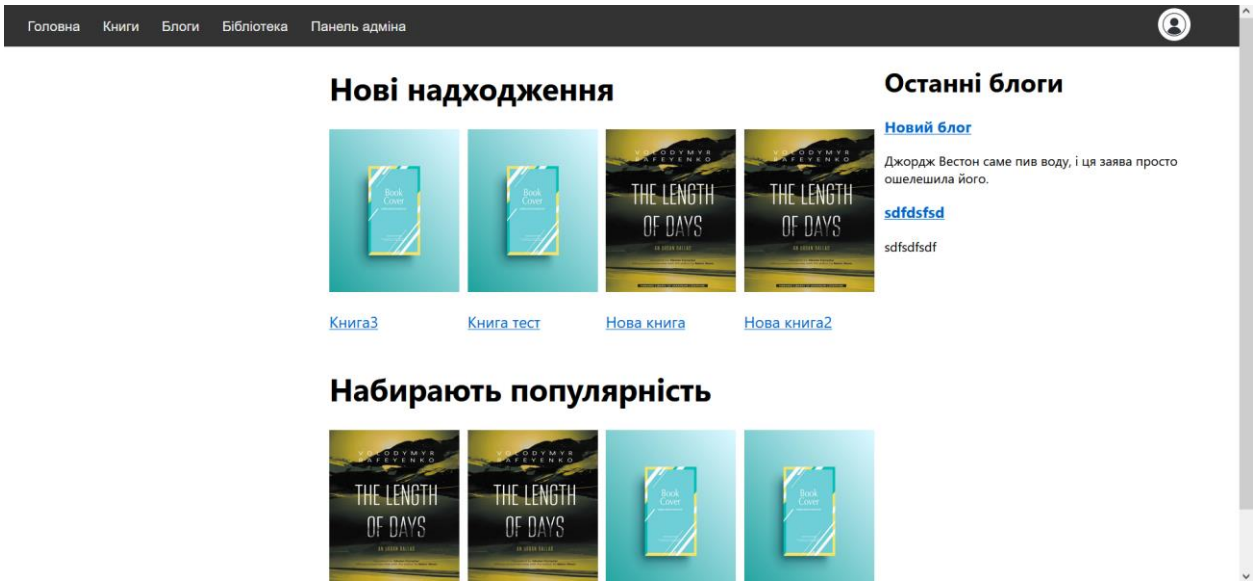

Рисунок 3.1 – Головна сторінка

Якщо користувач залогінився, то замість кнопки «Вхід» буде відображатися кнопка входу в особистий кабінет, де у користувачів є можливість налаштувати профіль і написати і опублікувати текст книги. Якщо користувач адмін, то на панелі навігації додатково буде відображатися посилання на панель адміністратора.

На рисунку 3.2 зображено сторінку книги.

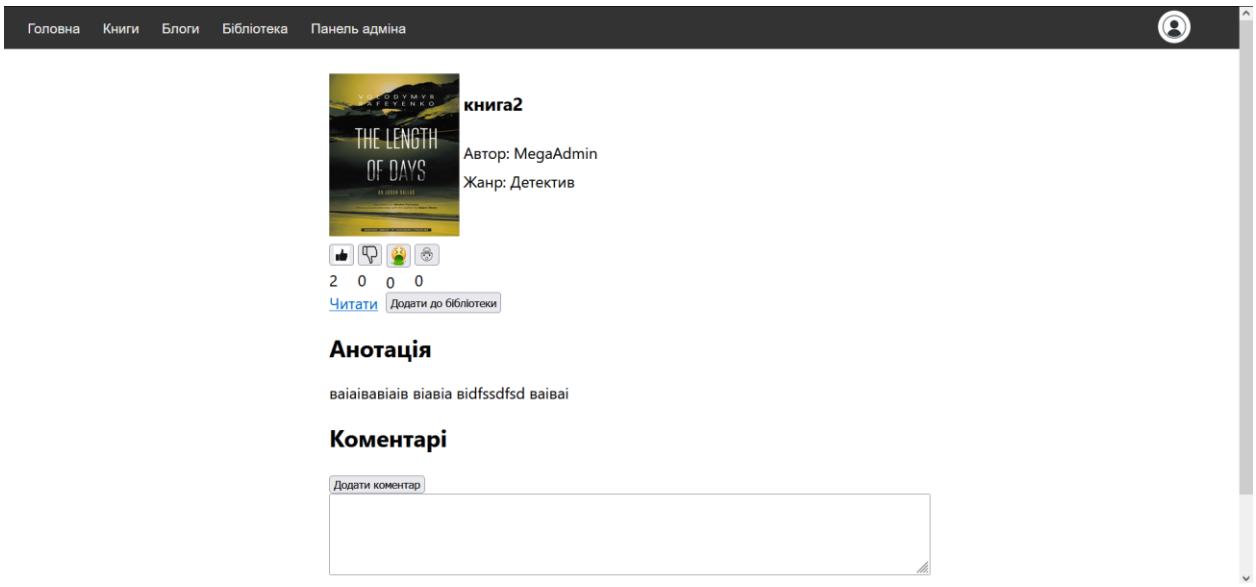

Рисунок 3.2 – Сторінка книги

На цій сторінці відображається назва книги, анотація, обкладинка, а також коментарі до книги.

На рисунку 3.3 зображено сторінка для читання книги.

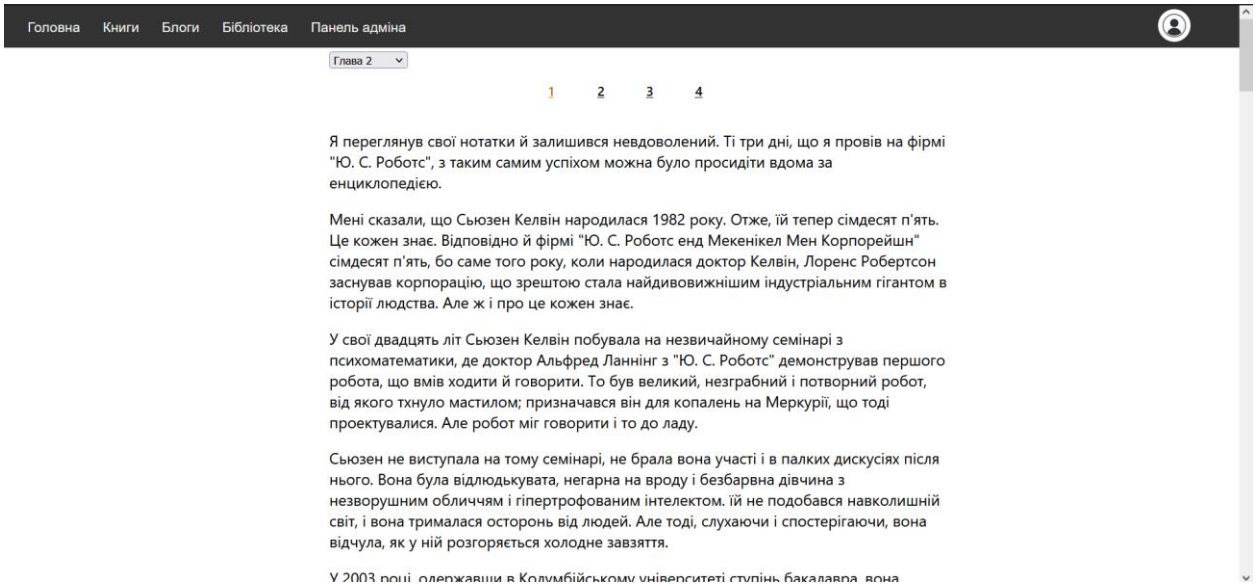

Рисунок 3.3 – Сторінка для читання книги

На цій сторінці можна прочитати главу книги.

На рисунку 3.4 зображено сторінку блогів.

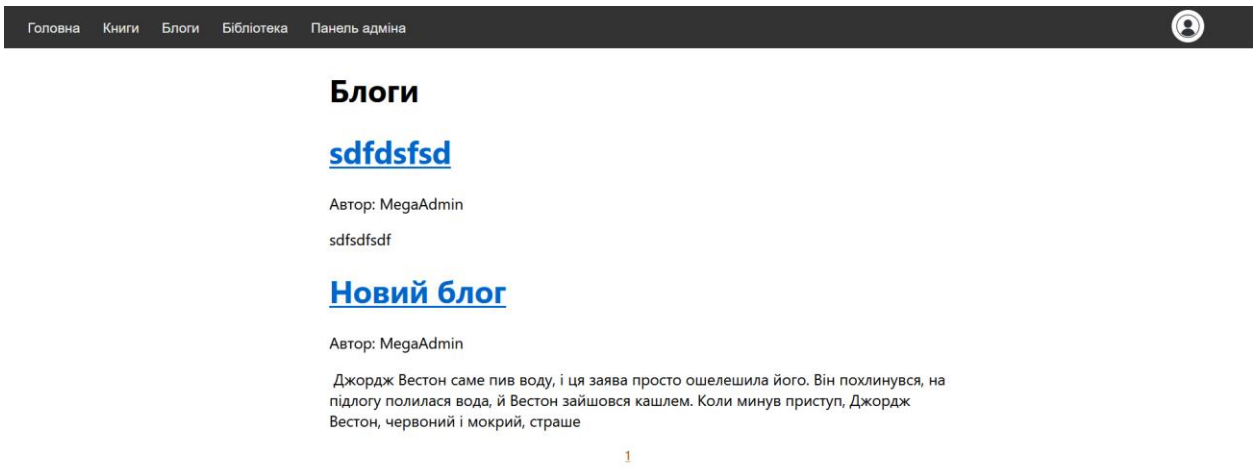

Рисунок 3.4 – Сторінка блогів

На цій сторінці відображаються блоги, які публікують користувачі.

На цій сторінці можна подивитися відомості про користувача.

3.1. Функції та модулі

Взаємодія з базою даних відбувається шляхом відсилання API-запитів на сервер. Для цього були написані функції **itemFetch**(), **itemsPost**(), **itemsPatch()** і **itemsDelete()**. Оскільки дані, які присилає сервер будуть використовуватись не один раз і в різних компонентах, для сайту був створений глобальний state за допомогою хука **use State**, який оновлюється за допомогою хука **useReducer**. Всі рути сайту були обернені в компонент **ContextData.Provider.** Таким чином стало можливим отримання state із контексту в будь-якому компоненті сайту і його оновлення за допомогою хука **useReducer**. Для відправки даних на сервер бекенду використовується метод axios.post(), для того, щоб мати можливість передавати csrf-токен в заголовках запиту.

На рисунку 3.6 зображений код функції **itemsFetch**().

```
import NET from "../network";

const itemsFetch = async (url, dispatchData, dispatchDataType) => {
    try {
        const response = await fetch(`${NET.APP_URL}${url}`)
        if (response.status === 200) {
            const result = await response.json()
            dispatchData({
                type: dispatchDataType,
                payload: result
            })
        }
    } catch (e) {
        console.log(e);
    }
}

export default itemsFetch
```

Рисунок 3.5 – Код функції itemsFetch()

Ця функція відправляє запит на адресу, яку ми вказуємо в параметрі url, а у відповідь отримує дані у форматі json, які потім заносяться в глобальний state.

На рисунку 3.7 зображений код функції itemsPost().

```
import NET from "../network";
import http from "./axios";

const itemsPost = async (url, data, navigate, navigateUrl) => {
    await http.post(`${NET.APP_URL}${url}`, data).then(async response => {
        if (response.status === 200) {
            navigate(`${navigateUrl}/${response.data.dataID.id}${navigateUrl === '/library' ? '?page=1' : ''}`)
        }
    }).catch((e) => {
        console.info(e);
    });
}

export default itemsPost
```

Рисунок 3.6 – Код функції item_post()

Ця функція відправляє на сервер POST запит. В тілі запиту передаються дані в форматі json. Використовується для вставки даних в базу даних.

На рисунку 3.8 зображений код функції itemsPatch().

```
import NET from "../network";
import http from "./axios";

const itemsPatch = async (url, data, navigate, navigateUrl) => {
    await http.patch(`${NET.APP_URL}${url}`, data).then(async response => {
        if (response.status === 200) {
            navigate(`${navigateUrl}/${response.data.dataID.id}`)
        }
    }).catch((e) => {
        console.info(e);
    });
}

export default itemsPatch
```

Рисунок 3.7 – Код функції itemsPatch()

Ця функція відправляє на сервер PATCH запит. В тілі запиту передаються дані в форматі json. Використовується для оновлення даних в базі даних.

На рисунку 3.9 зображений код функції itemsDelete().

```
import NET from "../network";
import http from "./axios";

const itemsDelete = async (url, data, navigate, navigateUrl) => {
    await http.delete(`${NET.APP_URL}${url}`, data).then(async response => {
        if (response.status === 200) {
            navigate(`${navigateUrl}`)
        }
    }).catch((e) => {
        console.info(e);
    });
}

export default itemsDelete
```

Рисунок 3.8 – Код функції itemsDelete()

Ця функція відправляє на сервер DELETE запит. В тілі запиту передаються дані в форматі json. Використовується для видалення даних із бази даних.

Щоб не відображати на одній сторінці одразу весь масив даних на сайті використовується пагінація. В параметрах запиту передається номер сторінки, який можна отримати за допомогою хука useSearchParams(). Щоб знайти індекс останнього елемента, який буде відображатися на сторінці, потрібно перемножити номер сторінки і кількість елементів на одній сторінці, а перший індекс це різниця між останнім індексом і кількістю елементів на сторінці.

Для авторизації на сайті використовується Laravel Sanctum і axios. Для передачі даних для авторизації використовується функція axios.post(), яка відправляє post-запит на вказаний нами url. За допомогою функції axios.get('/sanctum/csrf-cookie') ми можемо отримати csrf-токен, який будемо використовувати для отримання доступу для захищених маршрутів. Ця функція зберігає csrf-токен в куках і потім він буде використовуватися при відправці запитів.

Для того, щоб контролювати, який користувач в даний момент здійснив вхід на сайт, був створений AuthContext, який містить всі необхідні методи для відправки даних для авторизації і реєстрації на бекенд. Також цей контекст містить метод getUser(), який зберігає дані про користувача, який увійшов на сайт у змінну user. Потім цю змінну можна використовувати у будь-якому місці програми.

Щоб контролювати доступ користувача до сторінок сайту створено AdminLayout, GuestLayout і AuthLayout. Якщо обгорнути рути в AdminLayout, то до них матиме доступ лише користувач з роллю admin, всіх інших користувачів буде перенаправляти на головну сторінку. Аналогічно працюють і інші Layout.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було виконано літературний огляд існуючих аналогів інформаційних систем для створення, публікації і поширення електронних книг.

Були сформовані задачі, які необхідно вирішити в процесі розробки.

Для розробки були обрані фреймворки Laravel і React.js.

В процесі реалізації була створена серверна і клієнтська частина системи. Була розроблена логіка для публікації, пошуку і перегляду книг, а також інтерфейс для взаємодії користувача з інформаційною системою.

Результатом роботи є інформаційна система для створення, публікації та поширення електронних книг.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Web Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/web-development/.

2. Laravel Docs [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/9.x/installation.

3. HTML Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/html/

4. CSS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/css/

5. web server [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/Web-server.

6. tinymce [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/@tinymce/tinymce-react.

7. react-credit-cards [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/react-credit-cards.

8. Lets Understand The ReactJS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.phptpoint.com/reactjs-tutorial/

9. Laravel Sanctum [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/10.x/sanctum

10. axios [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/axios

11. Axios docs [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/docs/intro

12. Using the Effect Hook [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/hooks-effect.html

13. Using the State Hook [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/hooks-state.html

14. Hooks API Reference [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/hooks-reference.html

15. React useContext Hook [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/react/react_usecontext.asp

16. React useReducer Hook [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/react/react_usereducer.asp

17. React useMemo Hook [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/react/react_usememo.asp

18. React JSX [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/react/react_jsx.asp

19. JSX в глибину [Електронний ресурс] – Режим доступу до ресурсу: https://codeguida.com/post/363

20. JavaScript Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/

ДОДАТОК А

Клієнтська частина

Рути

```
import './App.css';
import { Routes, Route } from "react-router-dom"
import React from 'react';
import General from './routes/General/General';
import Books from './routes/Books/Books';
import Blogs from './routes/Blogs/Blogs';
import Library from './routes/Library/Library';
import Admin from './routes/Admin/Admin';
import Roles from './routes/Admin/Roles/Roles';
import RolesShow from './routes/Admin/Roles/Show/RolesShow'
import RolesCreate from './routes/Admin/Roles/Create/RolesCreate'
import RolesEdit from './routes/Admin/Roles/Edit/RolesEdit';
import ContextData from './context/Data/ContextData';
import StateData from './context/Data/StateData';
import ReducerData from './context/Data/ReduserData';
import Users from './routes/Admin/Users/Users';
import UsersShow from './routes/Admin/Users/Show/UsersShow';
import UsersCreate from './routes/Admin/Users/Create/UsersCreate';
import UsersEdit from './routes/Admin/Users/Edit/UsersEdit';
import Login from './routes/Auth/Login/Login';
import Registration from './routes/Auth/Registration/Registration';
import GuestLayout from './layouts/GuestLayot';
import AdminLayout from './layouts/AdminLayot';
import AuthLayot from './layouts/AuthLayot';
import Profile from './routes/Profile/Profile';
import ProfileEdit from './routes/Profile/Edit/ProfileEdit';
import Genres from './routes/Admin/Genres/Genres';
```

```
import GenresCreate from './routes/Admin/Genres/Create/GenresCreate';
import GenresShow from './routes/Admin/Genres/Show/GenresShow';
import GenresEdit from './routes/Admin/Genres/Edit/GenresEdit';
import ProfileBooks from './routes/Profile/Books/Books';
import BooksCreate from './routes/Profile/Books/Create/BooksCreate';
import ChapterCreate from './routes/Profile/Chapter/Create/ChapterCreate';
import BooksShow from './routes/Books/Show/BooksShow';
import Reader from './routes/Books/Reader/Reader';
import BlogsCreate from './routes/Blogs/Create/BlogsCreate';
import BlogsEdit from './routes/Blogs/Edit/BlogsEdit';
import BlogsShow from './routes/Profile/Blogs/BlogsShow';
import PostShow from './routes/Blogs/Show/PostShow';

function App() {
    const [stateData, dispatchData] = React.useReducer(ReducerData, StateData)
  return (
    <ContextData.Provider value={{stateData, dispatchData}}>
      <Routes>
        <Route index element={<General />} />
        <Route path='/genres' element={<Books />} />
        <Route path='/books/:id' element={<BooksShow />} />
        <Route path='/books/:id/reader' element={<Reader />} />
        <Route path='/blogs' element={<Blogs />} />
        <Route path='/blogs/:id' element={<PostShow />} />
        <Route element={<AdminLayout />}>
          <Route path='/admin' element={<Admin />}>
            <Route path='roles' element={<Roles />} />
            <Route path='roles/create' element={<RolesCreate />} />
            <Route path='roles/:id' element={<RolesShow />} />
```

```
                    <Route path='roles/:id/edit' element={<RolesEdit />} />
                    <Route path='users' element={<Users />} />
                    <Route path='users/create' element={<UsersCreate />} />
                    <Route path='users/:id' element={<UsersShow />} />
                    <Route path='users/:id/edit' element={<UsersEdit />} />
                    <Route path='genres' element={<Genres />} />
                    <Route path='genres/create' element={<GenresCreate />} />
                    <Route path='genres/:id' element={<GenresShow />} />
                    <Route path='genres/:id/edit' element={<GenresEdit />} />
                  </Route>
                </Route>
                <Route element={<GuestLayout />}>
                  <Route path='/login' element={<Login />} />
                  <Route path='/registration' element={<Registration />} />
                </Route>
                <Route element={<AuthLayot />}>
                  <Route path='/profile/blogs/create' element={<BlogsCreate />} />
                  <Route path='/profile/blogs/:id' element={<BlogsEdit />} />
                  <Route path='/profile/blogs' element={<BlogsShow />} />
                  <Route path='/profile' element={<Profile />} />
                  <Route path='/library/:id' element={<Library />} />
                  <Route path='/profile/edit' element={<ProfileEdit />} />
                  <Route path='/profile/books' element={<ProfileBooks />} />
                  <Route path='/profile/books/create' element={<BooksCreate />} />
                   <Route path='/profile/books/:id/chapter/create' element={<Chapter-
Create />} />
                </Route>
              </Routes>
            </ContextData.Provider>
          );
```

```
}

export default App;
```

Пагінація

```
import React from "react"
import classes from "./pagination.module.scss"
import { Link } from "react-router-dom";

const Pagination = ({totalItems, itemsPerPage, currentPage}) => {
   let pages = []
   for(let i = 1; i <= Math.ceil(totalItems/itemsPerPage); i++) {
      pages.push(i)
   }

   return (
      <div className={classes.pagination}>
         {
            pages.map((page, index) => {

               if (index < Number(currentPage) + 3 && index >= Number(cur-
rentPage) - 3) {
                  return <Link
                        to={`?page=${page}`}
                        key={index}
                              style={{color: page === Number(currentPage) ?
'#b96d2c' : '#0a0a0a'}}
                     >
                        {page}
                     </Link>
```

```
                }

            })
        }

        {
            pages.length > Number(currentPage) + 3 ?
            <>
                <div>...</div>
                <Link
                    to={`?page=${pages[pages.length - 1]}`}
                    key={pages.length - 1}
                        style={{color: pages[pages.length - 1] === Number(cur-
rentPage) ? '#ffe400' : '#0a0a0a'}}
                >
                    {pages[pages.length - 1]}
                </Link>
            </> : ''
        }

    </div>
  )
};

export default Pagination;
```

Отримання даних з бекенду

```
import NET from "../network";

const itemsFetch = async (url, dispatchData, dispatchDataType) => {
```

```
    try {
        const response = await fetch(`${NET.APP_URL}${url}`)
        if (response.status === 200) {
            const result = await response.json()

            dispatchData({
                type: dispatchDataType,
                payload: result
            })
        }
    } catch (e) {
        console.log(e);
    }
}

export default itemsFetch
```

Створення запиту

```
import axios from "axios"
import NET from "../network"

const http = axios.create({
    baseURL: NET.AXIOS_URL,
    headers: {
        'X-Requested-With': 'XMLHttpRequest',
    },
    withCredentials: true
})

export default http
```

Додавання даних

```
import NET from "../network";
import http from "./axios";

const itemsPost = async (url, data, navigate, navigateUrl) => {
    await http.post(`${NET.APP_URL}${url}`, data).then(async response =>
{
        if (response.status === 200) {
            navigate(`${navigateUrl}/${response.data.dataID.id}${navigateUrl
=== '/library' ? '?page=1' : ''}`)
        }
    }).catch((e) => {
        console.info(e);
    });
}

export default itemsPost
```

Оновлення даних

```
import NET from "../network";
import http from "./axios";

const itemsPatch = async (url, data, navigate, navigateUrl) => {
    await http.patch(`${NET.APP_URL}${url}`, data).then(async response =>
{
        if (response.status === 200) {
            navigate(`${navigateUrl}/${response.data.dataID.id}`)
        }
```

```
    }).catch((e) => {
        console.info(e);
    });
}

export default itemsPatch
```

Видалення даних

```
import NET from "../network";
import http from "./axios";

const itemsDelete = async (url, data, navigate, navigateUrl) => {
    await http.delete(`${NET.APP_URL}${url}`, data).then(async response
=> {
        if (response.status === 200) {
            navigate(`${navigateUrl}`)
        }
    }).catch((e) => {
        console.info(e);
    });
}

export default itemsDelete
```

Контекст для авторизації

```
import { createContext, useContext, useEffect, useState } from "react"
import http from "../../actions/axios"

import { useNavigate } from "react-router-dom"
```

```
const AuthContext = createContext({})

export const AuthProvider = ({ children }) => {
    const [user, setUser] = useState(null)
    const [errors, setErrors] = useState([])
    const csrf = () => http.get('/sanctum/csrf-cookie')
    const navigate = useNavigate()

    useEffect(() => {
        getUser()
    }, [])

    const getUser = async () => {
        const { data } = await http.get("/api/user")
        setUser(data)
    }

    const login = async (email, password) => {
        const data = {
            email: email,
            password: password
        }
        await csrf()
        try {
            setErrors([])
            await http.post('/api/login', data)
            await getUser()
            navigate('/')
        } catch (e) {
```

```
        if (e.response.status === 422) {
            setErrors(e.response.data.errors)
        }
    }

}

const register = async (title, email, password, passwordConfirmation) => {
    const data = {
        title: title,
        email: email,
        password: password,
        password_confirmation: passwordConfirmation
    }
    await csrf()
    try {
        setErrors([])
        await http.post('/api/register', data)
        await http.post('/api/login', { email, password })
        await getUser()
        navigate('/')
    } catch (e) {
        if (e.response.status === 422) {
            setErrors(e.response.data.errors)
        }
    }
}

const logout = () => {
    http.post("/api/logout").then(() => {
```

```
            setUser(null)
        })
    }

    return <AuthContext.Provider value={{ user, getUser, errors, login, regis-
ter, logout }}>
        {children}
    </AuthContext.Provider>
}

export default function useAuthContext() {
    return useContext(AuthContext)
}
```

Layout для адміна

```
import { Navigate, Outlet } from "react-router-dom";
import useAuthContext from "../context/Auth/AuthContext";

const AdminLayout = () => {
    const { user } = useAuthContext()
    return user && user.role_id === 2 ? <Outlet /> : <Navigate to="/" />
};

export default AdminLayout;
```

Layout для авторизованого користувача

```
import { Navigate, Outlet } from "react-router-dom";
import useAuthContext from "../context/Auth/AuthContext";

const AuthLayot = () => {
```

```
    const { user } = useAuthContext()
    return user ? <Outlet /> : <Navigate to="/login" />
};

export default AuthLayot;
```

Layout для неавторизованого користувача

```
import { Navigate, Outlet } from "react-router-dom";
import useAuthContext from "../context/Auth/AuthContext";

const GuestLayout = () => {
    const { user } = useAuthContext()
    return !user ? <Outlet /> : <Navigate to="/" />
};

export default GuestLayout;
```

ДОДАТОК Б

Серверна частина

Рути

```
Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

Route::group(['namespace' => 'App\Http\Controllers\Auth'], function () {
    Route::post('/login', LoginController::class);
    Route::post('/logout', LogoutController::class);
    Route::post('/register', RegisterController::class);
});

Route::group(['namespace' => 'App\Http\Controllers\Admin', 'prefix' => 'admin', 'middleware' => 'admin'], function () {
    Route::group(['namespace' => 'Roles', 'prefix' => 'roles'], function () {
        Route::get('/', IndexController::class);
        Route::post('/', StoreController::class);
        Route::get('/{role}', ShowController::class);
        Route::patch('/{role}', UpdateController::class);
        Route::delete('/{role}', DestroyController::class);
    });

    Route::group(['namespace' => 'Users', 'prefix' => 'users'], function () {
        Route::get('/', IndexController::class);
        Route::post('/', StoreController::class);
        Route::get('/{user}', ShowController::class);
        Route::patch('/{user}', UpdateController::class);
        Route::delete('/{user}', DestroyController::class);
```

```
        });

        Route::group(['namespace' => 'Genres', 'prefix' => 'genres'], function () {
            Route::post('/', StoreController::class);
            Route::patch('/{genre}', UpdateController::class);
            Route::delete('/{genre}', DestroyController::class);
        });
    });

    Route::group(['namespace' => 'App\Http\Controllers\Genres', 'prefix' => 'genres'], function () {
        Route::get('/', IndexController::class);
        Route::get('/{genre}', ShowController::class);
    });

    Route::group(['namespace' => 'App\Http\Controllers\Main'], function () {
        Route::get('/', IndexController::class);
    });

    Route::group(['namespace' => 'App\Http\Controllers\Blogs', 'prefix' => 'posts'], function () {
        Route::get('/{post}', ShowController::class);
        Route::get('/', IndexController::class);
    });

    Route::group(['namespace' => 'App\Http\Controllers\Books', 'prefix' => 'books'], function () {
        Route::post('/', StoreController::class);
        Route::get('/', IndexController::class);
        Route::get('/{book}', ShowController::class);
```

```
            Route::group(['namespace' => 'Reader', 'prefix' => '{book}/reader'], function () {
                Route::get('/', IndexController::class);
            });
        });

        Route::group(['namespace' => 'App\Http\Controllers\Profile', 'prefix' => 'profile', 'middleware' => 'auth'], function () {
            Route::group(['namespace' => 'Books', 'prefix' => '{user}/books'], function () {
                Route::get('/', IndexController::class);
            });

            Route::group(['namespace' => 'Blogs', 'prefix' => '{user}/blogs'], function () {
                Route::post('/', StoreController::class);
                Route::get('/', IndexController::class);
            });

            Route::group(['namespace' => 'Library', 'prefix' => '{user}/library'], function () {
                Route::get('/', IndexController::class);
                Route::post('/', StoreController::class);
                Route::delete('/{book}', DestroyController::class);
            });

            Route::group(['namespace' => 'Chapter', 'prefix' => 'chapters'], function () {
                Route::post('/', StoreController::class);
```

```
    });
});
```

Моделі

```
class User extends Authenticatable {
    use HasApiTokens, HasFactory, Notifiable, SoftDeletes;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'title',
        'email',
        'password',
        'role_id',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
```

```
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function roles() {
        return $this->belongsTo(Role::class, 'role_id', 'id');
    }

    public function books() {
        return $this->hasMany(Book::class, 'author_id', 'id');
    }

    public function posts() {
        return $this->hasMany(Post::class, 'author_id', 'id');
    }

    public function library() {
            return $this->belongsToMany(Book::class, 'libraries', 'user_id',
'book_id');
    }
}

<?php

namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Role extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'roles';
    protected $guarded = false;
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Book extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'books';
    protected $guarded = false;
    public $timestamps = false;
```

```
    public function images() {
        return $this->hasMany(Image::class, 'book_id', 'id');
    }

    public function users() {
        return $this->belongsTo(User::class, 'author_id', 'id');
    }

    public function genre() {
        return $this->belongsTo(BookGenre::class, 'book_genre_id', 'id');
    }

    public function content() {
        return $this->hasMany(Content::class, 'book_id', 'id');
    }

    public function comments() {
         return $this->belongsToMany(PostComment::class, 'book_comments', 'book_id', 'user_id');
    }

    public function reaction() {
        return $this->belongsToMany(BookUserReaction::class, 'book_user_reactions', 'book_id', 'user_id');
    }
}

<?php

namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class BookGenre extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'book_genres';
    protected $guarded = false;
}

class BookComment extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'book_comments';
    protected $guarded = false;
}

class BookUserReaction extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'book_user_reactions';
    protected $guarded = false;
```

```
}

class Content extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'contents';
    protected $guarded = false;
}

class Image extends Model
{
    use HasFactory;
    protected $guarded = false;
    protected $table = 'images';
}

class Library extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'libraries';
    protected $guarded = false;
}

class Post extends Model
{
    use HasFactory;
```

```
        use SoftDeletes;

        protected $table = 'posts';
        protected $guarded = false;

        public function users() {
            return $this->belongsTo(User::class, 'author_id', 'id');
        }

        public function comments() {
            return $this->belongsToMany(PostComment::class, 'post_comments', 'post_id', 'user_id');
        }
    }

    class PostComment extends Model
    {
        use HasFactory;
        use SoftDeletes;

        protected $table = 'post_comments';
        protected $guarded = false;
    }
```

Ресурси

```
class BookResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'name' => $this->name,
            'annotation' => $this->annotation,
            'public_start' => $this->public_start,
            'public_end' => $this->public_end,
            'author' => new UserResource($this->users),
            'genre' => new GenreResource($this->genre),
            'images' => ImageResource::collection($this->images),
                'comments'  =>  BookCommentsResource::collection($this->com-
ments)
        ];
    }
}

class BookCommentsResource extends JsonResource
{
    /**
```

```
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'message' => $this->message
        ];
    }
}

class GenreResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'title' => $this->title
        ];
    }
}
```

```
class ImageResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'url' => $this->url
        ];
    }
}

class LibraryResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' => $this->user_id
        ];
    }
```

```
}

class PostResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'title' => $this->title,
            'content' => $this->content,
            'author' => new UserResource($this->users),
            'comments' => PostCommentsResource::collection($this->comments)
        ];
    }
}

class PostCommentsResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
```

```
    public function toArray($request)
    {
        return parent::toArray($request);
    }
}

class UserResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'title' => $this->title
        ];
    }
}
```

Контролери

Контролери для адміна

```
class DestroyController extends Controller
{
    public function __invoke(BookGenre $genre)
    {
        $genre->delete();
        return response()->json(['success' => 'success', 200]);
    }
}

class StoreController extends Controller
{
    public function __invoke(Request $request)
    {
        $request->validate([
            'title' => 'required',
        ], [
            'required' => 'Це обов\'язкове поле'
        ]);

        $data = $request->all();
        try {
            DB::beginTransaction();
            $dataId = BookGenre::create($data);
            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            abort(500);
```

```
        }
        return response()->json(['success' => 'success', 200, 'dataID' => $dataId]);
    }
}

class UpdateController extends Controller
{
    public function __invoke(Request $request, BookGenre $genre)
    {
        $request->validate([
            'title' => 'required',
        ], [
            'required' => 'Це обов\'язкове поле'
        ]);

        $data = $request->all();
        try {
            DB::beginTransaction();
            $genre->update($data);
            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            abort(500);
        }
        return response()->json(['success' => 'success', 200, 'dataID' => $genre]);
    }
}

class DestroyController extends Controller
{
```

```
    public function __invoke(Role $role)
    {
        $role->delete();
        return response()->json(['success' => 'success', 200]);
    }
}

class IndexController extends Controller
{
    public function __invoke()
    {
        return Role::all();
    }
}

class ShowController extends Controller
{
    public function __invoke(Role $role)
    {
        return $role;
    }
}

class StoreController extends Controller
{
    public function __invoke(Request $request)
    {
        $request->validate([
            'title' => 'required',
        ], [
```

```
            'required' => 'Це обов\'язкове поле'
        ]);

        $data = $request->all();
        try {
            DB::beginTransaction();
            $dataId = Role::create($data);
            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            abort(500);
        }
       return response()->json(['success' => 'success', 200, 'dataID' => $dataId]);
    }
}

class UpdateController extends Controller
{
    public function __invoke(Request $request, Role $role)
    {
        $request->validate([
            'title' => 'required',
        ], [
            'required' => 'Це обов\'язкове поле'
        ]);

        $data = $request->all();
        try {
            DB::beginTransaction();
            $role->update($data);
```

```
            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            abort(500);
        }

        return response()->json(['success' => 'success', 200, 'dataID' => $role]);
    }
}

class DestroyController extends Controller
{
    public function __invoke(User $user)
    {
        $user->delete();
        return response()->json(['success' => 'success', 200]);
    }
}

class IndexController extends Controller
{
    public function __invoke()
    {
        return User::all();
    }
}

class ShowController extends Controller
{
    public function __invoke(User $user)
```

```
        {
            $user->role_id = $user->roles()->first()->title;
            return $user;
        }
    }

    class StoreController extends Controller
    {
        public function __invoke(Request $request)
        {
            $data = $request->all();
            $request->validate([
                'title' => 'required',
                'email' => 'required|email',
                  'password' => 'required|regex:/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-
Z\d]{8,}/u',

            ], [
                'required' => 'Це обов\'язкове поле',
                'email' => 'Введіть email у форматі example@gmail.com',
                'password.regex' => 'Пароль має містити хоча б 8 символів, хоча б
одну велику букву, хоча б одну маленьку букву і хоча б одну цифру',

            ]);

            try {
                DB::beginTransaction();
                $dataId = User::create($data);
                DB::commit();
            } catch (\Exception $exeption) {
```

```
                DB::rollBack();
                abort(500);
            }
            return response()->json(['success' => 'success', 200, 'dataID' => $dataId]);
        }
    }

    class UpdateController extends Controller
    {
        public function __invoke(Request $request, User $user)
        {
            $data = $request->all();
            $request->validate([
                'title' => 'required',
                'email' => 'required|email',
                  'password' => 'required|regex:/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-
Z\d]{8,}/u',

            ], [
                'required' => 'Це обов\'язкове поле',
                'email' => 'Введіть email у форматі example@gmail.com',
                'password.regex' => 'Пароль має містити хоча б 8 символів, хоча б
одну велику букву, хоча б одну маленьку букву і хоча б одну цифру',

            ]);

            try {
                DB::beginTransaction();
                $user->update($data);
                DB::commit();
```

```
        } catch (\Exception $exeption) {
            DB::rollBack();
            abort(500);
        }
        return response()->json(['success' => 'success', 200, 'dataID' => $user]);
    }
}
```

Контролери для авторизації

```
class LoginController extends Controller
{
    public function __invoke(Request $request)
    {
        $credentials = $request->validate([
            'email' => 'required|email',
              'password' => 'required|regex:/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}/u',
        ], [
            'required' => 'Це обов\'язкове поле',
            'email' => 'Введіть email у форматі example@gmail.com',
            'password.regex' => 'Пароль має містити хоча б 8 символів, хоча б одну велику букву, хоча б одну маленьку букву і хоча б одну цифру'
        ]);

        if (!Auth::attempt($credentials)) {
            throw ValidationException::withMessages([
                'email' => [
                    'Невірний логін або пароль'
                ],
                'password' => [
```

```
                'Невірний логін або пароль'
            ]
        ]);
    }

    $request->session()->regenerate();

    return response()->json([
        'status' => 200,
        'message' => 'Login Successfully',
    ]);;
  }
}

class LogoutController extends Controller
{
  public function __invoke(Request $request)
  {
    Auth::guard('web')->logout();

    $request->session()->invalidate();
    $request->session()->regenerateToken();

    return response()->noContent();
  }
}

class RegisterController extends Controller
{
  public function __invoke(Request $request) {
```

```
        $request->validate([
            'title' => 'required',
            'email' => 'required|email',
              'password' => 'required|regex:/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-
Z\d]{8,}/u',
            'password_confirmation' => 'required|same:password'
        ], [
            'required' => 'Це обов\'язкове поле',
            'email' => 'Введіть email у форматі example@gmail.com',
            'password.regex' => 'Пароль має містити хоча б 8 символів, хоча б
одну велику букву, хоча б одну маленьку букву і хоча б одну цифру',
            'same' => 'Паролі мають співпадати'
        ]);

        try {
            DB::beginTransaction();
            $user = User::create([
                'title' => $request->title,
                'email' => $request->email,
                'password' => bcrypt($request->password),
                'remember_token' => Str::random(40)
            ]);
            DB::commit();
        } catch (\Exception $exeption) {
            DB::rollBack();
            abort(500);
        }

        $request->session()->regenerate();
```

```
        return response()->json([
            'status' =>200,
            'username' =>$user->title,
            'message' =>'Registered Successfully',
        ]);
    }
}
```

Контролери для блогів

```
class IndexController extends Controller
{
    public function __invoke() {
        return PostResource::collection(Post::all());
    }
}

class ShowController extends Controller
{
    public function __invoke(Post $post) {
        return $post;
    }
}
```

Контролери для книг

```
class IndexController extends Controller
{
    public function __invoke(Book $book) {
        return $book->content;
```

```
    }
}

class IndexController extends Controller
{
    public function __invoke()
    {
        $books = Book::all();
        return BookResource::collection($books);
    }
}

class ShowController extends Controller
{
    public function __invoke(Book $book) {
        return new BookResource($book);
    }
}

class StoreController extends Controller
{
    public function __invoke(Request $request) {
        $data = $request->all();
        $image = $data['cover'];
        unset($data['cover']);
        $data['public_start'] = Carbon::now();
        try {
            DB::beginTransaction();
            $book = Book::create($data);
            DB::commit();
```

```
            } catch (\Exception $e) {
                DB::rollBack();
                abort(500);
            }
            $name = md5(Carbon::now() . '_' . $image->getClientOriginalName()) .
'.' . $image->getClientOriginalExtension();
            $filePath = Storage::disk('public')->putFileAs('/images', $image, $name);

            Image::create([
                'path' => $filePath,
                'url' => url('/storage/' . $filePath),
                'book_id' => $book->id
            ]);

            return response()->json(['success' => 'success', 200, 'dataID' => $book]);
        }
    }
```

Контролери для жанрів

```
class IndexController extends Controller
{
    public function __invoke()
    {
        return BookGenre::all();
    }
}

class ShowController extends Controller
{
    public function __invoke(BookGenre $genre)
```

```
    {
        return $genre;
    }
}
```

Головний контролер

```
class IndexController extends Controller
{
    public function __invoke() {
        $data = [];
        $booksCount = Book::all()->count();
        $data['random_books'] = $booksCount >= 4 ?
            BookResource::collection(Book::get()->random(4)) :
            BookResource::collection(Book::all()->take($booksCount));
        $data['latest_books'] = $booksCount >= 4 ?
                BookResource::collection(Book::orderBy('public_start', 'DESC')-
>get()->take(4)) :
                BookResource::collection(Book::orderBy('public_start', 'DESC')-
>get()->take($booksCount));
        return $data;
    }
}
```

Контролер для профілю користувача

```
class IndexController extends Controller
{
    public function __invoke(User $user) {
        return PostResource::collection($user->posts);
```

```
    }
}

class StoreController extends Controller
{
    public function __invoke(Request $request) {
        $data = $request->all();

        try {
            DB::beginTransaction();
            $post = Post::create($data);
            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            abort(500);
        }

        return response()->json(['success' => 'success', 200, 'dataID' => $post]);
    }
}

class IndexController extends Controller
{
    public function __invoke(User $user) {
        return BookResource::collection($user->books);
    }
}

class StoreController extends Controller
{
```

```
        public function __invoke(Request $request) {
            $data = $request->all();
            try {
                DB::beginTransaction();
                $content = Content::create($data);
                DB::commit();
            } catch (\Exception $e) {
                DB::rollBack();
                abort(500);
            }
             return response()->json(['success' => 'success', 200, 'dataID' => $con-
tent]);
        }
    }

    class DestroyController extends Controller
    {
        public function __invoke(User $user, Book $book) {
            $user->library()->detach($book->id);
            return response()->json(['success' => 'success', 200]);
        }
    }

    class IndexController extends Controller
    {
        public function __invoke(User $user) {
            return BookResource::collection($user->library);
        }
    }
```

```
class StoreController extends Controller
{
    public function __invoke(Request $request) {
        $data = $request->all();
        try {
            DB::beginTransaction();
            $library = Library::create($data);
            DB::commit();
        } catch (\Exception $e) {
            DB::rollBack();
            abort(500);
        }
        return response()->json(['success' => 'success', 200, 'dataID' => new Li-
braryResource($library)]);
    }
}
```