

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ОНЛАЙН ПЛАТФОРМИ
РОЗПОВСЮДЖЕННЯ І ПЕРЕГЛЯДУ ВЕБНОВЕЛ
DEVELOPMENT OF AN ONLINE PLATFORM FOR
DISTRIBUTING AND VIEWING WEB NOVELS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-120
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ХАЛІЗЕВ Н.С.

(ПРІЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ХАЛІЗЕВА Нікіти Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка онлайн платформи розповсюдження і перегляду вебновел. Development of an Online Platform for Distributing and Viewing Web Novels

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 3 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Обрання засобів та середовища розробки. 3. Опис програмної реалізації вебзастосунку перегляду вебновел. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Нікіта ХАЛІЗЕВ

(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Ольга ГЛАДКОВА

(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 154 с., 10 табл., 45 рис., 3 дод., 29 джерел.

ВЕБНОВЕЛА, ВЕБДОДАТОК ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, REACT, NODEJS, EXPRESS, MONGODB.

Об'єкт дослідження – процес розробки програмного забезпечення для перегляду та розповсюдження вебконтенту.

Предмет дослідження – програмне забезпечення для перегляду та розповсюдження вебновел .

Мета роботи – розробка програмного забезпечення для перегляду та розповсюдження вебновел.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування JavaScript, мова гіпертекстової розмітки HTML, фреймворк Express, бібліотека React, CSS фреймворк TailwindCSS, бібліотека React компонентів MaterialUI, база даних MongoDB, персональний комп'ютер з процесором AMD Ryzen 3 3600 під управлінням операційної системи Windows 10 Version 22H2.

Результати. Розроблено програмне забезпечення для перегляду та розповсюдження вебновел.

Висновки. Розроблене програмне забезпечення дозволяє переглядати та розповсюджувати вебновели.

Галузь використання – програмний засіб для розповсюдження та перегляду вебновел.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
154 pages, 10 tables, 45 figures, 3 appendixes, 29 sources.

WEBNOVEL, WEB APPLICATION, SOFTWARE, REACT, NODEJS,
EXPRESS, MONGODB.

The object of research is the process of developing software for viewing and distributing webcontent.

The subject of research is software for viewing and distributing webnovels.

Purpose - to develop software for viewing and distributing webnovels.

Materials, methods and techniques: object-oriented programming, JavaScript programming language, HTML hypertext markup language, Express framework, React library, TailwindCSS CSS framework, MaterialUI library of React components, MongoDB database, personal computer with AMD Ryzen 3 3600 processor running Windows 10 Version 22H2.

Results. The software for viewing and distributing webnovels has been developed.

Conclusions. The developed software allows viewing and distributing webnovels.

The field of use. Is a software tool for distributing and viewing webnovels.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Поняття вебновели.....	10
1.2 Аналіз існуючих рішень.....	10
1.3 Висновок за розділом.....	15
2 ОБРАННЯ ЗАСОБІВ ТА СЕРЕДОВИЩА РОЗРОБКИ.....	16
2.1 Вибір мови програмування.....	16
2.2 Фреймворки та база даних.....	17
2.3 Середовище розробки та редактор коду.....	19
2.4 Висновки за розділом.....	20
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ ВЕБНОВЕЛ.....	21
3.1 Структурна схема розробки.....	21
3.2 Опис архітектури програми.....	22
3.3 Опис серверної частини додатку.....	23
3.3.1 Використані залежності.....	23
3.3.2 Файлова структура серверної частини.....	24
3.3.3 Система маршрутизації.....	25
3.4 Опис клієнтської частини додатку.....	30
3.4.1 Використані залежності.....	30
3.4.2 Файлова структура клієнтської частини.....	31
3.5 Алгоритм роботи програми.....	33
3.5.1 Структурна схема сайту.....	33
3.5.2 Алгоритми роботи додатку.....	33
3.6 Реалізація онлайн вебдодатку.....	36
3.6.1 Головна сторінка.....	36

3.6.2 Сторінка перегляду інформації про вебновелу.....	37
3.6.3 Сторінка додавання нової вебновели до каталогу.....	40
3.6.4 Сторінка перегляду глави вебновели.....	41
3.6.5 Сторінка результат пошуку.....	42
3.6.6 Сторінки авторизації та реєстрації.....	43
3.6.7 Сторінка профілю користувача.....	44
3.6.8 Сторінка списку збережених вебновел.....	45
3.7 Висновки за розділом.....	46
4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ.....	47
4.1 Призначення й умови застосування програми.....	47
4.2 Тестування програми.....	47
4.2.1 Функціональне тестування.....	48
4.2.2 Тестування зручності сайту.....	50
4.3 Висновки за розділом.....	51
ВИСНОВКИ.....	52
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	53
ДОДАТОК А Технічне завдання.....	56
ДОДАТОК Б Текст програми.....	60
ДОДАТОК В Слайди презентації.....	146

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
JS	– JavaScript;
PHP	– Hypertext Preprocessor;
IDE	– Integrated Development Environment;
MVC	– Model-View-Controller;
SQL	– Structured Query Language;
NoSQL	– Non Structured Query Language;
HTML	– HyperText Markup Language;
CSS	– Cascading Style Sheets;
MERN	– MongoDB Express React NodeJS;
REST	– Representational State Transfer;
БД	– база даних;
СКБД	– система керування базами даних.

ВСТУП

Вебтехнології продовжують свій стрімкий розвиток, створюючи нові можливості для комунікації, розваг та навчання. Однією з таких можливостей є розповсюдження літературного контенту через інтернет.

З самого початку існування інтернету письменники почали ділитися своїми художніми творами з читачами онлайн. Вебсайти стали для письменників простором, де вони могли публікувати свої історії, знаходити аудиторію та отримувати відгук на свої роботи [1].

І саме через доступність та швидкість публікації творів онлайн почали набувати популярність вебновели.

Саме тому об'єктом дослідження стала розробка онлайн платформи для розповсюдження та перегляду вебновел. Розробка такої платформи дозволить дослідити повний цикл реалізації основних функцій платформи та надасть фундамент для подальшого розвитку та створення аналогічних платформ.

Метою даної дипломної роботи є розробка онлайн платформи для розповсюдження та перегляду вебновел.

Для досягнення поставленої мети у дипломній роботі були вирішенні такі завдання:

- провести огляд існуючих платформ для розповсюдження вебновел;
- розробити архітектуру та функціональні можливості онлайн платформи для розповсюдження та перегляду вебновел;
- реалізувати розроблену платформу та провести її тестування.

Результатом цієї роботи буде створення функціонального вебзастосунку, який забезпечить користувачам зручні інструменти для завантаження, перегляду, коментування та пошуку вебновел.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття вебновели

Вебновела — це роман, опублікований в інтернеті. Вона може бути безкоштовною або платною. Зазвичай вебновели є серійними, тобто автор публікує історію частинами — часто по одній главі за раз, і зазвичай виходять за розкладом. Публікація твору на вебплатформі, яка дозволяє автору безпосередньо взаємодіяти зі своєю аудиторією, а читачі можуть залишити відгук на кожне нове доповнення до історії [1].

Вебновели надзвичайно популярні в Китаї, Японії та Південній Кореї [1-2].

У Китаї індустрія вебновел процвітає. Платформи для самостійної публікації отримують до 40% всього інтернет трафіку у Китаї, та найпопулярніші автори можуть заробляти мільйони [1]- [2].

У Південній Кореї внутрішня індустрія вебновел коштує понад 400 мільйонів доларів США, а ринок зріс на 4000% між 2013 та 2019 роками [1]- [2].

У 2021 році південнокорейський технологічний гігант Какао придбав платформи для розповсюдження вебновел Radish і Tapas за 950 мільйонів доларів США, а його конкурент Naver придбав Wattpad приблизно за 600 мільйонів доларів [1].

1.2 Аналіз існуючих рішень

WebNovel – популярна онлайн платформа для перегляду вебновел та коміксів, яка надає користувачу великий вибір творів (рис.1.1).

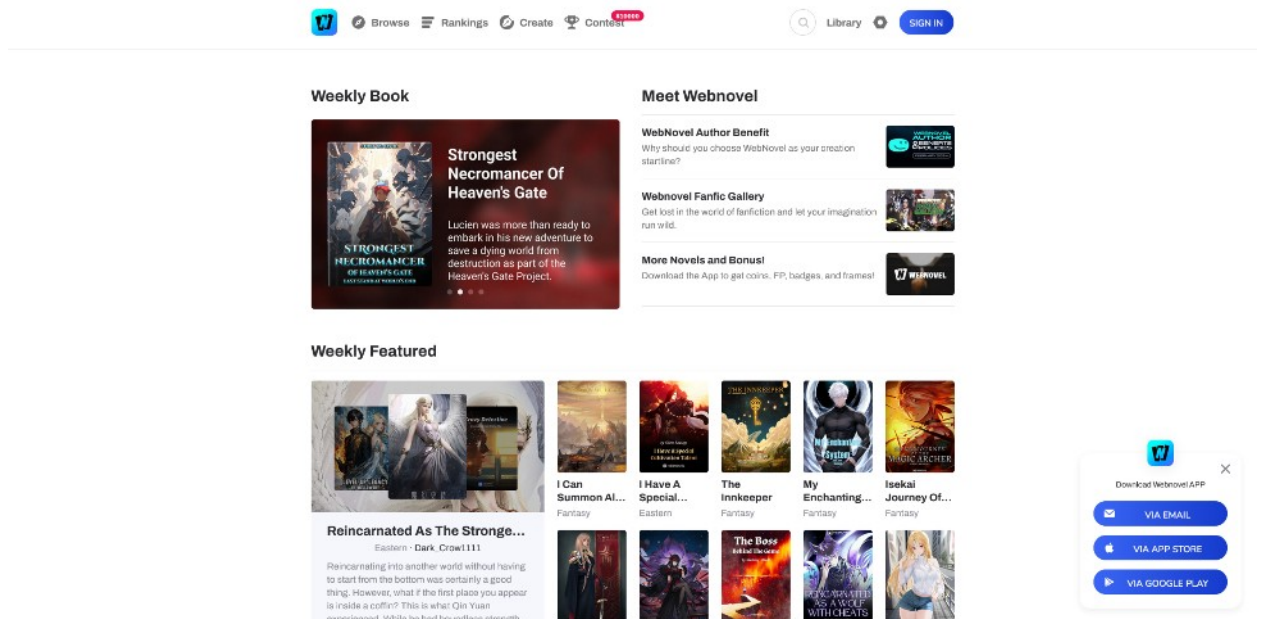


Рисунок 1.1 — Онлайн платформа WebNovel

Переваги:

- велика бібліотека творів різних жанрів й авторів;
- велика база відгуків на твори;
- можливість завантаження власного твору;
- інтерактивність та активна спільнота читачів та авторів;
- система тегів (дозволяє легко знаходити твори за інтересами користувача);
- можливість збереження творів у власній бібліотеці збережених.

Недоліки:

- монетизація (багато популярних творів доступні лише за оплату);
- якість контенту (оскільки новели можуть завантажуватися користувачами, не всі твори мають високу літературну якість);
- перевантажений інтерфейс.

WuxiaWorld – це провідний англомовний видавець китайських і корейських вебновел. Вони надають доступ до різноманітних творів, включно з історіями про безсмертних культиваторів, могутніх драконів та неймовірно сильних істот (рис.1.2).

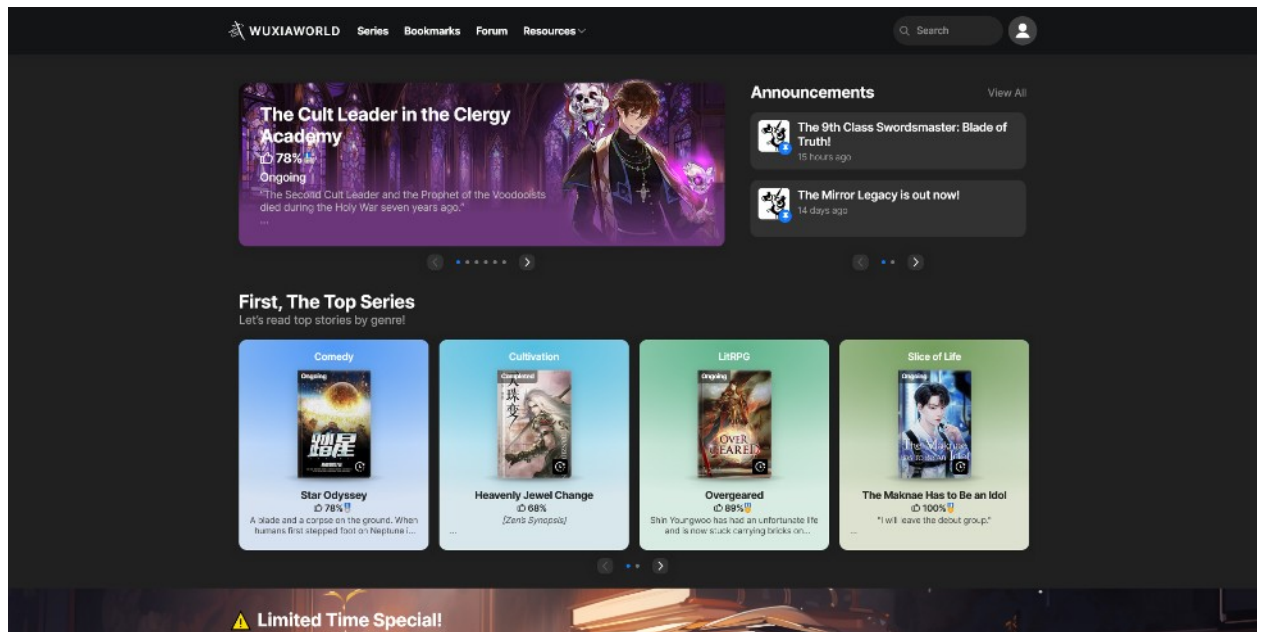


Рисунок 1.2 — Онлайн платформа WuxiaWorld

Переваги:

- велика бібліотека творів різних жанрів й авторів;
- безкоштовність контенту (більшість новел доступні для читання безкоштовно);
- простий та інтуїтивно зрозумілий інтерфейс;
- регулярні оновлення вебновел.

Недоліки:

- орієнтованість на творах жанру «уся», що може обмежити кількість доступних творів;
- монетизація (багато популярних творів доступні лише за оплату або обмежено доступ до нових глав);
- реклама (наявна реклама може заважати або відволікати при читанні новел).

Light Novel Pub – це вебсайт, де можна читати ранобе та вебновели, перекладені на англійську мову (рис.1.3).

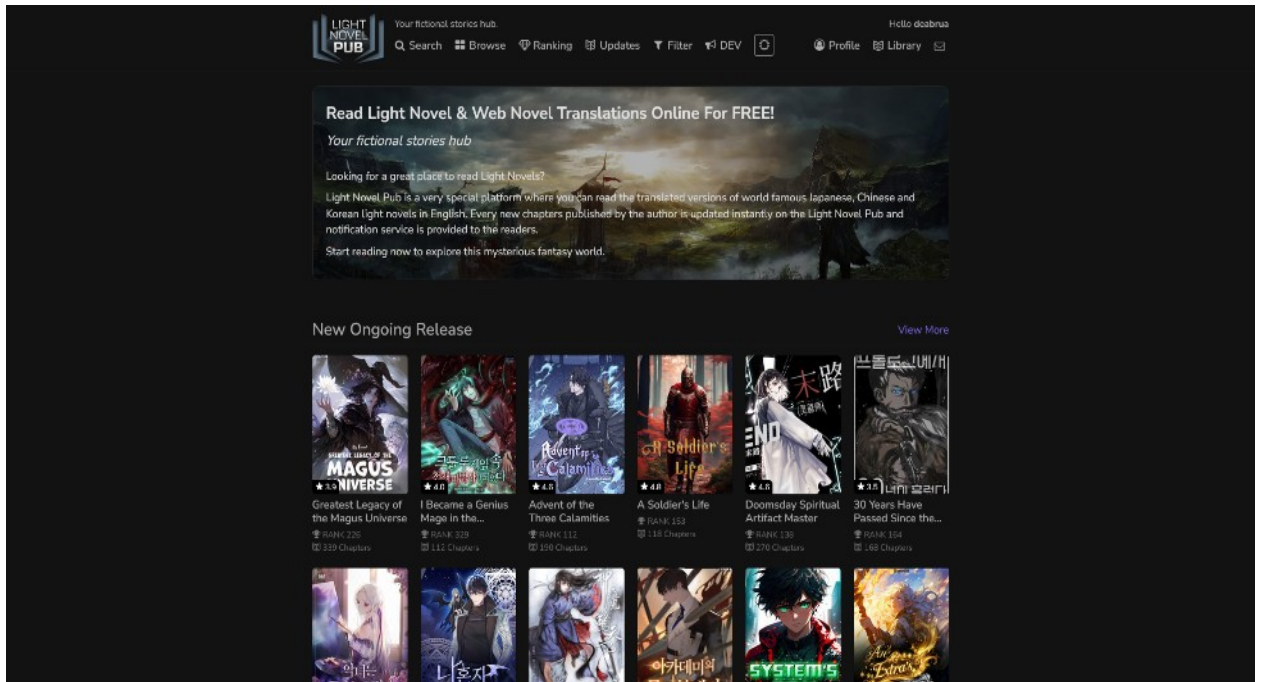


Рисунок 1.3 — Онлайн платформа Light Novel Pub

Переваги:

- велика колекція вебновел;
- безкоштовний доступ (багато творів доступні для читання безкоштовно);
- регулярні оновлення та нові випуски (сайт регулярно оновлює свою колекцію творів);
- простий та інтуїтивно зрозумілий інтерфейс.

Недоліки:

- реклама (наявна реклама може заважати або відволікати при читанні новел);
- низька якість перекладу (переклад може бути неякісним, може використовуватися машинний переклад);
- обмеженість доступу.

Порівняльний аналіз онлайн платформ для розповсюдження та перегляду вебновел наведено у табл. 1.1.

Таблиця 1.1 — Порівняльна таблиця платформ для розповсюдження та перегляду вебновел

Критерій порівняння	WebNovel	WuxiaWorld	Light Novel Pub
Велика колекція новел різноманітних жанрів	+	-	+
Соціальні елементи	+	+	+
Розвинена система жанрів	-	+	+
Простий та інтуїтивно зрозумілий інтерфейс	-	+	+
Наявність реклами	-	+	+
Платний доступ до творів/глав	+	+	-

Після проведеного аналізу існуючих рішень видно, що існує досить багато платформ для розповсюдження та перегляду вебновел. Проте, деякі платформа містять рекламну інформації або мають платний доступ до контенту, що ускладнює їх використання. Також, вебсайти з платним доступом можуть бути під впливом видавництв, що може призвести до необ'єктивності у просуванні творів різних авторів. Деякі вебсайти можуть надавати неякісний переклад новел, що напряду впливає на враження від твору.

Тому, при розробці онлайн платформи для розповсюдження та перегляду вебновел необхідно забезпечити такі функціональні вимоги:

- забезпечення функцій реєстрації та авторизації користувача;
- підтримка процесів відображення інформації про новели;
- можливість перегляду нових глав у творах;
- можливість коментування вмісту новел та глав;

- можливість оцінки вебновел;
- підтримка системи жанрів;
- можливість пошуку творів за назвою.

1.3 Висновок за розділом

У розділі було проведено аналіз предметної області. Було здійснено огляд та порівняння існуючих онлайн платформ, виявлено їх переваги та недоліки. Після чого було сформовано

2 ОБРАННЯ ЗАСОБІВ ТА СЕРЕДОВИЩА РОЗРОБКИ

2.1 Вибір мови програмування

При виборі мови програмування для реалізації програмного забезпечення необхідно враховувати не лише її здатність вирішувати поставлене завдання, але й масштабованість та складність підтримки майбутнього проєкту.

Популярність мови програмування забезпечує велику кількість матеріалів та фахівців, що дозволяє при появі деяких складнощів, у ході розробки проєкту, звернутися до них для пошуку рішень.

У поточний проміжок часу, найпопулярнішими мовами програмування для веброзробки є C#, Java, JavaScript та PHP. Кожна з перелічених мов має свої переваги та недоліки.

У табл. 2.1 наведено порівняння перелічених мов програмування, відповідно до потреб проєкту.

Таблиця 2.1 — Порівняння мов програмування

Критерій порівняння	C#	JavaScript	PHP
Велика та активна спільнота розробників	+ -	+	-
Велика кількість підтримуваних СКБД	+ -	+	+
Висока швидкість написання коду	-	+	+
Орієнтованість на веброзробку	+ -	+	+
Висока продуктивність	+	-	-

Отже, для розробки програмного забезпечення обрано мову програмування JavaScript, яка широко застосовується для розробки вебдодатків.

JavaScript — це високорівнева скриптова мова програмування, яка широко використовується для створення інтерактивних застосунків і сайтів [3]. Функції JavaScript можуть поліпшити зручність взаємодії користувача з вебсайтом. JavaScript є мовою програмування під час розроблення скриптів для виконання на стороні клієнта, що робить його однією з базових технологій у всесвітній мережі Інтернет. Наприклад, карусель зображення, меню, що випадає за кліком, і динамічно мінливі кольори елементів на веб-сторінці, які ви бачите під час перегляду сторінок в Інтернеті, виконані за допомогою JavaScript [4].

2.2 Фреймворки та база даних

Фреймворк — це набір повторно використовуваних програмних компонентів, які роблять розробку нових додатків більш ефективною. Повторне використання існуючих розробок і досліджень є важливим принципом у всіх інженерних галузях [5].

Фреймворки вебзастосунків складаються з інструментів програмування та ресурсів для створення веб-додатків, веб-сайтів, односторінкових додатків та їхніх відповідних серверних сервісів. Вони допомагають розробникам інтегрувати різні веб-компоненти та мови програмування для створення функціональних веб-додатків [5].

2.2.1 MERN-стек

MERN — це готовий стек технологій, заснований на технологіях JavaScript. MERN розшифровується як MongoDB, Express, React і NodeJS, на честь чотирьох ключових технологій, з яких складається стек [6].

Основні компоненти стека MERN включають: MongoDB, Express, React, NodeJS.

MongoDB: потужна система управління базами даних, яка використовує документ-орієнтовану модель даних. У порівнянні з традиційними реляційними базами даних, такими як MySQL або PostgreSQL, MongoDB пропонує більш гнучкий підхід до зберігання та опрацювання даних [7].

Express: серверний фреймворк, що працює всередині сервера NodeJS. Express.js позиціонує себе як «швидкий, незалежний, мінімалістичний веб-фреймворк для NodeJS», і це дійсно саме те, чим він є. Express.js має потужні моделі для маршрутизації URL-адрес (зіставлення вхідної URL-адреси з функцією сервера) та обробки HTTP-запитів і відповідей [6].

React: декларативний JavaScript-фреймворк для створення динамічних клієнтських додатків у форматі HTML. React дозволяє створювати складні інтерфейси за допомогою простих компонентів, підключати їх до даних на внутрішньому сервері та відображати у вигляді HTML [6].

NodeJS: програмне середовище, яке дозволяє запускати програми, написані мовою Javascript, поза браузером. Історично програми, написані на Javascript, на відміну від інших мов програмування, можна було запуснути лише у браузерах, які мали спеціальний вбудований движок виконання коду цієї мови. Поза браузером Javascript, можна сказати, не працював [8].

2.2.2 MongoDB

MongoDB – це документ-орієнтована система керування базами даних (СКБД) типу NoSQL. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Кожна колекція складається з документів, які в свою чергу складаються з полів. Поля є парами ключ-значення. Колекції також можуть бути індексовані, що покращує швидкість вибірки та сортування [7, 9-10].

Однією з особливостей MongoDB є використання BSON (Binary JSON) для представлення та обміну даними [7].

MongoDB дуже добре працює з NodeJS, при використанні бібліотеки Mongoose забезпечується зручна і ефективна робота з базою даних. Mongoose дозволяє легко створювати схеми для колекцій, перевіряти дані, виконувати запити та взаємодіяти з MongoDB у стилі ORM (Object-Relational Mapping) [7, 9-10]. Це значно спрощує розробку додатків, оскільки забезпечує абстракцію над низькорівневими операціями з базою даних та покращує читабельність і підтримку коду.

2.3 Середовище розробки та редактор коду

Вибір інтегрованого середовища розробки (IDE) або редактору безпосередньо впливає на продуктивність розробника. Правильний інструмент може значно спростити процес написання, тестування та налагодження коду.

На сьогоднішній день є кілька популярних IDE та редакторів коду для роботи з JavaScript. Їх загальна характеристика представлена у табл. 2.2.

Таблиця 2.2 — Загальна характеристика середовищ розробки

IDE	Особливості	Платформи	Вартість
Visual Studio Code	Легкий та масштабований редактор коду з великою кількістю плагінів	Windows, macOS, Linux	Безкоштовний
WebStorm	Повнофункціональне	Windows, macOS, Linux	Платний (безкоштовна пробна версія)
Sublime Text	Легкий редактор коду з великою кількістю плагінів	Windows, macOS, Linux	Платний (безкоштовна пробна версія)

Розглянемо переваги та недоліки даних IDE та редакторів коду у табл. 2.3.

Таблиця 2.3 — Переваги та недоліки середовищ розробки

IDE	Переваги	Недоліки
Visual Studio Code	Безкоштовний; відкритий вихідний код; велика кількість плагінів; часті оновлення; наявність офіційних плагінів найпопулярніших фреймворків.	Побудований на Electron (може бути повільним при великій кількості розширень).
WebStorm	Потужні інструменти для роботи з найпопулярнішими фреймворками JavaScript; швидкодія; вбудована підтримка інструментів III; повноцінна IDE.	Платний; може бути вимогливим до системних ресурсів порівняно з редакторами коду; деякі користувачі відзначають перевантажений інтерфейс.
Sublime Text	Легкий та швидкий на слабких ПК; велика кількість плагінів.	Платний; гірша інтеграція з фреймворками та VCS, порівняно з конкурентами

Visual Studio Code є найкращим вибором з огляду на вартість та підтримку використовуваних фреймворків.

2.4 Висновки за розділом

У даному розділі було обрано мову програмування – JavaScript, та стек для розробки вебдодатку — MERN. Також, базуючись на обраних фреймворках та мові програмування було обрано середовище розробки — Visual Studio Code.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ ВЕБНОВЕЛ

3.1 Структурна схема розробки

Загальна схема роботи даного вебдодатку включає три кроки перераховані нижче.

Крок 1. Клієнт за допомогою HTTP-запитів, використовуючи GET та POST, взаємодіє з серверною частиною додатку. Запити містять введену або запитувану користувачем інформацію, таку як дані для авторизації або реєстрації, або запити на отримання певних даних з бази даних. Клієнтська частина була розроблена з використанням React, TailwindCSS та MaterialUI.

Крок 2. Серверна частина вебдодатку приймає запити від клієнтської частини і обробляє їх. Вона виконує різноманітні операції, включаючи збереження, оновлення, видалення або витягування даних з бази даних. Серверна частина побудована з використанням фреймворку Express, а для взаємодії з базою даних використана бібліотека Mongoose.

Крок 3. База даних MongoDB зберігає дані, які були отримані від сервера. MongoDB зберігає дані у вигляді BSON-документів, без необхідності визначення схеми таблиць. Це зробить її гнучкою та масштабованою, оскільки забезпечує можливості подальшої модифікації структури бази даних.

Схематичне зображення схеми роботи вебдодатку представлено на рис. 3.1.

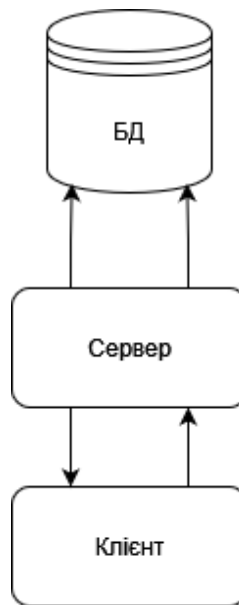


Рисунок 3.1 — Загальна схема роботи вебдодатку

3.2 Опис архітектури програми

Для побудови архітектури проєкту використовується шаблон проектування Model-View-Controller (MVC).

Model-View-Controller — це архітектурний шаблон, який розділяє додаток на три основні компоненти: модель, представлення і контролер [11-12].

«Модель» керує основною поведінкою та даними додатку. Потім «Представлення» відповідає за відображення цих даних, а «Контролер» обробляє вихідні дані від користувачів [13].

Однією з ключових характеристик архітектури MVC є її циклічність. Користувач взаємодіє з представленням, яке взаємодіє з контролером, щоб оновити або отримати дані з моделі. Потім оновлені дані відображаються у представленні, і цикл продовжується [13].

Схематичне зображення схеми роботи архітектурного шаблону MVC представлено на рис. 3.2.

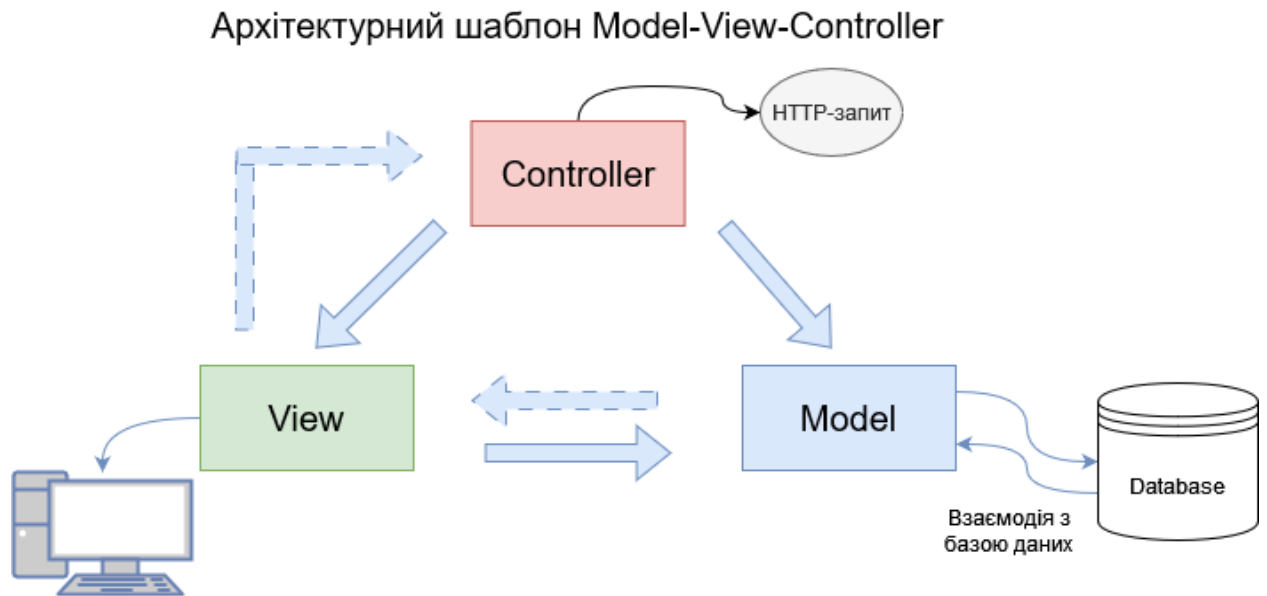


Рисунок 3.2 — Діаграма архітектурного шаблону MVC [13]

При використанні шаблону MVC та стеку MERN, у ролі моделі працює MongoDB. Вона описує структуру даних та надає методи для взаємодії з базою даних [11]- [13].

У стеці MERN представлення будується за допомогою React, популярної бібліотеки для побудови користувацьких інтерфейсів. Вона відповідає за визначення компонентів інтерфейсу, а файли JSX (JavaScript XML) використовуються для опису їхньої структури та зовнішнього вигляду [11]- [13].

Express, фреймворк, що працює на NodeJS, формує контролер. Він прослуховує HTTP-запити від клієнта, оброблює їх, взаємодіє з моделлю (MongoDB) для маніпуляцій з даними та надсилає відповідь назад клієнту [13].

3.3 Опис серверної частини додатку

3.3.1 Використані залежності

Для встановлення всіх залежностей був використаний пакетний менеджер Yarn.

Yarn – це відомий менеджер пакетів з відкритим вихідним кодом, який використовується для керування залежностями у проектах JavaScript. Він допомагає в процесі встановлення, оновлення, конфігурації та видалення залежностей пакетів, що в кінцевому підсумку допомагає швидше досягати поставлених цілей з меншою кількістю відволікаючих чинників [14-15].

При розробці серверної частини були використані наступні залежності:

- bcryptjs: для хешування паролів з використаннями алгоритму bcrypt;
- cloudinary: для управління зображеннями [16];
- cookie-parser: для аналізу HTTP-запитів з cookie;
- dotenv: для завантаження змінних середовища з файлу .env;
- express: фреймворк для розробки серверного додатку [17];
- express-validator: для валідації даних;
- jsonwebtoken: для роботи з JWT токенами;
- mongoose: бібліотека для роботи з MongoDB [18];
- nodemon: інструмент, який допомагає перезапускати сервер автоматично при зміні файлів.

3.3.2 Файлова структура серверної частини

Для серверної частини (рис. 3.3) після встановлення залежностей було сформовано основну структуру вебдодатку, це спростить подальшу розробку, обслуговування та оновлення додатку.

Основна структура складається з конфігураційних файлів та 7 тек:

- controllers: тека з контролерами вебсерверу;
- database: тека з фалами та модулями пов'язані з роботою БД;
- lib: тека з різноманітними допоміжними функціями;
- middleware: тека з функціями проміжної обробки запитів;
- models: тека з моделями даних для MongoDB;

- node_modules: тека де зберігаються всі залежності проєкту;
- routes: тека з файлами маршрутизації.

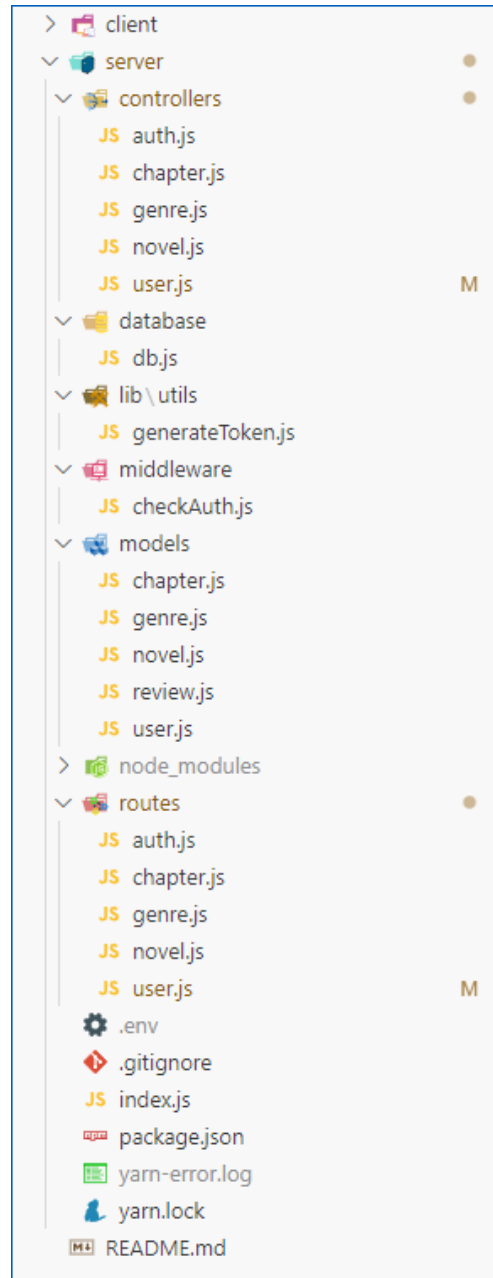


Рисунок 3.3 — Файлова структура серверної частини проєкту

3.3.3 Система маршрутизації

Система маршрутизації — це серце будь-якого сервера. Вона подібна до диспетчера, який розподіляє клієнтські запити до відповідних компонентів

серверу. Це ключовий елемент вебсервера, який забезпечує чітку та ефективну обробку запитів та відповідей.

Для розроблюваному програмному забезпеченні маршрутизація дозволяє здійснювати основні методи HTTP-запитів, а саме:

- GET: даний метод дозволяє здійснювати читання даних;
- POST: даний метод використовується для відправки даних до серверу;
- DELETE: даний метод дозволяє видаляти інформацію.

Всі розроблені маршрути можна поділити на наступні групи:

- «auth»: використовуються для здійснення процесу авторизації та реєстрації користувача;
- «user»: для здійснення дій пов'язаних з модифікацією та отриманням даних користувача;
- «novel»: для дій пов'язаних з модифікацією та отриманням даних вебновел;
- «genre»: для отриманням даних про жанри;
- «chapter»: для модифікації та отриманням даних про глави вебновел.

У табл. 3.1 — 3.5 наведено розроблені маршрути всіх груп, разом із їх функціональним призначенням та методом запиту.

Таблиця 3.1 — Маршрути групи «auth»

№	Маршрут	Метод	Призначення
1	2	3	4
1	/api/auth/me	GET	Отримання інформації про поточного авторизованого користувача
2	/api/auth/register	POST	Реєстрація нового користувача

Кінець таблиці 3.1

1	2	3	4
3	/api/auth/login	POST	Авторизація користувача
4	/api/auth/logout	POST	Вихід з системи

Таблиця 3.2 — Маршрути групи «user»

№	Маршрут	Метод	Призначення
1	2	3	4
1	/api/user/get/:id	GET	Отримати дані користувача
2	/api/user/saves/:id	GET	Отримати список збережених вебновел
3	/api/user/reviews/:id	GET	Отримати всі коментарі користувача
4	/api/user/get-by-review/:id	GET	Отримати дані користувача за id його коментаря
5	/api/user/update/:id	POST	Оновлення даних користувача

Таблиця 3.3 — Маршрути групи «novel»

№	Маршрут	Метод	Призначення
1	2	3	4
1	/api/novel/all	GET	Отримати всі вебновели
2	/api/novel/novels	GET	Отримати всі вебновели з урахування пангінації
3	/api/novel/get/:id	GET	Отримати дані конкретної вебновели

Кінець таблиці 3.3

1	2	3	4
4	/api/novel/reviews/:id	GET	Отримати всі коментарі до вебновели
5	/api/novel/genres/:id	GET	Отримати жанри вебновели
6	/api/novel/year/:year	GET	Отримати вебновели
7	/api/novel/ author/:author	GET	Отримати вебновели за автором
8	/api/novel/ search/:searchQuery	GET	Виконати пошук вебновел за запитом
9	/api/novel/ addViewCount/:id	POST	Додати до кількості переглядів вебновели +1
10	/api/novel/create	POST	Створити нову вебновелу
11	/api/novel/update/:id	POST	Оновити дані вебновели
12	/api/novel/save/:id	POST	Додати або прибрати з списку збережних користувача
13	/api/novel/review/:id	POST	Додати коментар до новели
14	/api/novel/rating/:id	POST	Встановити рейтинг вебновелі
15	/api/novel/:id	DELETE	Видалити вебновелу

Таблиця 3.4 — Маршрути групи «genre»

№	Маршрут	Метод	Призначення
1	2	3	4
1	/api/genre/all	GET	Отримати всі жанри

Кінець таблиці 3.4

1	2	3	4
2	/api/genre/:id	GET	Отримати дані конкретного жанру
3	/api/genre/novels/:id	GET	Отримати список новел за жанром

Таблиця 3.5 — Маршрути групи «chapter»

№	Маршрут	Метод	Призначення
1	2	3	4
1	/api/chapter/get/:id	GET	Отримати дані при главу
2	/api/chapter/next/:id	GET	Отримати дані про наступну главу
3	/api/chapter/prev/:id	GET	Отримати дані про попередню главу
4	/api/chapter/novel/:id	GET	Отримати всі глави новели
5	/api/chapter/review/:id	GET	Отримати всі коментарі до глави
6	/api/chapter/add	POST	Додати нову главу
7	/api/chapter/update	POST	Оновити дані глави
8	/api/chapter/review/:id	POST	Додати коментар до глави
9	/api/chapter/ addViewCount/:id	POST	Додати до кількості переглядів глави +1

Велика кількість подібних маршрутів обумовлена тим, що розроблювана система потребує актуальних даних у певний момент часу, що

дозволяє зробити запити, які надають максимально оптимізовану та чітку відповідь. Це гарантує, що користувачі завжди отримують найсвіжішу інформацію з мінімальним часом очікування.

Таким чином, створені маршрути забезпечують гнучкість та оптимальне навантаження системи. Їх можна легко використовувати та інтегрувати з клієнтським додатком, що підвищує ефективність і швидкість взаємодії між клієнтом і сервером.

3.4 Опис клієнтської частини додатку

3.4.1 Використані залежності

Для встановлення всіх залежностей був використаний пакетний менеджер Yarn [14]- [15].

При розробці клієнтської частини були використані наступні залежності:

- mui/material: це бібліотека компонентів для React, які реалізує Material Design від Google [19];
- mui/x-date-pickers: пакет для роботи з компонентами вибору дати та часу [20];
- uiw/react-md-editor: простий редактор розмітки з попереднім переглядом [21];
- dayjs: мінімалістична бібліотека, яка аналізує, перевіряє, маніпулює та відображає дати і час для сучасних браузерів [22];
- react: бібліотека для побудови інтерфейсів користувача [23];
- react-hot-toast: бібліотека, яка реалізує системні повідомлення для React додатку [24];
- react-router-dom: бібліотека для керування клієнтською навігацією у React додатках [25];

- tailwindcss: CSS-фреймворк, який надає набір базових утилітних класів, які можна компонувати для досягнення потрібного дизайну [26];
- mkyu/mui-search-bar: компонент пошукової стрічки, який реалізований з використанням Materail-UI v5+ [27].

3.4.2 Файлова структура клієнтської частини

Для клієнтської частини (рис. 3.4) після встановлення залежностей було сформовано основну структуру вебдодатку, це спростить подальшу розробку, обслуговування та оновлення додатку.

Основна структура складається з конфігураційних файлів та 7 тек:

- public: тека із статичними ресурсами, які необхідні для сервера та браузеру, ці файли не оброблюються вебпакетом і напряду копіюються до теки збірки проєкту;
- src: основна тека з вихідним кодом додатку, де зберігаються основні файли та підтеки для розробки;
- assets: тека для зберігання ресурсів, таких як зображення, шрифти, стилі CSS та інші статичні файли, які використовуються у додатку.;
- middleware: тека з функціями проміжної обробки запитів у форматі JSX;
- components: тека з файлами компонентів додатку у форматі JSX, кожен компонент являє собою автономний блок інтерфейсу користувача, який може бути повторно використаний у різних частинах додатку;
- node_modules: тека де зберігаються всі залежності проєкту;
- pages: тека з файлами сторінок додатку у форматі JSX, кожна сторінка відповідає за певний маршрут у додатку і може містити різні компоненти та логіку для відображення конкретного контенту.

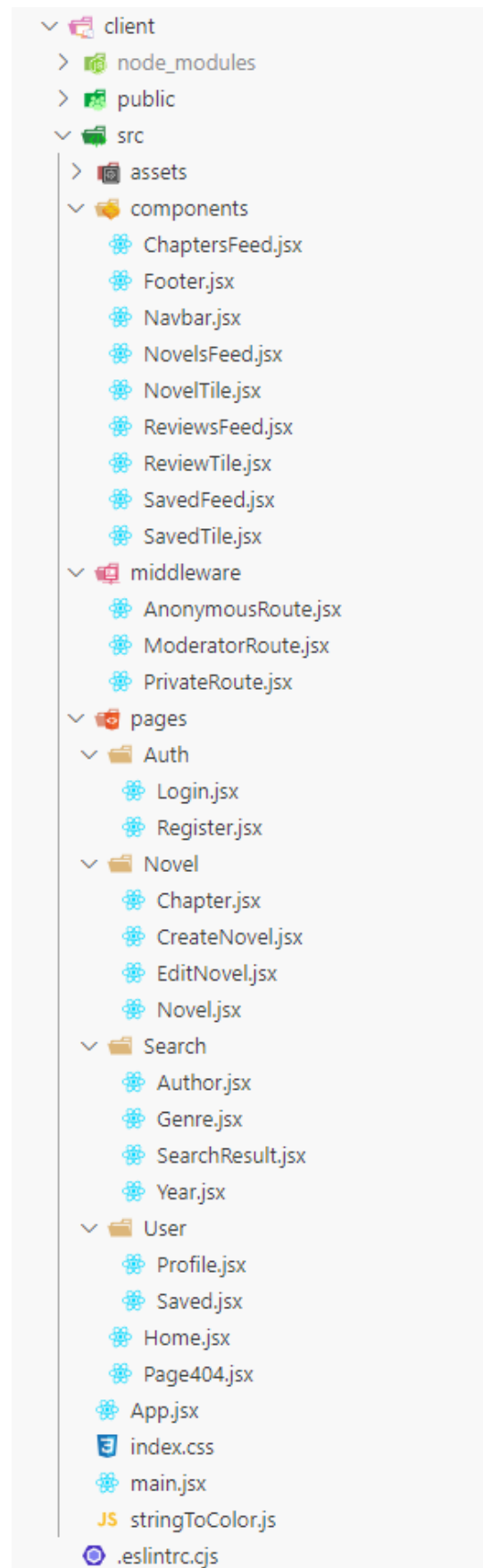


Рисунок 3.4 — Файлова структура клієнтської частини

3.5 Алгоритм роботи програми

3.5.1 Структурна схема сайту

Структурна схема сайту — це структура клієнтської частини вебдодатку. У ній продемонстровано всі сторінки згруповані по різному меню.

На рис. 3.5 представлена схема сайту.

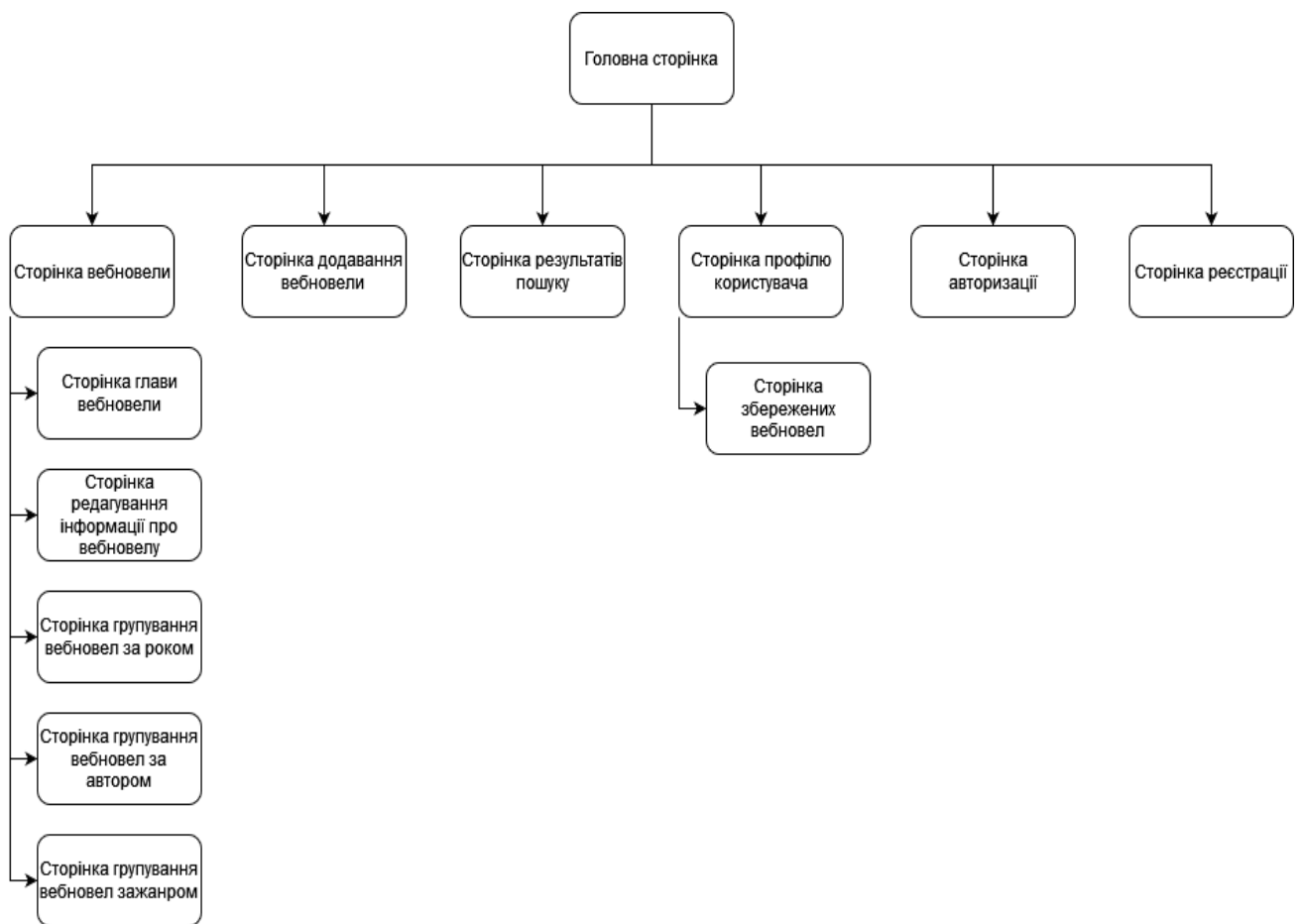


Рисунок 3.5 — Структурна схема сайту

3.5.2 Алгоритми роботи додатку

При створенні блок-схем алгоритму роботи додатку було виокремлено наступні функції:

- реєстрації користувача (рис. 3.6);

- авторизація користувача (рис. 3.7);
- додавання вебновели (рис. 3.8).

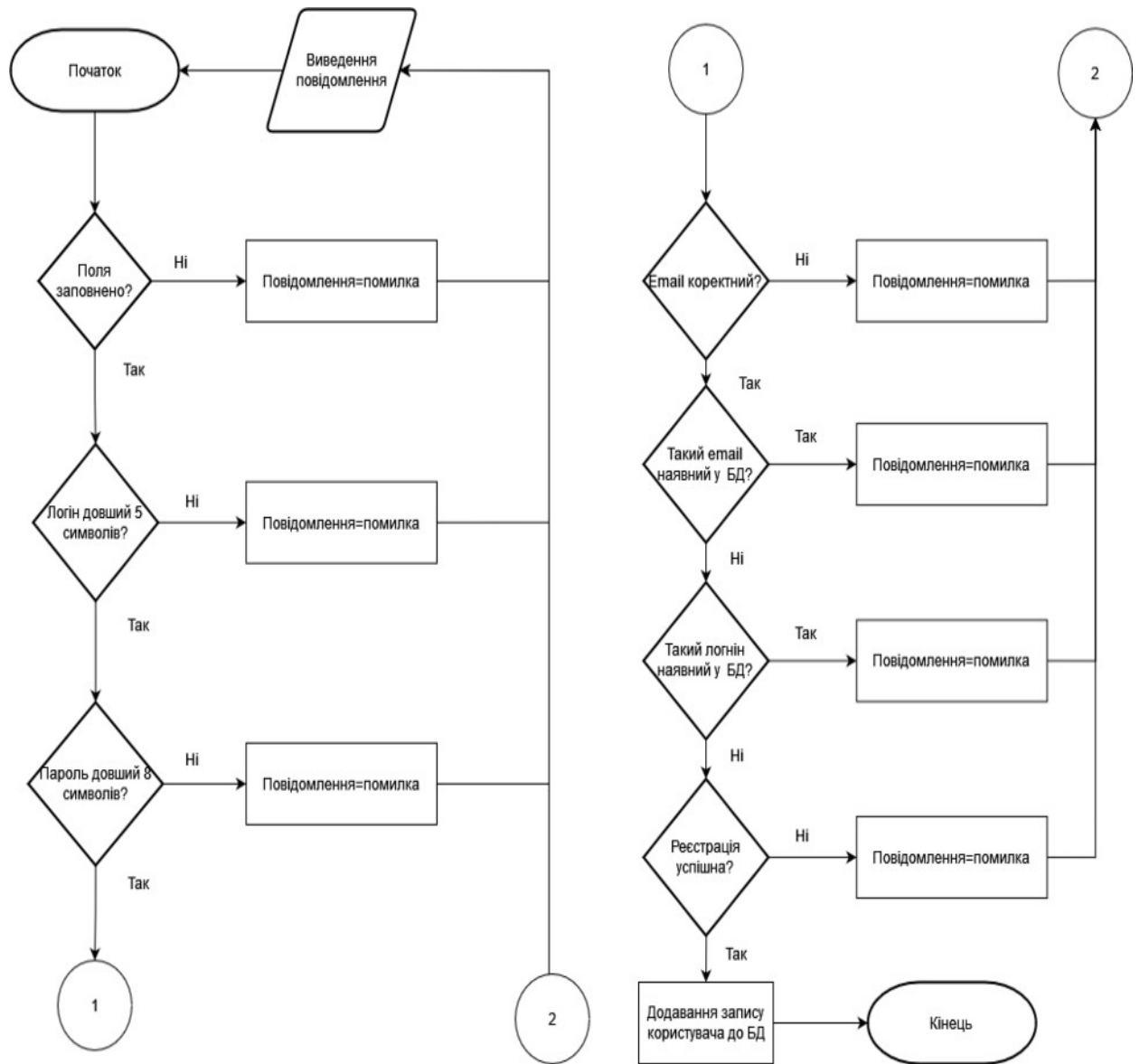


Рисунок 3.6 — Блок-схема алгоритму реєстрації користувача

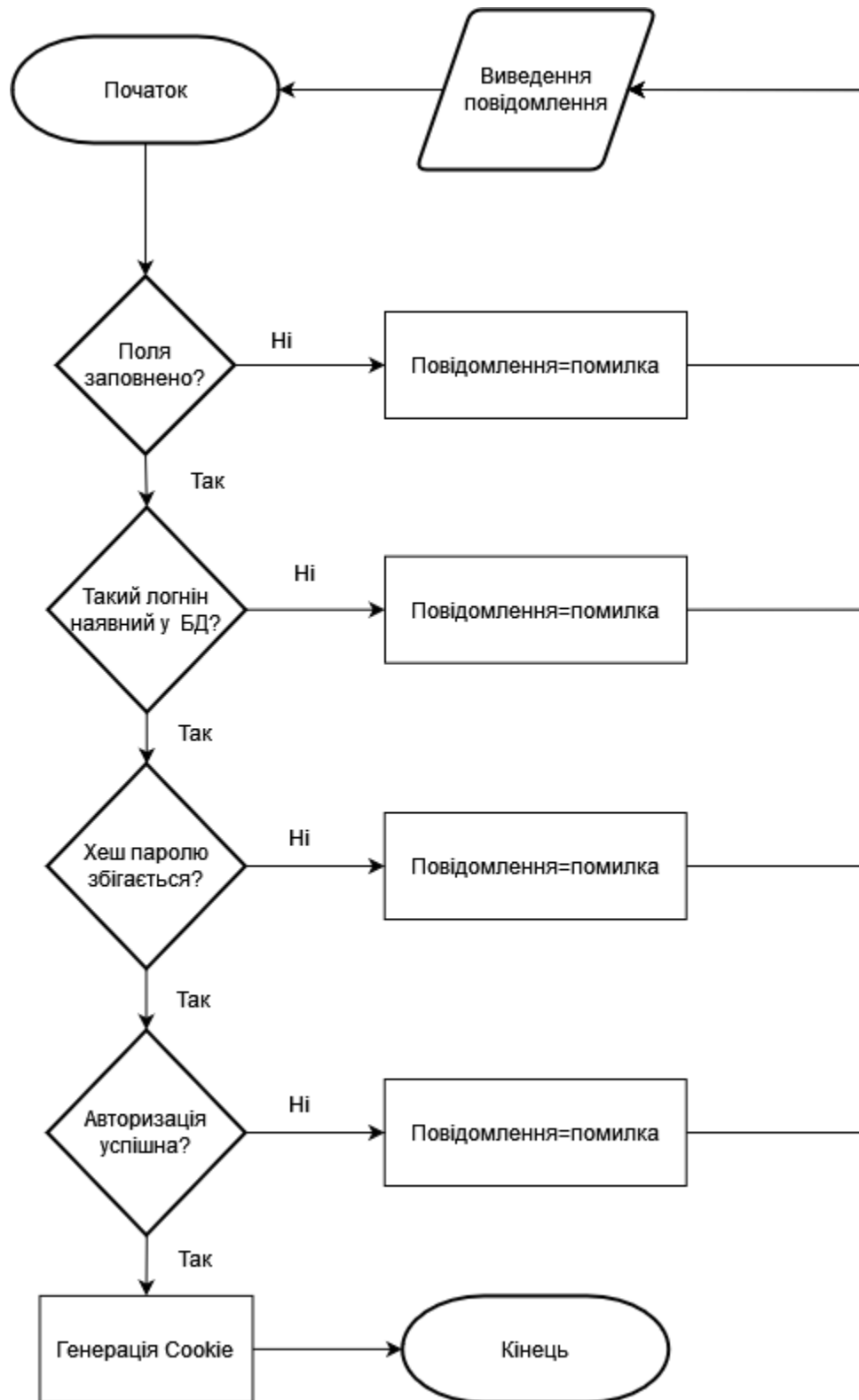


Рисунок 3.7 — Блок-схема алгоритму авторизації користувача

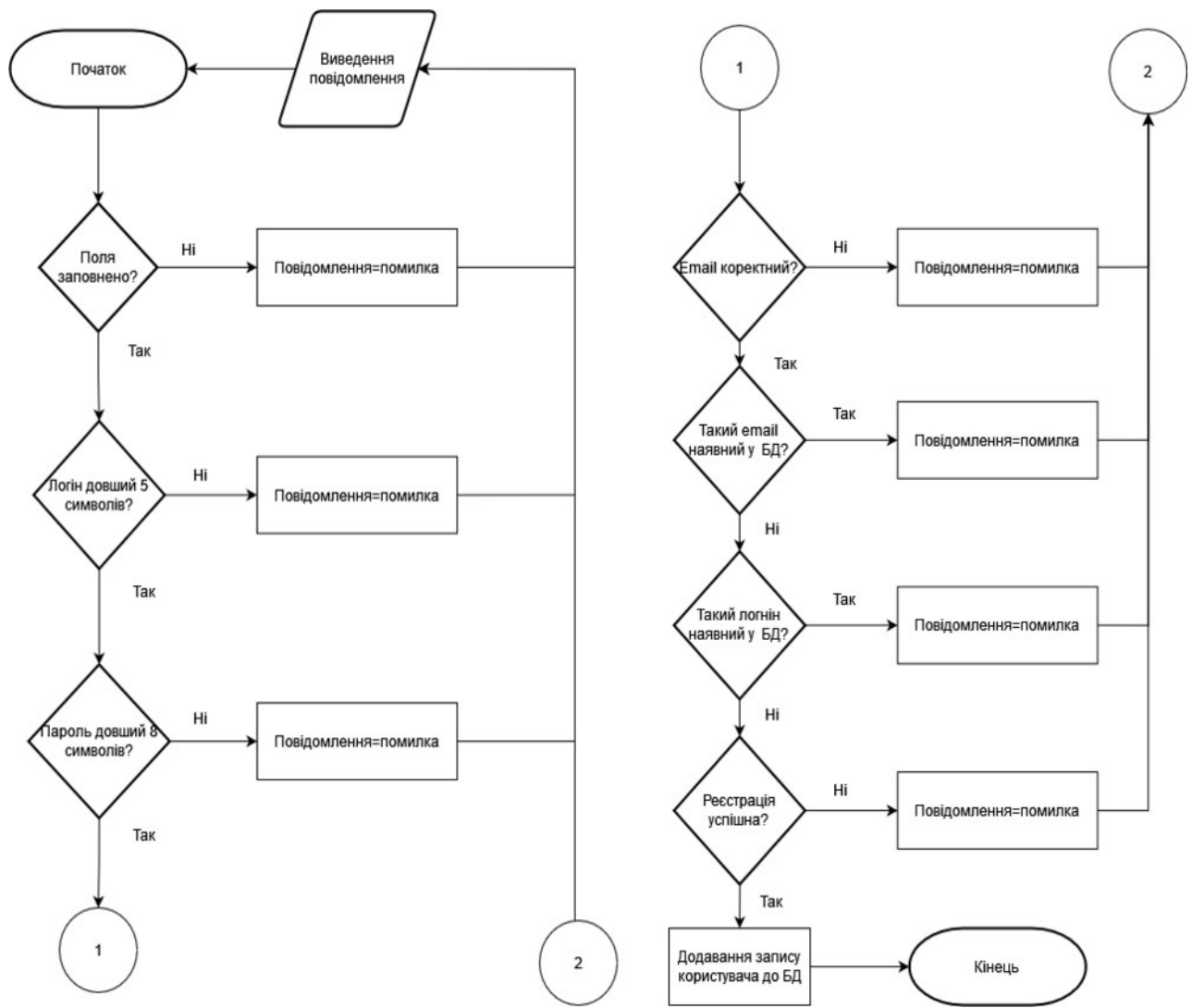


Рисунок 3.8 — Блок-схема алгоритму додавання вебновели

3.6 Реалізація онлайн вебдодатку

3.6.1 Головна сторінка

Головна сторінка (рис. 3.9) повинна відображати плитку вебновели, які упорядковані за датою останнього оновлення. Самі ж плитки повинні містити основну інформацію про вебновелу, а саме обкладинку, назву, опис, автора, оцінку та жанри твору.

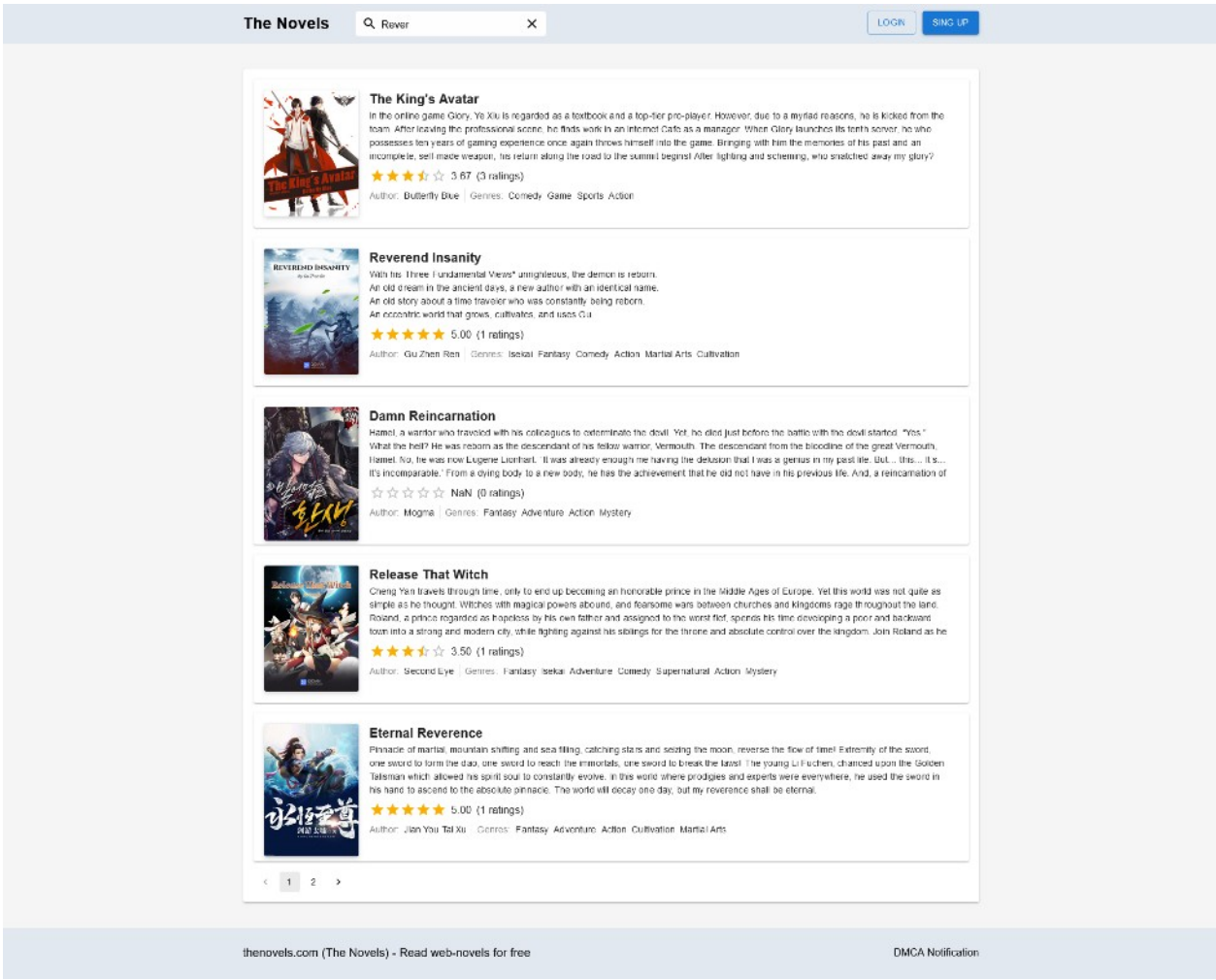


Рисунок 3.9 — Головна сторінка вебзастосунку

3.6.2 Сторінка перегляду інформації про вебновелу

Після натискання на обкладинку або назву новели, користувача переправить на саму сторінку новели (рис. 3.10), вона всі дані про новелу, а також список глав та коментарі до новели. Якщо користувача зареєстрований йому буде доступна кнопка для збереження новели у списку збережених.

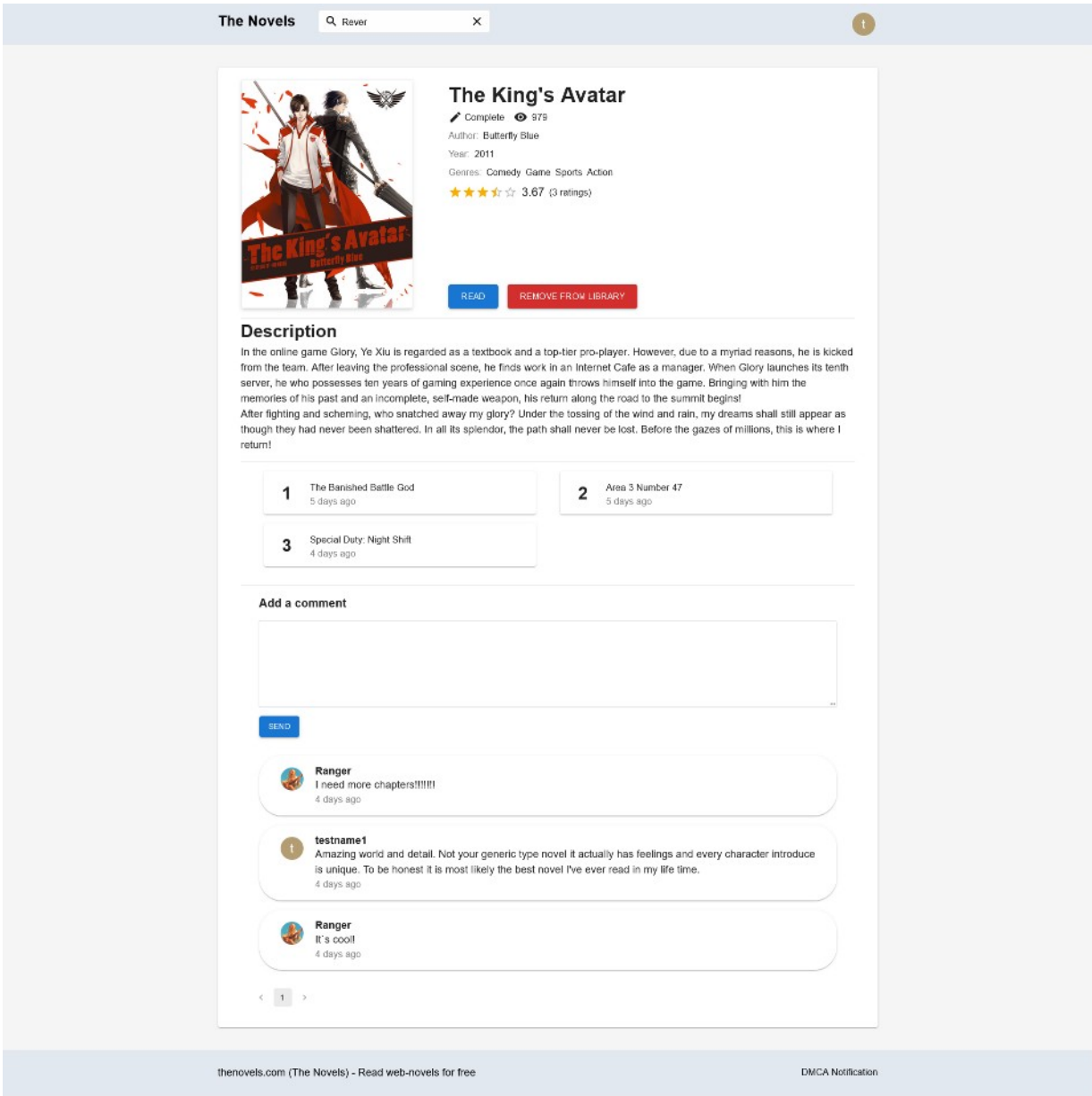


Рисунок 3.10 — Сторінка новели для користувача

Також, якщо користувач є модератором, йому стануть доступні наступні опції: видалення вебновели, зміна інформації про вебновелу та додавання нових глав до твору. Це зображено на рис. 3.11 — 3.13.

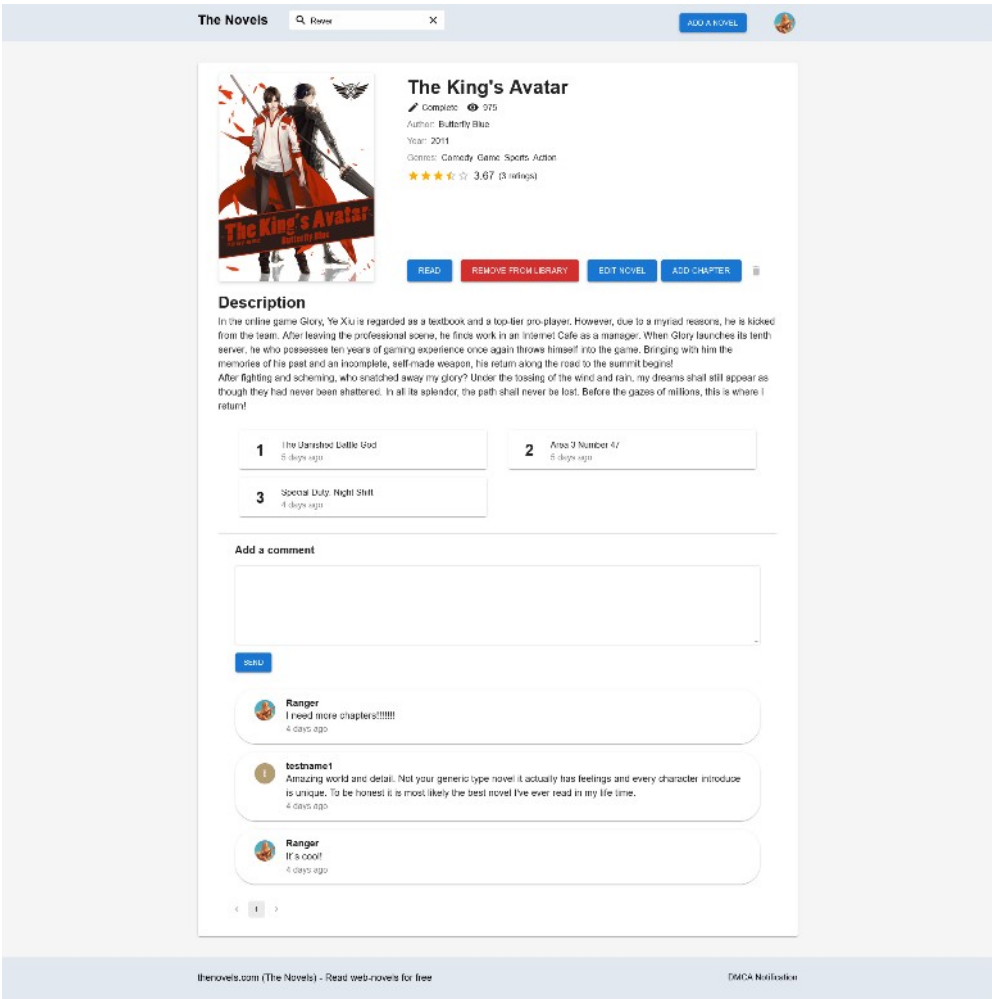


Рисунок 3.11 — Сторінка вебновели для модератора

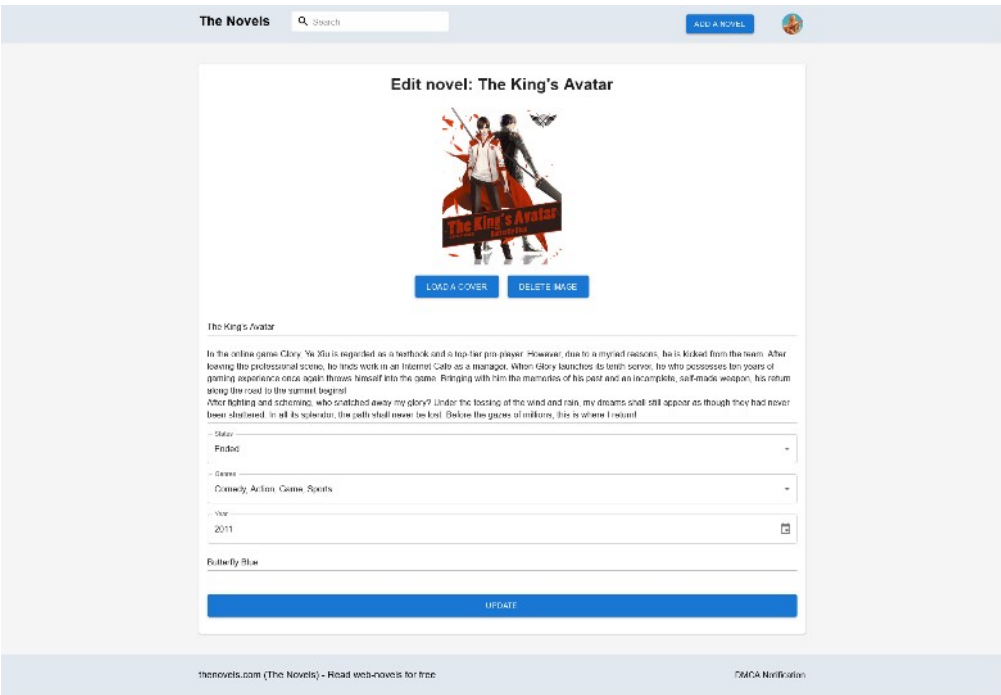


Рисунок 3.12 — Редагування інформації про вебновелу

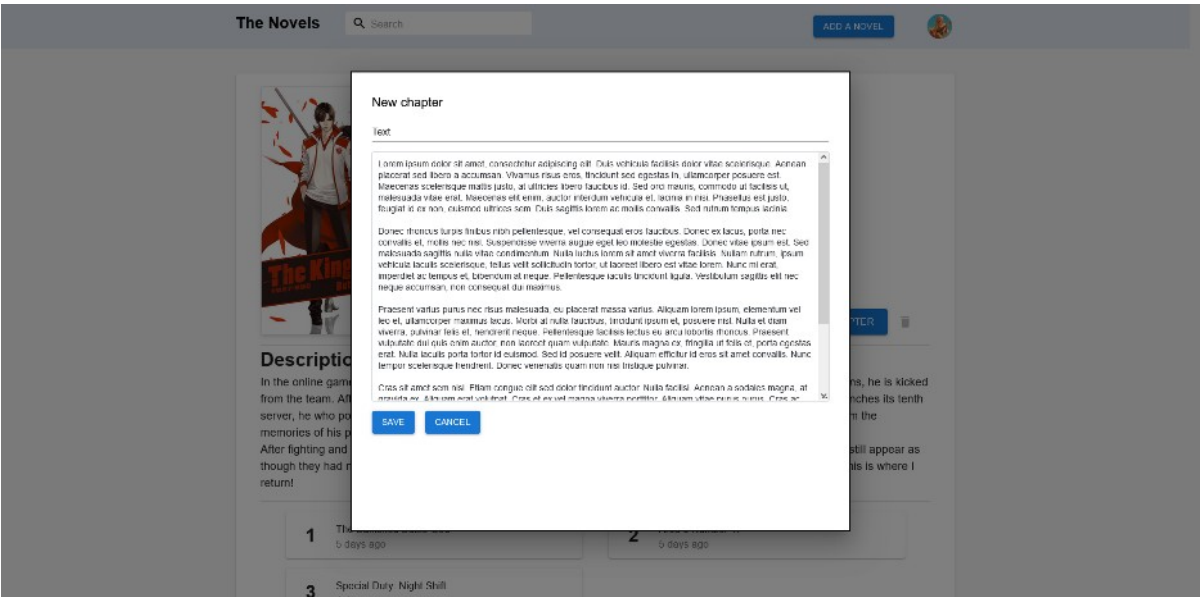


Рисунок 3.13 — Створення нової глави до вебновели

3.6.3 Сторінка додавання нової вебновели до каталогу

Дана сторінка (рис. 3.14) доступна лише для модератора сайту, на ній він може додати основну інформацію про вебновелу сайт, а саме: завантажити обкладинку; вказати назву, опис, автора, рік видання, жанри твору.

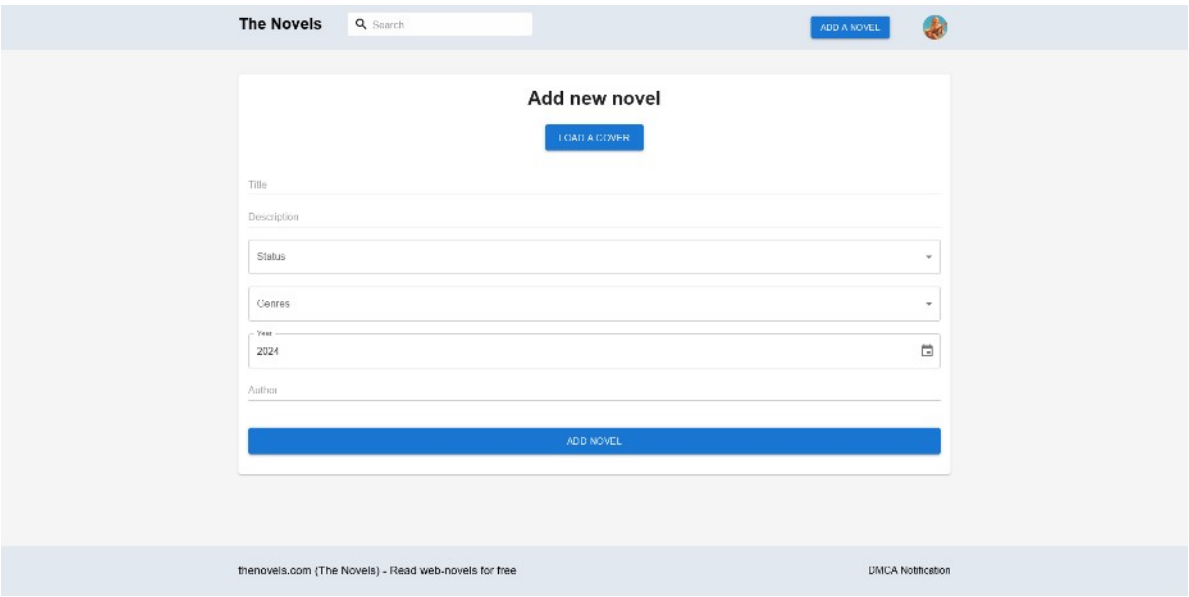


Рисунок 3.14 — Сторінка додавання нової вебновели

3.6.4 Сторінка перегляду глави вебновели

На даній сторінці (рис. 3.15 — 3.16) користувачі можуть побачити вміст глави вебновели, та коментарі до нього.

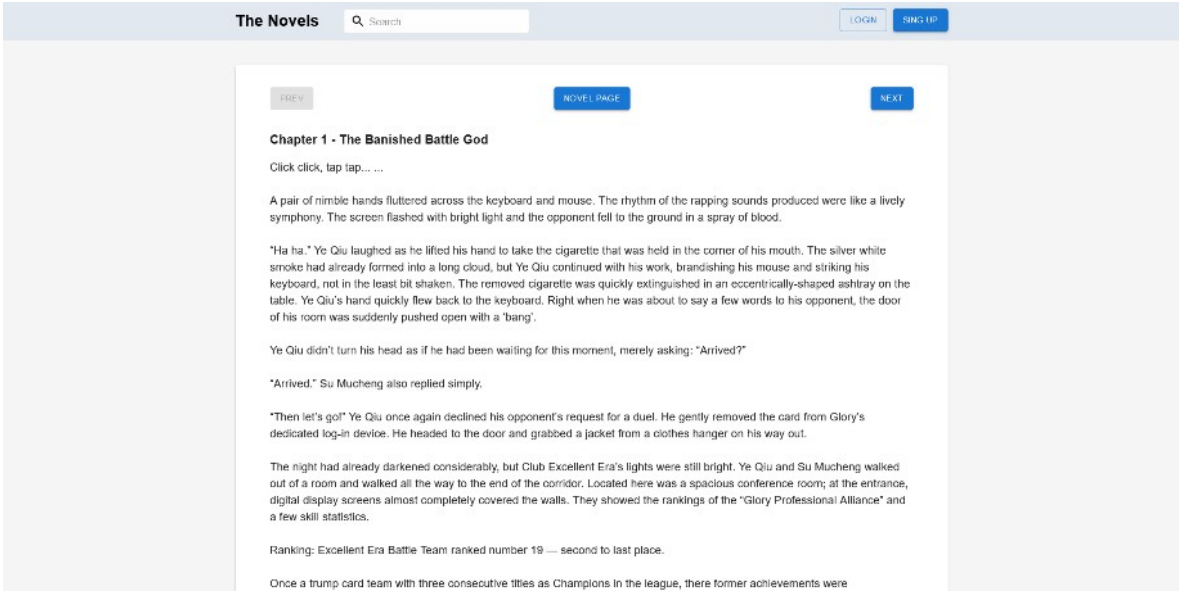


Рисунок 3.15 — Глава вебновели

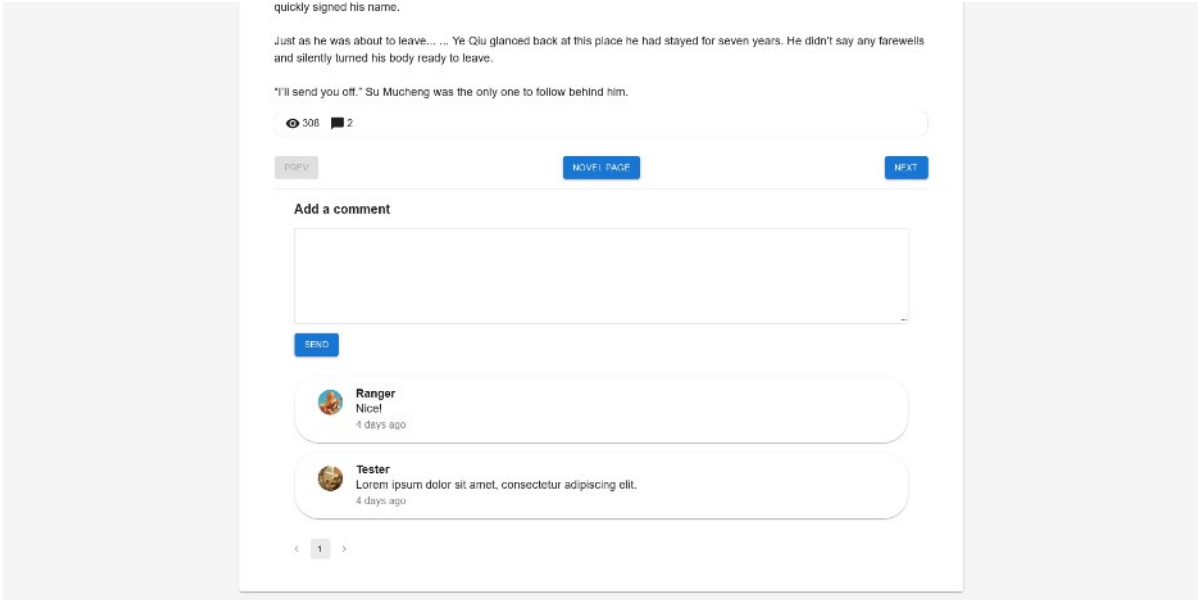


Рисунок 3.16 — Коментарі до глави

3.6.5 Сторінка результат пошуку

Дана сторінка (рис. 3.17 — 3.18) викликається у разі отримання запиту на пошук з пошукової стрічки або при натисненні на активні поля у інформації про вебновелу (рік видання, автор, жанри).

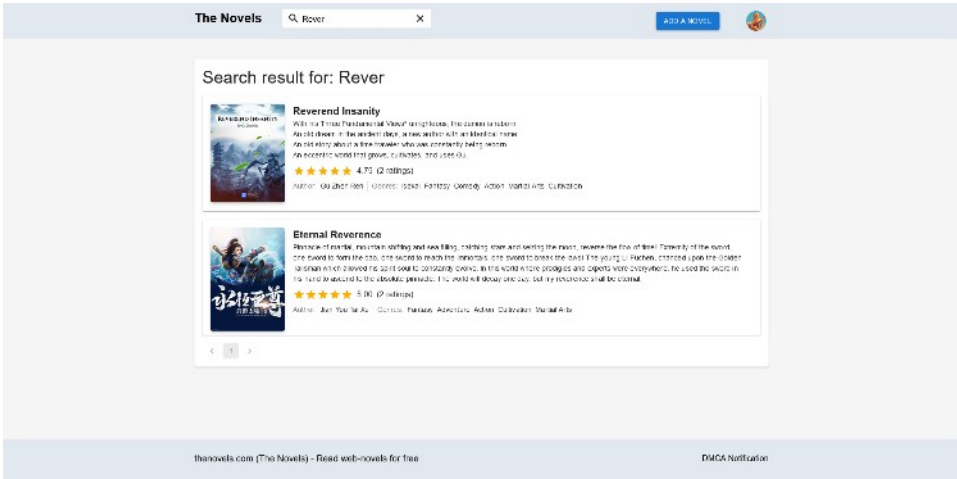


Рисунок 3.17 — Результат пошуку за часткою слова

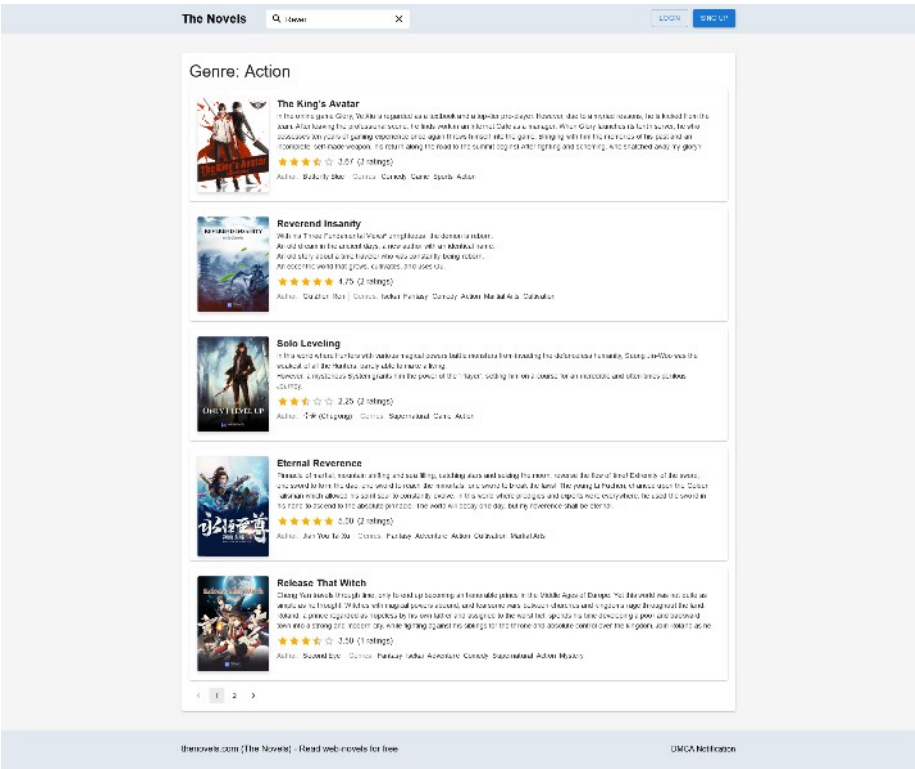
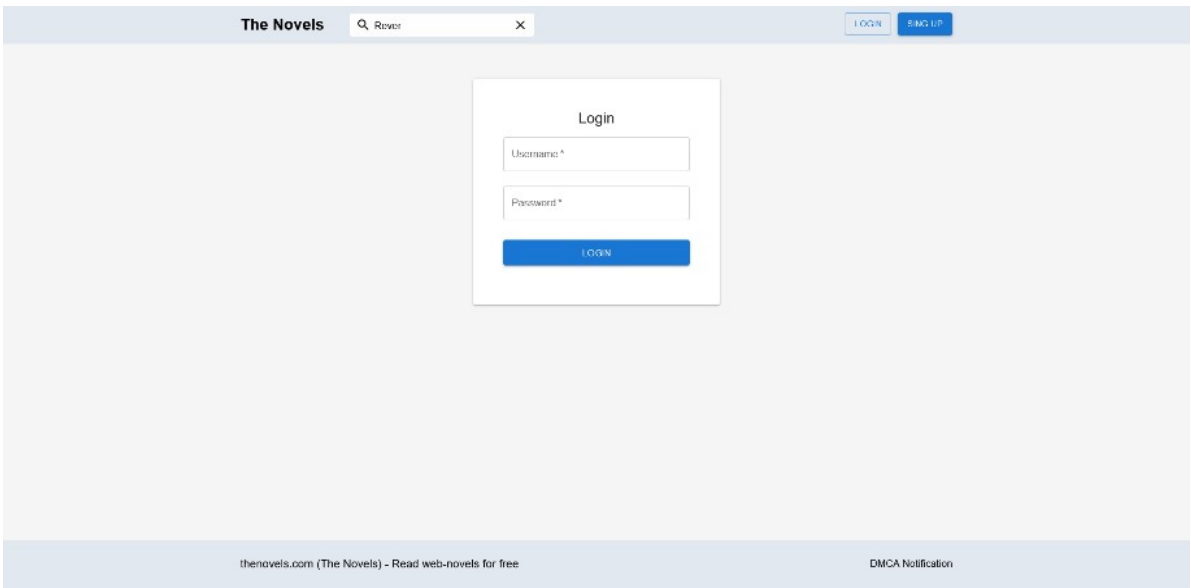


Рисунок 3.18 — Результат пошуку за активним полем з інформації про вебновелу

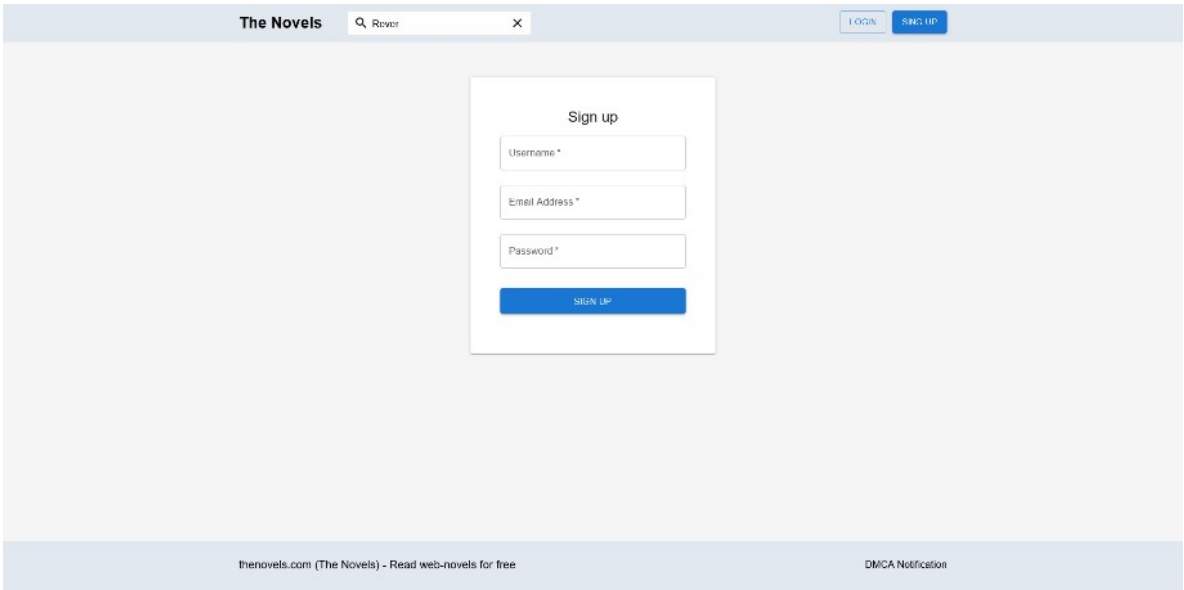
3.6.6 Сторінки авторизації та реєстрації

Сторінки авторизації та реєстрації (рис. 3.19 — 3.20), відповідають за авторизацію та реєстрацію користувача відповідно. Вони забезпечують процес аутентифікації користувача, відображення помилок при процесі авторизації або реєстрації користувача.



The screenshot shows the login page of 'The Novels' website. The header includes the site name 'The Novels', a search bar with 'Rover' entered, and 'LOGIN' and 'SIGN UP' buttons. The main content area features a 'Login' form with 'Username *' and 'Password *' input fields, and a blue 'LOGIN' button. The footer contains the text 'thenovels.com (The Novels) - Read web-novels for free' and a 'DMCA Notification' link.

Рисунок 3.19 — Сторінка авторизації користувача



The screenshot shows the registration page of 'The Novels' website. The header is identical to the login page. The main content area features a 'Sign up' form with 'Username *', 'Email Address *', and 'Password *' input fields, and a blue 'SIGN UP' button. The footer is also identical to the login page.

Рисунок 3.20 — Сторінка реєстрації користувача

3.6.7 Сторінка профілю користувача

Сторінка профілю користувача (рис. 3.21) відповідає за відображення інформації про користувача: ім'я користувача, аватарка, загальні дані про обліковий запис та коментарі самого користувача. Якщо користувач перейде на свою ж сторінку профілю, йому буде надана можливість зміни імені користувача та аварки через модальне вікно (рис. 3.22 — 3.23).

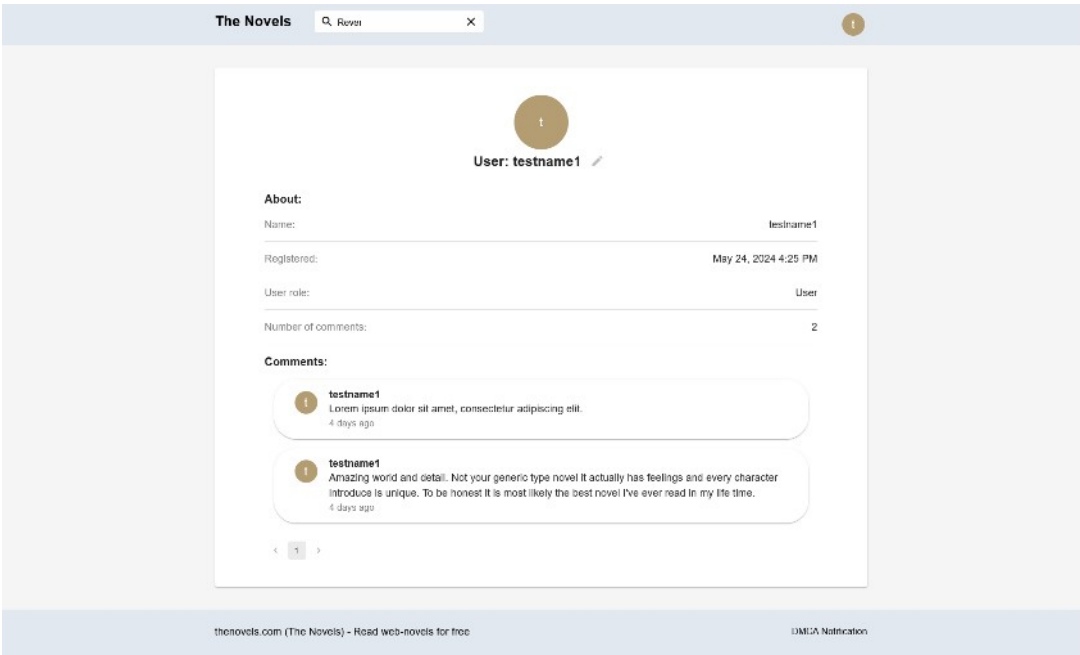


Рисунок 3.21 — Сторінка профілю користувача

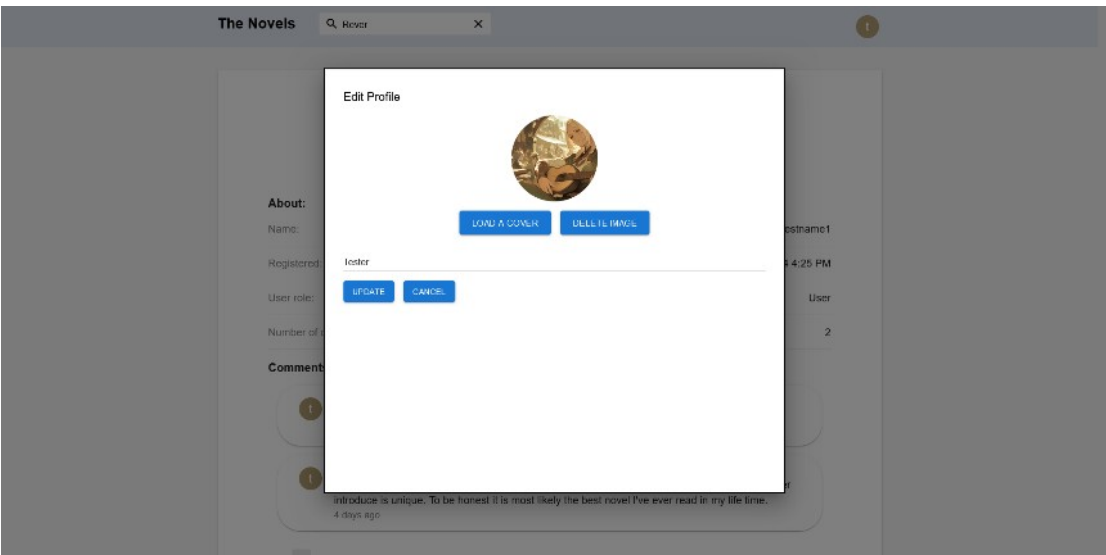


Рисунок 3.22 — Зміна імені та аватарки користувача

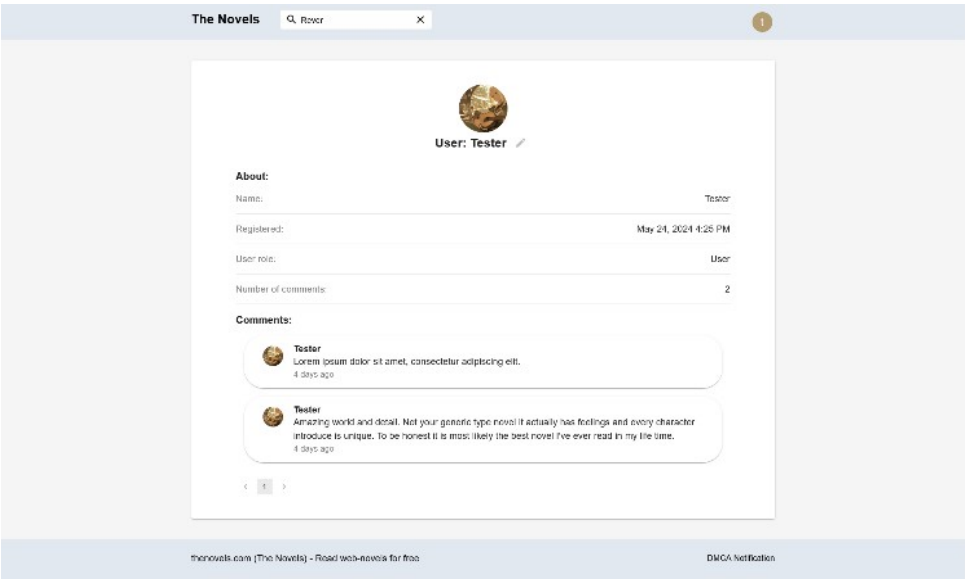


Рисунок 3.23 — Результат зміни імені та аватарки користувача

3.6.8 Сторінка списку збережених вебновел

Користувач також, маж можливість переглядати власний список збереженого (рис. 3.24). На даній сторінці відображаються збережені вебновели у вигляді плиток, що відображаю: обкладинку, назву, останню главу (якщо наявна), та оцінку вебновели. Натиснувши на аватарку або назву вебновели можна перейти на саму сторінку твору, а при натискані на главу буде виконане перекидання на сторінка відповідної глави вебновели.

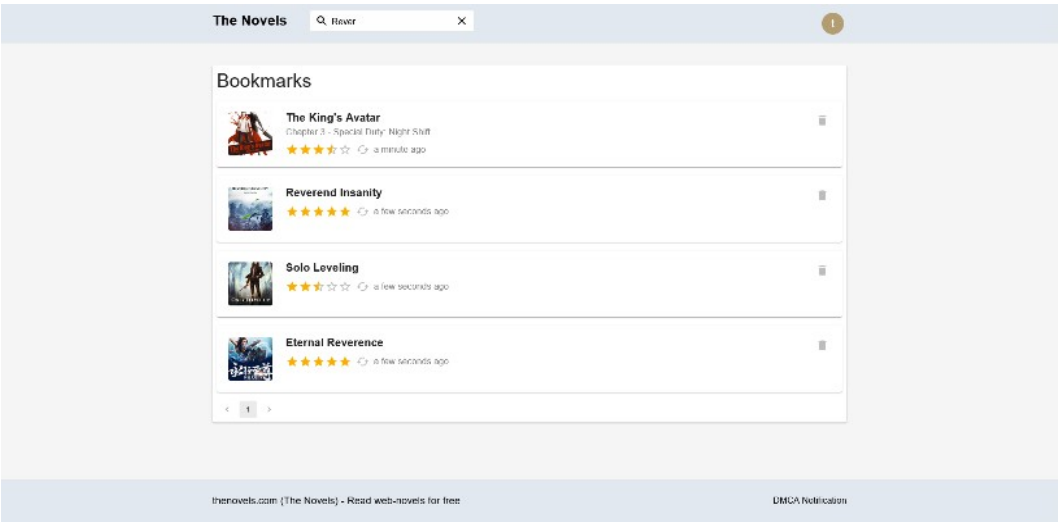


Рисунок 3.24 — Сторінка із збереженими вебновелами

3.7 Висновки за розділом

У даному розділі було описано основні рішення з розробки вебдодатку. Було описано структуру роботи та архітектура програми. Також, були описані файлові структури та залежності серверної та клієнтської частини додатку, та розглянуто реалізацію сторінок вебдодатку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Розроблений додаток використовується для зручного розповсюдження та перегляду вебновел. Основна мета програми полягає у наданні користувачам простого та ефективного інструменту для перегляду та завантаження вебновел.

Мінімальні системні вимоги та програмне забезпечення, що необхідні для функціонування серверної частини додатку:

- процесор Intel Xeon E5630 або аналогічний;
- 2 Gb оперативної пам'яті;
- HDD/SDD об'ємом від 128 Gb;
- вебсервер Apache або Nginx;
- програмне забезпечення: середовище NodeJS, менеджер пакетів yarn, база даних MongoDB.

Система має дворівневу архітектуру, що складається з клієнтського та серверного компонентів. Ці компоненти взаємодіють між собою, обмінюючись даними через API.

Користувацький інтерфейс відрізняється від аналогів зручністю, лаконічністю та простотою навігації. Серверна частина ж гарантує стабільну та безперебійну роботу системи.

4.2 Тестування програми

Тестування проводилось з метою перевірки якості та надійності роботи вебдодатку. Це допомагає виявити та виправити полики, які виникають під час роботи. Тестування дає змогу переконатися у тому, що додаток працює належним чином та допомагає забезпечити якість і надійність роботи системи.

Тестування розробленого вебдодатку включає наступні кроки.

Функціональне тестування: загальний термін для багатьох типів тестів, призначених для перевірки всієї функціональності програмного забезпечення. Функціональне тестування підтверджує, що програмне забезпечення працює належним чином і не містить помилок [28].

Тестування зручності сайту: полягає в тому, щоб визначити, чи будуть здатні користувачі, для яких призначений продукт, самостійно зрозуміти інтерфейс продукту і наскільки швидко у них це вийде [29].

4.2.1 Функціональне тестування

У процесі тестування функціональної частини вебдодатку, метою було переконатися в стабільній і коректній роботі ключових аспектів з позиції користувача. Нижче представлено чек-лист (табл. 4.1) з коротким оглядом тестування основних можливостей сайту.

Таблиця 4.1 — Чек-лист функціонального тестування

№	Тест	Результат
1	2	3
1	Аутентифікація	
a	Перевірити можливість авторизації користувача з використанням облікових даних	Passed
b	Перевірити можливість реєстрації користувача	Passed
2	Профіль користувача	
a	Перевірити наявність і коректність інформації користувача	Passed
b	Перевірити, що зміни профілю зберігаються	Passed

Продовження таблиці 4.1

1	2	3
3	Вебновела	
a	Перевірити наявність і коректність відображення інформації про вебновелу	Passed
b	Перевірити наявність і коректність відображення списку глав вебновели	Passed
4	Глави	
a	Перевірити можливість перелистування глав	Passed
b	Перевірити відображення вмісту глави	Passed
5	Коментарі	
a	Перевірити можливість залишати коментарі під вебновелою	Passed
b	Перевірити можливість залишати коментарі під главою	Passed
c	Перевірити, що коментарі коректно відображаються	Passed
6	Список збереженого	
a	Перевірити можливість додавання та видалення вебновел до списку збереженого	Passed
b	Перевірити наявність і коректність відображення списку збереженого	Passed

Кінець таблиці 4.1

1	2	3
7	Пошук	
a	Перевірити можливість пошуку за назвою	Passed
b	Перевірити можливість групування вебновел за роком видання, автором та жанром	Passed
8	Оцінювання	
a	Перевірити можливість залишити оцінку вебновелі	Passed
b	Перевірити коректність відображення оцінки вебновели	Passed

4.2.2 Тестування зручності сайту

Під час розгляду сайту, було помічено, що на сторінці наявні навігаційна панель та підвал сайту. У навігаційній панелі можна пошукову стрічку, та кнопки авторизації та реєстрації, які переносять вас на сторінки авторизації та реєстрації відповідно. Якщо ви вже авторизовані замість кнопок буде відображати аватарка вашого облікового запису, при натисканні на неї буде відображене випадаюче меню, у ньому наявні пункти «Profile» — перейти на сторінку профілю користувача, «Saved» — перехід на сторінку збережених вебновел, та «Logout» — вихід з облікового запису.

Весь сайт оформлений у приємних світлих кольорах, текст читається добре, що робить сайт досить приємним для ока.

Важливо відзначити, що завантаження сторінок відбувається досить швидко, але іноді можна побачити, як дані завантажуються, при їх завантаженні інтерфейс перекривається, що є досить неприємним досвідом для користувача.

Пошук по сайту демонструє високу ефективність, але він може шукати лише за повною або частковою назвою твору, можливе групування вебновел за автором, роком видання або жанром, але воно доступне лише з плиток новел або з самої сторінки вебновели.

При уважному розгляді, дизайн сайту має доволі стриманий та мінімалістичний вигляд. За межами навігаційної панелі, головного блоку та підвалу сайту, відсутні елементи, які могли б надати більше візуальної динаміки та різноманітності. Переосмислення дизайну може зробити платформу більш привабливою для користувачів.

Також, варто зазначити, що сайт, зовсім не адаптований для користувачів з обмеженими можливостями. Додаткові функціональності, такі як поліпшені варіанти навігації та альтернативні методи відтворення контенту, могли б створити більш доступний простір для всіх.

4.3 Висновки за розділом

У розділі було розглянуто призначення програми і визначені умови її використання. Програма призначена для розповсюдження та перегляду вебновел. Вона доступна через веббраузер з використанням Інтернет-з'єднання

Також було проведено функціональне та тестування зручності з метою перевірки якості та надійності роботи вебдодатку. Був складений чек-лист для функціонального тестування та розглянуто зручність використання сайту. Результати тестування дозволили виявити помилки та недоліки програми.

ВИСНОВКИ

У ході даної дипломної кваліфікаційної роботи було розроблено програмний продукт, який надає користувачам зручний інструмент для розповсюдження та читання вебновел. Проєкт був реалізований з використанням популярних фреймворків та технологій, таких як VS Code, JavaScript, NodeJS, Express, React, а також бази даних MongoDB.

У першому розділі було проведено аналіз предметної області, було визначено поняття вебновели та проаналізовано наявні рішення для їх розповсюдження та перегляду.

У другому розділі було обрано методи та технології розробки: JavaScript, як мову програмування, NodeJS та Express для реалізації серверної частини, React для клієнтської частини додатку, а також базу даних MongoDB для керування даними.

У третьому розділі було описано розробку програмного продукту та структуру додатку, зокрема архітектуру додатку, використані залежності та основні рішення при реалізації інтерфейсу користувача.

У четвертому розділі була описана експлуатації та тестування програми. Було визначено призначення й умови використання вебдодатку. Також, було проведено тестування програми з метою перевірки функціональності і відповідності вимогам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What's a Web Novel and Why Are They Exciting for Writers? [Electronic resource]. – Access mode: <https://medium.com/fiction-friends/whats-a-web-novel-and-why-should-you-be-excited-about-them-1181ae02be3b>
2. Why You Should Be Invested in Web Novels Instead of Book Novel [Electronic resource]. – Access mode: <https://digital-trans.asia/news-and-blogs/post/why-you-should-be-invested-web-novels-instead-book-novel>
3. Що таке JavaScript і для чого він потрібен [Електрон. ресурс] – Режим доступу: <https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/>
4. What is Javascript (JS)? [Electronic resource]. – Access mode: https://aws.amazon.com/what-is/javascript/?nc1=h_ls
5. What is a Framework in Programming and Engineering? [Electronic resource]. – Access mode: <https://aws.amazon.com/what-is/framework/>
6. MERN Stack Explained [Electronic resource]. – Access mode: <https://www.mongodb.com/resources/languages/mern-stack>
7. MongoDB що це, як з ним працювати [Електрон. ресурс] – Режим доступу: <https://wox.in.ua/threads/mongodb-scho-ce-jak-z-nim-pracjuvati.27/>
8. Що таке Node JS простими словами [Електрон. ресурс] – Режим доступу: <https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/>
9. Що таке MongoDB? вступ, Archітектура, функції та приклад [Електрон. ресурс] – Режим доступу: <https://www.guru99.com/uk/what-is-mongodb.html>
10. Основи MongoDB [Електрон. ресурс] – Режим доступу: <https://devzone.org.ua/post/osnovy-mongodb>
11. Understanding MVC Architecture in the MERN Stack [Electronic resource]. – Access mode: <https://medium.com/@ansari028amaan/understanding-mvc-architecture-in-the-mern-stack-5cc083828298>

12. Understanding the MVC Architecture in the MERN Stack [Electronic resource]. – Access mode: <https://shubhamjha25.medium.com/understanding-the-mvc-architecture-in-the-mern-stack-aff893abce50>
13. Leveraging MVC Architecture for Efficient Web Development [Electronic resource]. – Access mode: <https://www.blumandarin.com/blog/leveraging-mvc-architecture-for-efficient-web-development>
14. Introduction | Yarn [Electronic resource]. – Access mode: <https://yarnpkg.com/getting-started>
15. Yarn (package manager) [Електрон. ресурс] – Режим доступу: [https://en.wikipedia.org/wiki/Yarn_\(package_manager\)](https://en.wikipedia.org/wiki/Yarn_(package_manager))
16. Clodinary Frequently Asked Questions [Electronic resource]. – Access mode: <https://cloudinary.com/faq>
17. Express - Node.js web application framework [Electronic resource]. – Access mode: <https://expressjs.com/>
18. Mongoose ODM v8.4.1 [Electronic resource]. – Access mode: <https://mongoosejs.com/>
19. Material UI: React components that implement Material Design [Electronic resource]. – Access mode: <https://mui.com/material-ui/>
20. React Date Picker and Time Picker components - MUI X [Electronic resource]. – Access mode: <https://mui.com/x/react-date-pickers/>
21. @uiw/react-md-editor — npm [Electronic resource]. – Access mode: <https://www.npmjs.com/package/@uiw/react-md-editor>
22. dayjs — npm [Electronic resource]. – Access mode: <https://www.npmjs.com/package/dayjs>
23. React — Вікіпедія [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/React>
24. Documentation — react-hot-toast [Electronic resource]. – Access mode: <https://react-hot-toast.com/docs>

25. Feature Overview v6.23.1 | React Router [Electronic resource]. – Access mode: <https://reactrouter.com/en/main/start/overview>
26. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML [Electronic resource]. – Access mode: <https://tailwindcss.com/>
27. @mkyu/mui-search-bar — npm [Electronic resource]. – Access mode: <https://www.npmjs.com/package/@mkyu/mui-search-bar>
28. Різниця між функціональним і нефункціональним тестуванням [Електрон. ресурс]. – Режим доступу: <https://training.qatestlab.com/blog/technical-articles/difference-between-functional-and-non-functional-testing/>
29. Як і навіщо тестувати юзабіліті сайту [Електрон. ресурс]. – Режим доступу: <https://training.qatestlab.com/blog/technical-articles/how-and-why-usability-testing/>

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для підтримки онлайн платформи для розповсюдження та перегляду вебновел.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка онлайн платформи розповсюдження і перегляду вебновел», затверджене наказом Національного університету «Запорізька політехніка» № 168 від 24 квітня 2024 р.

A.2 Призначення розробки

Програмний продукт призначений для користувачів, що є читачами вебновел.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення вебсайту для розповсюдження і перегляду вебновел забезпечує виконання таких функцій:

- авторизацію та реєстрацію користувачів;
- можливість завантаження вебновел та глав до них;
- можливість перегляду вебновел та їх глав;
- можливість коментування вебновел та коментарів
- можливість оцінки вебновели;
- підтримка відображення інформації про вебновели та вмісту глав;
- можливість зміни інформації про вебновели та вмісту їх глав;

- можливість видалення вебновели;
- можливість зміни імені та аватарки для облікового запису.

A.3.2 Вимоги до інтерфейсу

Інтерфейс має бути мінімалістичним та інтуїтивно зрозумілим для користувачів. Необхідна зручна навігація та швидкий доступ до функцій вебсайту.

A.3.3 Вимоги до надійності

Вимоги до надійності вебдодатку включають стабільну та безперебійну роботу серверу та клієнту, що означає, що програма повинна функціонувати без критичних збоїв або непередбачуваних витоків пам'яті.

A.3.4 Умови експлуатації

Для експлуатації вебдодатку для розповсюдження і перегляду вебновел потрібен сучасний веббраузер, такий як Mozilla Firefox, Google Chrome, Microsoft Edge або Opera. Для доступу до вебсайту потрібне активне та безперебійне підключення до мережі Інтернет. Без підключення до мережі Інтернет користувач не зможе взаємодіяти з сервером, а отже й отримувати оновлення даних.

A.4 Вхідні дані

Вхідні дані до роботи програми включають інформацію про користувача, текстова інформація, у вигляді вмісту глав та основної

інформації про вебновели (назва, опис, автор, рік видання, жанри і т.д.), графічні дані (картинки для обкладинки вебновел та аватарки користувачів).

ДОДАТОК Б
Текст програми

B.1 Client

ChaptersFeed.jsx

```

/* eslint-disable react/prop-types */
import {
  Box,
  Card,
  CircularProgress,
  Typography,
  Backdrop,
  CardContent,
} from "@mui/material";
import dayjs from "dayjs";
import relativeTime from "dayjs/plugin/relativeTime";
import { useQuery } from "@tanstack/react-query";
import { Link } from "react-router-dom";

const ChaptersFeed = ({ novel }) => {
  dayjs.extend(relativeTime);
  const { data: chapters, isLoading } = useQuery({
    queryKey: ["chapters"],
    queryFn: async () => {
      try {
        const res = await fetch("/api/chapter/novel/" + novel._id);
        const data = await res.json();
        if (data.error) return null;
        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

  return (
    <Box>
      {isLoading && (
        <Backdrop
          sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
          open={true}
        >
          <CircularProgress color="inherit" />
        </Backdrop>
      )}
      {!isLoading && (
        <Box className="flex flex-wrap justify-between gap-4 my-4 px-
10">
          {chapters.map((chapter) => (
            <Box
              key={chapter._id}
              component={Link}
              to={"/chapter/" + chapter._id}
            >
              <Card sx={{ width: 475, height: 75 }}>
                <CardContent>
                  <Box className="flex gap-4">

```

```

        <Typography className="px-4" fontWeight={700}
fontSize={30}>
            {chapter.chapterNumber}
        </Typography>
        <Box>
            <Typography>{chapter.title}</Typography>
            <Typography color={"gray"}>
                {dayjs(chapter.createdAt).fromNow()}
            </Typography>
        </Box>
        </Box>
    </CardContent>
</Card>
</Box>
    )}
</Box>
    )}
</Box>
    );
};

export default ChaptersFeed;

```

Footer.jsx

```

import { Box, Container, Typography } from "@mui/material";

const Footer = () => {
    return (
        <Box className="flex bg-slate-200 border mt-auto h-20 w-full">
            <Container maxWidth="lg">
                <Box className="flex justify-between items-center pt-6">
                    <Typography fontSize={18}>
                        thenovels.com (The Novels) - Read web-novels for free
                    </Typography>
                    <a href="mailto:dmca@thenovels.com">
                        <Typography className="no-underline hover:underline
hover:text-cyan-600">
                            DMCA Notification
                        </Typography>
                    </a>
                </Box>
            </Container>
        </Box>
    );
};

export default Footer;

```

Navbar.jsx

```

import { NavLink, Link, useNavigate } from "react-router-dom";
import { useState } from "react";
import { useMutation, useQueryClient, useQuery } from
"@tanstack/react-query";
import { toast } from "react-hot-toast";
import {
    Container,
    Typography,
    Button,

```

```

    IconButton,
    Avatar,
    Menu,
    MenuItem,
    ListItemIcon,
    ListItemText,
    Divider,
    Box,
  } from "@mui/material";

import { Logout, Bookmark, AccountCircle } from "@mui/icons-material";

import SearchBar from "@mkyy/mui-search-bar";
import { stringToColor } from "../stringToColor";

const Navbar = () => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  const navigate = useNavigate();

  const [searchData, setSearchData] = useState("");

  const handleSearchRequest = () => {
    if (searchData.length !== 0) {
      navigate("/search/" + searchData);
    }
  };

  const [anchorEl, setAnchorEl] = useState(null);
  const open = Boolean(anchorEl);

  const handleMenu = (event) => {
    setAnchorEl(event.currentTarget);
  };
  const handleClose = () => {
    setAnchorEl(null);
  };

  const queryClient = useQueryClient();
  const { mutate: logout } = useMutation({
    mutationFn: async () => {
      try {
        const res = await fetch("/api/auth/logout", {
          method: "POST",
        });
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
      } catch (error) {
        throw new Error(error);
      }
    },
    onSuccess: () => {
      toast.success("Successfully logged out");
      queryClient.invalidateQueries({ queryKey: ["authUser"] });
      navigate("/");
    },
    onError: () => {
      toast.error("Logout failed");
    },
  });
};

```

```

return (
  <Box className="py-2.5 bg-slate-200 border">
    <Container maxWidth="lg">
      <Box className="flex justify-between flex-nowrap">
        <Box className="flex flex-none gap-10">
          <NavLink to="/">
            <Typography fontWeight={700} fontSize={25}>
              The Novels
            </Typography>
          </NavLink>
          <SearchBar
            className="border"
            value={searchData}
            onChange={(newValue) => setSearchData(newValue)}
            onSearch={handleSearchRequest}
          />
        </Box>

        <Box className="flex-none">
          {authUser ? (
            <Box>
              {authUser.isModerator && (
                <Button
                  variant="contained"
                  component={Link}
                  to="/add-novel"
                  sx={{ mr: 4 }}
                >
                  Add a novel
                </Button>
              )}
              <IconButton
                onClick={handleMenu}
                size="small"
                sx={{ ml: 2 }}
                aria-haspopup="true"
                aria-expanded={open ? "true" : undefined}
              >
                {authUser.profileImg ? (
                  <Avatar
                    sx={{ width: 40, height: 40 }}
                    src={authUser.profileImg}
                  />
                ) : (
                  <Avatar
                    sx={{
                      width: 40,
                      height: 40,
                      bgcolor: stringToColor(authUser.username),
                    }}
                  >
                    {authUser.username.charAt(0)}
                  </Avatar>
                )}
              </IconButton>
              <Menu
                id="account_menu"
                anchorEl={() => anchorEl}
                keepMounted
                open={open}
                onClose={handleClose}
                PaperProps={{
                  elevation: 0,

```

```

sx: {
  overflow: "visible",
  filter: "drop-shadow(0px 2px 8px
rgba(0,0,0,0.32))",
  mt: 1.5,
  "& .MuiAvatar-root": {
    width: 32,
    height: 32,
    ml: -0.5,
    mr: 1,
  },
  "&::before": {
    content: '""',
    display: "block",
    position: "absolute",
    top: 0,
    right: 14,
    width: 10,
    height: 10,
    bgcolor: "background.paper",
    transform: "translateY(-50%) rotate(45deg)",
    zIndex: 0,
  },
},
}}
transformOrigin={{ horizontal: "right", vertical:
"top" }}
anchorOrigin={{ horizontal: "right", vertical:
"bottom" }}
>
<MenuItem
  component={Link}
  to={"/profile/" + authUser._id}
  onClick={handleClose}
>
  <ListItemIcon>
    <AccountCircle sx={{ height: 30, width: 30 }} />
  </ListItemIcon>
  <ListItemText>Profile</ListItemText>
</MenuItem>
<Divider />
<MenuItem
  onClick={handleClose}
  component={Link}
  to={"/saved/" + authUser._id}
>
  <ListItemIcon>
    <Bookmark sx={{ height: 30, width: 30 }} />
  </ListItemIcon>
  <ListItemText>Saved</ListItemText>
</MenuItem>
<Divider />
<MenuItem
  onClick={(e) => {
    e.preventDefault();
    logout();
    handleClose();
  }}
>
  <ListItemIcon>
    <Logout sx={{ height: 30, width: 30 }} />
  </ListItemIcon>
  <ListItemText>Logout</ListItemText>

```

```

        </MenuItem>
      </Menu>
    </Box>
  ) : (
    <Box>
      <Link className="ml-2.5" to="/login">
        <Button variant="outlined">Login</Button>
      </Link>
      <Link className="ml-2.5" to="/register">
        <Button variant="contained">Sing up</Button>
      </Link>
    </Box>
  )}
</Box>
</Box>
</Container>
</Box>
);
};

export default Navbar;

```

NovelsFeed.jsx

```

import NovelTile from "../NovelTile";
import { Box } from "@mui/material";

// eslint-disable-next-line react/prop-types
const NovelsFeed = ({ novels }) => {
  return (
    <>
      {novels && (
        <Box>
          {novels.map((novel) => (
            <NovelTile key={novel._id} novel={novel} />
          ))}
        </Box>
      )}
    </>
  );
};

export default NovelsFeed;

```

NovelTile.jsx

```

/* eslint-disable react/prop-types */
import {
  Card,
  Box,
  CardMedia,
  CardContent,
  Typography,
  Rating,
  CardActionArea,
  Divider,
} from "@mui/material";
import { useQuery } from "@tanstack/react-query";
import { Link } from "react-router-dom";

```

```

const NovelTile = ({ novel }) => {
  const currentRating = novel.starRating
    ? novel.starRating.map((item) => item.rating).reduce((a, b) => a +
b, 0) /
    novel.starRating.length
    : 0;

  const getGenres = useQuery({
    queryKey: ["genres" + novel._id],
    queryFn: async () => {
      try {
        const res = await fetch("/api/novel/genres/" + novel._id);
        const data = await res.json();
        if (data.error) return null;
        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

  return (
    <Card className="flex flex-col p-4 my-4">
      <Box className="flex content-start">
        <CardActionArea
          component={Link}
          to={"/novel/" + novel._id}
          sx={{ height: 200, width: 150 }}
          className="flex-none"
        >
          <CardMedia
            component={"img"}
            className="border shadow-md rounded"
            image={novel.coverImg}
            alt="Novel Cover"
          />
        </CardActionArea>

        <CardContent className="flex flex-col" sx={{ pt: 0 }}>
          <Typography fontSize={20} fontWeight={700}>
            <Link to={"/novel/" + novel._id}>{novel.title}</Link>
          </Typography>
          <Typography
            fontSize={14}
            fontWeight={400}
            height={85}
            overflow={"hidden"}
            textOverflow={"ellipsis"}
          >
            {novel.description.split("\n").map((item, i) => {
              return (
                <span key={i}>
                  {item}
                  <br />
                </span>
              );
            })}
          </Typography>
          <Box className="mt-2 flex items-center gap-2">
            <Rating

```

```

        name="half-rating"
        value={currentRating}
        precision={0.5}
        readOnly
      />

<Typography>{parseFloat(currentRating).toFixed(2)}</Typography>
  <Typography>({novel.starRating.length}
ratings)</Typography>
</Box>
<Box className="mt-2 flex gap-2">
  <Box className="flex gap-2">
    <Typography fontSize={14} color={"gray"}>
      Author:{" "}
    </Typography>
    <Link to={"/author/" + novel.author}>
      <Typography
        fontSize={14}
        className="no-underline hover:underline hover:text-
cyan-600"
      >
        {novel.author}
      </Typography>
    </Link>
  </Box>
  {!getGenres.isLoading && getGenres.data && (
    <Box className="flex gap-2">
      <Divider orientation="vertical" />
      <Typography fontSize={14} color={"gray"}>
        Genres:
      </Typography>
      {getGenres.data.map((genre) => (
        <Link key={genre._id} to={"/genre/" + genre._id}>
          <Typography
            fontSize={14}
            className="no-underline hover:underline
hover:text-cyan-600"
          >
            {genre.name}
          </Typography>
        </Link>
      ))}
    </Box>
  )}
</Box>
</CardContent>
</Box>
</Card>
);
};

export default NovelTile;

```

ReviewsFeed.jsx

```

import { Box, Typography, Pagination, Button, Skeleton } from
"@mui/material";
import { useMutation, useQuery } from "@tanstack/react-query";
import toast from "react-hot-toast";
import ReviewTile from "../ReviewTile";
import { useEffect, useState } from "react";
import MDEditor from "@uiw/react-md-editor";

```

```

const ReviewsFeed = ({ chapter, novel }) => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  const [commentBody, setCommentBody] = useState("");
  const [page, setPage] = useState(1);
  const limit = 5;

  const {
    data: reviews,
    isLoading,
    refetch,
    isRefetching,
  } = useQuery({
    queryKey: ["reviews" + (chapter ? chapter._id : novel._id)],
    queryFn: async () => {
      try {
        const res = chapter
          ? await fetch(
              "/api/chapter/reviews/" +
              chapter._id +
              "?page=" +
              (page - 1) +
              "&limit=" +
              limit
            )
          : await fetch(
              "/api/novel/reviews/" +
              novel._id +
              "?page=" +
              (page - 1) +
              "&limit=" +
              limit
            );
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }

        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
    retry: 2,
  });

  const addNovelReview = useMutation({
    mutationFn: async () => {
      try {
        const userId = authUser._id;
        const res = await fetch("/api/novel/review/" + novel._id, {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({ userId: userId, text: commentBody }),
        });
        const data = await res.json();
        if (!res.ok) throw new Error(data.error || "Failed to create review");
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

```

```

    },
    onSuccess: () => {
      toast.success("Comment created successfully");
    },
    onError: (e) => {
      toast.error(e.message);
    },
  });
const addChapterReview = useMutation({
  mutationFn: async () => {
    try {
      const userId = authUser._id;
      const res = await fetch("/api/chapter/review/" + chapter._id,
{
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ userId: userId, text: commentBody }),
    });
      const data = await res.json();
      if (!res.ok) throw new Error(data.error || "Failed to create
review");
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    toast.success("Comment created successfully");
  },
  onError: (e) => {
    toast.error(e.message);
  },
});
useEffect(() => {
  refetch();
}, [page, refetch]);

const handlePageChange = (e, value) => {
  setPage(value);
};
const handleSend = (e) => {
  e.preventDefault();
  if (!authUser) {
    toast.error("To add a comment you must log in");
  } else {
    if (novel) {
      addNovelReview.mutate();
    } else if (chapter) {
      addChapterReview.mutate();
    }
  }
};
return (
  <Box className="p-4">
    <Box className="flex flex-col gap-4">
      <Typography className="px-4" fontSize={20} fontWeight={700}>
        Add a comment
      </Typography>
      <Box className="px-4">
        <MDEditor

```

```

        value={commentBody}
        onChange={setCommentBody}
        height={150}
        preview="edit"
        overflow="hidden"
        hideToolbar={true}
      />
    </Box>
    <Box className="mx-4 mb-4">
      <Button
        className="self-start"
        variant="contained"
        onClick={handleSend}
      >
        Send
      </Button>
    </Box>
  </Box>
  {(isLoading || isRefetching) && (
    <Skeleton variant="rectangular" height={170} />
  )}
  {!isLoading && reviews && (
    <Box>
      {!isRefetching && (
        <Box>
          {reviews.pageData.map((review) => (
            <ReviewTile
              key={review._id}
              id={review._id}
              review={review}
              linksDisabled={false}
            />
          ))}
          <Pagination
            count={Math.ceil(parseInt(reviews.totalCount) /
limit)}
            page={page}
            onChange={handlePageChange}
            shape="rounded"
          />
        </Box>
      )}
    </Box>
  )}
</Box>
);
};

export default ReviewsFeed;

```

ReviewTile.jsx

```

import dayjs from "dayjs";
import relativeTime from "dayjs/plugin/relativeTime";
import { useQuery } from "@tanstack/react-query";
import {
  Box,
  Typography,
  IconButton,
  Avatar,
  Paper,
  Grid,

```



```

        sx={{
          width: 40,
          height: 40,
          bgcolor: stringToColor(user.username),
        }}
      >
        {user.username.charAt(0)}
      </Avatar>
    )}
  </IconButton>
)}
</Grid>
<Grid justifyContent="left" item xs>
  {linksDisabled ? (
    <Typography variant="h4" fontSize={18}
fontWeight={700}>
      {user.username}
    </Typography>
  ) : (
    <Link to={"/profile/" + user._id}>
      <Typography variant="h4" fontSize={18}
fontWeight={700}>
        {user.username}
      </Typography>
    </Link>
  )}
  <Typography fontSize={18}>{review.body}</Typography>
  <Typography color={"gray"}>
    {dayjs(review.createdAt).fromNow()}
  </Typography>
</Grid>
</Grid>
</Paper>
)}
</Box>
);
};

export default ReviewTile;

```

SavedFeed.jsx

```

import SavedTile from "../SavedTile";
import { Box } from "@mui/material";

// eslint-disable-next-line react/prop-types
const SavedFeed = ({ novels }) => {
  return (
    <>
      {novels && (
        <Box>
          {novels.map((novel) => (
            <SavedTile key={novel._id} novel={novel} />
          ))}
        </Box>
      )}
    </>
  );
};

export default SavedFeed;

```

SavedTile.jsx

```

/* eslint-disable react/prop-types */
import {
  Card,
  Box,
  Typography,
  Rating,
  IconButton,
  Avatar,
  Grid,
} from "@mui/material";
import { useMutation, useQuery, useQueryClient } from
"@tanstack/react-query";
import { Link } from "react-router-dom";
import toast from "react-hot-toast";
import { Cached, Delete } from "@mui/icons-material";
import dayjs from "dayjs";
import relativeTime from "dayjs/plugin/relativeTime";

const SavedTile = ({ novel }) => {
  dayjs.extend(relativeTime);
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });

  const currentRating = novel.starRating
    ? novel.starRating.map((item) => item.rating).reduce((a, b) => a +
b, 0) /
    novel.starRating.length
    : 0;
  const chapters = novel.chapters ? novel.chapters : [];
  const { data: chapter, isLoading } = useQuery({
    queryKey: ["chapter" + (chapters.length !== 0 ? chapters.at(-1) :
"test")],
    queryFn: async () => {
      try {
        if (chapters.length === 0) {
          throw new Error("Chapters not found");
        }
        const res = await fetch("/api/chapter/get/" + chapters.at(-
1));
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }

        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
    retry: 1,
    onError: (error) => {
      toast.error(error.message);
    },
  });

  const queryClient = useQueryClient();
  const { mutate: unSaveNovel } = useMutation({
    mutationFn: async () => {
      try {

```

```

    if (!authUser) {
      throw new Error("User not found");
    }
    const res = await fetch("/api/novel/save/" + novel._id, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
    });
    const data = await res.json();
    if (!res.ok) {
      throw new Error(data.error || "Something went wrong");
    }

    return data;
  } catch (error) {
    throw new Error(error);
  }
},
onSuccess: () => {
  queryClient.invalidateQueries({ queryKey: ["novels" +
authUser._id] });
},
onError: (error) => {
  toast.error(error.message);
},
});

const handleUnSave = (e) => {
  e.preventDefault();
  unSaveNovel();
};

return (
  <Card className="flex p-4 my-4">
    <Grid container spacing={2}>
      <Grid item xs={1}>
        <IconButton
          size="small"
          disableRipple={true}
          component={Link}
          to={"/novel/" + novel._id}
        >
          <Avatar sx={{ width: 80, height: 80 }} variant="rounded">
            <img src={novel.coverImg} alt="Novel Cover" />
          </Avatar>
        </IconButton>
      </Grid>
      <Grid item xs={10}>
        <Box className="ml-4">
          <Typography fontSize={20} fontWeight={700}>
            <Link to={"/novel/" + novel._id}>{novel.title}</Link>
          </Typography>
          <{!isLoading && chapter && (
            <Link to={"/chapter/" + chapter._id}>
              <Typography color={"gray"}>
                Chapter {chapter.chapterNumber} - {chapter.title}
              </Typography>
            </Link>
          )}>
        </Box>
        <Box className="mt-2 flex items-center gap-2">
          <Rating value={currentRating} precision={0.5}
readOnly />

```

```

        <Box className="flex align-middle gap-2 flex-nowrap">
          <Cached color="disabled" />
          <Typography color={"gray"}>
            {dayjs(novel.updatedAt).fromNow()}
          </Typography>
        </Box>
      </Box>
    </Box>
  </Grid>
  <Grid item xs={1}>
    <Box className="flex justify-end items-center">
      <IconButton onClick={handleUnSave}>
        <Delete color="disabled" />
      </IconButton>
    </Box>
  </Grid>
</Grid>
</Card>
);
};

export default SavedTile;

```

AnonymousRoute.jsx

```

import { useQuery } from "@tanstack/react-query";
import { Outlet, Navigate } from "react-router-dom";

const AnonymousRoute = () => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  return authUser ? <Navigate to="/" replace /> : <Outlet />;
};

export default AnonymousRoute;

```

ModeratorRoute.jsx

```

import { useQuery } from "@tanstack/react-query";
import { Outlet, Navigate } from "react-router-dom";

const ModeratorRoute = () => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  return authUser.isModerator ? <Outlet /> : <Navigate to="/login"
  replace />;
};

export default ModeratorRoute;

```

PrivateRoute.jsx

```

import { useQuery } from "@tanstack/react-query";
import { Outlet, Navigate } from "react-router-dom";

const PrivateRoute = () => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  return authUser ? <Outlet /> : <Navigate to="/login" replace />;
};

```

```
export default PrivateRoute;
```

Login.jsx

```
import { useState } from "react";
import { useMutation, useQueryClient } from "@tanstack/react-query";
import { Box, Button, Paper, TextField, Typography } from
"@mui/material";
import { useNavigate } from "react-router-dom";
import toast from "react-hot-toast";

const Login = () => {
  const [formData, setFormData] = useState({
    username: "",
    password: "",
  });
  const queryClient = useQueryClient();
  const navigate = useNavigate();

  const { mutate: loginMutation, isPending } = useMutation({
    mutationFn: async ({ username, password }) => {
      try {
        const res = await fetch("/api/auth/login", {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({ username, password }),
        });

        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
      } catch (error) {
        throw new Error(error);
      }
    },
    onSuccess: () => {
      toast.success("Successfully logged in");
      queryClient.invalidateQueries({ queryKey: ["authUser"] });
      navigate("/");
    },
    onError: (e) => {
      toast.error(e.message);
    },
  });

  const handleSubmit = (e) => {
    e.preventDefault();
    loginMutation(formData);
  };

  const handleInputChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  return (
    <Paper className="w-[25rem] p-12 border my-14 mx-auto">
      <Typography className="text-bold text-center" variant="h5">
        Login
      </Typography>
    </Paper>
  );
};
```

```

    </Typography>
    <Box component="form" onSubmit={handleSubmit}>
      <TextField
        margin="normal"
        required
        fullWidth
        id="username"
        label="Username"
        name="username"
        autoFocus
        onChange={handleInputChange}
        value={formData.username}
      />
      <TextField
        margin="normal"
        required
        fullWidth
        name="password"
        label="Password"
        type="password"
        id="password"
        onChange={handleInputChange}
        value={formData.password}
      />
      <Button
        type="submit"
        size="large"
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
        fullWidth
      >
        {isPending ? "Loading..." : "Login"}
      </Button>
    </Box>
  </Paper>
);
};

export default Login;

```

Register.jsx

```

import { useState } from "react";
import { useMutation, useQueryClient } from "@tanstack/react-query";
import { Box, Button, Paper, TextField, Typography } from
"@mui/material";

import { toast } from "react-hot-toast";

const Register = () => {
  const [formData, setFormData] = useState({
    username: "",
    email: "",
    password: "",
  });

  const queryClient = useQueryClient();

  const { mutate, isError, isPending, error } = useMutation({
    mutationFn: async ({ username, email, password }) => {
      try {
        const res = await fetch("/api/auth/register", {

```

```

        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ username, email, password }),
    });
    const data = await res.json();
    if (!res.ok) throw new Error(data.error || "Failed to create
account");
    return data;
  } catch (error) {
    console.error(error);
    throw error;
  }
},
onSuccess: () => {
    toast.success("Account created successfully");
    queryClient.invalidateQueries({ queryKey: ["authUser"] });
},
onError: (e) => {
    toast.error(e.message);
},
});

const handleSubmit = (e) => {
    e.preventDefault();
    mutate(formData);
};

const handleInputChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
};

return (
    <Paper className="w-[25rem] p-12 border my-14 mx-auto">
        <Typography className="text-bold text-center" variant="h5">
            Sign up
        </Typography>
        <Box component="form" onSubmit={handleSubmit}>
            <TextField
                margin="normal"
                required
                fullWidth
                id="username"
                label="Username"
                name="username"
                autoComplete="email"
                autoFocus
                onChange={handleInputChange}
                value={formData.username}
            />
            <TextField
                margin="normal"
                required
                fullWidth
                id="email"
                label="Email Address"
                name="email"
                autoComplete="email"
                onChange={handleInputChange}
                value={formData.email}
            />
            <TextField

```

```

        margin="normal"
        required
        fullWidth
        name="password"
        label="Password"
        type="password"
        id="password"
        autoComplete="current-password"
        onChange={handleInputChange}
        value={formData.password}
      />
      {isError && toast.success(error.message)}

      <Button
        type="submit"
        size="large"
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
        fullWidth
      >
        {isPending ? "Loading..." : "Sign up"}
      </Button>
    </Box>
  </Paper>
);
};

export default Register;

```

Chapter.jsx

```

import { useEffect, useState } from "react";
import {
  Backdrop,
  Box,
  Button,
  CircularProgress,
  Divider,
  Modal,
  Paper,
  TextField,
  Typography,
} from "@mui/material";
import { useMutation, useQuery, useQueryClient } from
"@tanstack/react-query";
import { Link, Navigate, useParams } from "react-router-dom";
import toast from "react-hot-toast";
import ReviewsFeed from "../../components/ReviewsFeed";

import { ChatBubble, Visibility } from "@mui/icons-material";
import MDEditor from "@uiw/react-md-editor";

const Chapter = () => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  const { chapterId } = useParams();
  const [chapterBody, setChapterBody] = useState("");
  const [chapterTitle, setChapterTitle] = useState("");
  const [open, setOpen] = useState(false);

  const queryClient = useQueryClient();

  const {

```

```

    data: chapter,
    isLoading,
    isRefetching,
  } = useQuery({
    queryKey: ["chapter"],
    queryFn: async () => {
      try {
        const res = await fetch("/api/chapter/get/" + chapterId);
        const data = await res.json();
        if (data.error) return null;
        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

const { data: next_chapter } = useQuery({
  queryKey: ["next_chapter"],
  queryFn: async () => {
    try {
      const res = await fetch("/api/chapter/next/" + chapterId);
      const data = await res.json();
      if (data.error) return null;
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
});

const { data: prev_chapter } = useQuery({
  queryKey: ["prev_chapter"],
  queryFn: async () => {
    try {
      const res = await fetch("/api/chapter/prev/" + chapterId);
      const data = await res.json();
      if (data.error) return null;
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const { mutate: viewCountIncrease } = useMutation({
  mutationFn: async () => {
    try {
      const res = await fetch("/api/chapter/addViewCount/" +
chapterId, {
        method: "POST",

```

```

        headers: {
          "Content-Type": "application/json",
        },
      });
      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const { mutate: updateChapter, isPending } = useMutation({
  mutationFn: async () => {
    try {
      const res = await fetch("/api/chapter/update/", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ chapterId, chapterTitle,
chapterBody }),
      });
      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    toast.success("Chapter created successfully");
    handleClose();
    queryClient.refetchQueries();
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const onPageLoad = () => {
  viewCountIncrease();
};

const handleOpen = () => {
  if (!isLoading && chapter) {
    setChapterTitle(chapter.title);
    setChapterBody(chapter.body);
  }
  setOpen(true);
};

const handleClose = () => {
  setChapterTitle("");
  setChapterBody("");
  setOpen(false);
};

```

```

};

const handleChapterSave = (e) => {
  e.preventDefault();
  updateChapter();
};

useEffect(() => {
  onPageLoad();
}, []);

useEffect(() => {
  queryClient.refetchQueries();
  onPageLoad();
}, [chapterId]);

return (
  <Paper className="py-5 px-10 mt-10">
    {(isLoading || isRefetching) && (
      <Backdrop
        sx={{
          color: "#fff",
          zIndex: (theme) => theme.zIndex.drawer + 1,
        }}
        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {!isLoading && !chapter && <Navigate to={"/404"} />}
    {!isLoading && chapter && (
      <Box className="p-4 flex flex-col gap-4">
        <Box className="flex justify-between">
          <Button
            component={Link}
            variant="contained"
            disabled={!prev_chapter}
            to={prev_chapter && "/chapter/" + prev_chapter._id}
          >
            Prev
          </Button>
          <Button
            component={Link}
            variant="contained"
            to={"/novel/" + chapter.novel}
          >
            Novel Page
          </Button>
        <Box className="flex gap-2">
          {authUser && authUser.isModerator && (
            <Button onClick={handleOpen}>Edit</Button>
          )}
          <Button
            component={Link}
            variant="contained"
            disabled={!next_chapter}
            to={next_chapter && "/chapter/" + next_chapter._id}
          >
            Next
          </Button>
        </Box>
      </Box>
      <Typography variant="h6" fontWeight={700} className="pt-4">

```

```

Chapter {chapter.chapterNumber} - {chapter.title}
</Typography>
<Typography fontSize={18} sx={{ whiteSpace: "pre-wrap" }}>
  {chapter.body.split("\n").map((item, i) => {
    return (
      <span key={i}>
        {item}
        <br />
      </span>
    );
  })}
</Typography>
<Box className="border rounded-xl h-10 flex flex-nowrap
items-center px-4 gap-4">
  <Box className="flex gap-1">
    <Visibility />
    <Typography fontSize={16} fontWeight={400}>
      {chapter.viewsCount.toLocaleString()}
    </Typography>
  </Box>
  <Box className="flex gap-1">
    <ChatBubble />
    <Typography fontSize={16} fontWeight={400}>
      {chapter.reviews.length.toLocaleString()}
    </Typography>
  </Box>
</Box>
<Box className="flex justify-between pt-4">
  <Button
    component={Link}
    variant="contained"
    disabled={!prev_chapter}
    to={prev_chapter && "/chapter/" + prev_chapter._id}
  >
    Prev
  </Button>
  <Button
    component={Link}
    variant="contained"
    to={"/novel/" + chapter.novel}
  >
    Novel Page
  </Button>
  <Button
    component={Link}
    variant="contained"
    disabled={!next_chapter}
    to={next_chapter && "/chapter/" + next_chapter._id}
  >
    Next
  </Button>
</Box>
<Box>
  <Divider />
  <ReviewsFeed chapter={chapter} />
</Box>
<Modal
  open={open}
  onClose={handleClose}
  aria-labelledby="modal-modal-title"
  aria-describedby="modal-modal-description"
>
  <Box className="flex flex-col gap-4" sx={modalStyle}>

```

```

        <Typography variant="h6" component="h2">
          New chapter
        </Typography>
        <TextField
          variant="standard"
          placeholder="Title"
          value={chapterTitle}
          onChange={ (e) => setChapterTitle(e.target.value)}
          fullWidth
        />
        <MDEditor
          value={chapterBody}
          onChange={setChapterBody}
          height={400}
          preview="edit"
          overflow="hidden"
          hideToolbar={true}
        />
        <Box className="flex flex-nowrap gap-4">
          <Button
            variant="contained"
            size="medium"
            onClick={handleChapterSave}
          >
            {isPending ? "Loading..." : "Save"}
          </Button>
          <Button variant="contained" size="medium"
onClick={handleClose}>
            Cancel
          </Button>
        </Box>
      </Box>
    </Modal>
  </Box>
  )}
</Paper>
);
};

const modalStyle = {
  position: "absolute",
  top: "50%",
  left: "50%",
  transform: "translate(-50%, -50%)",
  width: 800,
  height: 740,
  bgcolor: "background.paper",
  border: "2px solid #000",
  boxShadow: 24,
  p: 4,
};

export default Chapter;

```

CreateNovel.jsx

```

import { useState, useRef } from "react";
import { useMutation, useQuery } from "@tanstack/react-query";
import { useNavigate } from "react-router-dom";
import {
  Button,
  MenuItem,

```

```

    Paper,
    Select,
    TextField,
    Box,
    InputLabel,
    FormControl,
    Backdrop,
    CircularProgress,
    Typography,
    Skeleton,
  } from "@mui/material";
import dayjs from "dayjs";
import { AdapterDayjs } from "@mui/x-date-pickers/AdapterDayjs";
import { LocalizationProvider } from
"@mui/x-date-pickers/LocalizationProvider";
import { DatePicker } from "@mui/x-date-pickers/DatePicker";
import toast from "react-hot-toast";

const CreateNovel = () => {
  const navigate = useNavigate();

  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [status, setStatus] = useState("");
  const [date, setDate] = useState(dayjs());
  const [author, setAuthor] = useState("");
  const [genres, setGenres] = useState([]);
  const [coverImg, setCoverImg] = useState(null);
  const imgRef = useRef(null);

  const { mutate: CreateNovel, isPending } = useMutation({
    mutationFn: async () => {
      try {
        const year = date.year();
        const res = await fetch("/api/novel/create", {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            title,
            description,
            status,
            year,
            author,
            genres,
            coverImg,
          }),
        });

        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong.");
        }
      } catch (error) {
        throw new Error(error);
      }
    },
    onSuccess: () => {
      navigate("/");
    },
    onError: (e) => {

```

```

        toast.error(e.message);
      },
    ));

const { data: a_genres, isLoading } = useQuery({
  queryKey: ["a_genres"],
  queryFn: async () => {
    try {
      const res = await fetch("/api/genre/all");
      const data = await res.json();

      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onError: (e) => {
    toast.error(e.message);
  },
});

const handleSubmit = (e) => {
  e.preventDefault();
  CreateNovel();
};

const handleImgChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    const reader = new FileReader();
    reader.onload = () => {
      setCoverImg(reader.result);
    };
    reader.readAsDataURL(file);
  }
};

const handleClearImage = () => {
  if (coverImg) {
    setCoverImg(null);
  }
};

return (
  <>
    <Paper className="p-4 mt-10">
      {(isLoading || !a_genres || a_genres.isLoading) && (
        <Backdrop
          sx={{ color: "#fff", zIndex: (theme) =>
theme.zIndex.drawer + 1 }}
          open={true}
        >
          <CircularProgress color="inherit" />
        </Backdrop>
      )}
      {(isLoading || a_genres.isLoading) && (
        <Skeleton variant="rectangular" height={500} />
      )}
      {!isLoading && !a_genres.isLoading && (
        <Box

```

```

        className="flex flex-col gap-5"
        component="form"
        onSubmit={handleSubmit}
    >
        <Typography className="self-center" fontSize={30}
fontWeight={700}>
            Add new novel
        </Typography>
        {coverImg && (
            <img
                className="self-center"
                src={coverImg}
                alt="Test"
                height={300}
                width={225}
            />
        )}
        <Box className="self-center flex gap-4">
            <Button variant="contained" size="large"
component="label">
                Load a cover
                <input
                    accept="image/png, image/jpeg"
                    type="file"
                    hidden
                    ref={imgRef}
                    onChange={handleImgChange}
                />
            </Button>
            {coverImg && (
                <Button
                    variant="contained"
                    size="large"
                    onClick={handleClearImage}
                >
                    Delete Image
                </Button>
            )}
        </Box>
        <br />
        <br />
        <TextField
            variant="standard"
            placeholder="Title"
            value={title}
            onChange={ (e) => setTitle(e.target.value) }
            fullWidth
        />
        <TextField
            multiline
            variant="standard"
            placeholder="Description"
            value={description}
            onChange={ (e) => setDescription(e.target.value) }
            fullWidth
        />
        <FormControl>
            <InputLabel>Status</InputLabel>
            <Select
                label="Status"
                value={status}
                onChange={ (e) => setStatus(e.target.value) }
            >

```

```

        <MenuItem value="inProgress">In progress</MenuItem>
        <MenuItem value="hiatus">Hiatus</MenuItem>
        <MenuItem value="ended">Ended</MenuItem>
      </Select>
    </FormControl>
    <FormControl>
      <InputLabel>Genres</InputLabel>
      <Select
        label="Genres"
        multiple
        value={genres}
        onChange={(e) => setGenres(e.target.value)}
      >
        {a_genres.map((genre) => (
          <MenuItem key={genre._id} value={genre._id}>
            {genre.name}
          </MenuItem>
        ))}
      </Select>
    </FormControl>
    <LocalizationProvider dateAdapter={AdapterDayjs}>
      <DatePicker
        label={"Year"}
        views={["year"]}
        value={date}
        onChange={setDate}
      />
    </LocalizationProvider>
    <TextField
      variant="standard"
      placeholder="Author"
      value={author}
      onChange={(e) => setAuthor(e.target.value)}
      fullWidth
    />
    <Button
      type="submit"
      size="large"
      variant="contained"
      sx={{ mt: 3, mb: 2 }}
      fullWidth
    >
      {isPending ? "Loading..." : "Add Novel"}
    </Button>
  </Box>
) }
</Paper>
</>
);
};

export default CreateNovel;

```

EditNovel.jsx

```

import { useState, useEffect, useRef } from "react";
import { useMutation, useQuery } from "@tanstack/react-query";
import { useNavigate, useParams } from "react-router-dom";
import {
  Button,
  MenuItem,
  Paper,

```

```

    Select,
    TextField,
    Box,
    InputLabel,
    FormControl,
    Backdrop,
    CircularProgress,
    Typography,
    Skeleton,
  } from "@mui/material";
import dayjs from "dayjs";
import customParseFormat from "dayjs/plugin/customParseFormat";
import { AdapterDayjs } from "@mui/x-date-pickers/AdapterDayjs";
import { LocalizationProvider } from
"@mui/x-date-pickers/LocalizationProvider";
import { DatePicker } from "@mui/x-date-pickers/DatePicker";
import toast from "react-hot-toast";

const EditNovel = () => {
  dayjs.extend(customParseFormat);
  const navigate = useNavigate();
  const { novelId } = useParams();

  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [status, setStatus] = useState("");
  const [date, setDate] = useState(dayjs());
  const [author, setAuthor] = useState("");
  const [genres, setGenres] = useState([]);
  const [coverImg, setCoverImg] = useState(null);
  const imgRef = useRef(null);

  const { data: novel, isLoading } = useQuery({
    queryKey: [novelId],
    queryFn: async () => {
      try {
        const res = await fetch("/api/novel/get/" + novelId);
        const data = await res.json();
        if (data.error) return null;
        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

  const { mutate: UpdateNovel, isPending } = useMutation({
    mutationFn: async () => {
      try {
        const year = date.year();
        const res = await fetch("/api/novel/update/" + novelId, {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            title,
            description,
            status,
            year,
          })
        );
      }
    }
  });

```

```

        author,
        genres,
        coverImg,
      )),
    });

    const data = await res.json();

    if (!res.ok) {
      throw new Error(data.error || "Something went wrong.");
    }
  } catch (error) {
    throw new Error(error);
  }
},
onSuccess: () => {
  navigate("/novel/" + novelId);
},
onError: (e) => {
  toast.error(e.message);
},
});

const a_genres = useQuery({
  queryKey: ["a_genres" + novelId],
  queryFn: async () => {
    try {
      const res = await fetch("/api/genre/all");
      const data = await res.json();

      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onError: (e) => {
    toast.error(e.message);
  },
});

const handleSubmit = (e) => {
  e.preventDefault();
  UpdateNovel();
};

useEffect(() => {
  if (!isLoading && !a_genres.isLoading) {
    setTitle(novel.title);
    setDescription(novel.description);
    setStatus(novel.status);
    setGenres(Array.from(novel.genres));
    setDate(dayjs(novel.year, "YYYY"));
    setAuthor(novel.author);
    setCoverImg(novel.coverImg);
  }
}, [isLoading, a_genres.isLoading]);

const handleImgChange = (e) => {
  const file = e.target.files[0];

```

```

    if (file) {
      const reader = new FileReader();
      reader.onload = () => {
        setCoverImg(reader.result);
      };
      reader.readAsDataURL(file);
    }
  };
  const handleClearImage = () => {
    if (coverImg) {
      setCoverImg(null);
    }
  };

  return (
    <>
      <Paper className="p-4 mt-10">
        {(isLoading || a_genres.isLoading) && (
          <Backdrop
            sx={{ color: "#fff", zIndex: (theme) =>
theme.zIndex.drawer + 1 }}
            open={true}
          >
            <CircularProgress color="inherit" />
          </Backdrop>
        )}
        {(isLoading || a_genres.isLoading) && (
          <Skeleton variant="rectangular" height={500} />
        )}
        {!isLoading && a_genres && !a_genres.isLoading && (
          <Box
            className="flex flex-col gap-5"
            component="form"
            onSubmit={handleSubmit}
          >
            <Typography className="self-center" fontSize={30}
fontWeight={700}>
              Edit novel: {novel.title}
            </Typography>
            {coverImg && (
              <img
                className="self-center"
                src={coverImg}
                alt="Test"
                height={300}
                width={225}
              />
            )}
            <Box className="self-center flex gap-4">
              <Button variant="contained" size="large"
component="label">
                Load a cover
                <input
                  accept="image/png, image/jpeg"
                  type="file"
                  hidden
                  ref={imgRef}
                  onChange={handleImgChange}
                />
              </Button>
              {coverImg && (
                <Button
                  variant="contained"

```

```

        size="large"
        onClick={handleClearImage}
      >
        Delete Image
      </Button>
    )}
  </Box>
<br />
<br />
<TextField
  variant="standard"
  placeholder="Title"
  value={title}
  onChange={(e) => setTitle(e.target.value)}
  fullWidth
/>
<TextField
  multiline
  variant="standard"
  placeholder="Description"
  value={description}
  onChange={(e) => setDescription(e.target.value)}
  fullWidth
/>
<FormControl>
  <InputLabel>Status</InputLabel>
  <Select
    label="Status"
    value={status}
    onChange={(e) => setStatus(e.target.value)}
  >
    <MenuItem value="inProgress">In progress</MenuItem>
    <MenuItem value="hiatus">Hiatus</MenuItem>
    <MenuItem value="ended">Ended</MenuItem>
  </Select>
</FormControl>
<FormControl>
  <InputLabel>Genres</InputLabel>
  <Select
    label="Genres"
    multiple
    value={genres}
    onChange={(e) => setGenres(e.target.value)}
  >
    {a_genres.data.map((genre) => (
      <MenuItem key={genre._id} value={genre._id}>
        {genre.name}
      </MenuItem>
    ))}
  </Select>
</FormControl>
<LocalizationProvider dateAdapter={AdapterDayjs}>
  <DatePicker
    label={"Year"}
    views={["year"]}
    value={date}
    onChange={setDate}
  />
</LocalizationProvider>
<TextField
  variant="standard"
  placeholder="Author"
  value={author}

```

```

        onChange={ (e) => setAuthor(e.target.value) }
        fullWidth
      />
      <Button
        type="submit"
        size="large"
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
        fullWidth
      >
        {isPending ? "Loading..." : "Update"}
      </Button>
    </Box>
  )}
</Paper>
</>
);
};

export default EditNovel;

```

Novel.jsx

```

import { useEffect, useState } from "react";
import {
  Backdrop,
  Box,
  Button,
  CardMedia,
  CircularProgress,
  Divider,
  Grid,
  IconButton,
  Modal,
  Paper,
  Rating,
  Skeleton,
  TextField,
  Typography,
} from "@mui/material";
import { Delete, Mode, Visibility } from "@mui/icons-material";
import { useMutation, useQuery } from "@tanstack/react-query";
import MDEditor from "@uiw/react-md-editor";
import { Link, Navigate, useNavigate, useParams } from "react-router-dom";
import toast from "react-hot-toast";

import ReviewsFeed from "../../components/ReviewsFeed";
import ChaptersFeed from "../../components/ChaptersFeed";

const Novel = () => {
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });
  const { novelId } = useParams();

  const [chapterBody, setChapterBody] = useState("");
  const [chapterTitle, setChapterTitle] = useState("");
  const [rating, setRating] = useState(0.0);
  const [open, setOpen] = useState(false);

  const getGenres = useQuery({
    queryKey: ["genres" + novelId],
    queryFn: async () => {

```

```

    try {
      const res = await fetch("/api/novel/genres/" + novelId);
      const data = await res.json();
      if (data.error) return null;
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
});

const {
  data: novel,
  isLoading,
  refetch,
} = useQuery({
  queryKey: [novelId],
  queryFn: async () => {
    try {
      const res = await fetch("/api/novel/get/" + novelId);
      const data = await res.json();
      if (data.error) return null;
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
});

const { mutate: ratingNovel, isPending } = useMutation({
  mutationFn: async () => {
    try {
      const res = await fetch(`/api/novel/rating/${novel._id}`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          rating: rating,
        }),
      });

      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    refetch();
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

```

```

    },
  });

const { mutate: viewCountIncrease } = useMutation({
  mutationFn: async () => {
    try {
      const res = await fetch("/api/novel/addViewCount/" + novelId,
{
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
    });
      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    refetch();
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const addChapter = useMutation({
  mutationFn: async () => {
    try {
      const res = await fetch("/api/chapter/add/", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ novelId, chapterTitle,
chapterBody }),
      });
      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }
      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    toast.success("Chapter created successfully");
    handleClose();
    refetch();
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const { mutate: saveNovel } = useMutation({
  mutationFn: async () => {

```

```

    try {
      if (!authUser) {
        throw new Error("User not found");
      }
      const res = await fetch("/api/novel/save/" + novelId, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
      });
      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    refetch();
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const navigate = useNavigate();

const { mutate: deleteNovel, isPending: isDeleting } = useMutation({
  mutationFn: async () => {
    try {
      if (!authUser && !authUser.isModerator) {
        throw new Error("Permission denied");
      }
      const res = await fetch("/api/novel/" + novelId, {
        method: "Delete",
      });
      const data = await res.json();
      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    toast.apply("Novel deleted successfully");
    navigate("/");
  },
  onError: (error) => {
    toast.error(error.message);
  },
});

const handleRatingSet = (e) => {
  e.preventDefault();
  setRating(e.target.value);
  ratingNovel();
};

```

```

const handleChapterSave = (e) => {
  e.preventDefault();
  addChapter.mutate();
};

const handleNovelSave = (e) => {
  e.preventDefault();
  saveNovel();
};

const handleOpen = () => setOpen(true);
const handleClose = () => {
  setChapterTitle("");
  setChapterBody("");
  setOpen(false);
};

const handleDelete = () => {
  deleteNovel();
};
const onPageLoad = () => {
  viewCountIncrease();
};

useEffect(() => {
  onPageLoad();
}, []);

useEffect(() => {
  if (novel) {
    const currentRating = novel.starRating
      ? novel.starRating
        .map((item) => item.rating)
        .reduce((a, b) => a + b, 0) / novel.starRating.length
      : 0;
    setRating(currentRating);
  }
}, [novel]);

return (
  <Paper className="py-5 px-10 mt-10">
    {(isLoading || isDeleting) && (
      <Backdrop
        sx={{
          color: "#fff",
          zIndex: (theme) => theme.zIndex.drawer + 1,
        }}
        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {isLoading && <Skeleton variant="rectangular" height={1000} />}
    {!isLoading && !novel && <Navigate to={"/404"} />}
    {!isLoading && novel && (
      <Grid container direction={"row"} spacing={2}>
        <Grid item xs={4} justifySelf={"center"}>
          <CardMedia
            className="border shadow-md rounded"
            component="img"
            height={200}
            width={150}

```

```

        sx={{ height: 400, width: 300 }}
        image={novel.coverImg}
        alt="Novel Cover"
      />
    </Grid>
    <Grid
      item
      xs={8}
      container
      direction={"column"}
      justifyContent={"space-between"}
    >
      <Grid item>
        <Typography fontSize={36} fontWeight={700}>
          {novel.title}
        </Typography>
        <Box className="flex align-middle gap-4">
          <Box className="flex align-middle gap-1 flex-nowrap">
            <Mode />
            <Typography fontSize={16}>
              {getStatus(novel.status)}
            </Typography>
          </Box>
          <Box className="flex align-middle gap-2 flex-nowrap">
            <Visibility />
            <Typography fontSize={16}>
              {novel.viewsCount.toLocaleString()}
            </Typography>
          </Box>
        </Box>
        <Box className="flex gap-2 my-2">
          <Typography color={"gray"}>Author: </Typography>
          <Link to={"/author/" + novel.author}>
            <Typography className="no-underline hover:underline
hover:text-cyan-600">
              {novel.author}
            </Typography>
          </Link>
        </Box>
        <Box className="flex gap-2 my-2">
          <Typography color={"gray"}>Year: </Typography>
          <Link to={"/year/" + novel.year}>
            <Typography className="no-underline hover:underline
hover:text-cyan-600">
              {novel.year}
            </Typography>
          </Link>
        </Box>
        {!getGenres.isLoading && getGenres.data && (
          <Box className="flex gap-2 my-2">
            <Typography color={"gray"}>Genres:</Typography>
            {getGenres.data.map((genre) => (
              <Link key={genre._id} to={"/genre/" + genre._id}>
                <Typography className="no-underline
hover:underline hover:text-cyan-600">
                  {genre.name}
                </Typography>
              </Link>
            ))}
          </Box>
        )}
      </Grid>
    </Box>
    {(isPending || isLoading) && (

```

```

        <Box className="flex items-center gap-2">
          <Rating
            value={parseFloat(rating)}
            precision={0.5}
            readOnly
          />
          <Typography fontSize={20}>
            {parseFloat(rating).toFixed(2)}
          </Typography>
          <Typography>({novel.starRating.length}
ratings)</Typography>
        </Box>
      )}
      {!isPending && authUser && (
        <Box className="flex items-center gap-2">
          <Rating
            value={parseFloat(rating)}
            onChange={ (e) => handleRatingSet(e) }
            precision={0.5}
          />
          <Typography fontSize={20}>
            {parseFloat(rating).toFixed(2)}
          </Typography>
          <Typography>({novel.starRating.length}
ratings)</Typography>
        </Box>
      )}
      {!isPending && !authUser && (
        <Box className="flex items-center gap-2">
          <Rating
            value={parseFloat(rating)}
            onChange={ (e) => handleRatingSet(e) }
            precision={0.5}
            readOnly
          />
          <Typography fontSize={20}>
            {parseFloat(rating).toFixed(2)}
          </Typography>
          <Typography>({novel.starRating.length}
ratings)</Typography>
        </Box>
      )}
    </Box>
  </Grid>
  <Grid item justifySelf={"end"} mt={10}>
    <Box className="flex flex-nowrap gap-4">
      <Button
        variant="contained"
        size="large"
        component={Link}
        to={
          novel.chapters.length > 0
            ? "/chapter/" + novel.chapters[0]
            : ""
        }
      >
        Read
      </Button>
      {authUser &&
        !Array.from(novel.savedBy).includes(authUser._id) &&
        (
          <Button
            variant="contained"

```

```

        size="large"
        onClick={handleNovelSave}
      >
        Add to library
      </Button>
    )}
    {authUser &&
      Array.from(novel.savedBy).includes(authUser._id) &&
      (
        <Button
          variant="contained"
          size="large"
          onClick={handleNovelSave}
          color="error"
        >
          Remove from library
        </Button>
      )}
    {authUser && authUser.isModerator && (
      <Box className="flex gap-2">
        <Button
          variant="contained"
          size="large"
          component={Link}
          to={"/update/" + novel._id}
        >
          Edit novel
        </Button>
        <Button
          variant="contained"
          size="large"
          onClick={handleOpen}
        >
          Add chapter
        </Button>
        <IconButton onClick={handleDelete}>
          <Delete color="disabled" />
        </IconButton>
      </Box>
    )}
  </Box>
</Grid>
</Grid>
<Grid item xs={12}>
  <Box>
    <Divider />
    <Typography fontSize={30} fontWeight={700}>
      Description
    </Typography>
    <Typography fontSize={18} whiteSpace={"pre-wrap"}>
      {novel.description.split("\n").map((item, i) => {
        return (
          <span key={i}>
            {item}
            <br />
          </span>
        );
      })}
    </Typography>
  </Box>
</Grid>
{novel.chapters.length > 0 && (
  <Grid item xs={12}>

```

```

        <Divider />
        <Box className="flex justify-center">
            <ChaptersFeed novel={novel} />
        </Box>
    </Grid>
  )}
  <Grid item xs={12}>
    <Divider />
    <ReviewsFeed novel={novel} />
  </Grid>
  <Modal
    open={open}
    onClose={handleClose}
    aria-labelledby="modal-modal-title"
    aria-describedby="modal-modal-description"
  >
    <Box className="flex flex-col gap-4" sx={modalStyle}>
      <Typography variant="h6" component="h2">
        New chapter
      </Typography>
      <TextField
        variant="standard"
        placeholder="Title"
        value={chapterTitle}
        onChange={(e) => setChapterTitle(e.target.value)}
        fullWidth
      />
      <MDEditor
        value={chapterBody}
        onChange={setChapterBody}
        height={400}
        preview="edit"
        overflow="hidden"
        hideToolbar={true}
      />
      <Box className="flex flex-nowrap gap-4">
        <Button
          variant="contained"
          size="medium"
          onClick={handleChapterSave}
        >
          {addChapter.isPending ? "Loading..." : "Save"}
        </Button>
        <Button variant="contained" size="medium"
onClick={handleClose}>
          Cancel
        </Button>
      </Box>
    </Box>
  </Modal>
</Grid>
  )}
</Paper>
);
};

function getStatus(status) {
  switch (status) {
    case "inProgress":
      return "In Progress";
    case "ended":
      return "Complete";
    case "hiatus":

```

```

        return "Hiatus";
      default:
        return "unknown";
    }
  }

const modalStyle = {
  position: "absolute",
  top: "50%",
  left: "50%",
  transform: "translate(-50%, -50%)",
  width: 800,
  height: 740,
  bgcolor: "background.paper",
  border: "2px solid #000",
  boxShadow: 24,
  p: 4,
};

export default Novel;

```

Author.jsx

```

import { useQuery } from "@tanstack/react-query";
import NovelsFeed from "../../components/NovelsFeed";
import {
  Box,
  Skeleton,
  Backdrop,
  CircularProgress,
  Pagination,
  Paper,
  Typography,
} from "@mui/material";
import toast from "react-hot-toast";
import { useEffect, useState } from "react";
import { Navigate, useParams } from "react-router-dom";

const Author = () => {
  const { author } = useParams();
  const [page, setPage] = useState(1);
  const limit = 5;

  const {
    data: novels,
    isLoading,
    refetch,
    isRefetching,
  } = useQuery({
    queryKey: ["novels"],
    queryFn: async () => {
      try {
        const res = await fetch(
          "/api/novel/author/" +
            author +
            "?page=" +
            (page - 1) +
            "&limit=" +
            limit
        );
        const data = await res.json();

```

```

    if (!res.ok) {
      throw new Error(data.error || "Something went wrong");
    }

    return data;
  } catch (error) {
    throw new Error(error);
  }
},
onError: (error) => {
  toast.error(error.message);
},
});

useEffect(() => {
  refetch();
}, [page, refetch]);

const handlePageChange = (e, value) => {
  setPage(value);
};

return (
  <Paper className="flex flex-col flex-nowrap gap-4 p-4 mt-10">
    {(isLoading || isRefetching) && (
      <Backdrop
        sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {(isLoading || isRefetching) && (
      <Box>
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
      </Box>
    )}
    {!isLoading && !novels && <Navigate to={"/404"} />}
    {!isLoading && !isRefetching && novels && (
      <Box>
        <Typography variant="h4">Author: {author}</Typography>
        <NovelsFeed novels={novels.pageData} />
        <Pagination
          count={Math.ceil(parseInt(novels.totalCount) / limit)}
          page={page}
          onChange={handlePageChange}
          shape="rounded"
        />
      </Box>
    )}
  </Paper>
);
};

export default Author;

```

Genre.jsx

```
import { useQuery } from "@tanstack/react-query";
```

```

import NovelsFeed from "../../components/NovelsFeed";
import {
  Box,
  Skeleton,
  Backdrop,
  CircularProgress,
  Pagination,
  Paper,
  Typography,
} from "@mui/material";
import toast from "react-hot-toast";
import { useEffect, useState } from "react";
import { Navigate, useParams } from "react-router-dom";

const Genre = () => {
  const { genreId } = useParams();
  const [page, setPage] = useState(1);
  const limit = 5;

  const genre = useQuery({
    queryKey: ["genre" + genreId],
    queryFn: async () => {
      try {
        const res = await fetch("/api/genre/" + genreId);
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }

        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
    onError: (error) => {
      toast.error(error.message);
    },
  });

  const {
    data: novels,
    isLoading,
    refetch,
    isRefetching,
  } = useQuery({
    queryKey: ["novels"],
    queryFn: async () => {
      try {
        const res = await fetch(
          "/api/genre/novels/" +
            genreId +
            "?page=" +
            (page - 1) +
            "&limit=" +
            limit
        );
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
      }
    },
  });

```

```

        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
    onError: (error) => {
      toast.error(error.message);
    },
  });

  useEffect(() => {
    refetch();
  }, [page, refetch]);

  const handlePageChange = (e, value) => {
    setPage(value);
  };

  return (
    <Paper className="flex flex-col flex-nowrap gap-4 p-4 mt-10">
      {(isLoading || isRefetching || genre.isLoading) && (
        <Backdrop
          sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
          open={true}
        >
          <CircularProgress color="inherit" />
        </Backdrop>
      )}
      {(isLoading || isRefetching || genre.isLoading) && (
        <Box>
          <Skeleton variant="rectangular" height={232} />
          <Skeleton variant="rectangular" height={232} />
          <Skeleton variant="rectangular" height={232} />
        </Box>
      )}
      {!isLoading && !novels && <Navigate to={"/404"} />}
      {!isLoading && !isRefetching && novels && !genre.isLoading && (
        <Box>
          <Typography variant="h4">Genre:
{genre.data.name}</Typography>
          <NovelsFeed novels={novels.pageData} />
          <Pagination
            count={Math.ceil(parseInt(novels.totalCount) / limit)}
            page={page}
            onChange={handlePageChange}
            shape="rounded"
          />
        </Box>
      )}
    </Paper>
  );
};

export default Genre;

```

SearchResult.jsx

```

import { useQuery } from "@tanstack/react-query";
import NovelsFeed from "../../components/NovelsFeed";
import {
  Box,

```

```

    Skeleton,
    Backdrop,
    CircularProgress,
    Typography,
    Paper,
    Pagination,
  } from "@mui/material";
import toast from "react-hot-toast";
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";

const SearchResult = () => {
  const { searchQuery } = useParams();
  const [page, setPage] = useState(1);
  const limit = 5;

  const {
    data: novels,
    isLoading,
    refetch,
    isRefetching,
  } = useQuery({
    queryKey: ["searchedNovels" + searchQuery],
    queryFn: async () => {
      try {
        const res = await fetch(
          "/api/novel/search/" +
            searchQuery +
            "?page=" +
            (page - 1) +
            "&limit=" +
            limit
        );
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }

        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
    onError: (error) => {
      toast.error(error.message);
    },
  });

  useEffect(() => {
    refetch();
  }, [searchQuery, refetch]);
  useEffect(() => {
    refetch();
  }, [page, refetch]);
  const handlePageChange = (e, value) => {
    setPage(value);
  };

  return (
    <Paper className="flex flex-col flex-nowrap gap-4 p-4 mt-10">
      {(isLoading || isRefetching) && (
        <Backdrop
          sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}

```

```

        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {(isLoading || isRefetching) && (
      <Box>
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
      </Box>
    )}
    {!isLoading && !isRefetching && novels && (
      <Box>
        <Typography variant="h4">Search result for:
{searchQuery}</Typography>
        <Box>
          <NovelsFeed novels={novels.pageData} />
          <Pagination
            count={Math.ceil(parseInt(novels.totalCount) / limit)}
            page={page}
            onChange={handlePageChange}
            shape="rounded"
          />
        </Box>
      </Box>
    )}
  </Paper>
);
};

export default SearchResult;

```

Year.jsx

```

import { useQuery } from "@tanstack/react-query";
import NovelsFeed from "../components/NovelsFeed";
import {
  Box,
  Skeleton,
  Backdrop,
  CircularProgress,
  Pagination,
  Paper,
  Typography,
} from "@mui/material";
import { useEffect, useState } from "react";
import { Navigate, useParams } from "react-router-dom";

const Year = () => {
  const { year } = useParams();
  const [page, setPage] = useState(1);
  const limit = 5;

  const {
    data: novels,
    isLoading,
    refetch,
    isRefetching,
  } = useQuery({
    queryKey: ["novels"],
    queryFn: async () => {

```

```

    try {
      const res = await fetch(
        "/api/novel/year/" + year + "?page=" + (page - 1) +
        "&limit=" + limit
      );
      const data = await res.json();

      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
});
useEffect(() => {
  refetch();
}, [page, refetch]);
const handlePageChange = (e, value) => {
  setPage(value);
};
return (
  <Paper className="flex flex-col flex-nowrap gap-4 p-4 mt-10">
    {(isLoading || isRefetching) && (
      <Backdrop
        sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {(isLoading || isRefetching) && (
      <Box>
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
      </Box>
    )}
    {!isLoading && !novels && <Navigate to={"/404"} />}
    {!isLoading && !isRefetching && novels && (
      <Box>
        <Typography variant="h4">Year: {year}</Typography>
        <NovelsFeed novels={novels.pageData} />
        <Pagination
          count={Math.ceil(parseInt(novels.totalCount) / limit)}
          page={page}
          onChange={handlePageChange}
          shape="rounded"
        />
      </Box>
    )}
  </Paper>
);
};
export default Year;

```

Profile.jsx

```

import { useMutation, useQuery } from "@tanstack/react-query";
import {
  Box,
  Backdrop,
  CircularProgress,
  Paper,
  Typography,
  Avatar,
  Divider,
  Pagination,
  Skeleton,
  Button,
  TextField,
  Modal,
  IconButton,
} from "@mui/material";
import dayjs from "dayjs";
import { Link, Navigate, useNavigate, useParams } from "react-router-dom";
import ReviewTile from "../../components/ReviewTile";
import { useEffect, useRef, useState } from "react";
import { stringToColor } from "../../stringToColor";
import toast from "react-hot-toast";
import { Edit } from "@mui/icons-material";

const Profile = () => {
  const { userId } = useParams();
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });

  const navigate = useNavigate();

  const [page, setPage] = useState(1);
  const limit = 5;

  const [open, setOpen] = useState(false);

  const [username, setUsername] = useState("");
  const [coverImg, setCoverImg] = useState(null);
  const imgRef = useRef(null);

  const {
    data: user,
    isLoading,
    isPending,
    isError,
    refetch: refetchUser,
  } = useQuery({
    queryKey: ["getUserProfile" + userId],
    queryFn: async () => {
      try {
        const res = await fetch("/api/user/get/" + userId);
        const data = await res.json();
        if (data.error) {
          throw new Error(data.error || "Something went wrong");
        }
        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

```

```

    retry: 1,
  });

const {
  data: userReviews,
  isRefetching,
  refetch,
} = useQuery({
  queryKey: ["userReviews"],
  queryFn: async () => {
    try {
      const res = await fetch("/api/user/reviews/" + userId);
      const data = await res.json();

      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
});

const { mutate: updateUserInfo, isPending: isUpdating } =
useMutation({
  mutationFn: async () => {
    try {
      if (userId !== authUser._id) {
        throw new Error("Permission denied");
      }
      const res = await fetch("/api/user/update/" + userId, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          username: username,
          profileImg: coverImg,
        }),
      });
    } catch (error) {
      throw new Error(error);
    }
  },
  onSuccess: () => {
    navigate("/profile/" + userId);
    toast.success("Profile updated successfully");
    handleClose();
    refetchUser();
    refetch();
  },
  onError: (e) => {
    toast.error(e.message);
  },
});

```

```

useEffect(() => {
  refetch();
}, [userId]);

useEffect(() => {
  refetch();
}, [page, refetch]);

const handlePageChange = (e, value) => {
  setPage(value);
};

const handleProfileUpdate = (e) => {
  e.preventDefault();
  if (userId === authUser._id) {
    updateUserInfo();
  }
};

const handleImgChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    const reader = new FileReader();
    reader.onload = () => {
      setCoverImg(reader.result);
    };
    reader.readAsDataURL(file);
  }
};

const handleClearImage = () => {
  if (coverImg) {
    setCoverImg(null);
  }
};

const handleOpen = () => setOpen(true);
const handleClose = () => {
  setOpen(false);
};

useEffect(() => {
  if (!isLoading && userReviews && !userReviews.isLoading) {
    setUsername(user.username);
    setCoverImg(user.profileImg);
  }
}, [isLoading]);

return (
  <Paper className="p-2 mt-10">
    {(isLoading || isPending || isUpdating) && (
      <Backdrop
        sx={{
          color: "#fff",
          zIndex: (theme) => theme.zIndex.drawer + 1,
        }}
        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {(isLoading || isPending) && (
      <Skeleton variant="rectangular" height={400} />
    )}
  </Paper>
);

```

```

    ))
    {!isLoading && user && (
      <Box fullWidth className="mb-10">
        <Box className="flex flex-col w-full px-20 gap-4">
          <Box className="flex flex-col justify-between items-center
w-full px-20 mt-10">
            {user.profileImg ? (
              <Avatar sx={{ width: 96, height: 96 }}
src={user.profileImg} />
            ) : (
              <Avatar
                sx={{
                  width: 96,
                  height: 96,
                  bgcolor: stringToColor(user.username),
                }}
              >
                {user.username.charAt(0)}
              </Avatar>
            )}
          <Box className="flex gap-2 justify-center items-center">
            <Typography fontSize={24} fontWeight={700}>
              User: {user.username}
            </Typography>
            {authUser && userId === authUser._id && (
              <IconButton onClick={handleOpen}>
                <Edit color="disabled" />
              </IconButton>
            )}
          </Box>
        </Box>
      <Box className="mt-4">
        <Typography fontSize={20} fontWeight={700}>
          About:
        </Typography>
        <Typography fontSize={18}>{user.about}</Typography>
      </Box>
      <Box className="flex justify-between items-center ">
        <Typography fontSize={18} color={"gray"}>
          Name:
        </Typography>
        <Typography fontSize={18}>{user.username}</Typography>
      </Box>
      <Divider />
      <Box className="flex justify-between items-center ">
        <Typography fontSize={18} color={"gray"}>
          Registered:
        </Typography>
        <Typography fontSize={18}>
          {dayjs(user.createdAt).format("MMMM D, YYYY h:mm A")}
        </Typography>
      </Box>
      <Divider />
      <Box className="flex justify-between items-center ">
        <Typography fontSize={18} color={"gray"}>
          User role:
        </Typography>
        <Typography fontSize={18}>
          {user.isModerator ? "Moderator" : "User"}
        </Typography>
      </Box>
      <Divider />
      <Box className="flex justify-between items-center ">

```

```

        <Typography fontSize={18} color={"gray"}>
          Number of comments:
        </Typography>
        <Typography
fontSize={18}>{user.reviews.length}</Typography>
        </Box>

    <Divider />
    {userReviews &&
      userReviews.totalCount > 0 &&
      !(isLoading || isRefetching) && (
        <Box>
          <Typography fontSize={20} fontWeight={700}>
            Comments:
          </Typography>
          <Box>
            {userReviews.pageData.map((review) => (
              <Link
                key={review._id}
                to={
                  "/" +
                  (review.novel
                    ? "novel/" + review.novel
                    : "chapter/" + review.chapter)
                }
              >
                <ReviewTile review={review}
linksDisabled={true} />
              </Link>
            ))}
            <Pagination
              count={Math.ceil(
                parseInt(userReviews.totalCount) / limit
              )}
              page={page}
              onChange={handlePageChange}
              shape="rounded"
            />
          </Box>
        </Box>
      )}
    </Box>
    <Modal
      open={open}
      onClose={handleClose}
      aria-labelledby="modal-modal-title"
      aria-describedby="modal-modal-description"
    >
      <Box className="flex flex-col gap-4" sx={modalStyle}>
        <Typography variant="h6" component="h2">
          Edit Profile
        </Typography>
        {coverImg && (
          <Avatar
            className="self-center"
            sx={{ width: 150, height: 150 }}
            src={coverImg}
          />
        )}
        <Box className="self-center flex gap-4">
          <Button variant="contained" size="large"
component="label">
            Load a cover

```

```

        <input
          accept="image/png, image/jpeg"
          type="file"
          hidden
          ref={imgRef}
          onChange={handleImgChange}
        />
      </Button>
    {coverImg && (
      <Button
        variant="contained"
        size="large"
        onClick={handleClearImage}
      >
        Delete Image
      </Button>
    )}
  </Box>
<br />
<br />
<TextField
  variant="standard"
  placeholder="Username"
  value={username}
  onChange={(e) => setUsername(e.target.value)}
  fullWidth
/>
<Box className="flex flex-nowrap gap-4">
  <Button
    variant="contained"
    size="medium"
    onClick={handleProfileUpdate}
  >
    {isUpdating ? "Loading..." : "Update"}
  </Button>
  <Button variant="contained" size="medium"
onClick={handleClose}>
    Cancel
  </Button>
</Box>
</Box>
</Modal>
</Box>
  )}
  {isError && <Navigate to={"/404"} />}
</Paper>
);
};

const modalStyle = {
  position: "absolute",
  top: "50%",
  left: "50%",
  transform: "translate(-50%, -50%)",
  width: 800,
  height: 740,
  bgcolor: "background.paper",
  border: "2px solid #000",
  boxShadow: 24,
  p: 4,
};

export default Profile;

```

Saved.jsx

```
import { useQuery } from "@tanstack/react-query";
import {
  Box,
  CircularProgress,
  Paper,
  Typography,
  Backdrop,
  Pagination,
  Skeleton,
} from "@mui/material";
import SavedFeed from "../../components/SavedFeed";
import { Navigate, useNavigate, useParams } from "react-router-dom";
import { useEffect, useState } from "react";

const Saved = () => {
  const { userId } = useParams();
  const { data: authUser } = useQuery({ queryKey: ["authUser"] });

  const [page, setPage] = useState(1);
  const limit = 5;

  const navigate = useNavigate();

  const {
    data: novels,
    isLoading,
    refetch,
    isRefetching,
    isError,
  } = useQuery({
    queryKey: ["novels" + userId],
    queryFn: async () => {
      try {
        const res = await fetch(
          "/api/user/saves/" +
            userId +
            "?page=" +
            (page - 1) +
            "&limit=" +
            limit
        );
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }

        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
  });

  useEffect(() => {
    refetch();
  }, [page, refetch]);

  const handlePageChange = (e, value) => {
```

```

    setPage(value);
  };

  if (userId !== authUser._id) {
    navigate("/");
  }
  return (
    <Paper className="flex flex-col flex-nowrap gap-4 p-2 mt-10">
      {(isLoading || isRefetching) && (
        <Backdrop
          sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
          open={true}
        >
          <CircularProgress color="inherit" />
        </Backdrop>
      )}
      {(isLoading || isRefetching) && (
        <Box>
          <Skeleton variant="rectangular" height={150} />
          <Skeleton variant="rectangular" height={150} />
          <Skeleton variant="rectangular" height={150} />
        </Box>
      )}
      {!isLoading && novels && (
        <Box>
          <Typography variant="h4">Bookmarks</Typography>
          <SavedFeed novels={novels.pageData} />
          <Pagination
            count={Math.ceil(parseInt(novels.totalCount) / limit)}
            page={page}
            onChange={handlePageChange}
            shape="rounded"
          />
        </Box>
      )}
      {!isLoading && <Navigate to={"/404"} />}
    </Paper>
  );
};

export default Saved;

```

Home.jsx

```

import { useQuery } from "@tanstack/react-query";
import NovelsFeed from "../components/NovelsFeed";
import {
  Box,
  Skeleton,
  Backdrop,
  CircularProgress,
  Pagination,
  Paper,
} from "@mui/material";
import toast from "react-hot-toast";
import { useEffect, useState } from "react";

const Home = () => {
  const [page, setPage] = useState(1);
  const limit = 5;

```

```

const {
  data: novels,
  isLoading,
  refetch,
  isRefetching,
} = useQuery({
  queryKey: ["novels"],
  queryFn: async () => {
    try {
      const res = await fetch(
        "/api/novel/novels?page=" + (page - 1) + "&limit=" + limit
      );
      const data = await res.json();

      if (!res.ok) {
        throw new Error(data.error || "Something went wrong");
      }

      return data;
    } catch (error) {
      throw new Error(error);
    }
  },
  onError: (error) => {
    toast.error(error.message);
  },
});
useEffect(() => {
  refetch();
}, [page, refetch]);
const handlePageChange = (e, value) => {
  setPage(value);
};
return (
  <Paper className="flex flex-col flex-nowrap gap-4 px-4 pb-4 mt-
10">
    {isLoading && (
      <Backdrop
        sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
        open={true}
      >
        <CircularProgress color="inherit" />
      </Backdrop>
    )}
    {(isLoading || isRefetching) && (
      <Box>
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
        <Skeleton variant="rectangular" height={232} />
      </Box>
    )}
    {!isLoading && !isRefetching && novels && (
      <Box>
        <NovelsFeed novels={novels.pageData} />
        <Pagination
          count={Math.ceil(parseInt(novels.totalCount) / limit)}
          page={page}
          onChange={handlePageChange}
          shape="rounded"
        />
      </Box>
    )}
  </Paper>
)

```

```

    </Paper>
  );
};

export default Home;

```

Page404.jsx

```

import { Box, Typography } from "@mui/material";

const Page404 = () => {
  return (
    <Box className="flex flex-col justify-center items-center gap-4 my-40">
      <Typography variant="h2" fontSize={96} fontWeight={700}>
        404
      </Typography>
      <Typography fontSize={36}>Not Found</Typography>
    </Box>
  );
};

export default Page404;

```

App.jsx

```

import { Toaster } from "react-hot-toast";
import { Route, Routes } from "react-router-dom";
import { useQuery } from "@tanstack/react-query";
import { Backdrop, Box, CircularProgress, Container } from
"@mui/material";

import Home from "../pages/Home";
import Page404 from "../pages/Page404";

import Login from "../pages/Auth/Login";
import Register from "../pages/Auth/Register";

import Novel from "../pages/Novel/Novel";
import Chapter from "../pages/Novel/Chapter";
import CreateNovel from "../pages/Novel/CreateNovel";

import Profile from "../pages/User/Profile";
import Saved from "../pages/User/Saved";

import Year from "../pages/Search/Year";
import Genre from "../pages/Search/Genre";
import Author from "../pages/Search/Author";

import Footer from "../components/Footer";
import Navbar from "../components/Navbar";

import AnonymousRoute from "../middleware/AnonymousRoute";
import ModeratorRoute from "../middleware/ModeratorRoute";
import PrivateRoute from "../middleware/PrivateRoute";
import EditNovel from "../pages/Novel/EditNovel";
import SearchResult from "../pages/Search/SearchResult";

function App() {
  const { data: authUser, isLoading } = useQuery({

```

```

    queryKey: ["authUser"],
    queryFn: async () => {
      try {
        const res = await fetch("/api/auth/me");
        const data = await res.json();
        if (data.error) return null;
        if (!res.ok) {
          throw new Error(data.error || "Something went wrong");
        }
        return data;
      } catch (error) {
        throw new Error(error);
      }
    },
    retry: false,
  });

  if (isLoading) {
    return (
      <>
        <Backdrop
          sx={{ color: "#fff", zIndex: (theme) => theme.zIndex.drawer
+ 1 }}
        >
          <CircularProgress color="inherit" />
        </Backdrop>
      </>
    );
  }
  return (
    <Box className="flex flex-col min-h-screen">
      <Navbar />
      <Container maxWidth="lg" className="mb-10">
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="*" element={<Page404 />} />
          <Route path="/year/:year" element={<Year />} />
          <Route path="/novel/:novelId" element={<Novel />} />
          <Route path="/genre/:genreId" element={<Genre />} />
          <Route path="/author/:author" element={<Author />} />
          <Route path="/profile/:userId" element={<Profile />} />
          <Route path="/chapter/:chapterId" element={<Chapter />} />
          <Route path="/search/:searchQuery" element={<SearchResult
/>} />
          <Route element={<AnonymousRoute />}>
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
          </Route>
          <Route Route element={<ModeratorRoute />}>
            <Route path="/add-novel" element={<CreateNovel />} />
            <Route path="/update/:novelId" element={<EditNovel />} />
          </Route>
          <Route Route element={<PrivateRoute />}>
            <Route path="/saved/:userId" element={<Saved />} />
          </Route>
        </Routes>
      </Container>
      <Footer />
      <Toaster />
    </Box>
  );
}

```

```
export default App;
```

index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  background-color: #F5F5F5 !important;
}

.txt:hover {
  text-decoration: underline;
}
```

main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";

import { BrowserRouter } from "react-router-dom";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      refetchOnWindowFocus: false,
    },
  },
});

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <BrowserRouter>
      <QueryClientProvider client={queryClient}>
        <App />
      </QueryClientProvider>
    </BrowserRouter>
  </React.StrictMode>
);

stringToColor.js
export function stringToColor(string) {
  let hash = 0;
  let i;

  /* eslint-disable no-bitwise */
  for (i = 0; i < string.length; i += 1) {
    hash = string.charCodeAt(i) + ((hash << 5) - hash);
  }

  let color = "#";

  for (i = 0; i < 3; i += 1) {
    const value = (hash >> (i * 8)) & 0xff;
```

```

    color += `00${value.toString(16)}`.slice(-2);
  }
  /* eslint-enable no-bitwise */

  return color;
}

```

Б.2 Server

/controllers/auth.js

```

import bcrypt from "bcryptjs";
import { body } from "express-validator";
import { generateTokenAndSetCookie } from
"../lib/utils/generateToken.js";
import User from "../models/user.js";

export const register = async (req, res) => {
  try {
    const { username, email, password } = req.body;

    if (!body("email").isEmail) {
      return res.status(400).json({ error: "Invalid email format" });
    }

    const userExist = await User.findOne({ username });
    if (userExist) {
      return res.status(400).json({ error: "Username is already taken"
});
    }

    const existingEmail = await User.findOne({ email });
    if (existingEmail) {
      return res.status(400).json({ error: "Email is already
taken" });
    }
    if (username.length < 5) {
      return res
        .status(400)
        .json({ error: "Username must be at least 5 characters
long" });
    }

    if (password.length < 8) {
      return res
        .status(400)
        .json({ error: "Password must be at least 8 characters
long" });
    }

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const newUser = new User({
      username,
      email,
      password: hashedPassword,
    });
  }
}

```

```

    if (newUser) {
      generateTokenAndSetCookie(newUser._id, res);
      await newUser.save();

      res.status(201).json({
        _id: newUser._id,
        username: newUser.username,
        email: newUser.email,
        isModerator: newUser.isModerator,
        profileImg: newUser.profileImg,
        about: newUser.about,
        likedReview: newUser.likedReview,
        disLikedReview: newUser.disLikedReview,
        comments: newUser.comments,
        saves: newUser.saves,
        followedNovels: newUser.followedNovels,
      });
    } else {
      res.status(400).json({ error: "Invalid user data" });
    }
    return res.status(200).json();
  } catch (error) {
    console.log("Error in : ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const login = async (req, res) => {
  try {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
    const isPasswordCorrect = await bcrypt.compare(
      password,
      user?.password || ""
    );

    if (!user || !isPasswordCorrect) {
      return res.status(400).json({ error: "Invalid username or password" });
    }

    generateTokenAndSetCookie(user._id, res);

    res.status(201).json({
      _id: user._id,
      username: user.username,
      email: user.email,
      isModerator: user.isModerator,
      profileImg: user.profileImg,
      about: user.about,
      likedReview: user.likedReview,
      disLikedReview: user.disLikedReview,
      comments: user.comments,
      saves: user.saves,
      followedNovels: user.followedNovels,
    });
  } catch (error) {
    console.log("Error in : ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const logout = async (req, res) => {

```

```

    try {
      res.clearCookie("jwt");
      res.status(200).json({ message: "Logged out successfully" });
    } catch (error) {
      console.log("Error in : ", error.message);
      res.status(500).json({ error: error.message });
    }
  };

export const getAuthUser = async (req, res) => {
  try {
    const user = await User.findById(req.user._id).select("-password");
    res.status(200).json(user);
  } catch (error) {
    console.log("Error in : ", error.message);
    res.status(500).json({ error: error.message });
  }
};

```

controllers/chapter.js

```

import Novel from "../models/novel.js";
import Chapter from "../models/chapter.js";
import User from "../models/user.js";
import Review from "../models/review.js";

export const addChapter = async (req, res) => {
  try {
    const { novelId, chapterTitle, chapterBody } = req.body;

    const novel = await Novel.findById(novelId);
    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }
    if (
      novelId.length === 0 &&
      chapterTitle.length === 0 &&
      chapterBody.length === 0
    ) {
      return res
        .status(400)
        .json({ error: "Required fields are not filled in" });
    }
    const chapterNumber = novel.chapters.length + 1;
    const newChapter = new Chapter({
      chapterNumber: chapterNumber,
      novel: novelId,
      title: chapterTitle,
      body: chapterBody,
    });
    await newChapter.save();
    await Novel.updateOne(
      { _id: novelId },
      { $push: { chapters: newChapter._id } }
    );
    res.status(201).json(newChapter);
  } catch (error) {
    console.log("Error in addChapter: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

```

```

export const updateChapter = async (req, res) => {
  try {
    const { chapterId, chapterTitle, chapterBody } = req.body;

    const chapter = await Chapter.findById(chapterId);
    if (!chapter) {
      return res.status(404).json({ error: "Chapter not found" });
    }

    if (
      chapterId.length === 0 &&
      chapterTitle.length === 0 &&
      chapterBody.length === 0
    ) {
      return res
        .status(400)
        .json({ error: "Required fields are not filled in" });
    }
    await Chapter.findByIdAndUpdate(chapterId, {
      title: chapterTitle,
      body: chapterBody,
    });
    res.status(201).json("Ok");
  } catch (error) {
    console.log("Error in addChapter: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getChapter = async (req, res) => {
  try {
    const chapterId = req.params.id;
    const chapter = await Chapter.findById(chapterId);

    if (!chapter) {
      return res.status(404).json({ error: "Chapter not found" });
    }
    res.status(200).json(chapter);
  } catch (error) {
    console.log("Error in getChapter: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getAllChapters = async (req, res) => {
  try {
    const novelId = req.params.id;
    const chapters = await Chapter.find({ novel: novelId });
    if (chapters.length === 0) {
      return res.status(200).json([]);
    }
    res.status(200).json(chapters);
  } catch (error) {
    console.log("Error in getAllChapters: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const addViewCount = async (req, res) => {
  try {
    const chapterId = req.params.id;
    const chapter = await Chapter.findById(chapterId);

```

```

    chapter.viewsCount += 1;
    chapter.save();
    res.status(200).json(chapter);
  } catch (error) {
    console.log("Error in addViewCountChapter: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getNext = async (req, res) => {
  try {
    const chapterId = req.params.id;
    const current_chapter = await Chapter.findById(chapterId);

    if (!current_chapter) {
      return res.status(404).json({ error: "Chapter not found" });
    }
    const next_chapter = await Chapter.findOne({
      chapterNumber: current_chapter.chapterNumber + 1,
      novel: current_chapter.novel,
    });
    if (!next_chapter) {
      return res.status(404).json({ error: "Chapter not found" });
    }
    res.status(200).json(next_chapter);
  } catch (error) {
    console.log("Error in getNext: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getPrev = async (req, res) => {
  try {
    const chapterId = req.params.id;
    const current_chapter = await Chapter.findById(chapterId);

    if (!current_chapter || current_chapter.chapterNumber === 1) {
      return res.status(404).json({ error: "Chapter not found" });
    }
    const prev_chapter = await Chapter.findOne({
      chapterNumber: current_chapter.chapterNumber - 1,
      novel: current_chapter.novel,
    });
    if (!prev_chapter) {
      return res.status(404).json({ error: "Chapter not found" });
    }
    res.status(200).json(prev_chapter);
  } catch (error) {
    console.log("Error in getPrev: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getReviews = async (req, res) => {
  try {
    const chapterId = req.params.id;
    const chapter = await Chapter.findById(chapterId);
    if (!chapter) {
      return res.status(404).json({ error: "Chapter not found" });
    }
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const reviews = await Review.find({ chapter: chapterId }, null, {

```

```

        skip: page * limit,
        limit: limit,
    }).sort({ createdAt: -1 });

    if (reviews.length === 0) {
        return res.status(404).json({ error: "Reviews not found" });
    }
    const total_reviews = await Review.find({ chapter: chapterId });
    res
        .status(200)
        .json({ pageData: reviews, totalCount: total_reviews.length });
} catch (error) {
    console.log("Error in : ", error.message);
    res.status(500).json({ error: error.message });
}
};

export const reviewOnChapter = async (req, res) => {
    try {
        const { userId, text } = req.body;
        const chapterId = req.params.id;

        if (!text) {
            return res.status(400).json({ error: "Text field is required" });
        }
        const chapter = await Chapter.findById(chapterId);
        const user = await User.findById(userId);

        if (!chapter) {
            return res.status(404).json({ error: "Novel not found" });
        }
        if (!user) {
            return res.status(404).json({ error: "User not found" });
        }

        const review = new Review({ user: userId, chapter: chapter, body: text });

        chapter.reviews.push(review);
        user.reviews.push(review);
        await review.save();
        await chapter.save();
        await user.save();

        res.status(200);
    } catch (error) {
        console.log("Error in reviewOnChapter: ", error);
        res.status(500).json({ error: "Internal server error" });
    }
};

```

/controllers/genre.js

```

import Genre from "../models/genre.js";
import Novel from "../models/novel.js";

export const getAll = async (req, res) => {
    try {
        const genres = await Genre.find();
        if (genres.length === 0) {
            return res.status(200).json([]);
        }
    }
};

```

```

    }
    res.status(200).json(genres);
  } catch (error) {
    console.log("Error in getAll: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getGenre = async (req, res) => {
  try {
    const genreId = req.params.id;

    const genre = await Genre.findById(genreId);

    if (!genre) {
      return res.status(404).json({ error: "Genre not found" });
    }
    res.status(200).json(genre);
  } catch (error) {
    console.log("Error in getAll: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getNovelsWithGenre = async (req, res) => {
  try {
    const genreId = req.params.id;
    const genre = await Genre.findById(genreId);

    if (!genre) {
      return res.status(404).json({ error: "Genre not found" });
    }
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const novels = await Novel.find(
      { genres: { $elemMatch: { $eq: genreId } } },
      null,
      {
        skip: page * limit,
        limit: limit,
      }
    );
    if (novels.length === 0) {
      return res.status(404).json({ error: "Novels not found" });
    }
    const total_novels = await Novel.find({
      genres: { $elemMatch: { $eq: genreId } },
    });
    res.status(200).json({ pageData: novels, totalCount:
total_novels.length });
  } catch (error) {
    console.log("Error in getNovelsWithGenre: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

```

/controllers/novel.js

```

import User from "../models/user.js";
import Novel from "../models/novel.js";
import Genre from "../models/genre.js";
import Review from "../models/review.js";

```

```

import Chapter from "../models/chapter.js";

import { v2 as cloudinary } from "cloudinary";

export const createNovel = async (req, res) => {
  try {
    const { title, description, status, year, author, genres } =
req.body;
    let { coverImg } = req.body;

    if (
      !(title && description && status && year && author &&
genres.length !== 0)
    ) {
      return res
        .status(400)
        .json({ error: "Required fields are not filled in" });
    }
    if (coverImg) {
      const uploadResponse = await
cloudinary.uploader.upload(coverImg);
      coverImg = uploadResponse.secure_url;
    }
    for (const genreId of genres) {
      const find = await Genre.findById(genreId);
      if (!find) {
        return res.status(404).json({ error: "Genre not found" });
      }
    }
    const newNovel = new Novel({
      title: title,
      description: description,
      status: status,
      year: year,
      genres: genres,
      author: author,
      coverImg: coverImg,
    });
    await newNovel.save();
    res.status(201).json(newNovel);
  } catch (error) {
    console.log("Error in createNovel: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const updateNovel = async (req, res) => {
  try {
    const novelId = req.params.id;
    const { title, description, status, year, author, genres } =
req.body;
    let { coverImg } = req.body;
    const novel = await Novel.findById(novelId);
    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }

    if (
      !(title && description && status && year && author &&
genres.length !== 0)
    ) {
      return res
        .status(400)

```

```

        .json({ error: "Required fields are not filled in" });
    }

    if (coverImg && coverImg !== novel.coverImg) {
        if (novel.coverImg) {
            await cloudinary.uploader.destroy(
                novel.coverImg.split("/").pop().split(".")[0],
                {
                    resource_type: "image",
                }
            );
        }
        const uploadResponse = await
cloudinary.uploader.upload(coverImg);
        coverImg = uploadResponse.secure_url;
    } else if (coverImg && novel.coverImg) {
        await cloudinary.uploader.destroy(
            novel.coverImg.split("/").pop().split(".")[0],
            {
                resource_type: "image",
            }
        );
    }
}

for (const genreId of genres) {
    const find = await Genre.findById(genreId);
    if (!find) {
        return res.status(404).json({ error: "Genre not found" });
    }
}

await Novel.findByIdAndUpdate(novelId, {
    title: title,
    description: description,
    status: status,
    year: year,
    genres: genres,
    author: author,
    coverImg: coverImg,
});
res.status(201).json("Ok");
} catch (error) {
    console.log("Error in updateNovel: ", error.message);
    res.status(500).json({ error: error.message });
}
};

export const reviewOnNovel = async (req, res) => {
    try {
        const { userId, text } = req.body;
        const novelId = req.params.id;

        if (!text) {
            return res.status(400).json({ error: "Text field is
required" });
        }
        const novel = await Novel.findById(novelId);
        const user = await User.findById(userId);

        if (!novel) {
            return res.status(404).json({ error: "Novel not found" });
        }
        if (!user) {

```

```

        return res.status(404).json({ error: "User not found" });
    }

    const review = new Review({ user: userId, novel: novel, body: text
});

    novel.reviews.push(review);
    user.reviews.push(review);
    await review.save();
    await novel.save();
    await user.save();

    res.status(200);
  } catch (error) {
    console.log("Error in reviewOnNovel: ", error);
    res.status(500).json({ error: "Internal server error" });
  }
};

export const getAll = async (req, res) => {
  try {
    const novels = await Novel.find();
    if (novels.length === 0) {
      return res.status(200).json([]);
    }
    res.status(200).json(novels);
  } catch (error) {
    console.log("Error in getAll: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getNovels = async (req, res) => {
  try {
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const novels = await Novel.find({}, null, {
      skip: page * limit,
      limit: limit,
    }).sort({ updatedAt: -1 });
    if (novels.length === 0) {
      return res.status(200).json(0);
    }
    const total_novels = await Novel.find();
    res.status(200).json({ pageData: novels, totalCount:
total_novels.length });
  } catch (error) {
    console.log("Error in : ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getNovel = async (req, res) => {
  try {
    const novelId = req.params.id;
    const novel = await Novel.findById(novelId);
    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }
    res.status(200).json(novel);
  } catch (error) {
    console.log("Error in getNovel: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

```

```

    }
  };

export const addViewCount = async (req, res) => {
  try {
    const novelId = req.params.id;
    const novel = await Novel.findById(novelId);
    novel.viewsCount += 1;
    await novel.save();
    res.status(200).json(novel);
  } catch (error) {
    console.log("Error in addViewCountNovel: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const addStarRating = async (req, res) => {
  try {
    const novelId = req.params.id;
    const userId = req.user._id;
    const { rating } = req.body;
    const novel = await Novel.findById(novelId);
    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }
    const userStarRatingExist = novel.starRating.find(
      (obj) => obj.user.toString() === userId.toString()
    );
    if (userStarRatingExist) {
      await Novel.updateOne(
        { _id: novelId, "starRating.user": userId },
        { $set: { "starRating.$.rating": rating } }
      );
    } else {
      novel.starRating.push({ user: userId, rating: rating });
      await novel.save();
    }
    res.status(200).json(novel.starRating);
  } catch (error) {
    console.log("Error in addStarRating: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const searchForNovel = async (req, res) => {
  try {
    const searchQuery = req.params.searchQuery;
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const novels = await Novel.find({ title: { $regex:
searchQuery } }, null, {
      skip: page * limit,
      limit: limit,
    }).sort({ updatedAt: -1 });
    if (novels.length === 0) {
      return res.status(200).json("Novels not found");
    }
    const total_novels = await Novel.find({ title: { $regex:
searchQuery } });
    res.status(200).json({ pageData: novels, totalCount:
total_novels.length });
  } catch (error) {
    console.log("Error in searchForNovel: ", error.message);
  }
};

```

```

    res.status(500).json({ error: error.message });
  }
};

export const getReviews = async (req, res) => {
  try {
    const novelId = req.params.id;
    const novel = await Novel.findById(novelId);
    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const reviews = await Review.find({ novel: novelId }, null, {
      skip: page * limit,
      limit: limit,
    }).sort({ createdAt: -1 });

    if (reviews.length === 0) {
      return res.status(404).json({ error: "Reviews not found" });
    }
    const total_reviews = await Review.find({ novel: novelId });
    res
      .status(200)
      .json({ pageData: reviews, totalCount: total_reviews.length });
  } catch (error) {
    console.log("Error in getReviews: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getGenres = async (req, res) => {
  try {
    const novelId = req.params.id;
    const novel = await Novel.findById(novelId);
    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }
    const genres = [];
    for (const genreId of novel.genres) {
      const genre = await Genre.findById(genreId);
      if (genre) genres.push(genre);
    }
    if (genres.length === 0) {
      return res.status(404).json({ error: "Genres not found" });
    }
    res.status(200).json(genres);
  } catch (error) {
    console.log("Error in getReviews: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const savedUnSavedByUser = async (req, res) => {
  try {
    const userId = req.user._id;
    const novelId = req.params.id;

    const novel = await Novel.findById(novelId);

    if (!novel) {
      return res.status(404).json({ error: "Novel not found" });
    }
  }
};

```

```

    const userSaveNovel = novel.savedBy.includes(userId);

    if (userSaveNovel) {
      await Novel.updateOne({ _id: novelId }, { $pull: { savedBy:
userId } });
      await User.updateOne({ _id: userId }, { $pull: { saves:
novelId } });

      const updatedSaves = novel.savedBy.filter(
        (id) => id.toString() !== userId.toString()
      );
      res.status(200).json(updatedSaves);
    } else {
      await User.updateOne({ _id: userId }, { $push: { saves:
novelId } });
      novel.savedBy.push(userId);
      await novel.save();
      res.status(200).json(novel.savedBy);
    }
  } catch (error) {
    console.log("Error in savedUnSavedByUser: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getNovelsByAuthor = async (req, res) => {
  try {
    const author = req.params.author;
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const novels = await Novel.find({ author: author }, null, {
      skip: page * limit,
      limit: limit,
    }).sort({ updatedAt: -1 });
    if (novels.length === 0) {
      return res.status(404).json({ error: "Novels not found" });
    }
    const total_novels = await Novel.find({ author: author });
    res.status(200).json({ pageData: novels, totalCount:
total_novels.length });
  } catch (error) {
    console.log("Error in getNovelsByAuthor: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getNovelsByYear = async (req, res) => {
  try {
    const year = req.params.year;
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    const novels = await Novel.find({ year: year }, null, {
      skip: page * limit,
      limit: limit,
    }).sort({ updatedAt: -1 });
    if (novels.length === 0) {
      return res.status(404).json({ error: "Novels not found" });
    }
    const total_novels = await Novel.find({ year: year });
    res.status(200).json({ pageData: novels, totalCount:
total_novels.length });
  } catch (error) {

```

```

        console.log("Error in getNovelsByYear: ", error.message);
        res.status(500).json({ error: error.message });
    }
};

export const deleteNovel = async (req, res) => {
    try {
        const novelId = req.params.id;
        const novel = await Novel.findById(novelId);
        if (!novel) {
            return res.status(404).json({ error: "Novel not found" });
        }
        if (novel.coverImg) {
            await cloudinary.uploader.destroy(
                novel.coverImg.split("/").pop().split(".")[0],
                {
                    resource_type: "image",
                }
            );
        }
        for (const chapter of novel.chapters) {
            await Chapter.findByIdAndDelete(chapter);
        }
        for (const user of novel.savedBy) {
            await User.updateOne({ _id: user }, { $pull: { saves:
novelId } });
        }
        await Novel.findByIdAndDelete(novelId);
        res.status(200).json({ message: "Post deleted successfully" });
    } catch (error) {
        console.log("Error in getNovel: ", error.message);
        res.status(500).json({ error: error.message });
    }
};

```

/controllers/user.js

```

import { v2 as cloudinary } from "cloudinary";

import User from "../models/user.js";
import Novel from "../models/novel.js";
import Review from "../models/review.js";

export const updateUser = async (req, res) => {
    try {
        const userId = req.params.id;
        const { username } = req.body;
        let { profileImg } = req.body;
        const user = await User.findById(userId);
        if (!user) {
            return res.status(404).json({ error: "User not found" });
        }

        if (!username) {
            return res
                .status(400)
                .json({ error: "Required fields are not filled in" });
        }
        if (/s/.test(username)) {
            return res.status(400).json({ error: "Username can't contain a
spaces" });
        }
    }
};

```

```

    if (username.length < 5) {
      return res
        .status(400)
        .json({ error: "Username must be at least 5 characters
long" });
    }

    if (profileImg && profileImg !== user.profileImg) {
      if (user.profileImg) {
        await cloudinary.uploader.destroy(
          user.profileImg.split("/").pop().split(".")[0],
          {
            resource_type: "image",
          }
        );
      }
      const uploadResponse = await
cloudinary.uploader.upload(profileImg);
      profileImg = uploadResponse.secure_url;
    } else if (profileImg && user.profileImg) {
      await cloudinary.uploader.destroy(
        user.profileImg.split("/").pop().split(".")[0],
        {
          resource_type: "image",
        }
      );
    }

    await User.findByIdAndUpdate(userId, {
      username: username,
      profileImg: profileImg,
    });
    res.status(201).json("Ok");
  } catch (error) {
    console.log("Error in updateUser: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getUser = async (req, res) => {
  try {
    const userId = req.params.id;

    const user = await User.findById(userId).select("-password");
    if (!user) return res.status(404).json({ message: "User not found"
});

    res.status(200).json(user);
  } catch (error) {
    console.log("Error in getUser: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getUserByReview = async (req, res) => {
  try {
    const reviewId = req.params.id;

    const review = await Review.findById(reviewId);
    if (!review) return res.status(404).json({ message: "Review not
found" });

    const user = await User.findById(review.user).select("-password");

```

```

    if (!user) return res.status(404).json({ message: "User not found"
  });
  res.status(200).json(user);
} catch (error) {
  console.log("Error in getUserByReview: ", error.message);
  res.status(500).json({ error: error.message });
}
};

export const getReviews = async (req, res) => {
  try {
    const userId = req.params.id;
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ error: "User not found" });
    }
    const reviews = await Review.find({ user:
user }).sort({ createdAt: -1 });
    if (reviews.length === 0) {
      return res.status(404).json({ error: "Reviews not found" });
    }
    const total_reviews = await Review.find({ user: user });
    res
      .status(200)
      .json({ pageData: reviews, totalCount: total_reviews.length });
  } catch (error) {
    console.log("Error in getReviews: ", error.message);
    res.status(500).json({ error: error.message });
  }
};

export const getSaves = async (req, res) => {
  try {
    const userId = req.params.id;
    const user = await User.findById(userId);
    const page = parseInt(req.query.page);
    const limit = parseInt(req.query.limit);
    if (!user) {
      return res.status(404).json({ error: "User not found" });
    }
    const novels = await Novel.find(
      {
        savedBy: { $elemMatch: { $eq: userId } },
      },
      null,
      {
        skip: page * limit,
        limit: limit,
      }
    );
    if (novels.length === 0) {
      return res.status(404).json({ error: "Novels not found" });
    }
    const total_novels = await Novel.find({
      savedBy: { $elemMatch: { $eq: userId } },
    });

    res.status(200).json({ pageData: novels, totalCount:
total_novels.length });
  } catch (error) {
    console.log("Error in getSaves: ", error.message);
    res.status(500).json({ error: error.message });
  }
}

```

```
};
```

db.js

```
import mongoose from "mongoose";

const connectMongoDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(`Error connection to mongoDB: ${error.message}`);
    process.exit(1);
  }
};

export default connectMongoDB;
```

generateToken.js

```
import jwt from "jsonwebtoken";

export const generateTokenAndSetCookie = (userId, res) => {
  const token = jwt.sign({ userId }, process.env.JWT_SECRET, {
    expiresIn: "14d",
  });

  res.cookie("jwt", token, {
    maxAge: 14 * 24 * 60 * 60 * 1000,
    httpOnly: true,
    sameSite: "strict",
    secure: process.env.NODE_ENV !== "development",
  });
};
```

checkAuth.js

```
import User from "../models/user.js";
import jwt from "jsonwebtoken";

export const checkAuth = async (req, res, next) => {
  try {
    const token = req.cookies.jwt;
    if (!token) {
      return res.status(401).json({ error: "Unauthorized: No Token Provided" });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    if (!decoded) {
      return res.status(401).json({ error: "Unauthorized: Invalid Token" });
    }

    const user = await User.findById(decoded.userId).select("-password");

    if (!user) {

```

```

    return res.status(404).json({ error: "User not found" });
  }

  req.user = user;
  next();
} catch (err) {
  console.log("Error in checkAuth middleware", err.message);
  return res.status(500).json({ error: "Internal Server Error" });
}
};

```

/models/chapter.js

```

import mongoose from "mongoose";

const chapterSchema = new mongoose.Schema(
  {
    chapterNumber: {
      type: Number,
      required: true,
    },
    novel: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Novel",
      required: true,
    },
    title: {
      type: String,
      default: "",
      required: true,
    },
    body: {
      type: String,
      required: true,
    },
    reviews: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Review",
        default: [],
      },
    ],
    viewsCount: {
      type: Number,
      default: 0,
    },
  },
  { timestamps: true }
);

const Chapter = mongoose.model("Chapter", chapterSchema);

export default Chapter;

```

/models/genre.js

```

import mongoose from "mongoose";

```

```

const genreSchema = new mongoose.Schema({
  name: {
    type: String,
    require: true,
  },
});

const Genre = mongoose.model("Genre", genreSchema);

export default Genre;

```

/models/novel.js

```

import mongoose from "mongoose";

const novelSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
    },
    coverImg: {
      type: String,
      default: "",
    },
    description: {
      type: String,
      required: true,
    },
    status: {
      type: String,
      required: true,
      enum: ["inProgress", "ended", "hiatus"],
    },
    year: {
      type: String,
      required: true,
      default: "",
    },
    author: {
      type: String,
      required: true,
    },
    genres: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Genre",
        default: [],
      },
    ],
    chapters: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Chapter",
        default: [],
      },
    ],
    viewsCount: {
      type: Number,
      default: 0,
    },
    reviews: [

```

```

    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Review",
      default: [],
    },
  ],
  savedBy: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      default: [],
    },
  ],
  starRating: [
    {
      rating: { type: Number, min: 0, max: 5, default: 0 },
      user: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
    },
  ],
},
{ timestamps: true }
);

const Novel = mongoose.model("Novel", novelSchema);

export default Novel;

```

/models/review.js

```

import mongoose from "mongoose";

const reviewSchema = new mongoose.Schema(
  {
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    novel: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Novel",
    },
    chapter: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Chapter",
    },
    body: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

const Review = mongoose.model("Review", reviewSchema);

export default Review;

```

/models/user.js

```

import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
      minLength: 8,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    isModerator: {
      type: Boolean,
      required: true,
      default: false,
    },
    profileImg: {
      type: String,
      default: "",
    },
    about: {
      type: String,
      default: "",
    },
    reviews: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Review",
        default: [],
      },
    ],
    saves: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Novel",
        default: [],
      },
    ],
  },
  { timestamps: true }
);

const User = mongoose.model("User", userSchema);

export default User;

```

/routes/auth.js

```

import { Router } from "express";
import { login, logout, register, getAuthUser } from
"../controllers/auth.js";
import { checkAuth } from "../middleware/checkAuth.js";

const router = Router();

```

```

router.get("/me", checkAuth, getAuthUser);
router.post("/register", register);
router.post("/login", login);
router.post("/logout", logout);

```

```

export default router;

```

/routes/chapter.js

```

import { Router } from "express";
import {
  addChapter,
  getAllChapters,
  getChapter,
  getNext,
  addViewCount,
  getPrev,
  reviewOnChapter,
  getReviews,
  updateChapter,
} from "../controllers/chapter.js";
import { checkAuth } from "../middleware/checkAuth.js";

const router = Router();

router.get("/get/:id", getChapter);
router.get("/next/:id", getNext);
router.get("/prev/:id", getPrev);
router.get("/novel/:id", getAllChapters);
router.get("/reviews/:id", getReviews);

router.post("/add/", checkAuth, addChapter);
router.post("/update/", updateChapter);
router.post("/review/:id", checkAuth, reviewOnChapter);
router.post("/addViewCount/:id", addViewCount);

export default router;

```

/routes/genre.js

```

import { Router } from "express";
import {
  getAll,
  getGenre,
  getNovelsWithGenre,
} from "../controllers/genre.js";

const router = Router();

router.get("/all", getAll);
router.get("/:id", getGenre);
router.get("/novels/:id", getNovelsWithGenre);

export default router;

```

/routes/novel.js

```

import { Router } from "express";
import {
  getAll,
  createNovel,
  reviewOnNovel,
  addStarRating,
  getNovels,
  searchForNovel,
  getNovel,
  addViewCount,
  getReviews,
  getGenres,
  savedUnSavedByUser,
  getNovelsByAuthor,
  getNovelsByYear,
  updateNovel,
  deleteNovel,
} from "../controllers/novel.js";
import { checkAuth } from "../middleware/checkAuth.js";

const router = Router();

router.get("/all", getAll);

router.get("/novels", getNovels);
router.get("/get/:id", getNovel);
router.get("/reviews/:id", getReviews);
router.get("/genres/:id", getGenres);

router.get("/year/:year", getNovelsByYear);
router.get("/author/:author", getNovelsByAuthor);
router.get("/search/:searchQuery", searchForNovel);

router.post("/addViewCount/:id", addViewCount);
router.post("/create", checkAuth, createNovel);
router.post("/update/:id", checkAuth, updateNovel);
router.post("/save/:id", checkAuth, savedUnSavedByUser);
router.post("/review/:id", checkAuth, reviewOnNovel);
router.post("/rating/:id", checkAuth, addStarRating);

router.delete("/:id", checkAuth, deleteNovel);

export default router;

```

/routes/user.js

```

import express from "express";
import { checkAuth } from "../middleware/checkAuth.js";
import {
  updateUser,
  getUser,
  getUserByReview,
  getReviews,
  getSaves,
} from "../controllers/user.js";

const router = express.Router();

router.get("/get/:id", getUser);
router.get("/saves/:id", checkAuth, getSaves);
router.get("/reviews/:id", getReviews);
router.get("/get-by-review/:id", getUserByReview);

```

```
router.post("/update/:id", checkAuth, updateUser);

export default router;
```

index.js

```
import express from "express";
import dotenv from "dotenv";
import cookieParser from "cookie-parser";
import { v2 as cloudinary } from "cloudinary";

import authRoutes from "./routes/auth.js";
import userRoutes from "./routes/user.js";
import novelRoutes from "./routes/novel.js";
import genreRoutes from "./routes/genre.js";
import chapterRoutes from "./routes/chapter.js";

import connectMongoDB from "./database/db.js";

dotenv.config();

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

const PORT = process.env.PORT || 5000;

const app = express();

app.use(express.json({ limit: "5mb" }));
app.use(express.urlencoded({ extended: true }));

app.use(cookieParser());

app.use("/api/auth", authRoutes);
app.use("/api/user", userRoutes);
app.use("/api/novel", novelRoutes);
app.use("/api/genre", genreRoutes);
app.use("/api/chapter", chapterRoutes);

app.listen(PORT, () => {
  console.log("Server run on ", PORT);
  connectMongoDB();
});
```

ДОДАТОК В
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

дипломна робота на тему:

Розробка онлайн платформи розповсюдження і перегляду вебновел.

Development of an Online Platform for Distributing and Viewing Web Novels.

Виконав: студент гр. КНТ-120 Нікіта ХАЛІЗЕВ

Керівник: к.т.н., доцент Ольга ГЛАДКОВА

Рисунок В.1 — Слайд № 1

Об'єкт, предмет, мета роботи

- Об'єкт дослідження – процес розробки програмного забезпечення для перегляду та розповсюдження вебконтенту.
- Предмет дослідження – програмне забезпечення для перегляду та розповсюдження вебновел.
- Мета роботи – розробка програмного забезпечення для перегляду та розповсюдження вебновел.

Рисунок В.2 — Слайд № 2

Аналіз предметної області

Критерій порівняння	WebNovel	WuxiaWorld	Light Novel Pub
Велика колекція новел різноманітних жанрів	+	-	+
Соціальні елементи	+	+	+
Розвинена система жанрів	-	+	+
Простий та інтуїтивно зрозумілий інтерфейс	-	+	+
Наявність реклами	-	+	+
Платний доступ до творів/глав	+	+	-

3

Рисунок В.3 — Слайд № 3

Вибір мови програмування

Критерій порівняння	C#	JavaScript	PHP
Велика та активна спільнота розробників	+-	+	-
Велика кількість підтримуваних СКБД	+-	+	+
Висока швидкість написання коду	-	+	+
Орієнтованість на веброзробку	+-	+	+
Висока продуктивність	+	-	-

4

Рисунок В.4 — Слайд № 4

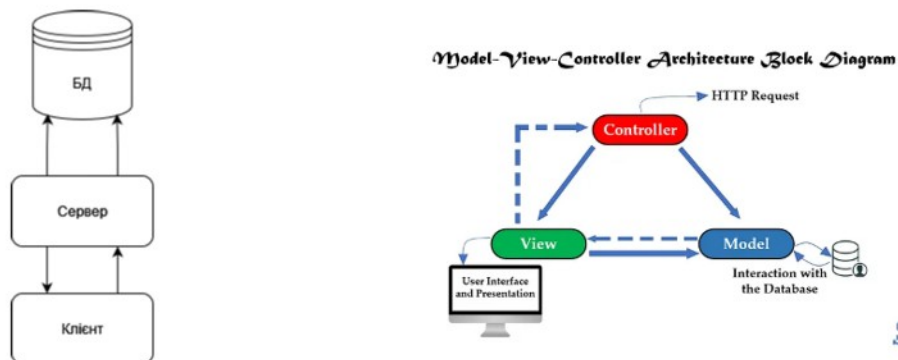
Обрання середовища розробки

IDE	Особливості	Платформи	Вартість
Visual Studio Code	Легкий та масштабований редактор коду з великою кількістю плагінів	Windows, macOS, Linux	Безкоштовний
WebStorm	Повнофункціональне	Windows, macOS, Linux	Платний
Sublime Text	Легкий редактор коду з великою кількістю плагінів	Windows, macOS, Linux	Платний

5

Рисунок В.5 — Слайд № 5

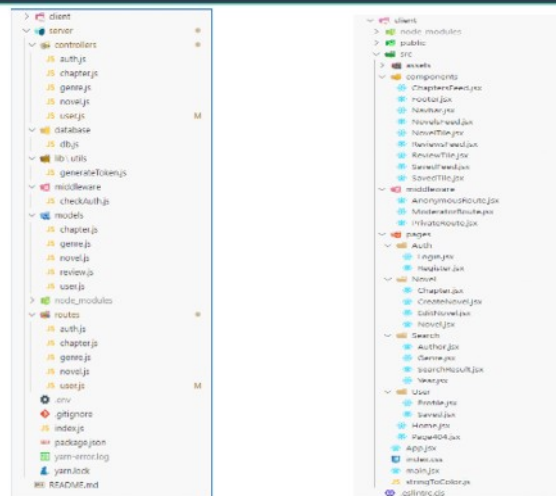
Схема роботи програми



6

Рисунок В.6 — Слайд № 6

Файлова структура вебдодатку



7

Рисунок В.7 — Слайд № 7

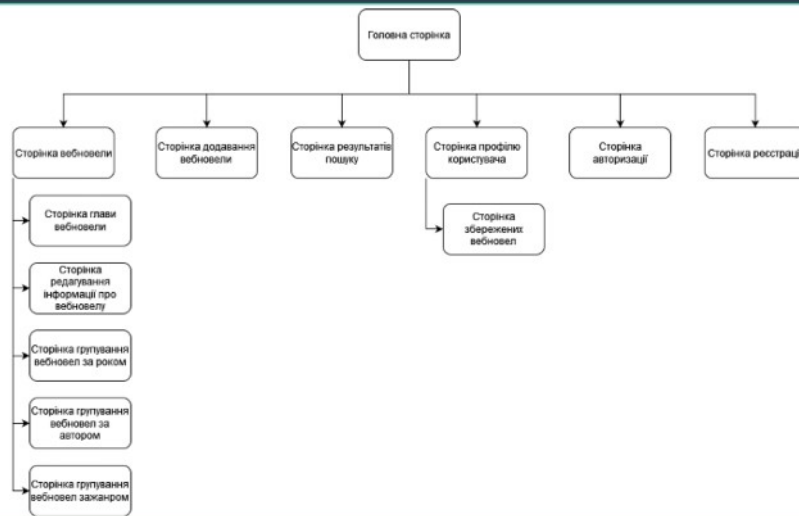
Маршрутизація

№	Маршрут	Метод	Призначення
1	2	3	4
1	/api/user/get/:id	GET	Отримати дані користувача
2	/api/user/saves/:id	GET	Отримати список збережених вебновел
3	/api/user/reviews/:id	GET	Отримати всі коментарі користувача
4	/api/user/get-by-review/:id	GET	Отримати дані користувача за id його коментаря
5	/api/user/update/:id	POST	Оновлення даних користувача

8

Рисунок В.8 — Слайд № 8

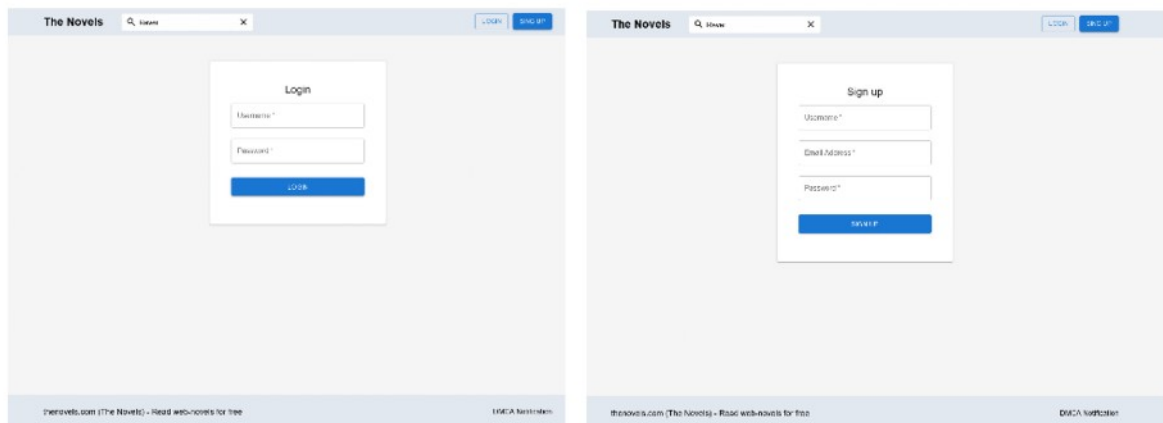
Структурна схема вебдодатку



9

Рисунок В.9 — Слайд № 9

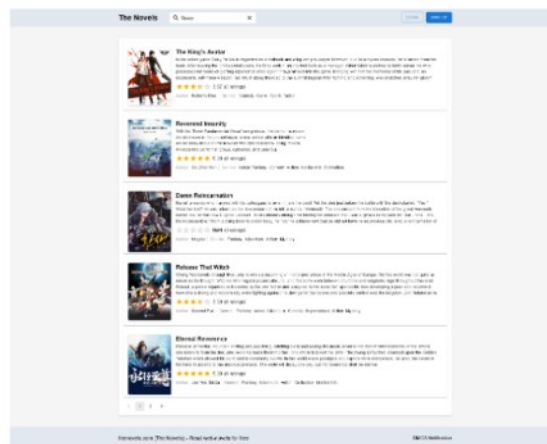
Екранні форми



10

Рисунок В.10 — Слайд № 10

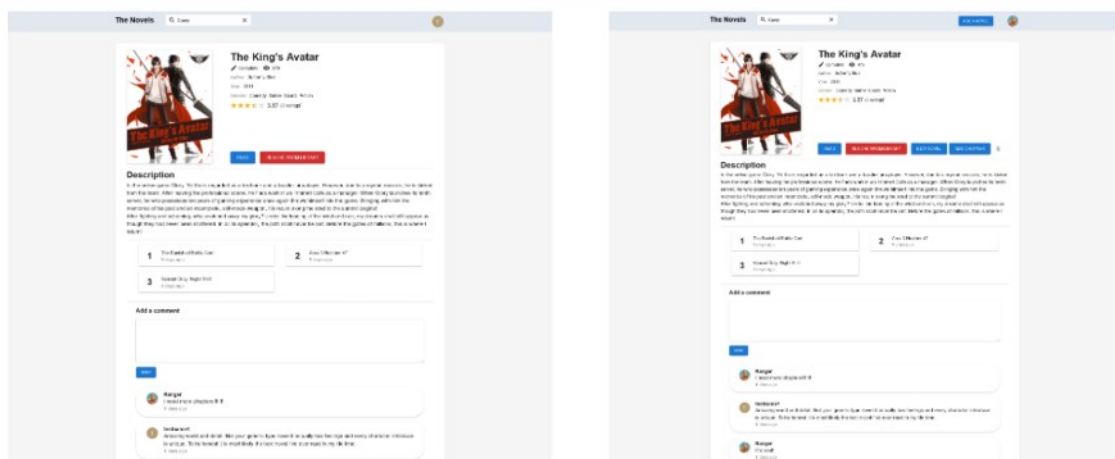
Екранні форми



11

Рисунок В.11 — Слайд № 11

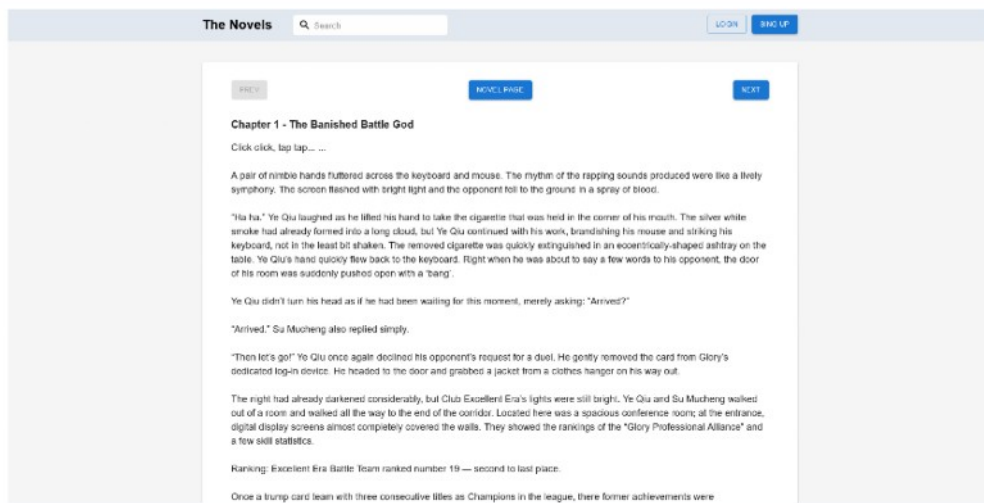
Екранні форми



12

Рисунок В.12 — Слайд № 12

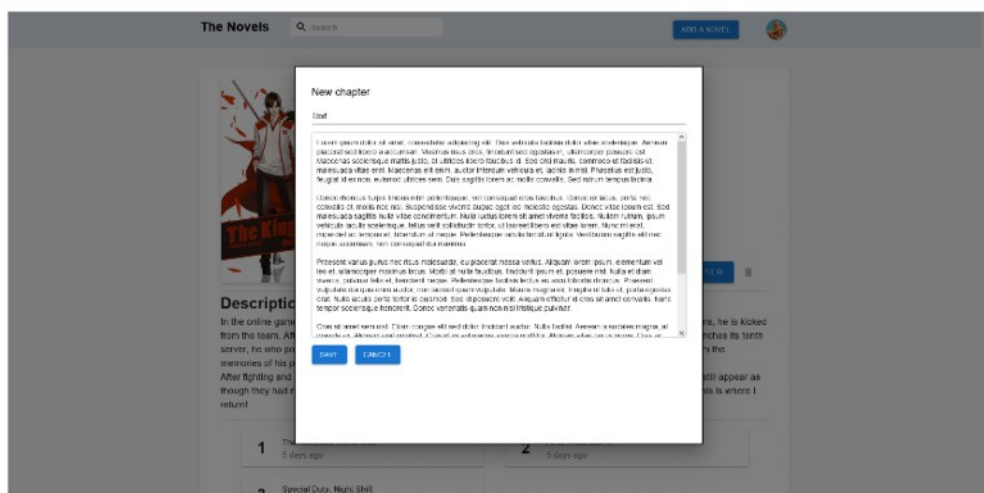
Екранні форми



13

Рисунок В.13 — Слайд № 13

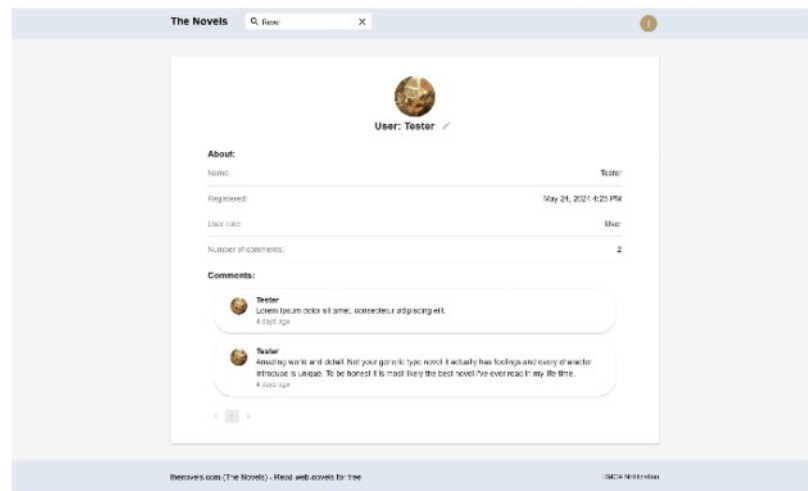
Екранні форми



14

Рисунок В.14 — Слайд № 14

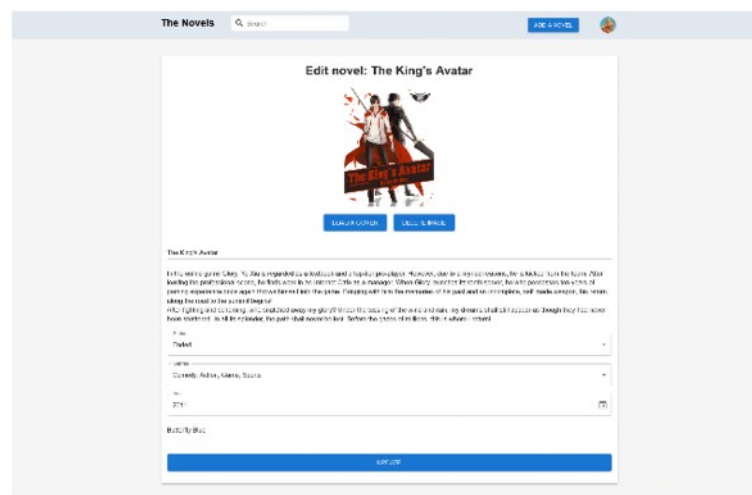
Екранні форми



15

Рисунок В.15 — Слайд № 15

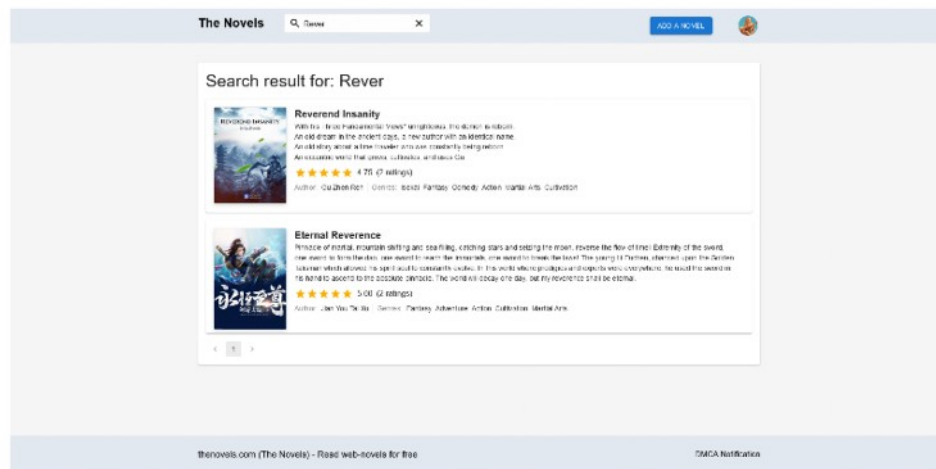
Екранні форми



16

Рисунок В.16 — Слайд № 16

Екранні форми



17

Рисунок В.17 — Слайд № 17

Висновки

У ході даної дипломної кваліфікаційної роботи було розроблено програмний продукт, який надає користувачам зручний інструмент для розповсюдження та читання вебновел. Проєкт був реалізований з використанням популярних фреймворків та технологій, таких як VS Code, JavaScript, NodeJS, Express, React, а також бази даних MongoDB.

18

Рисунок В.18 — Слайд № 18