

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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**До захисту допущено:
В.о. завідувача кафедри
Михайло НОВОТАРСЬКИЙ**

(підпис)

“__” _____ 2025 р.

**Дипломний проект
на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”**

на тему: Платформа для написання та публікації робіт для письменників

Виконав : студент 4 курсу, групи ІО-11
(шифр групи)

Денисенко Богдан Романович
(прізвище, ім’я, по батькові) _____
(підпис)

Керівник ас. каф., ОТ, Нікольський С.С.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____
(підпис)

Консультант (нормоконтроль) ас. каф., ОТ, Нікольський С.С.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) _____
(підпис)

Рецензент старший викладач Радченко К.О.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____
(підпис)

Засвідчую, що у цьому дипломному
проекті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025 р.

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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

Михайло НОВОТАРСЬКИЙ

(підпис)

“__” _____ 2025 р.

ЗАВДАННЯ

на бакалаврський дипломний проект студента

Денисенка Богдана Романовича

1. Тема проекту Платформа для написання та публікації робіт для письменників.

керівник проекту Нікольський Сергій Сергійович, ас. каф., ОТ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 23 травня 2025 року № 1705-с

2. Термін здачі студентом закінченого проекту 05 червня 2025 р.

3. Вихідні дані до проекту технічна документація, теоретичні дані.

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

Розділ 1. Огляд існуючих підходів платформ для публікації.

Розділ 2. Огляд та вибір технологій для розробки системи.

Розділ 3. Деталі розробки веб-додатка.

Розділ 4. Дослідження та аналіз розробленого веб-додатка.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Нікольський С.С.		

7. Дата видачі завдання «10» вересня 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проекту	Терміни виконання етапів проекту	Примітки
1.	<i>Затвердження теми проекту</i>	<i>10.11.2024-24.11.2024</i>	
2.	<i>Вивчення та аналіз завдання</i>	<i>25.11.2024-20.01.2025</i>	
3.	<i>Розробка архітектури та загальної структури системи</i>	<i>21.01.2025-10.02.2025</i>	
4.	<i>Розробка структур окремих підсистем</i>	<i>11.02.2025-22.02.2025</i>	
5.	<i>Програмна реалізація системи</i>	<i>23.02.2025-10.05.2025</i>	
6.	<i>Тестування розробленої системи</i>	<i>11.05.2025-18.05.2025</i>	
7.	<i>Оформлення пояснювальної записки</i>	<i>19.05.2025-30.05.2025</i>	
8.	<i>Передзахист</i>	<i>04.06.2025</i>	
9.	<i>Захист</i>	<i>18.06.2025</i>	

Студент-дипломник _____ Богдан ДЕНИСЕНКО
(підпис)

Керівник проекту _____ Сергій НІКОЛЬСЬКИЙ
(підпис)

АНОТАЦІЯ

У даній роботі був спроектований та розроблений веб-додаток, для публікації та написання робіт письменниками. Були проаналізовані існуючі рішення, та на основі цих рішень був реалізований веб-додаток. Був поставлений ряд питань, які вебсайт повинен вирішувати, що були враховані під час розробки. Для її виконання був підібраний стек технологій, що задовольняє технічні потреби і є оптимальним варіантом для реалізації проекту, та завдяки якому була реалізована працююча система. Програмний код серверної та клієнтської частин був написаний на мові програмування 'Typescript'.

Ключові слова: вебсайти, книги, електронні читачі, текстові редактори, публікації, Typescript, GraphQL.

ANNOTATION

In this work, a web application was designed and developed for writers to publish and write their works. Existing solutions were analyzed, and based on them, a web application was implemented. A number of questions that the website should address were formulated and taken into account during development. A technology stack was selected that meets the technical requirements and is an optimal choice for implementing the project, enabling the creation of a working system. The server-side and client-side code was written in the TypeScript programming language.

Keywords: websites, books, e-readers, text editors, publications, TypeScript, GraphQL.

сп ра вк и	Ф ор ма т	Значення			Найменування	Кіл. лист ів	№ екзе мпля ра	Додаток
					Документація загальна			
					Знову розроблена			
	A4	ІАЛЦ.467200.002 ТЗ			Платформа для написання та публікації робіт для письменників	4		
					Технічне завдання			
	A4	ІАЛЦ.467200.003 ПЗ			Платформа для написання та публікації робіт для письменників	83		
					Пояснювальна записка			
	A4	ІАЛЦ.467200.004 ДА			Платформа для написання та публікації робіт для письменників	1		
					Схема алгоритму роботи системи			
	A4	ІАЛЦ.4672008.005 ДБ			Платформа для написання та публікації робіт для письменників	1		
					Структурна схема веб-застосунку			
	A4	ІАЛЦ.467200.006 ДВ			Платформа для написання та публікації робіт для письменників	1		
					Схема бази даних			
	A4	ІАЛЦ.467200.007 ДГ			Платформа для написання та публікації робіт для письменників	114		
					Текст програмного коду			
					ІАЛЦ.467200.001 ОА			
Зм	Лист	№ докум.		Підп				
Розробив		Денисенко Б.Р.			Платформа для написання та публікації робіт для письменників Опис альбому	Літ.	Аркуш	Аркушів
Перевірів		Нікольський С.С.					1	1
Реценз		Радченко К.О.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Н. Контр.		Нікольський С.С.						
Затвердив								

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЕКТУ

на тему: «Платформа для написання та публікації робіт для письменників»

Київ – 2025

ЗМІСТ

1.	НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ	2
2.	ПІДСТАВИ ДЛЯ РОЗРОБКИ	2
3.	МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
4.	ДЖЕРЕЛА РОЗРОБКИ.....	3
5.	ТЕХНІЧНІ ВИМОГИ.....	3
	5.1. Вимоги до розробленого продукту	3
	5.2. Вимоги до програмного забезпечення.....	3
	5.3. Вимоги до апаратної частини	3
6.	ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ.467200.002 ТЗ			
		№ докум.	Підпис	Дата				
Розробив	Денисенко Б.Р.				Платформа для написання та публікації робіт для письменників Технічне завдання	Літ.	Аркуш	Аркушів
Перевірив	Нікольський С.С.						1	3
Реценз.	Радченко К.О.					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Н. Контр.	Нікольський С.С.							
Затвердив								

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку веб-додатку для створення, редагування та публікації літературних творів письменниками. У рамках завдання передбачено також подальшу підтримку, масштабування та вдосконалення розробленої системи.

Областю застосування цієї системи є літературна діяльність у цифровому середовищі: створення авторських текстів, публікація електронних книг, робота з текстовими редакторами у веб-середовищі, а також організація взаємодії між авторами та читачами. Така система є актуальною у зв'язку з активним розвитком цифрового книговидання, популяризацією самвидаву та потребою авторів у зручних онлайн-інструментах для роботи з текстом.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

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

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка веб-додатку для створення, редагування та публікації текстів письменниками. Система забезпечує базову функціональність для роботи з літературними творами у веб-середовищі.

					ІАЛЦ.467200.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		2

4. ДЖЕРЕЛА РОЗРОБКИ

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

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

Простий та інтуїтивний інтерфейс
Можливість написання літературних творів
Можливість публікації літературних творів
Можливість перегляду літературних творів

5.2. Вимоги до програмного забезпечення

ОС Linux.
NodeJS v22.5.1 і вище, npm v10.9.1 і вище.

5.3. Вимоги до апаратної частини

ЦП з показниками не менш ніж 1.4GHz BCM2837B0.
RAM не менш ніж 1GB.
ROM не менш ніж(збільшується з розширенням бази даних) 1GB.

					ІАЛЦ.467200.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		3

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.11.2024-24.11.2024
Вивчення та аналіз завдання	25.11.2024-20.01.2025
Розробка архітектури та загальної структури системи	21.01.2025-10.02.2025
Розробка структур окремих частин системи	11.02.2025-22.02.2025
Програмна реалізація системи	23.02.2025-10.05.2025
Виправлення помилок	11.05.2025-18.05.2025
Оформлення пояснювальної записки	19.05.2025-30.05.2025

					ІАЛЦ.467200.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		4

**ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЕКТУ**

на тему: «Платформа для написання та публікації робіт для письменників»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ	4
ВСТУП.....	5
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ПЛАТФОРМ ДЛЯ ПУБЛІКАЦІЇ	7
1.1. Підхід до користувачів як до читачів	7
1.2. Підхід до користувачів як до письменників	8
1.3. Комбінований підхід	9
ВИСНОВОК ДО РОЗДІЛУ 1	11
РОЗДІЛ 2. ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ.....	12
2.1. Поділ додатку на клієнтську та серверну частину	12
2.2. Клієнтська частина	13
2.3. Серверна частина	14
2.4. Комунікація між клієнтською та серверною частинами.....	15
ВИСНОВОК ДО РОЗДІЛУ 2.....	17
РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ ВЕБ-ДОДАТКА.....	19
3.1. Розробка архітектури веб-додатка	19
3.2. Реалізація шаблону проекту	25
3.3. Розробка системи авторизації та реєстрації	37
3.4. Розробка навігації	51
3.5. Розробка головної сторінки	53
3.6. Розробка сторінки огляду книги	56
3.7. Розробка редактора та переглядача глав	58
3.8. Розробка системи відгуків	63
ВИСНОВОК ДО РОЗДІЛУ 3.....	67
РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОГО ВЕБ-ДОДАТКА	69
4.1. Головна сторінка.....	69
4.2. Сторінки входу та реєстрації	69
4.3. Інтерактивні елементи та навігація	71
4.4. Створення та редакція книги	73
4.5. Система зворотнього зв'язку	77
4.6. Вихід з облікового запису	78

					ІАЛЦ.467200.003 ПЗ				
Зм.	Арк.	№ докум.	Підпис	Дата					
Розробив		Денисенко Б.Р.			Платформа для написання та публікації робіт для письменників	Літ.	Аркуш	Аркушів	
Перевірив		Нікольський С.С.					1	106	
Реценз.		Радченко К.О.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11			
Н. Контр.		Нікольський С.С.							
Затвердив									
					Пояснювальна записка				

ВИСНОВОК ДО РОЗДІЛУ 4.....79

ВИСНОВКИ.....81

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ82

ПЕРЕЛІК СКОРОЧЕНЬ

TS	(TypeScript) мова програмування TypeScript
JS	(JavaScript) мова програмування JavaScript
GQL	(GraphQL) мова запитів для API
NPM	(Node Package Manager) менеджер пакетів JavaScript
MUI	(MaterialUI) бібліотека компонентів
CUI	(ChakraUI) бібліотека компонентів
UI	(User Interface) інтерфейс користувача
REST	(Representational State Transfer) тип інтерфейсу комунікації
API	(Application Programming Interface) програмний інтерфейс

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		4

ВСТУП

Упродовж усього розвитку літератури питання самостійного видання власних творів залишалося актуальним для авторів. Зі стрімким розвитком сучасних інформаційних технологій відкрилися нові можливості для письменників у сфері створення, публікації та поширення текстів.

На сьогодні, завдяки соціальним мережам, електронним маркетплейсам та спеціалізованим веб-додаткам, автори можуть самостійно видавати свої роботи, оминаючи традиційні видавничі бар'єри. Це відкриває доступ до ширшої аудиторії, надає більше творчої свободи та дозволяє авторам зосередитися на написанні та розвитку власних творів. Така тенденція сприяє зростанню популярності цифрового книговидання та формує попит на зручні онлайн-інструменти для підтримки письменницької діяльності.

У даній лабораторній роботі було проаналізовано існуючі веб-платформи для публікації літературних творів, а також поставлено мету — спроектувати та розробити власне рішення, яке поєднує простий і інтуїтивно зрозумілий інтерфейс та надає користувачам можливість:

- створювати літературні твори;
- публікувати їх на платформі;
- переглядати вже опубліковані роботи інших авторів.

Окрім цього, було поставлено завдання реалізувати систему з урахуванням можливості подальшого масштабування та розширення функціональності. Архітектура проєкту має бути побудована таким чином, щоб

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		5

уникати обмежень монолітного підходу та забезпечувати гнучкість у додаванні нових модулів без порушення вже реалізованого функціоналу.

					ІАЛЦ.467200.003 ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ПЛАТФОРМ ДЛЯ ПУБЛІКАЦІЇ

У сучасному світі доступ до інформаційних мереж є невід’ємною складовою повсякденного життя. Найпоширенішим засобом взаємодії користувача з цими мережами є веб-браузер — універсальний інструмент, що забезпечує зручний інтерфейс для отримання, обробки та передачі даних у мережевому середовищі. Саме через веб-браузер здійснюється більшість цифрових дій: від перегляду новинних стрічок, споживання мультимедійного контенту — до виконання фінансових операцій, оформлення онлайн-покупок та взаємодії з державними чи корпоративними системами.

1.1. Підхід до користувачів як до читачів

У зв’язку з цим веб-технології набули особливого значення у процесі створення сучасних інформаційних систем. Фактично, майже кожен багатофункціональний пристрій - комп’ютер, планшет, смартфон, телевізор - оснащений веб-браузером, що робить веб-додатки максимально доступними та універсальними. Саме тому більшість сучасних сервісів, незалежно від їхнього призначення, реалізуються у вигляді веб-рішень, що дозволяють охопити широку аудиторію користувачів без необхідності встановлення додаткового програмного забезпечення.

На сьогодні автори мають широкий вибір сервісів для публікації своїх творів. Серед найбільш популярних рішень можна виділити такі платформи, як:

Kindle Direct Publishing (Amazon);

Apple Books;

Google Play Books.

Кожен із цих сервісів має свої переваги, недоліки та фокус, орієнтований на певну аудиторію або екосистему пристроїв. Зокрема, Google Play Books

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		7

пропонує широкі можливості для публікації та читання творів через кросплатформенні додатки, проте найбільшу зручність він забезпечує саме для користувачів пристроїв на базі Android. Apple Books, у свою чергу, є глибоко інтегрованим у власну екосистему Apple і не підтримує офіційного доступу до контенту поза межами цієї платформи, зокрема через веб-браузер. Kindle Direct Publishing від Amazon фокусується на власних пристроях Kindle, однак водночас забезпечує зручну підтримку для читання та зберігання книг як у своїх мобільних застосунках, так і безпосередньо через веб-інтерфейс.

Таким чином, лише Kindle та Google Play Books надають відносно незалежний від платформи доступ до контенту, тоді як Apple Books залишається обмеженим для користувачів за межами пристроїв Apple. Проте, незважаючи на ці відмінності, усі згадані сервіси зосереджуються переважно на публікації вже готових творів. Вони не передбачають інтегрованого середовища для створення літературного контенту, що ускладнює процес для авторів, які перебувають у процесі написання тексту. Письменники, як правило, завершують роботу над твором у зовнішньому середовищі, а вже потім завантажують готовий файл для публікації.

Хоча в рамках екосистем Google та Apple і існують текстові редактори, вони не пов'язані безпосередньо з процесом публікації, що розділяє робочий процес автора на окремі, часто розірвані етапи.

1.2. Підхід до користувачів як до письменників

Окрім цього, на ринку присутні сервіси, орієнтовані виключно на допомогу авторам у процесі публікації. Вони виконують роль посередника між письменником і згаданими вище платформами - наприклад, допомагають з редагуванням, форматуванням, публікацією, а в деяких випадках навіть із друком і маркетингом. Проте такий підхід має ознаки традиційного

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		8

видавничого процесу, який нерідко вимагає значних часових і ресурсних витрат, не завжди виправданих для нових або незалежних авторів.

Водночас для письменників, які вже мають сформовану читацьку аудиторію та достатній рівень популярності, такі сервіси можуть бути ефективним рішенням. Вони дозволяють делегувати публікаційний етап професіоналам, зосередившись виключно на творчій складовій.

1.3. Комбінований підхід

Однак вищезазначені сервіси не вичерпують усього спектру сучасних підходів до публікації літературного контенту. Окрім платформ, орієнтованих виключно на одну зі сторін процесу - або автора як постачальника контенту, або читача як його кінцевого споживача, - існує окремий клас рішень, що намагаються поєднати ці ролі в межах єдиної екосистеми. Такі платформи розглядають процес створення та споживання літературного продукту як взаємозалежні складові одного циклу, де автор і читач взаємодіють безпосередньо.

Подібна модель має багато спільного з принципами функціонування сучасних соціальних мереж. Автори мають можливість розпочати роботу над твором без завершеного тексту, поступово публікуючи окремі розділи чи фрагменти. У процесі написання вони можуть залучати аудиторію, яка не лише читає й оцінює публікації, а й активно коментує, надає зворотний зв'язок і тим самим впливає на подальший розвиток твору. Такий підхід дозволяє автору одразу відчувати реакцію потенційної читацької аудиторії, скоригувати стиль, сюжет або навіть структуру твору на основі живої взаємодії з читачем.

Для читачів же відкривається унікальна можливість бути не лише пасивними споживачами літературного продукту, а й активними учасниками творчого процесу. Це сприяє формуванню спільноти навколо твору або

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		9

конкретного автора, підвищує зацікавленість аудиторії та створює умови для довготривалої взаємодії.

У результаті подібна платформа перетворюється на динамічне середовище, де творчість поєднується з комунікацією, а сам процес написання стає публічним і соціально залученим. Цей підхід стимулює авторів залишатися в тісному контакті з читачем, своєчасно реагувати на запити аудиторії, адаптувати зміст, тематику чи подачу текстів. У підсумку це створює новий формат цифрової літературної взаємодії, де межа між творцем і споживачем стає дедалі умовнішою.

Слід зазначити, що у середовищі, де межа між автором і читачем є умовною, зростає зацікавленість користувачів у творчому самовираженні. Постійна взаємодія з авторами, можливість впливати на хід розвитку твору, а також відкритий доступ до процесу написання сприяють формуванню середовища, у якому читачі поступово набувають рішучості спробувати себе в ролі автора. Такий формат платформи не лише активізує читацьку аудиторію, а й стимулює появу нових літературних ініціатив без потреби звертатися до традиційних видавничих каналів, що робить отриману у результаті платформу здійну самопідживлюватися за рахунок цього процесу.

Самими популярними платформами з ознаками такої взаємодії є платформи Wattpad та Royal Road. Wattpad має дуже велику базу користувачів та сучасний інтерфейс. Royal Road має риси що схлоняються у бік більш традиційних онлайн форумів. Але у будь-якому підході окрім позитивних сторін будуть і негативні. У Wattpad є така проблема як неналежна модерація та відсутність можливості експорту робіт, що значно впливає на готовність користувачів використовувати їх вбудовані інструменти для письма. Royal Road навідміну від Wattpad дозволяє експортувати авторську роботу але має фокус на жанри фентезі, наукової фантастики та літРПГ.

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		10

ВИСНОВОК ДО РОЗДІЛУ 1

У ході дослідження були розглянуті різні підходи до побудови сучасних платформ для публікації літературних творів. На основі аналізу наявних рішень було виокремлено три основні типи платформ: сервіси, що орієнтовані переважно на читача як кінцевого споживача контенту; платформи, які фокусуються на авторах та процесі публікації творів; а також комбіновані рішення, які поєднують можливості для створення, публікації та споживання літературного контенту в єдиному середовищі.

Кожен із цих підходів має як свої переваги, так і недоліки. Наприклад, сервіси з орієнтацією на читача (такі як Google Play Books або Apple Books) пропонують зручний доступ до літератури, проте практично не надають інструментів для створення чи публікації творів безпосередньо на платформі. Водночас платформи, зосереджені на процесі публікації, надають авторам інструменти для розповсюдження своїх творів, але часто не передбачають можливості безпосереднього створення тексту або тісної взаємодії з аудиторією.

Особливу увагу було приділено платформам змішаного типу, які дозволяють автору не лише публікувати, а й створювати твори безпосередньо на сайті, водночас забезпечуючи функціонал для залучення аудиторії. Такі платформи як Wattpad або Royal Road стимулюють зворотний зв'язок між автором і читачем у реальному часі, формуючи активні творчі спільноти.

На основі аналізу виявлено, що платформи змішаного типу мають найбільший потенціал для розвитку в умовах цифрової епохи, оскільки створюють максимально гнучке середовище як для письменників-початківців, так і для досвідчених авторів. Саме такий підхід було обрано як базову концепцію для подальшої реалізації веб-додатку, розробленого в межах даної роботи.

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		11

РОЗДІЛ 2. ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

Для успішного створення веб-додатку, який буде задовольняти поставлені перед ним задачі, необхідно ретельно підібрати набір технологій, що дозволить реалізувати гнучку, масштабовану та надійну систему. Одним із ключових завдань є забезпечення можливості подальшого розширення функціональності без порушення вже реалізованого функціоналу, а також уникнення обмежень, які притаманні монолітним архітектурним підходам. Система повинна залишатися гнучкою у плані оновлення, легкою у підтримці та готовою до змін залежно від потреб проєкту.

2.1. Поділ додатку на клієнтську та серверну частину

Для веб-додатків типовим є поділ на дві частини: клієнтську та серверну. Клієнтська частина відповідає за взаємодію з користувачем, тому важливо забезпечити сучасний, зручний та інтуїтивно зрозумілий інтерфейс. У той час як серверна частина має обробляти запити, відповідати за логіку додатку, зберігання та обробку даних.

З метою уніфікації підходу до розробки та спрощення обміну між клієнтом і сервером була обрана мова програмування TypeScript, яка забезпечує переваги над звичайним JavaScript завдяки підтримці статичної типізації. Це значно полегшує розробку, підвищує стабільність та дозволяє уникнути багатьох помилок на етапі написання коду. Крім того, TypeScript сприяє кращій структурованості проєкту, особливо при розробці великих або розширюваних систем.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		12

2.2. Клієнтська частина

Для реалізації клієнтської частини було обрано фреймворк React, який дозволяє створювати динамічний інтерфейс, використовуючи компонентний підхід. Це надає можливість повторного використання окремих елементів інтерфейсу та полегшує розширення функціоналу у майбутньому. React також добре інтегрується з TypeScript, що дозволяє досягти ще більшої надійності при розробці.

Особливо важливим етапом проектування клієнтської частини веб-додатка є організація маршрутизації сторінок. Обраний підхід до реалізації раутингу безпосередньо впливає на архітектуру програми, її підтримуваність, масштабованість та зручність розробки. У рамках даної роботи було розглянуто два основних підходи: класичну маршрутизацію за допомогою бібліотеки React Router, а також ‘file-based’ маршрутизацію, реалізовану у TanStack Router. Перший підхід передбачає явне визначення маршрутів у коді, тоді як другий дозволяє створювати маршрути автоматично на основі файлової структури проєкту, що наближує його до моделі, яку використовують сучасні фреймворки, такі як Next.js. З метою покращення структурованості коду, зменшення ймовірності помилок під час навігації та загального спрощення процесу розробки було прийнято рішення на користь TanStack Router. Цей підхід забезпечує більш декларативний стиль роботи з маршрутами, спрощує масштабування додатка та покращує читабельність структури проєкту.

Для полегшення та пришвидшення розробки клієнтської частини веб-додатка доцільним є використання готових бібліотек інтерфейсних компонентів. Серед можливих варіантів були розглянуті Material UI (MUI) та Chakra UI. Обидві бібліотеки мають зрілу екосистему та підтримують широкі можливості з темізації й кастомізації інтерфейсу. Material UI має довший цикл розвитку, більше готових компонентів та є кращим вибором для більш формального дизайну. Проте при глибокій кастомізації можуть виникати

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		13

труднощі, оскільки компоненти MUI часто мають складну внутрішню структуру.

Chakra UI у цьому плані більш гнучка та легша в налаштуванні. Також Chakra підтримує динамічне завантаження компонентів, що зменшує розмір бандлу. Однак недоліком Chakra UI є те, що вона змінює стилі базових HTML-елементів, через що можуть виникати проблеми з відображенням сторонніх компонентів, наприклад, EditorJS. У такому випадку для коректного відображення можна використати компонент Prose, який частково вирішує цю проблему за рахунок базових стилів для контенту.

У результаті порівняння було обрано бібліотеку Chakra UI, оскільки вона забезпечує гнучкіший підхід до створення інтерфейсу, дозволяє швидко реалізовувати компоненти з високим рівнем кастомізації та має просту і зрозумілу структуру.

2.3. Серверна частина

Враховуючи вибір TypeScript для обох частин додатку, логічним рішенням для серверної частини стало використання Node.js. Це середовище дозволяє писати серверний код на тій самій мові, що й клієнтський, що в свою чергу спрощує обмін даними, використання загальних типів та моделей, а також зменшує технічні бар'єри між різними частинами додатку.

Для зберігання даних у розроблюваному веб-додатку доцільним є використання реляційної бази даних, оскільки система передбачає велику кількість зв'язків між внутрішніми елементами даних. У якості бази даних у для проекту є доцільним використання MariaDB. Вона забезпечує необхідну функціональність для роботи зі складною структурою даних, підтримує міжтабличні зв'язки та легко масштабується.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		14

2.4. Комунікація між клієнтською та серверною частинами

Однією з ключових складових веб-додатку є організація взаємодії між клієнтською та серверною частинами. Від обраного методу комунікації залежить як ефективність обміну даними, так і можливості масштабування системи. У межах даного проєкту було розглянуто два підходи: REST API та GraphQL.

REST API базується на принципі створення окремих кінцевих точок (endpoints) для кожної операції. Цей підхід є класичним та добре підтримуваним, однак має низку обмежень. Зокрема, REST не передбачає вбудованої типізації, що потребує ручного опису типів отриманих і надісланих об'єктів, підвищуючи ризик помилок. Крім того, реалізація підтримки реального часу потребує додаткових рішень, оскільки REST не має вбудованої підтримки WebSocket.

GraphQL, навпаки, використовує єдиний endpoint для всіх запитів, забезпечуючи гнучкий механізм взаємодії через чітко типізовані запити та відповіді. Основні операції GraphQL поділяються на query (запит даних), mutation (зміна даних) та subscription (отримання даних у режимі реального часу). Такий підхід значно полегшує клієнтську розробку та мінімізує надмірність переданих даних. Водночас GraphQL має низку обмежень, зокрема ускладнене створення зовнішніх API для сторонніх розробників та менш прозору реалізацію авторизації на етапі ініціалізації сесії.

З урахуванням переваг і недоліків обох підходів, було прийнято рішення використовувати комбіновану архітектуру: REST API застосовується для реалізації зовнішніх інтерфейсів та процесів авторизації, у той час як GraphQL використовується для основної взаємодії між клієнтом і сервером усередині системи. Такий підхід дозволяє поєднати сильну типізацію, гнучкість запитів та ефективність обробки даних у реальному часі.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		15

Окремо слід зазначити, що доцільним є використання шаблону ViteExpress, що дозволить поєднати клієнт та сервер у один монорепозіторій. Зокрема, це також дозволить нам знехтувати налаштуванням Express JS (що запроваджує REST API), та сфокусуватися на інтеграції GraphQL.

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		16

ВИСНОВОК ДО РОЗДІЛУ 2

У результаті аналізу технічних вимог і функціональних особливостей майбутнього веб-додатку, з метою забезпечення ефективної розробки, масштабованості, розширюваності та підтримки високої якості користувацького досвіду, було сформовано набір сучасних технологій. Обрані інструменти дозволяють створити надійну архітектуру, що відповідає принципам модульності та поділу відповідальностей, забезпечуючи чіткий розподіл між клієнтською та серверною частинами системи.

Архітектура додатку умовно поділяється на дві частини.

Клієнтська частина (Frontend), яка відповідає за взаємодію з користувачем, інтерфейс та логіку представлення:

- React – основна бібліотека для створення інтерфейсу користувача, яка забезпечує компонентну структуру та високу продуктивність;
- TanStack Router – інструмент маршрутизації, що дозволяє організувати структуру сторінок відповідно до файлової системи проєкту, спрощуючи навігацію та підтримку;
- ChakraUI – бібліотека UI-компонентів, яка забезпечує швидку розробку адаптивного та естетичного інтерфейсу з можливістю гнучкого стилізування;
- Apollo Client – клієнтська бібліотека для взаємодії з GraphQL API, яка дозволяє ефективно кешувати дані, управляти станом запитів і реалізувати реактивну модель обміну інформацією.

Серверна частина (Backend), яка відповідає за логіку обробки запитів, зберігання даних та бізнес-логіку:

- Node.js – платформа, яка дозволяє виконувати JavaScript на стороні сервера, забезпечуючи асинхронну обробку запитів і високу масштабованість;

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		17

- Express.js – легкий фреймворк для побудови REST API, що використовується для реалізації частини функціоналу, зокрема процесів авторизації та створення зовнішніх інтерфейсів;
- Apollo Server – серверна частина для обробки GraphQL-запитів, яка забезпечує зручну інтеграцію з клієнтом, типізацію даних і підтримку підписок у режимі реального часу.
- MariaDB - реляційна база даних з відкритим кодом.
- ViteExpress – шаблон проекту, імплементує REST та дозволяє скористуватися перевагами Vite JS

Для обох частин було також обрано використання мови TypeScript, для чіткої типізації із метою зменшення вирогідності людської помилки.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		18

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ ВЕБ-ДОДАТКА

З обраними технологіями та визначеним підходом до реалізації можна переходити безпосередньо до розробки програмної частини проєкту. З метою створення масштабованої системи, здатної виконувати поставлені задачі — зокрема, написання, публікація та перегляд створених робіт — реалізацію буде здійснено шляхом побудови клієнтської та серверної частин. Обидві частини, у свою чергу, будуть поділені за блочною структурою на окремі модулі, кожен з яких відповідатиме за конкретний функціонал. У сукупності вони утворюють цілісний веб-додаток.

3.1. Розробка архітектури веб-додатка

Перед початком розробки необхідно виконати проєктування системи. Основні функціональні вимоги до веб-застосунку включають: створення і редагування текстових робіт, зберігання структурованого контенту, пошук, перегляд опублікованих матеріалів, а також можливість залишати відгуки. Таким чином, можна відокремити чотири ключові функціональні аспекти, які формують основу архітектури системи (рис. 3.1).

Виходячи з цього, формується модель даних, яка забезпечить реалізацію необхідного функціоналу.

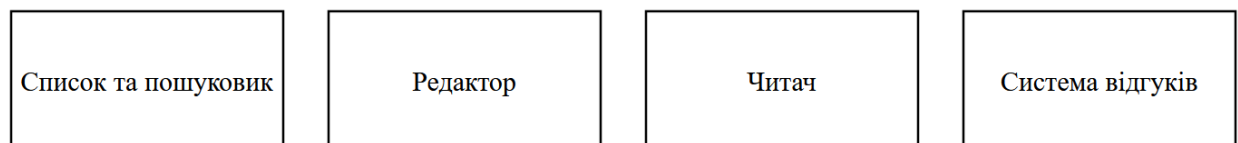


Рисунок 3.1 – Ключові аспекти системи

Основним суб'єктом взаємодії із системою є користувач (рис. 3.2).

Основні атрибути сутності User:

- email – адреса електронної пошти, що є унікальним ідентифікатором користувача та може використовуватись для відновлення доступу до облікового запису;
- username – псевдонім користувача;
- password – збережений хеш пароллю, що використовується для аутентифікації;
- refreshToken – токен, що забезпечує оновлення сесії авторизації;
- description – додаткове, необов'язкове текстове поле для опису профілю;
- isDarkMode – вказує на обрану тему інтерфейсу;
- books – список об'єктів типу Book, які були створені користувачем;
- reviews – відгуки, залишені користувачем на книги чи окремі розділи.

```
10 model User {  
11   id String @id  
12  
13   createdAt DateTime @default(now())  
14   updatedAt DateTime @updatedAt  
15  
16   email String @unique @db.VarChar(255)  
17   username String @unique @db.VarChar(255)  
18   password String  
19  
20   refreshToken String? @unique @db.VarChar(255)  
21  
22   description String? @db.VarChar(500)  
23  
24   isDarkMode Boolean @default(true)  
25  
26   books Book[]  
27  
28   reviews Review[]  
29   ratings Rating[]  
30 }
```

Рисунок 3.2 – Модель користувача

Користувач може взаємодіяти з книгами як у ролі автора, так і як читач – залишаючи відгуки або оцінки. Кожна книга є окремою сутністю, що відповідає за зберігання інформації про зміст, структуру та взаємодії з користувачами,

вона містить посилання на відповідні глави, автора, а також зберігає зібрані відгуки та оцінки читачів. Крім того, для підвищення продуктивності системи доцільно зберігати агреговану інформацію, таку як загальна кількість слів і глав, що дозволяє зменшити кількість звернень до пов'язаних таблиць у базі даних (рис. 3.3).

Основні атрибути моделі Book:

- title – назва книги;
- progressStatus – статус написання (“у процесі”, “заморожено”, “завершено”); дозволяє читачеві орієнтуватися в актуальному стані твору;
- author – зв’язок з автором книги, реалізований через зовнішній ключ на модель User;
- wordsTotal – збережена загальна кількість слів у книзі;
- chaptersTotal – кількість глав, що входять до складу книги;
- chapters – зв’язок із відповідними главами (BookChapter);
- reviews – відгуки користувачів до всієї книги;
- ratings – оцінки книги, поставлені користувачами.

```

32 enum WorkingProgressStatus {
33     inProgress
34     frozen
35     complete
36 }
37
38 model Book {
39     id String @id
40
41     createdAt DateTime @default(now())
42     updatedAt DateTime @updatedAt
43
44     title          String          @db.VarChar(255)
45     description    String          @db.MediumText
46     progressStatus WorkingProgressStatus @default(inProgress)
47
48     author User @relation(fields: [authorId], references: [id])
49     authorId String
50
51     wordsTotal    Int @default(0)
52     chaptersTotal Int @default(0)
53
54     chapters BookChapter[]
55
56     reviews Review[]
57     ratings Rating[]
58 }

```

Рисунок 3.3 – Модель книги

Глави книги моделюються за допомогою сутності BookChapter. Подібно до основної моделі книги, кожна глава містить інформацію про зміст, а також агреговані метрики, зокрема кількість слів. Це дозволяє оптимізувати запити до бази даних та забезпечити швидкий доступ до статистичних даних. Глави жорстко прив'язані до конкретної книги, і в межах однієї книги мають унікальну послідовність (рис. 3.4).

```

60 model BookChapter {
61   id String @id
62
63   createdAt DateTime @default(now())
64   updatedAt DateTime @updatedAt
65
66   sequenceNumber Int
67
68   title String @db.VarChar(255)
69
70   foreword String? @db.MediumText
71   footnote String? @db.MediumText
72
73   content Json @db.Json
74
75   wordsTotal Int @default(0)
76
77   book Book @relation(fields: [bookId], references: [id], onDelete: Cascade)
78   bookId String @db.VarChar(255)
79
80   reviews Review[]
81   ratings Rating[]
82
83   @@unique([bookId, sequenceNumber])
84 }

```

Рисунок 3.4 – Модель BookChapter

- Основні атрибути моделі BookChapter:
- sequenceNumber — унікальний номер глави в межах книги, визначає порядок її розміщення;
- title – назва глави;
- foreword – опціональна передмова до глави;
- footnote – опціональна виноска або післяслово;
- content – основний текст глави, збережений у форматі JSON для підтримки структурованого контенту;
- wordsTotal – кількість слів у межах глави;
- book – зв'язок з батьківською книгою (Book);
- bookId – зовнішній ключ, що ідентифікує книгу;
- reviews – відгуки користувачів безпосередньо до окремої глави;

- ratings – оцінки глави, що дозволяють деталізувати вподобання читачів на рівні окремих частин книги.

Для реалізації можливості залишення відгуків до книг або окремих глав було створено модель Review (рис. 3.5). Дана модель дозволяє користувачам взаємодіяти з контентом, залишаючи текстові коментарі, які можуть бути пов'язані або з усією книгою, або з конкретною главою. Цей підхід забезпечує гнучкість у зворотному зв'язку та сприяє розвитку спільноти навколо окремих творів.

Основні атрибути моделі Review:

- text – вміст відгуку, представлений у вигляді текстового поля;
- user – зв'язок з користувачем, який залишив відгук;
- userId – зовнішній ключ, що ідентифікує автора відгуку;
- bookChapter – опціональний зв'язок із конкретною главою, якщо відгук адресовано їй;
- bookChapterId – ідентифікатор глави, до якої відноситься відгук;
- book – опціональний зв'язок з книгою, якщо відгук стосується книги загалом;
- bookId – ідентифікатор відповідної книги.

```

86 model Review {
87   id String @id
88
89   createdAt DateTime @default(now())
90   updatedAt DateTime @updatedAt
91
92   text String @db.MediumText
93
94   user User @relation(fields: [userId], references: [id])
95   userId String @db.VarChar(255)
96
97   bookChapter BookChapter? @relation(fields: [bookChapterId], references: [id], onDelete: Cascade)
98   bookChapterId String? @db.VarChar(255)
99
100  book Book? @relation(fields: [bookId], references: [id], onDelete: Cascade)
101  bookId String?
102 }

```

Рисунок 3.5 – Модель Review

Для реалізації системи оцінювання контенту було створено модель Rating (рис. 3.6). Вона дозволяє користувачам висловлювати свою думку шляхом

виставлення позитивної або негативної оцінки як книзі загалом, так і окремим її главам. Оцінки фіксуються у вигляді записів, що згодом агрегуються у вигляді загальної суми голосів.

Атрибути моделі Rating:

- type – тип оцінки: позитивна або негативна. Значення впливає на сумарний рейтинг об'єкта;
- user – зв'язок із користувачем, що залишив оцінку;
- userId – ідентифікатор користувача, який здійснив голосування;
- bookChapter – опціональний зв'язок із главою, якщо оцінка стосується конкретної частини книги;
- bookChapterId – ідентифікатор відповідної глави;
- book – опціональний зв'язок із книгою, якщо оцінка надана на рівні всієї роботи;
- bookId – ідентифікатор відповідної книги.

```

104 enum RatingType {
105     up
106     down
107 }
108
109 model Rating {
110     id String @id
111
112     createdAt DateTime @default(now())
113     updatedAt DateTime @updatedAt
114
115     type RatingType
116
117     user User @relation(fields: [userId], references: [id])
118     userId String @db.VarChar(255)
119
120     bookChapter BookChapter? @relation(fields: [bookChapterId], references: [id])
121     bookChapterId String? @db.VarChar(255)
122
123     book Book? @relation(fields: [bookId], references: [id])
124     bookId String?
125
126     @@unique([userId, bookId])
127     @@unique([userId, bookChapterId])
128 }

```

Рисунок 3.6 – Модель Rating

Окремо слід зазначити, що були також виведені типи enum RatingType та WorkingProgressStatus. RatingType – відображає напрямлення рейтингу, позитивне або негативне. WorkingProgressStatus – є відображенням стану написання проекту. Стан може бути “у прогресі”, “заморожений”, “закінчений”.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		24

3.2. Реалізація шаблону проекту

Створивши модель даних та зазначивши ключові елементи, можемо приступити до ініціалізації проекту. Для цього викликаємо команду:

npm create vite-express

У створеній директорії розпочинається подальше налаштування середовища. Для спрощення управління конфігураційними параметрами проекту доцільним є створення окремої функції для обробки файлу змінних середовища `.env`. У середовищі Vite даний файл за замовчуванням підтримується для зчитування змінних, які використовуються під час виконання як клієнтської, так і серверної частин застосунку. Файл `.env` міститиме конфіденційну інформацію, зокрема IP-адреси серверів, мережеві порти, секрети для авторизаційних токенів та URL-адресу підключення до бази даних. Саме тому він обов'язково включається до `.gitignore`, щоб уникнути випадкового потрапляння до публічних репозиторіїв. Приклад конфігураційного файлу представлено на рисунку 3.7. Далі розглянемо функцію обробки конфігурації `toAppConfig` (рис. 3.8).



```
1 VITE_MODE="dev"
2 VITE_HOST=192.168.1.2
3 VITE_HOST_REMOTE=123.123.123.123
4 VITE_DOMAIN_NAME=n0q.dev
5 VITE_PORT=443
6 VITE_APP_NAME="echoes"
7 ACCESS_TOKEN_SECRET="encoded string"
8 REFRESH_TOKEN_SECRET="encoded string"
9 DATABASE_URL="mysql://user@localhost:3306/project_in"
```

Рисунок 3.7 – Приклад файлу середовища

```

export function toAppConfig(): AppConfig {
  const MODE = process.env.VITE_MODE || 'prod';
  const isProd = MODE === 'prod';

  const host = process.env.VITE_HOST || 'localhost';
  const port = parseInt(process.env.VITE_PORT || '3000');

  const hostRemote = process.env.VITE_HOST_REMOTE || 'localhost';
  const domainName = process.env.VITE_DOMAIN_NAME || 'localhost';

  const urlIp = `http://${hostRemote}:${port}`;
  const urlDomain = `http://${domainName}:${port}`;
  const urlRemote = `https://${host}:${port}`;
  const urlLocal = host !== 'localhost' ? `https://localhost:${port}` : undefined;

  const limit = { limit: '50mb' };

  const origin = urlLocal !== null ? [urlLocal, urlRemote, urlIp, urlDomain] : [urlRemote];
  const corsOptions: cors.CorsOptions = {
    origin,
    credentials: true
  };

  const accessTokenExpiresIn = 24 * 60 * 60;
  const refreshTokenExpiresIn = 15 * 24 * 60 * 60;

  const accessTokenSecret = process.env.ACCESS_TOKEN_SECRET;
  const refreshTokenSecret = process.env.REFRESH_TOKEN_SECRET;
  if (accessTokenSecret == null || refreshTokenSecret == null) throw new Error('No secrets found');

  const accessToken: TokenConfig = {
    secret: accessTokenSecret,
    expiresIn: accessTokenExpiresIn
  };

  const refreshToken: RefreshTokenConfig = {
    secret: refreshTokenSecret,
    expiresIn: refreshTokenExpiresIn,
    maxAge: refreshTokenExpiresIn * 1000
  };

  const greeting = `
  \n\n\x1b[35m-server is listening at...\x1b[0m\n\n${urlRemote}\n${urlLocal}\n\n`;

  return {
    isProd,
    host,
    port,
    limit,
    origin,
    corsOptions,
    accessToken,
    refreshToken,
    greeting
  };
}

```

Рисунок 3.8 – Реалізація “toAppConfig”

Оскільки значення змінних середовища, що зчитуються з файлу .env, не мають суворої типізації та вводяться вручну, у функції передбачена перевірка їх наявності. У випадку відсутності значень – задаються дефолтні параметри. Отримані дані використовуються для побудови конфігураційного об’єкта, який надалі застосовується у ініціалізації серверної частини. Деякі параметри, зокрема limit (обмеження розміру запитів) та accessTokenExpiresIn (час життя токенів доступу), визначаються безпосередньо у тілі функції – це спрощує структуру та підвищує зрозумілість коду.

Після обробки всі параметри зводяться у конфігураційний об’єкт, який повертається з функції й використовується як єдине джерело налаштувань застосунку. Структура вихідної конфігурації представлена на рисунку 3.9.

					ІАЛП.467200.003 ПЗ	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

```
interface AppConfig {
  readonly isProd: boolean;
  readonly host: string;
  readonly port: number;
  readonly limit: { readonly limit: string };
  readonly origin: string[];
  readonly corsOptions: cors.CorsOptions;
  readonly greeting: string;
  readonly accessToken: TokenConfig;
  readonly refreshToken: RefreshTokenConfig;
}
```

Рисунок 3.9 – Вихідний тип конфігурації

Далі потрібно зробити початкову конфігурацію файлу tsconfig, це дозволить нам організувати файли проекту більше ефективно (рис. 3.10).

```
1 {
2   "compilerOptions": {
3     "target": "ESNext",
4     "useDefineForClassFields": true,
5     "lib": ["DOM", "DOM.Iterable", "ESNext"],
6     "allowJs": false,
7     "jsx": "react-jsx",
8     "skipLibCheck": true,
9     "esModuleInterop": true,
10    "allowSyntheticDefaultImports": true,
11    "strict": true,
12    "forceConsistentCasingInFileNames": true,
13    "module": "ESNext",
14    "moduleResolution": "Bundler",
15    "resolveJsonModule": true,
16    "isolatedModules": true,
17    "noEmit": true,
18    "paths": {
19      "@app/*": ["../app/*"],
20      "@client/*": ["../app/client/*"],
21      "@server/*": ["../app/server/*"],
22      "@generated/*": ["../__generated__/*"],
23      "@root/*": ["./*"]
24    }
25  },
26  "exclude": ["../app/client/assets", "../__generated__/*", "../dist"]
27 }
28
```

Рисунок 3.10 – Файл tsconfig

Додаток тепер є розподіленим на клієнт та сервер та ігнорує шляхи, що є згенерованими генератором типів, чи результатами транспіляції, дозволяючи нам скористуватись утілітою перевірки типів від TypeScript.

Для запуску серверної частини проєкту, основний файл якої знаходиться у файлі main.ts, використовується комбінація Express.js та Apollo Server. У даній

архітектурі Apollo Server інтегрується як middleware до Express JS, що забезпечує безконфліктну взаємодію між цими двома компонентами.

Для забезпечення захищеного з'єднання використовується протокол HTTPS, з відповідними сертифікатами, які попередньо створюються за допомогою утиліти Certbot. Це дозволяє підвищити рівень безпеки під час обміну даними між клієнтом і сервером.

Ініціалізація Apollo Server включає імпорт схем typeDefs та resolvers, що реалізують обробку GraphQL-запитів і мутацій. Після цього налаштовується сервер Express, до якого підключаються всі необхідні middleware, зокрема:

- CORS – відповідає за міждоменно-допустимий обмін ресурсами (Cross-Origin Resource Sharing), дозволяючи або обмежуючи доступ до API з інших джерел;
- JSON-парсер – забезпечує обробку JSON-об'єктів, що передаються у тілі HTTP-запитів;
- Cookie-parser – дозволяє зчитувати дані з cookie-клієнта;
- expressMiddleware – спеціальний модуль Apollo Server, який підключається як middleware та отримує контекстні дані для кожного запиту, зокрема дані аутентифікації.

Окремим маршрутом виступає refreshToken – ендпоінт, що відповідає за оновлення сесії авторизованого користувача. Це дозволяє зберігати тривалу аутентифікацію без потреби повторного введення облікових даних. Файл наведений на рис. 3.11.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		28

```

18 export const config = toAppConfig();
19
20 const __dirname = import.meta.dirname;
21 const key = fs.readFileSync(path.join(__dirname + '../../certs/selfsigned.key'));
22 const cert = fs.readFileSync(path.join(__dirname + '../../certs/selfsigned.crt'));
23
24 const app = express();
25 const server = https.createServer({ key, cert }, app);
26
27 const typeDefs = fs.readFileSync(path.join(__dirname, 'graphql', 'schema.graphql'), 'utf8');
28
29 const resolvers = {
30   JSON: GraphQLJSON,
31   Query,
32   Mutation
33   // Subscription,
34 };
35
36 const apollo = new ApolloServer<IdentityContext>({
37   typeDefs,
38   resolvers,
39   introspection: !config.isProd,
40   plugins: [ApolloServerPluginDrainHttpServer({ httpServer: server })],
41   formatError: formattedError => {
42     process.stdout.write('\n```\n');
43     console.error(formattedError);
44     process.stdout.write('\n```\n');
45     return formattedError;
46   }
47 });
48
49 await apollo.start();
50
51 app.use(cors<cors.CorsRequest>(config.corsOptions));
52 app.use(express.json(config.limit));
53 app.use(cookie());
54 app.get('/api/refreshAccessToken', refreshTokenController);
55 app.use(
56   '/graphql',
57   expressMiddleware(apollo, {
58     context: getIdentityContext
59   })
60 );
61
62 await new Promise<void>(resolve =>
63   server.listen({ hostname: config.host, port: config.port }, resolve)
64 );
65
66 process.stdout.write(config.greeting);
67
68 ViteExpress.bind(app, server);

```

Рисунок 3.11 – Файл main.ts

Серверна частина розташована у папці server (рис. 3.12) та організована за модульним принципом з урахуванням логічного поділу на функціональні блоки. Така структура полегшує підтримку, масштабування та розширення проєкту. Нижче наведено опис основних компонентів:

- express – каталог, який містить логіку, пов’язану з REST-інтерфейсами та взаємодією з HTTP-запитами:
 - auth – модуль аутентифікації, зокрема обробка токенів авторизації (наприклад, refreshTokenController).
- graphql – каталог, що відповідає за реалізацію GraphQL-сервісу:

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		29

- `resolvers` – містить файли реалізації *резолверів* (наприклад, `graphql.ts`), які обробляють запити, визначені в схемі.
- `schema.graphql` – файл, що містить декларацію GraphQL-схеми (типи, запити, мутації), яку використовує Apollo Server.
- `prisma` – каталог для роботи з ORM-бібліотекою Prisma, що забезпечує взаємодію з базою даних. Містить файли генерації типів та моделі на основі Prisma Schema (не відображено на зображенні, але логічно пов'язаний із сервісами).
- `service` – каталог, де знаходяться функції ексклюзивні для відповідних компонентів проекту:
 - `book.ts` — містить функції для обробки логіки, пов'язаної з книгами.
- `config.ts` – файл, у якому реалізується логіка обробки змінних середовища (`.env`) через функцію `toAppConfig`, формуючи конфігурацію серверу.
- `helpers.ts` – утилітарні функції, які підтримують загальну логіку проекту, використовуються у декількох модулях.
- `identity.ts` – реалізація механізму контексту ідентифікації користувача для GraphQL (наприклад, витяг даних користувача з запиту та їх подальша передача у резолвери).
- `main.ts` – основний вхідний файл серверної частини. Ініціалізує Express.js, Apollo Server, middleware, HTTPS-сервер, підключає маршрути, та запускає застосунок.
- `common.ts` – спільні типи, константи або утиліти, які використовуються у обох, клієнта та сервера.

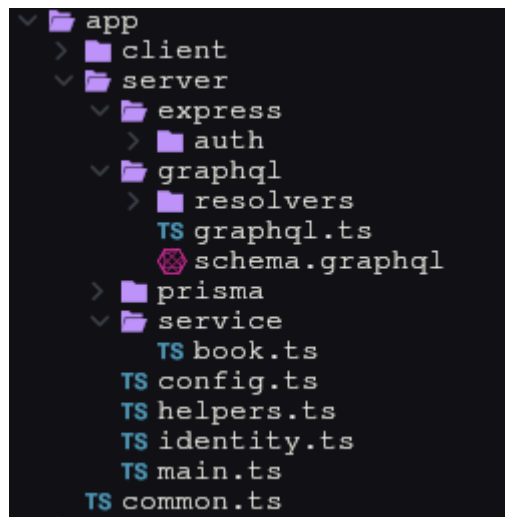


Рисунок 3.12 – Файлова структура серверної частини

З клієнтського боку необхідно здійснити налаштування та підключення GraphQL, Chakra UI, а також реалізувати маршрутизацію за допомогою TanStack Router. Для конфігурації клієнтського середовища замість використання toAppConfig застосовується стандартна обробка файлу .env. Це зумовлено тим, що, на відміну від серверної частини, клієнтська не повинна мати доступу до всіх змінних середовища. Зокрема, інформація про відкриті порти, адресу розгортання чи спосіб запуску додатку може містити чутливі дані, які з міркувань безпеки доцільно обмежити лише серверною частиною.

Завдяки використанню інструменту Vite (рис. 3.13), реалізація такого підходу стає можливою завдяки використанню префіксу VITE_ для змінних середовища. За замовчуванням, конфігурація Vite дозволяє передавати у клієнтське середовище лише ті змінні, що позначені відповідним префіксом, зберігаючи конфіденційність усіх інших.

```
1 VITE_MODE="dev"
2 VITE_HOST=192.168.1.2
3 VITE_HOST_REMOTE=123.123.123.123
4 VITE_DOMAIN_NAME=n0q.dev
5 VITE_PORT=443
6 VITE_APP_NAME="echoes"
7 ACCESS_TOKEN_SECRET="encoded string"
8 REFRESH_TOKEN_SECRET="encoded string"
9 DATABASE_URL="mysql://user@localhost:3306/project_ln"
```

Рисунок 3.13 – Приклад використання префіксу

Далі, аналогічно до серверної частини, ми можемо отримати змінні середовища, перевірити їх на наявність та, у разі відсутності, призначити значення за замовчуванням (рис. 3.14).

```
35 const mode = import.meta.env.VITE_MODE || 'prod';
36 const isProd = mode === 'prod';
37
38 export const appName = import.meta.env.VITE_APP_NAME || 'project-1n';
39
40 const host = import.meta.env.VITE_DOMAIN_NAME || import.meta.env.VITE_HOST || 'localhost';
41 const port = import.meta.env.VITE_PORT || '3000';
42 export const url = `https://${host}:${port}`;
43 const graphqlUri = `${url}/graphql`;
44
```

Рисунок 3.14 – Опрацювання змінних середовища

Наступним етапом є підключення бібліотеки Chakra UI. Відповідно до офіційної документації, спочатку необхідно налаштувати провайдер, який генерується за допомогою команди:

`npx @chakra-ui/cli snippet add`

Після виконання цієї команди за шляхом components/ui/provider буде згенеровано відповідний компонент. З огляду на доцільність структурування коду, усі компоненти, пов'язані з Chakra UI, рекомендовано зберігати в окремій папці. Оскільки всі компоненти проєкту зберігаються у каталозі components, папку ui доцільно перенести відповідно до цієї структури.

Отриманий у результаті компонент (рис. 3.15) можливо модифікувати таким чином, щоб забезпечити зміну теми за допомогою ColorModeProvider (рис 3.16).

```
1 'use client';
2
3 import { ChakraProvider, createSystem, defaultConfig, Theme } from '@chakra-ui/react';
4 import { ColorModeProvider, type ColorModeProviderProps } from './color-mode';
5
6 export function Provider(props: ColorModeProviderProps) {
7   const system = createSystem(defaultConfig, {
8     theme: {
9       tokens: {}
10    }
11  });
12  return (
13    <ChakraProvider value={system}>
14      <Theme colorPalette="cyan">
15        <ColorModeProvider defaultTheme="dark">{props.children}</ColorModeProvider>
16      </Theme>
17    </ChakraProvider>
18  );
19 }
```

Рисунок 3.15 – Оновлений ChakraProvider

Його підключення та використання є аналогічними до встановлення основного провайдера, як і у випадку з іншими компонентами бібліотеки Chakra UI.

```

1 'use client';
2
3 import type { IconButtonProps } from '@chakra-ui/react';
4 import { ClientOnly, IconButton, Skeleton } from '@chakra-ui/react';
5 import { ThemeProvider, useTheme } from 'next-themes';
6 import type { ThemeProviderProps } from 'next-themes';
7 import * as React from 'react';
8 import { LuMoon, LuSun } from 'react-icons/lu';
9
10 export interface ColorModeProviderProps extends ThemeProviderProps {}
11
12 export function ColorModeProvider(props: ColorModeProviderProps) {
13   return <ThemeProvider attribute="class" disableTransitionOnChange {...props} />;
14 }
15
16 export function useColorMode() {
17   const { resolvedTheme, setTheme } = useTheme();
18   const toggleColorMode = () => {
19     setTheme(resolvedTheme === 'light' ? 'dark' : 'light');
20   };
21   return {
22     colorMode: resolvedTheme,
23     setColorMode: setTheme,
24     toggleColorMode
25   };
26 }
27
28 export function useColorModeValue<T>(light: T, dark: T) {
29   const { colorMode } = useColorMode();
30   return colorMode === 'light' ? light : dark;
31 }
32
33 export function ColorModeIcon() {
34   const { colorMode } = useColorMode();
35   return colorMode === 'light' ? <LuSun /> : <LuMoon />;
36 }
37
38 interface ColorModeButtonProps extends Omit<IconButtonProps, 'aria-label'> {}
39
40 export const ColorModeButton = React.forwardRef<HTMLButtonElement, ColorModeButtonProps>(_props, ref) {
41   const { toggleColorMode } = useColorMode();
42   return (
43     <ClientOnly fallback={<Skeleton boxSize="8" />}>
44       <IconButton
45         onClick={toggleColorMode}
46         variant="ghost"
47         aria-label="Toggle color mode"
48         size="sm"
49         ref={ref}
50         {..._props}
51         css={{
52           _icon: {
53             width: '5',
54             height: '5'
55           }
56         }}
57       >
58         <ColorModeIcon />
59       </IconButton>
60     </ClientOnly>
61   );
62 }
63
64

```

Рисунок 3.16 – Оновлений ColorModeProvider

Це дозволяє реалізувати функціональність перемикання між світлою та темною темами інтерфейсу, що є сучасною практикою підвищення зручності користування вебзастосунком. Проте слід зазначити, що функціональність

зміни теми є окремим елементом загального функціоналу, реалізація якого значною мірою залежить від обраної палітри кольорів. Палітра, яка використовується за замовчуванням, не завжди є оптимальною для одночасного застосування як у світлій, так і в темній темах, що потребує додаткового налаштування або кастомізації кольорової схеми для забезпечення належної візуальної адаптивності інтерфейсу. Компонент `ColorModeProvider` викликається в межах основного провайдера, забезпечуючи можливість керування темою інтерфейсу. У той час як компонент `ColorModeButton` планується використовувати в подальшій розробці для перемикання між темною та світлою темами.

Для коректної інтеграції компонентів `Chakra UI` у застосунок, основну структуру програми необхідно обгорнути у компонент `ChakraProvider`, що дозволяє забезпечити доступ до всіх стилістичних налаштувань бібліотеки в межах усього клієнтського інтерфейсу.

Окремо встановимо та підключимо компонент `Toaster` (рис 3.17), який йде окремо від `ChakraProvider`, щоб мати можливість викликати відповідні компоненти.

```
<ChakraProvider>
  <Toaster />
  <Stack maxWidth="100vw">
    <RouterProviderWrapper authHeaders={authHeaders} />
  </Stack>
</ChakraProvider>
```

Рисунок 3.17 – Використання `ChakraProvider` та `Toaster`

Після ініціалізації компонента `ChakraProvider` можна перейти до налаштування файлового маршрутизаційного механізму за допомогою `TanStack Router`. Для цього створюється екземпляр роутера, до якого передається автоматично згенероване дерево маршрутів. Це дерево передається як атрибут об'єкта, що використовується при виклику функції `createRouter`.

З метою забезпечення коректної підтримки типів у середовищі TypeScript, додатково виконується декларація відповідних модулів, що дозволяє інтегрувати систему маршрутизації з типобезпечною архітектурою клієнтського застосунку. Додатково надамо початкову інформацію (рис. 3.18) щодо стану авторизації для перевірки та дій до початку процесу завантаження маршрутів.

```
// TanStackRouter - Import the generated route tree
import { Session } from '@generated/types';
import { routeTree } from './routeTree.gen';

// Create a new router instance
export const router = createRouter({
  routeTree,
  context: {
    isAuthenticated: false
  }
});

// Register the router instance for type safety
declare module '@tanstack/react-router' {
  interface Register {
    router: typeof router;
  }
}
```

Рисунок 3.18 – TanStack декларація маршрутизатора

Роутер з окремим контекстом авторизації у свою чергу передаються у RouterProvider (рис. 3.19):

```
<RouterProvider router={router} context={{ isAuthenticated: authHeaders != null }} />
```

Рисунок 3.19 – Router Provider

Останньою частиною побудови початкового шаблону проекту є підключення Apollo Client, який забезпечить зв'язок з серверною частиною Graph QL (Apollo Server). У рамках конфігурації Apollo Client, одним із ключових компонентів є утиліта HttpLink (рис. 3.20).

```
const httpLink = new HttpLink({
  uri: graphqlUri,
  credentials: 'include',
  headers: authHeaders != null ? { authorization: JSON.stringify(authHeaders) } : undefined
});
```

Рисунок 3.20 – Apollo HttpLink

Вона забезпечує встановлення HTTP-з'єднання з GraphQL-сервером та виконує роль транспортного шару між клієнтом та сервером, дозволяючи явно вказати URI ендпоінту, налаштувати політику cookies через параметр credentials: "include", а також динамічно додавати HTTP-заголовки, зокрема заголовок авторизації, що є важливим для аутентифікованого доступу.

Окрему роль у побудові зв'язку між клієнтом і сервером відіграє ErrorLink. Це спеціалізована ланка у ланцюжку посилань Apollo, яка відповідає за перехоплення та обробку помилок, що виникають під час виконання GraphQL-запитів або мутацій. Її використання є особливо актуальним у двох контекстах:

- Детальний опис помилок під час розробки – коли виникає необхідність у детальній діагностиці помилок, особливо в неуспішних запитах до сервера.
- Реалізації менеджменту сесій, де ErrorLink дозволяє реалізувати стратегію автоматичного відновлення сесії. Зокрема, у випадку, коли відповідь сервера містить повідомлення про закінчення строку дії JWT-токена (наприклад, jwt expired), клієнтська логіка ініціює оновлення токена за допомогою виклику функції refreshToken(), після чого оригінальний запит повторно надсилається до сервера без необхідності перезавантаження сторінки чи втручання користувача.

Завершальним етапом є створення екземпляра Apollo Client (рис. 3.21), у конфігурацію якого передаються обидва посилання: httpLink та errorLink.

```
const client = new ApolloClient({
  uri: graphqlUri,
  link: from([errorLink, httpLink]),
  cache: new InMemoryCache(),
  connectToDevTools: !isProd,
  credentials: 'include'
});
```

Рисунок 3.21 – Apollo Client

Вони об'єднуються за допомогою функції `from`, яка дозволяє формувати послідовність обробки запитів. Додатково клієнт конфігурується з використанням кешу `InMemoryCache` та параметра `connectToDevTools`, який активується у режимі розробки для забезпечення інтеграції з браузерними інструментами відладки `Apollo`.

3.3. Розробка системи авторизації та реєстрації

Після реалізації основних компонентів як на стороні клієнта, так і на стороні сервера, застосунок набуває повноцінної функціональності — з'являється можливість відображення інтерфейсних компонентів, створення власних маршрутизаторів і сторінок, а також здійснення повноцінної комунікації з сервером. Для цього використовуються такі інструменти, як `Apollo Client` або `Fetch API`, що забезпечують ефективний обмін даними через `GraphQL` або `REST`-протоколи відповідно.

Це дозволяє перейти до реалізації одного з найважливіших аспектів сучасного вебзастосунку — взаємодії з моделлю користувача та менеджменту сесії. До складу цієї функціональності входять реєстрація нового користувача, вхід до системи, автоматичне відновлення сесії, ануляція сесії та отримання персоніфікованих даних користувача на основі поточних сеансових даних.

Для побудови системи аутентифікації обрано підхід, заснований на використанні `JWT (JSON Web Token)` у поєднанні з серверною обробкою `cookies`. На відміну від традиційного зберігання токенів у `LocalStorage`, даний підхід не передбачає жодної взаємодії клієнтської частини з даними сесії, окрім одноразового токена доступу, який зберігається та використовується для відстеження сесії клієнту. `RefreshToken` у свою чергу передається через `HttpOnly cookie`. Це забезпечує суттєво вищий рівень безпеки, зокрема захист від `XSS`-атак, оскільки доступ до `cookie` неможливий через `JavaScript`.

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		37

Обробка життєвого циклу сесії реалізується автоматично. У випадку завершення строку дії access-токена, сервер повертає відповідну помилку (jwt expired), яка перехоплюється на клієнті за допомогою механізму ErrorLink, інтегрованого в Apollo Client. У відповідь клієнт ініціює запит на оновлення сесії, після чого здійснюється повторна відправка запиту без потреби у взаємодії з боку користувача. Таким чином, безперервність користувацького досвіду зберігається, а весь процес авторизації та валідації залишається прозорим для кінцевого споживача.

Обраний спосіб реалізації є технічно складнішим у порівнянні зі збереженням токенів на клієнті, однак цей підхід дозволяє досягти вищого рівня безпеки та відповідності сучасним вимогам до захисту персональних даних у вебзастосунках. Для цього реалізації почнемо з опису даних сесії та користувача у схемі GraphQL (рис. 3.22).

```
input SignUpInput {
  email: String!
  username: String!
  password: String!
}

input LoginInput {
  email: String!
  password: String!
}

type Session {
  userId: ID!
  accessToken: String!
}

type Identity {
  id: ID!
  email: String!
  isDarkTheme: Boolean!
  user: User!
}

type User {
  id: ID!
  username: String!
  description: String!
  createdAt: String!
  updatedAt: String!
}

input UserOneInput {
  id: ID
  username: String
}
```

Рисунок 3.22 – Типи schema.graphql для необхідні для авторизації

Модель користувача, що реалізована у структурі бази даних, була концептуально розділена на дві складові: User та Identity. Такий підхід дозволяє забезпечити гнучкий розподіл доступу до даних користувача відповідно до його ролі у системі та статусу аутентифікації.

Модель User містить загальнодоступну інформацію, яка може бути використана будь-якими іншими користувачами системи. До таких даних можуть належати, наприклад, публічне ім'я або дата реєстрації. Ці атрибути не містять чутливої інформації, тому їх наявність у відкритому доступі не становить загрози для безпеки або конфіденційності.

Натомість модель Identity є строго прив'язаною до поточного авторизованого користувача та містить виключно конфіденційні поля, такі як дані електронної пошти. Така сегментація чітко відмежовує дані, які можуть бути доступні публічно, від тих, що мають обмежений доступ, і дозволяє ефективно реалізувати контроль над авторизацією на рівні GraphQL-запитів.

Крім того, це унеможливорює спроби отримання чутливої інформації сторонніми користувачами навіть у випадку спроби здійснення запиту вручну через інструменти типу GraphQL Playground або аналогічні. Модель Identity доступна лише у контексті поточної аутентифікованої сесії, і її використання обумовлюється перевіркою сесійного токена на серверному рівні. Такий підхід значно підвищує загальний рівень безпеки системи та дозволяє зберігати персональні дані відповідно до принципів захисту конфіденційної інформації.

Натомість типи SignUpInput та LoginInput відповідають за реєстрацію та авторизацію.

Відповідно до побудованих типів, можемо створити відповідні мутації та запити. Згідно рис. 3.23 та рис. 3.24 маємо запити та мутації, які ми повинні реалізувати. Але перед цим, кожна модифікація схеми, повинна бути сопереженою командою для регенерації типів:

npm run gql-types

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		39

```
identity: Identity!  
userOne(input: UserOneInput!): User!
```

Рисунок 3.23 – Запити для авторизації

```
signUp(input: SignUpInput!): Void  
login(input: LoginInput!): Session!  
logout: Void
```

Рисунок 3.24 – Мутації для авторизації

Дана команда викликає graphql-codegen (рис 3.25), який автоматично згенерує типи для використання у клієнті та сервері

```
"scripts": {  
  "dev": "VITE_MODE=dev nodemon -w app/server -x tsx app/server/main.ts",  
  "start": "NODE_ENV=production VITE_MODE=prod tsx app/server/main.ts",  
  "build": "vite build",  
  "genssl": "openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout ./certs/selfsigned.key -out ./certs/selfsigned.crt",  
  "gql-types": "graphql-codegen --config codegen.ts",  
  "checkup": "opinionated checkup",  
  "prisma": "prisma",  
  "chakra": "chakra"  
},
```

Рисунок 3.25 – Скороченні команди додатку

Після регенерації типів можемо перейти до реалізації резолверів, маперів та контексту для їх подальшого використання у системі. Перш за все, необхідно розробити контекст, який ми зможемо використовувати для перевірки даних поточного користувача.

Контекст буде супроводжувати кожен GraphQL-запит і дозволить отримувати інформацію про авторизованого користувача, а також забезпечуватиме базу для перевірок прав доступу та виконання логіки, що залежить від статусу сесії. Саме через нього можна буде звертатися до моделі Identity, яка є доступною лише для автентифікованого користувача, що критично важливо для забезпечення безпеки взаємодії з API.

Таким чином, правильно реалізований контекст(рис. 3.26) є обов'язковим кроком перед розробкою резолверів, адже він забезпечує доступ до сеансових даних і дозволяє реалізовувати механізми авторизації на рівні запитів.

```

TS identity.ts X | TS login.ts X | TS logout.ts X | TS signUp.ts X |
1 import { Session } from '@generated/types';
2 import 'dotenv/config';
3 import { Request, Response } from 'express';
4 import { GraphQLError } from 'graphql';
5 import jwt from 'jsonwebtoken';
6 import { config } from './main';
7 import { prisma } from './prisma/client';
8
9 export interface IdentityContext {
10   req: Request;
11   res: Response;
12   identity?: {
13     id: string;
14     email: string;
15     username: string;
16   };
17 }
18
19 export async function getIdentityContext({
20   req,
21   res
22 }): Promise<IdentityContext> {
23   req: Request;
24   res: Response;
25 } {
26   try {
27     const authHeaders = req.headers.authorization;
28     if (authHeaders == null) return { req, res };
29
30     const session: Session = JSON.parse(authHeaders);
31     const { accessToken } = session;
32
33     const decoded = jwt.verify(accessToken, config.accessToken.secret) as {
34       userId: string;
35       iat: number;
36       exp: number;
37     };
38
39     const identity = await prisma.user.findUnique({
40       where: { id: decoded.userId },
41       select: {
42         id: true,
43         email: true,
44         username: true
45       }
46     });
47     if (identity == null) throw new Error("Couldn't find the user.");
48     return { req, res, identity };
49   } catch (error) {
50     if (error instanceof Error)
51       throw new GraphQLError(error.message, {
52         extensions: {
53           code: 'UNAUTHENTICATED',
54           http: { status: 401 }
55         }
56       });
57     throw new GraphQLError(typeof error === 'string' ? error : 'Unknown error', {
58       extensions: {
59         code: 'UNAUTHENTICATED',
60         http: { status: 401 }
61       }
62     });
63   }
64 }

```

Рисунок 3.26 – Функція формування контексту

Згідно з кодом, ми маємо вихідний тип контексту, який складається з об'єктів запиту та відповіді, що надаються з боку Express.js. Це зумовлено тим, що раніше було створено Apollo Server, який використовується як middleware у рамках Express-додатку.

Маючи визначений вихідний тип, можемо сформуванати логіку отримання контексту, яка починається з обробки сесії користувача. Сесійні дані витягуються із заголовків запиту, після чого виконується парсинг отриманого значення для доступу до токена аутентифікації та ідентифікатора користувача.

Для цілей верифікації токена, а також забезпечення надійного шифрування, використовується бібліотека `jsonwebtoken`, яка дозволяє виконати декодування та перевірку аутентичності отриманих даних. Після успішної верифікації, виконується запит до бази даних для перевірки існування користувача за вказаним ідентифікатором.

У разі, якщо користувача не знайдено, ініціюється відповідна помилка. Якщо ж користувач існує, у контекст повертаються необхідні дані відповідно до попередньо визначеної схеми. З метою забезпечення стабільності виконання та уникнення раптових збоїв, реалізовано обробку винятків, яка супроводжується встановленням статусу відповіді 401 у випадках невдалої авторизації.

Задля ініціалізації (рис. 3.27) створеного контексту надаємо тип до екземпляру ApolloServer та передаємо функцію до атрибуту context.

```
const apollo = new ApolloServer<IdentityContext>({
  typeDefs,
  resolvers,
  introspection: !config.isProd,
  plugins: [ApolloServerPluginDrainHttpServer({ httpServer: server })],
  formatError: formattedError => {
    process.stdout.write('\n```\n');
    console.error(formattedError);
    process.stdout.write('\n```\n');
    return formattedError;
  }
});

await apollo.start();

app.use(cors<cors.CorsRequest>(config.corsOptions));
app.use(express.json(config.limit));
app.use(cookies());
app.get('/api/refreshAccessToken', refreshTokenController);
app.use(
  '/graphql',
  expressMiddleware(apollo, {
    context: getIdentityContext
  })
);
```

Рисунок 3.27 – Ініціалізація контексту

Далі створюється резолвер для обробки реєстрації користувача (рис. 3.28). На першому етапі передаються всі необхідні аргументи з чітким

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		42

визначенням їхніх типів. З огляду на те, що TypeScript не підтримує автоматичну типізацію при використанні більш ніж одного параметра у функції резолвера, слід бути уважним під час визначення вхідних типів.

Першим параметром функції виступає об'єкт типу `any`, оскільки резолвери можуть бути вкладеними (сабресолверами) і звертатися один до одного, передаючи дані з батьківського рівня. Другим параметром є вхідна інформація, отримана від клієнта. Оскільки мова йде про мутацію, а також з урахуванням попередньо згенерованих типів, ми можемо використати відповідний вхідний тип, що було згенеровано для даного резолвера. Третій параметр — це контекст, який було налаштовано раніше, і який містить усю необхідну інформацію про поточний сеанс або запит.

```
TS login.ts X | TS logout.ts X | TS signUp.ts X |
1 import { IdentityContext } from '@app/server/identity';
2 import { prisma, uuid } from '@app/server/prisma/client';
3 import { MutationSignUpArgs } from '@generated/types';
4 import bcrypt from 'bcrypt';
5
6 export async function SignUpResolver(
7   _: any,
8   { input }: MutationSignUpArgs,
9   { req, res, identity }: IdentityContext
10 ): Promise<void> {
11   const existingUsername = await prisma.user.findUnique({
12     where: { username: input.username },
13     select: { id: true }
14   });
15   if (existingUsername != null) throw new Error('User with this username already exists.');
```

```
16
17   const existingEmail = await prisma.user.findUnique({
18     where: { email: input.email },
19     select: { id: true }
20   });
21   if (existingEmail != null) throw new Error('Email is already in use.');
```

```
22
23   const hashedPassword = await bcrypt.hash(input.password, 10);
24
25   await prisma.user.create({
26     data: {
27       id: uuid(),
28       email: input.email,
29       username: input.username,
30       password: hashedPassword
31     }
32   });
33 }
```

Рис 3.28 – Резолвер реєстрації

Для підтвердження можливості реєстрації необхідно спершу переконатися у відсутності вже зареєстрованого користувача з такою ж

електронною поштою або логіном. У випадку виявлення дублікатів відповідна помилка повертається клієнту.

Якщо перевірка пройшла успішно, пароль користувача хешується із застосуванням криптографічної функції, після чого створюється новий запис у базі даних. Відповідно до обраної стратегії безпеки, жодних сеансових чи авторизаційних даних після завершення операції не повертається — користувач повинен пройти авторизацію окремо після завершення реєстрації.

Для реалізації авторизації (рис. 3.29), аналогічно до резолвера реєстрації, виконується типізація вхідних та вихідних даних. На відміну від процесу реєстрації, цього разу у відповідь повертається сесія користувача.

```
TS login.ts X | TS logout.ts X | TS signUp.ts X |
1 import { IdentityContext } from '@app/server/identity';
2 import { prisma } from '@app/server/prisma/client';
3 import { MutationLoginArgs, Session } from '@generated/types';
4 import bcrypt from 'bcrypt';
5 import jwt from 'jsonwebtoken';
6 import { config } from '@app/server/main';
7
8 export async function LoginResolver(
9   _: any,
10  { input }: MutationLoginArgs,
11  { req, res, identity }: IdentityContext
12 ): Promise<Session> {
13   const error = 'Invalid password or email';
14
15   const user = await prisma.user.findUnique({
16     where: { email: input.email }
17   });
18   if (user == null) throw new Error(error);
19
20   const match = await bcrypt.compare(input.password, user.password);
21   if (!match) throw new Error(error);
22
23   const accessToken = jwt.sign({ userId: user.id }, config.accessToken.secret, {
24     expiresIn: config.accessToken.expiresIn
25   });
26   const refreshToken = jwt.sign({ userId: user.id }, config.refreshToken.secret, {
27     expiresIn: config.refreshToken.expiresIn
28   });
29
30   await prisma.user.update({
31     where: { id: user.id },
32     data: { refreshToken }
33   });
34
35   res.cookie('jwt', refreshToken, {
36     httpOnly: true,
37     sameSite: 'none',
38     secure: true,
39     maxAge: config.refreshToken.maxAge
40   });
41
42   return {
43     userId: user.id,
44     accessToken
45   };
46 }
```

Рисунок 3.29 – Резолвер авторизації

На початку реалізації оголошується потенційна помилка, яка може бути повернута у разі невірних або некоректних вхідних даних. Далі відбувається перевірка наявності користувача з відповідною електронною поштою. Якщо такий користувач відсутній — повертається помилка. У випадку, коли користувач знайдено, виконується перевірка введеного пароля. У разі невідповідності паролю також повертається помилка.

Якщо автентифікація проходить успішно, створюються токени доступу: access token та refresh token. Після цього оновлений refresh token зберігається у базі даних відповідно до поточного користувача.

Варто зазначити, що обрана стратегія не підтримує одночасну авторизацію на декількох пристроях, що, однак, підвищує загальний рівень безпеки системи за рахунок обмеження множинного використання сесії.

На завершення, у відповідь від сервера повертається cookie з флагом httpOnly, а також об'єкт сесії, який може бути використаний на клієнтській стороні для подальшої взаємодії.

У разі, якщо користувач має намір анулювати власну сесію, викликається резолвер Logout (рис. 3.30).

У цьому випадку відсутні вхідні параметри, за винятком стандартних об'єктів request, response та identity, які передаються через контекст.

На першому етапі отримується cookie поточного користувача, з якого витягується refreshToken. За цим токеном здійснюється пошук користувача у базі даних. Якщо відповідного користувача не знайдено, сесія анулюється шляхом виконання операції clearCookie, що видаляє відповідний токен з клієнтської сторони.

У разі, якщо користувача знайдено, відповідний токен у базі даних оновлюється (анулюється), після чого також повертається відповідь з викликом clearCookie, що остаточно припиняє дію сесії на стороні клієнта. Такий підхід гарантує безпечне завершення авторизованої взаємодії між клієнтом та сервером.

```

1 import { IdentityContext } from '@app/server/identity';
2 import { prisma } from '@app/server/prisma/client';
3
4 export async function LogoutResolver(
5   _: any,
6   __: any,
7   { req, res, identity }: IdentityContext
8 ): Promise<void> {
9   const cookie: { jwt?: string | null } = req.cookies;
10  if (cookie == null || cookie.jwt == null) return;
11
12  const refreshToken = cookie.jwt;
13
14  const user = await prisma.user.findUnique({
15    where: { refreshToken },
16    select: { id: true }
17  });
18
19  if (user == null) {
20    res.clearCookie('jwt', { httpOnly: true });
21    return;
22  }
23
24  await prisma.user.update({
25    where: {
26      id: user.id
27    },
28    data: { refreshToken: null }
29  });
30
31  res.clearCookie('jwt', { httpOnly: true });
32

```

Рисунок 3.30 – Logout резолвер

Оскільки при обробці запитів GraphQL необхідно мати доступ до інформації про сесію користувача на стороні клієнта ще до ініціалізації ApolloClient, виникає проблема циклічних залежностей. Зокрема, без створеного екземпляра Apollo Client неможливо виконати запити, пов'язані з Apollo Server. Для вирішення цієї проблеми використовується допоміжний REST API (рис. 3.31), який відповідає за обробку токена refresh, що дозволяє відновити сесію користувача ще до декларації Apollo Client.

Для реалізації цього підходу створюється окремий ендпоінт за допомогою Express JS. У функцію передаються параметри запиту та відповіді (req, res), після чого перевіряється наявність cookies у запиті. У випадку їх відсутності одразу повертається відповідь зі статусом 401.

Якщо cookies присутні, здійснюється пошук користувача у базі даних за токеном refresh. Якщо користувача не знайдено — повертається помилка. У разі наявності користувача виконується декодування токена з використанням бібліотеки jsonwebtoken. Якщо токен виявляється недійсним або зіпсованим — сервер повертає повідомлення про недійсність та відповідний статус 401.

У випадку успішної валідації refresh токена, користувачу повертається оновлена сесія, яка в подальшому може бути використана для ініціалізації ApolloClient та виконання GraphQL-операцій без додаткової взаємодії з боку користувача. Такий підхід забезпечує безперервність сесії та покращує користувацький досвід без порушення безпекових вимог.

```

1 import { config } from '@app/server/main';
2 import { prisma } from '@app/server/prisma/client';
3 import { Session } from '@generated/types';
4 import 'dotenv/config';
5 import { Request, Response } from 'express';
6 import jwt from 'jsonwebtoken';
7
8 export async function refreshTokenController(req: Request, res: Response) {
9   try {
10     const cookie: { jwt?: string | null } = req.cookies;
11
12     if (cookie == null || cookie.jwt == null) {
13       res.sendStatus(401);
14       return;
15     }
16
17     const refreshToken = cookie.jwt;
18
19     const user = await prisma.user.findUnique({
20       where: { refreshToken },
21       select: { id: true }
22     });
23
24     if (user == null) {
25       res.statusMessage = 'User not found.';
26       res.status(401).end();
27       return;
28     }
29
30     const decoded = jwt.verify(refreshToken, config.refreshToken.secret) as {
31       userId: string;
32       iat: number;
33       exp: number;
34     };
35
36     if (user.id !== decoded.userId) {
37       res.statusMessage = 'Invalid token.';
38       res.status(401).end();
39       return;
40     }
41
42     const accessToken = jwt.sign({ userId: user.id }, config.accessToken.secret, {
43       expiresIn: config.accessToken.expiresIn
44     });
45
46     const session: Session = { userId: user.id, accessToken };
47     res.json(session);
48   } catch (error) {
49     console.error(error);
50   }
51 }
52

```

Рисунок 3.31 – Контролер refreshToken

Дана комбінація резолверів та REST-ендпоінту дозволяє реалізувати повноцінну систему авторизації, базовану на JSON Web Token, з обробкою на стороні сервера. Для організації менеджменту сесії на клієнтській стороні

необхідно розробити допоміжний контекст, кастомний хук, а також відповідні сторінки для авторизації.

Першим кроком є створення контексту та хука авторизації (рис. 3.32). Розроблений хук відповідає за керування процесами входу та виходу користувача зі системи, а також за оновлення сесії у разі її відновлення через токен refresh.

```
interface AuthContextProps {
  headers: Session | null;
  setHeaders: (session: Session | null) => void;
}

export const AuthContext = createContext<AuthContextProps | null>(null);
export const AuthProvider = AuthContext.Provider;

interface UseAuthOutput extends AuthContextProps {
  isAuthenticated: boolean;
  login: (value: { email: string; password: string }) => Promise<Session | null | undefined>;
  logout: () => Promise<void>;
}

export function useAuth(): UseAuthOutput {
  const authContext = useContext(AuthContext);
  const { update: login } = useToastMutation<{ login: Session }>({
    mutation: loginMutation,
    successMsg: 'Logged in successfully.'
  });

  const { update: logout } = useToastMutation({
    mutation: logoutMutation,
    successMsg: 'Logged out successfully.'
  });

  if (authContext == null) throw new Error('useAuth must be used within an AuthProvider');

  return {
    ...authContext,
    isAuthenticated: authContext.headers != null,
    login: async ({ email, password }: { email: string; password: string }) => {
      const { data } = await login({
        variables: {
          input: {
            email,
            password
          }
        }
      });
      if (data != null) {
        authContext.setHeaders(data.login);
        return data.login;
      }
      return data;
    },
    logout: async () => {
      await logout();
      authContext.setHeaders(null);
    }
  };
}
```

Рисунок 3.32 – Hook useAuth

У межах реалізації авторизації використовується кастомний хук useToastMutation, який поєднує можливості стандартного хука useMutation з пакету Apollo Client та компоненту Toaster з бібліотеки Chakra UI. Такий підхід дозволяє не лише обробляти мутації, а й відображати користувачу інформативні

повідомлення про результати виконання запиту, включаючи успішне завершення або можливі помилки. Це підвищує зручність взаємодії з додатком і покращує загальний користувацький досвід.

Цей хук залежить від даних, що надаються через AuthContextProvider, що стало причиною виникнення циклічних залежностей у процесі ініціалізації додатку. Зокрема, проблема полягала в тому, що useAuth звертається до контексту, який, у свою чергу, не може бути створений до ініціалізації ApolloClient, а той залежить від наявності сесії.

Для вирішення цієї ситуації було прийнято рішення винести функціонал відновлення сесії у REST API ендпоінт, що дозволило виконувати операцію оновлення сесії ще до створення клієнта Apollo.

У зв'язку з цим, логіку взаємодії з REST-ендпоінтом було винесено з useAuth в окремий хук useRestAuth (рис. 3.33).

```
interface UseRestAuthOutput {
  refreshToken: () => Promise<Session | null | undefined>;
}

export function useRestAuth(): UseRestAuthOutput {
  return {
    refreshToken: async () => {
      try {
        const response = await fetch(`${url}/api/refreshAccessToken`, {
          credentials: 'include'
        });
        if (!response.ok) {
          throw new Error(`[error]: ${response.status}`);
        }
        const session = await response.json();
        return session;
      } catch (error) {
        console.error(error);
      }
    }
  };
}

const loginMutation = gql`
  mutation AuthLogin($input: LoginInput!) {
    login(input: $input) {
      ...Session
    }
  }
  ${sessionFragment}
`;

const logoutMutation = gql`
  mutation AuthLogout {
    logout
  }
`;
```

Рисунок 3.33 – Hook useResAuth

Даний хук більше не залежить від контексту AuthContextProvider і може бути викликаний незалежно на ранніх етапах ініціалізації. Це дозволило уникнути циклічної залежності та забезпечити стабільну роботу клієнтської частини додатку.

Далі реалізуємо логіку хука useIdentity (рис. 3.34), що включатиме в себе всю релевантну інформацію про поточного користувача. Такий підхід дозволяє централізовано керувати даними користувача, уникати дублювання запитів у різних компонентах додатку та спрощує перевірку авторизаційного стану в межах будь-якої частини клієнтського інтерфейсу.

```
import { ApolloError, ApolloQueryResult, gql, OperationVariables, useQuery } from '@apollo/client';
import { Identity, IdentityRootQuery } from '@generated/types';
import { createContext, useContext } from 'react';

export interface IdentityContext {
  data: Identity;
  refetch: () => void;
}

export const IdentityContext = createContext<IdentityContext | null>(null);
export const IdentityProvider = IdentityContext.Provider;

export function useIdentity(): IdentityContext | null {
  const identityContext = useContext(IdentityContext);
  if (identityContext == null) return null;
  return identityContext;
}

export function useIdentityQuery({ skip }: { skip?: boolean }): {
  data: IdentityRootQuery | undefined;
  loading: boolean;
  error: ApolloError | undefined;
  refetch: () => Promise<ApolloQueryResult<IdentityRootQuery>>;
} {
  const { data, loading, error, refetch } = useQuery<IdentityRootQuery>(query, {
    skip
  });
  return {
    data,
    loading,
    error,
    refetch
  };
}

const query = gql`
  query IdentityRoot {
    identity {
      id
      email
      isDarkTheme
      user {
        id
        username
        createdAt
        updatedAt
      }
    }
  }
`
```

Рисунок 3.34 – Hook useIdentity

Основою функціонування useIdentity є допоміжний хук useIdentityQuery, який використовується для отримання даних про користувача безпосередньо з Apollo Server. Отримані дані слугують джерелом для ініціалізації

IdentityContext, що у подальшому надає доступ до ідентифікаційної інформації у всіх дочірніх компонентах React-застосунку.

Для реєстрації користувача, створюємо форму (рис. 3.35), яка валідується за допомогою valibot, та контролюється React Hook Form. Після заповнення даних, при успішній реєстрації користувач буде перенесено на сторінку входу, яка побудована аналогічним чином, але використовує useAuth хук, замість мутації.

```
const schema = v.pipe(
  v.object({
    email: v.pipe(v.string(), v.nonEmpty('Email is required'), v.email('Invalid email')),
    username: v.pipe(v.string(), v.nonEmpty('Username is required')),
    password: v.pipe(
      v.string(),
      v.minLength(8, 'Your password is too short.'),
      v.maxLength(45, 'Your password is too long.'),
      v.regex(/[a-z]/, 'Your password must contain a lowercase letter.'),
      v.regex(/[A-Z]/, 'Your password must contain a uppercase letter.'),
      v.regex(/[0-9]/, 'Your password must contain a number.')
    ),
    confirm_password: v.pipe(v.string(), v.nonEmpty('Confirm your password'))
  }),
  v.forward(
    v.check(input => input.password === input.confirm_password, 'Passwords do not match'),
    ['confirm_password']
  )
);

const {
  register,
  handleSubmit,
  formState: { errors }
} = useForm<FormValues>({ resolver: valibotResolver(schema) });

const onSubmit = handleSubmit(async values => {
  await signUp({
    variables: {
      input: {
        email: values.email,
        username: values.username,
        password: values.password
      }
    }
  });
  navigate({
    to: '/login'
  });
});
```

Рисунок 3.35 – Створення схеми valibot та форми react-hook-form

3.4. Розробка навігації

Важливою частиною вебдодатку є навігація, адже саме завдяки їй користувач отримує можливість переходити між основними сторінками

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		51

інтерфейсу. Основна навігація реалізована у секції «header», яка виконує роль центрального елементу орієнтації в межах додатку.

До складу цієї навігаційної панелі входять:

- логотип додатку, який слугує посиланням на головну сторінку;
- редактор набросків;
- посилання на сторінку створення книги;
- а також, залежно від стану авторизації, відображається або набір опцій для входу та реєстрації, або інформація про профіль користувача з можливістю анулювання поточної сесії.

Такий підхід дозволяє не лише зручно організувати переміщення між функціональними модулями, але й динамічно адаптувати інтерфейс відповідно до статусу автентифікації (рис. 3.36) користувача.

```
export const Route = createRootRouteWithContext<RouterContext>() ({
  component: () => {
    const { logout } = useAuth();
    const identity = useIdentity();
    const navigate = useNavigate();
```

Рисунок 3.36 – Використання useAuth та useIdentity

Згідно попередньо розробленим хукам, можемо розробити меню, яке в залежності від даних користувача буде мати опцію авторизації та реєстрації, або аватару з меню (рис. 3.37), де користувач може перейти до свого профілю, або деавторизуватися.

```

{identity != null ? (
  <Menu.Root>
    <Menu.Trigger css={{ '&:hover': { cursor: 'pointer' } }}>
      <Avatar size="sm" fallback={identity.data.user.username[0]} />
    </Menu.Trigger>
    <Menu.Positioner mt={2}>
      <Menu.Content backgroundColor={'bg.subtle'}>
        <Link to={` /profile/${identity.data.user.username}`}>
          <Menu.Item value="/">
            <Tooltip content="My profile">
              <Stack direction="row">
                <Icon color="colorPalette.500" fontSize="xl">
                  <i className="nf nf-fa-user" />
                </Icon>
                My profile
              </Stack>
            </Tooltip>
          </Menu.Item>
        </Link>

        <Menu.Item value="logout" onClick={async () => logout()}>
          <Tooltip content="Log out">
            <Stack direction="row">
              <Icon color="colorPalette.500" fontSize="xl">
                <i className="nf nf-fa-sign_out" />
              </Icon>
              Log out
            </Stack>
          </Tooltip>
        </Menu.Item>
      </Menu.Content>
    </Menu.Positioner>
  </Menu.Root>
) : (
  <>
    <Separator orientation="vertical" height={5} />
    <Link to="/login">
      <Button height={8}>Sign in</Button>
    </Link>
    <Link to="/register">
      <Button height={8}>Sign up</Button>
    </Link>
  </>
)
)}

```

Рисунок 3.37 – Опціональні кнопки авторизації та меню користувача

3.5. Розробка головної сторінки

Головною сторінкою створено список доступних для прочитання книг. Компонент BookSearchList (рис. 3.38) відповідає за відображення списку книжок з підтримкою пошуку та пагінації. Він використовує стан для збереження поточного значення пошуку, активної сторінки та кількості елементів на сторінці. Пошукове значення передається у запит через хук usePaginatedQuery, який виконує GraphQL-запит BooksListQuery та повертає

список книжок відповідно до заданого фільтру (ключові слова, відступ та розмір сторінки).

Компонент також відображає форму пошуку, сам список результатів (WorkingList) і, якщо загальна кількість книжок перевищує розмір сторінки, — компонент пагінації. Поки дані завантажуються, показується скелет сторінки (PaginatedListSkeleton). Таким чином, BookSearchList забезпечує зручну взаємодію користувача з великою колекцією книжок, дозволяючи ефективно шукати та переглядати результати частинами.

Усі розроблені компоненти є модульними, що дозволяє використовувати їх і у інших частинах проекту.

```
export function BookSearchList() {
  const [workingSearchValue, setWorkingSearchValue] = useState<string>('');
  const [workingsListPage, setWorkingsListPage] = useState<number>(1);
  const [workingsListPageSize, setWorkingsListPageSize] = useState<number>(8);
  const startIndex = (workingsListPage - 1) * workingsListPageSize;

  const { data } = usePaginatedQuery<BooksListQuery>({
    query: bookListQuery,
    input: {
      filter: {
        terms: workingSearchValue,
        offset: startIndex,
        size: workingsListPageSize
      }
    }
  });

  return (
    <>
      <ListSearch placeholder="Search works..." onSubmit={value => setWorkingSearchValue(value)} />
      {data != null ? (
        <>
          <WorkingList workingsList={ [...data.bookMany.items] } />
          {data.bookMany.total > workingsListPageSize && (
            <Stack justifyContent="center" alignItems="center">
              <Pagination
                total={data.bookMany.total}
                page={workingsListPage}
                setPage={setWorkingsListPage}
                pageSize={workingsListPageSize}
                setPageSize={setWorkingsListPageSize}
              />
            </Stack>
          )}
        </>
      ) : (
        <PaginatedListSkeleton />
      )}
    </>
  );
}
```

Рисунок 3.38 – BookSearchList – головна сторінка

Компонент WorkingList використовує дочірній компонент WorkingListCard для відображення кожного окремого елементу списку книг у

вигляді картки. WorkingListCard побудований на основі моделі Book (рис. 3.39), тому в ньому відображається лише та інформація, яка передбачена цією моделлю – наприклад, назва книги, автор, короткий опис, статус книги та дані про відгуки та рейтинги. Це дозволяє уніфіковано і візуально структуровано представити дані про кожну книгу.

```
type BookManyOutput {
  items: [Book!]!
  total: Int!
}

type Book {
  id: ID!
  title: String!
  description: String!
  author: User!
  createdAt: String!
  updatedAt: String!
  progressStatus: WorkingProgressStatus!
  wordsTotal: Int!
  chaptersTotal: Int!
  ratingTotal: Int!
  reviewsTotal: Int!
  personalRating: RatingType!
}
```

Рисунок 3.39 – Book – schema.graphql

Задля передачі книг на клієнтську частину, відповідно до моделі, були розроблені окремі резолвери та мапери, які відповідають за подачу даних, як показано на прикладі рис. 3.40.

```
export async function toBookSchema(
  model: Book,
  { identity }: IdentityContext
): Promise<BookSchema> {
  return {
    id: model.id,
    title: model.title,
    description: model.description,
    author: await call(async () => {
      const userModel = await prisma.user.findUniqueOrThrow({ where: { id: model.authorId } })
      return toUserSchema(userModel)
    }),
    createdAt: model.createdAt.toISOString(),
    updatedAt: model.updatedAt.toISOString(),
    progressStatus: call(() => {
      ensureWorkingProgressStatus(model.progressStatus)
      return model.progressStatus
    })
  }
}
```

Рисунок 3.40 – Book – мапер

З кожної картки книги, користувач має можливість перейти на сторінку автора, чи на сторінку перегляду книги.

3.6. Розробка сторінки огляду книги

Компонент реалізує сторінку з оглядом книги. Він динамічно завантажує дані про конкретну книгу за її ідентифікатором (bookId) через GraphQL-запит BookOverview. Якщо дані ще не отримані — відображається скелет сторінки BookOverviewSkeleton, а після успішного завантаження — основний компонент BookOverview(рис.3.41). Відображається за маршрутом 'overview' (рис. 3.42).

```
function BookOverview({ book }: { readonly book: BookOne }) {
  return (
    <Stack width={{ base: '100%', md: '80%' }} maxWidth="840px" gap={3}>
      <Stack direction="row" justifyContent="space-between" alignItems="center">
        <Heading color="colorPalette.400" size="2xl" overflow="auto">
          {book.title}
        </Heading>
      </Stack>

      <Stack direction="column" gap={1}>
        <Stack direction="row">
          <Link>
            <Text fontWeight="bold">by:</Text>
          </Link>
          <Link to={` /profile/${book.author.username}`}>
            <Text
              color="colorPalette.400"
              _hover={{
                textDecoration: 'underline',
                textDecorationColor: 'colorPalette.700'
              }}
            >
              {book.author.username}
            </Text>
          </Link>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">created at:</Text>
          <Text>{toFormattedDate({ value: book.createdAt })}</Text>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">last updated:</Text>
          <Text>{toFormattedDate({ value: book.updatedAt })}</Text>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">chapters:</Text>
          <Text>{book.chaptersTotal}</Text>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">words:</Text>
          <Text>{book.wordsTotal}</Text>
        </Stack>
      </Stack>

      <Stack direction="row" justifyContent="left" alignItems="center">
        <WorkingRatingBadge ratingTotal={book.ratingTotal} />
        <WorkingReviewsBadge reviewsTotal={book.reviewsTotal} />
        <WorkingProgressStatusBadge progressStatus={book.progressStatus} />
      </Stack>

      <Text>{book.description}</Text>
    </Stack>
  )
}
```

Рисунок 3.41 – BookOverview

Компонент BookOverview виводить заголовок книги, ім'я автора (з посиланням на профіль), дату створення й останнього оновлення, кількість розділів та загальну кількість слів. Окрім цього, рендеряться бейджі з рейтингом, кількістю рецензій і статусом прогресу написання. Внизу розміщується основний текст опису книги. Компонент також підтримує вкладені маршрути через Outlet, що дозволяє розміщати різні секції за відповідним маршрутом. Серед цих секцій є: список глав, відгуки, редактор даних книги (тільки для автора).

```
function RouteComponent() {
  const { bookId: id } = Route.useParams();

  const identity = useIdentity();

  const { data } = useQuery<BookOverviewRatingQuery>(bookRatingQuery, {
    variables: { id }
  });

  const [ratingUpdate] = useMutation<BookOverviewRatingQuery>(bookRatingMutation, {
    refetchQueries: [bookRatingQuery]
  });

  return (
    <>
      {data != null ? (
        <>
          <BookOverviewTabs
            isAuthor={data?.bookOne.author.id === identity?.data.id}
            bookId={id}
            currentTab="contents"
          />

          <BookChapterList
            id={data.bookOne.id}
            isAuthor={identity?.data.id === data.bookOne.author.id}
          />

          {identity != null &&
            identity?.data.id !== data.bookOne.author.id &&
            data.bookOne.chaptersTotal > 0 && (
              <RatingPanel
                personalRating={data.bookOne.personalRating}
                onRatingSet={async ratingType => {
                  await ratingUpdate({
                    variables: {
                      input: {
                        bookId: data.bookOne.id,
                        ratingType
                      }
                    }
                  });
                }}
              />
            )}
        ) : (
          <Skeleton height="200px" />
        )
      ) : null}
    </>
  );
}
```

Рисунок 3.42 – Маршрут “overview” книги

3.7. Розробка редактора та переглядача глав

Редактором глав (рис. 3.43) є компонент BookChapterForm, що реалізує повноцінний редактор глави книги, який дозволяє створювати або редагувати вміст глави з використанням інтуїтивного інтерфейсу.

Форма включає поля для заголовку (title), передмови (foreword), основного вмісту (content) та примітки внизу (footnote). Основна частина глави редагується через компонент EditorSlate, який використовує структуру Descendant[] зі Slate.js для багатоформатного редагування тексту.

Для перевірки введених даних використовується бібліотека valibot, яка задає схему валідації. Вона забезпечує обов'язковість заголовку, а також обмеження довжини.

```
<form onSubmit={onSubmit}>
  <Stack justifyContent="center" alignItems="center" minHeight="88vh">
    <Stack maxWidth="840px" minHeight="80vh" justifyContent="center" alignItems="center">
      <Stack width="100%" direction="row" justifyContent="space-between" alignItems="start">
        <Box>
          <Field invalid={errors.title != null} errorText={errors.title?.message}>
            <Input {...register('title')} placeholder="Chapter title" />
          </Field>
        </Box>
        <Stack direction="row">
          <Button onClick={async () => onSubmit()}>Submit</Button>
          <Link to={`\work/book/${bookId}/overview/contents`} >
            <Button>Go back</Button>
          </Link>
        </Stack>
      </Stack>

      <Field invalid={errors.foreword != null} errorText={errors.foreword?.message}>
        <Textarea
          color="fg.muted"
          variant="subtle"
          placeholder="Foreword"
          resize="none"
          maxLength={500}
          {...register('foreword')}
        />
      </Field>

      <EditorSlate
        initialValue={initialValues?.content}
        onChange={values => setContent(values)}
      />

      <Field invalid={errors.footnote != null} errorText={errors.footnote?.message}>
        <Textarea
          color="fg.muted"
          variant="subtle"
          placeholder="Footnote"
          resize="none"
          maxLength={500}
          {...register('footnote')}
        />
      </Field>
    </Stack>
  </Stack>
</form>
```

Рисунок 3.43 – Редактор глав

Стани помилок обробляються через react-hook-form, і відповідні повідомлення відображаються у UI.

Користувач може надіслати форму через кнопку Submit, що викликає функцію onChapterSubmit, або повернутися назад до вмісту книги через посилання Go back.

Сам редактор було розроблено за допомогою бібліотеки Slate.js, що забезпечує гнучку та розширювану архітектуру для створення WYSIWYG-редакторів. Його інтеграція в інтерфейс користувача реалізована через компонент EditorSlate, який виступає обгорткою над функціональним модулем EditorViewport. Останній ініціалізує екземпляр редактора з підключенням модулів withReact та withHistory, що дозволяє забезпечити як інтеграцію з React-компонентами, так і підтримку історії редагувань (undo/redo).

Редактор підтримує різноманітні функції форматування тексту, включаючи жирне накреслення, курсив, підкреслення, моноширинний шрифт, а також зміну структури документа за допомогою заголовків першого та другого рівнів, списків (нумерованих і маркованих), цитат і блоків з вирівнюванням тексту (вліво, вправо, по центру, за шириною). Ці можливості реалізовані як через панель інструментів, доступну користувачеві, так і через обробку гарячих клавіш.

Кожен елемент контенту обробляється окремим рендерером. Компонент Element відповідає за відображення структурних елементів документа (параграфів, заголовків, списків тощо), тоді як компонент Leaf відповідає за стилістичне оформлення окремих фрагментів тексту. Обидва ці компоненти враховують як тип елемента або ознаку (наприклад, bold чи italic), так і можливість вирівнювання тексту через атрибут align.

Редактор дозволяє зберігати дані у вигляді структури Descendant[], що є базовим форматом внутрішнього представлення Slate. Це забезпечує зручне збереження вмісту глави та подальшу інтеграцію з інтерфейсом форми

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		59

BookChapterForm. У разі зміни вмісту відповідна функція зворотного виклику (onChange) синхронізує актуальні дані з головним станом компонента.

Таким чином, розроблений редактор забезпечує гнучкий інструментарій для структурованого створення та редагування текстових глав у межах розроблюваної платформи, поєднуючи високу кастомізованість, інтерактивність та підтримку базових видавничих вимог.

Редактор також дозволяє перейти у режим читання, за допомогою відповідного пропу, як зображено на рис.3.44.

```
{data.bookChapterOne.foreword && (  
  <>  
    <Stack width="91%" justifyContent="left" alignItems="start" padding<  
      <Text fontWeight="bold">Foreword:</Text>  
      <Text color="fg.muted">{data.bookChapterOne.foreword}</Text>  
    </Stack>  
    <Separator width="91%" />  
  </>  
)}  
<EditorSlate initialValue={data.bookChapterOne.content} readOnly />  
{data.bookChapterOne.footnote && (  
  <>  
    <Separator width="91%" />  
    <Stack width="91%" justifyContent="left" alignItems="start" padding<  
      <Text fontWeight="bold">Footnote:</Text>  
      <Text color="fg.muted">{data.bookChapterOne.foreword}</Text>  
    </Stack>  
  </>  
)}
```

Рисунок 3.44 – Редактор у режимі читання

Задля створення або обробки змін у главах, були створенні резолвери мутацій.

BookChapterCreateResolver (рис.3.45) реалізує резолвер GraphQL для створення нового розділу книги в інформаційній системі. Вона забезпечує валідацію вхідних даних, дотримання авторських прав доступу та підтримання цілісності структури книги.

На початковому етапі виконується перевірка аутентичності користувача за допомогою функції ensureIdentity, що гарантує доступ лише авторизованим особам. Далі здійснюється валідація довжини введених текстових полів: заголовок (title) не повинен перевищувати 60 символів, а вступ (foreword) і примітка (footnote), якщо надані, — 500 символів кожне. У разі порушення цих

обмежень генеруються відповідні виключення, що запобігають запису невалідних даних до бази.

Після проходження перевірок виконується підрахунок кількості вже наявних розділів у книзі. Це значення використовується для визначення порядкового номера (sequenceNumber) нового розділу, який завжди додається в кінець книги.

Далі створюється новий запис у таблиці bookChapter за допомогою методу create. Кожен новий розділ отримує унікальний ідентифікатор (uuid()), встановлюється його належність до конкретної книги (bookId), записуються передані текстові поля, а також обчислюється загальна кількість слів (wordsTotal) на основі функції countWords.

```
import { countWords, ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma, uuid } from '@app/server/prisma/client';
import { updateBookStats } from '@app/server/service/book';
import { BookChapter, MutationBookChapterCreateArgs } from '@generated/types';
import { toBookChapterSchema } from '../mappers/bookChapterMapper';

export async function BookChapterCreateResolver(
  _: any,
  { input }: MutationBookChapterCreateArgs,
  { req, res, identity }: IdentityContext
): Promise<BookChapter> {
  ensureIdentity(identity);

  if (input.title.length > 60) throw new Error('Title is too long.');
```

```
  if (input.foreword != null && input.foreword.length > 500)
    throw new Error('Foreword is too long.');
```

```
  if (input.footer != null && input.footer.length > 500)
    throw new Error('Footnote is too long.');
```

```
  const chapterCount = await prisma.bookChapter.count({ where: { bookId: input.bookId } });

  const model = await prisma.bookChapter.create({
    data: {
      id: uuid(),
      bookId: input.bookId,
      title: input.title,
      foreword: input.foreword,
      footnote: input.footer,
      content: input.content,
      wordsTotal: countWords(input.content),
      sequenceNumber: chapterCount + 1
    }
  });

  await updateBookStats(input.bookId);
  return toBookChapterSchema(model, { req, res, identity });
}
```

Рисунок 3.45 – BookChapterCreateResolver

Після створення глави викликається updateBookStats, що оновлює агреговані дані книги (сумарну кількість слів та кількість розділів) для підтримання актуальності статистики.

На завершальному етапі створений розділ перетворюється у відповідний формат GraphQL-схеми (BookChapter) за допомогою функції toBookChapterSchema, після чого повертається як результат виконання запиту.

BookChapterUpdate робить ідентичні маніпуляції, але замість створення нової книги, він викликає оновлення за ідентифікатором.

BookChapterDeleteResolver (рис. 3.46), який відповідає за видалення розділу книги. Його основна мета — забезпечити узгоджене вилучення глави, з урахуванням авторства та оновлення порядкових номерів інших глав у книзі.

На початку функції викликається ensureIdentity, що перевіряє наявність автентифікованого користувача. Далі за отримується метадані про розділ, який планується видалити, зокрема його номер у послідовності (sequenceNumber) та ідентифікатор і автора відповідної книги.

Верифікація прав доступу виконується шляхом порівняння ідентифікатора поточного користувача з автором книги. У разі невідповідності кидається помилка, що перериває подальше виконання.

Після перевірки авторизації виконується видалення розділу за вказаним id з таблиці bookChapter. Потім, щоб зберегти логічну послідовність порядкових номерів інших глав книги, які йдуть після видаленого розділу, усі наступні глави зменшуються на одиницю через updateMany.

Наступним кроком викликається updateBookStats, який, ймовірно, перераховує статистику книги після змін (наприклад, кількість глав, загальний обсяг тощо).

На завершення, книга повторно завантажується з бази (findUniqueOrThrow) і трансформується у відповідну схему GraphQL-типу Book за допомогою toBookSchema, після чого повертається як відповідь.

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		62

```

import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { updateBookStats } from '@app/server/service/book';
import { Book, MutationBookChapterDeleteArgs } from '@generated/types';
import { toBookSchema } from '../mappers/bookMapper';

export async function BookChapterDeleteResolver(
  _: any,
  { id }: MutationBookChapterDeleteArgs,
  { req, res, identity }: IdentityContext
): Promise<Book> {
  ensureIdentity(identity);
  const model = await prisma.bookChapter.findUniqueOrThrow({
    where: { id },
    select: {
      sequenceNumber: true,
      book: { select: { id: true, authorId: true } }
    }
  });
  if (identity.id !== model.book.authorId) throw new Error('Operation is not permitted.');
```

```

  await prisma.bookChapter.delete({ where: { id } });

  await prisma.bookChapter.updateMany({
    where: {
      bookId: model.book.id,
      sequenceNumber: {
        gt: model.sequenceNumber
      }
    },
    data: {
      sequenceNumber: {
        decrement: 1
      }
    }
  });

  await updateBookStats(model.book.id);
  const book = await prisma.book.findUniqueOrThrow({ where: { id: model.book.id } });
  return toBookSchema(book, { req, res, identity });
}

```

Рисунок 3.46 – BookChapterDeleteResolver

3.8. Розробка системи відгуків

Основна функціональність зосереджена в компоненті за шляхом “reviews” (рис 3.47), який забезпечує завантаження необхідних даних про рецензії, ідентифікацію користувача, можливість створення нової рецензії, а також виведення списку вже існуючих коментарів з пагінацією.

На початку компонента отримується ідентифікатор книги з параметрів маршруту. Далі, використовуючи кастомний хук usePaginatedQuery, ініціалізується запит bookReviewsQuery, який отримує список рецензій, враховуючи поточну сторінку та розмір сторінки. Окремо виконується запит bookAuthorQuery для визначення, чи є поточний користувач автором книги, що

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		63

використовується для коректного відображення вкладок через компонент BookOverviewTabs.

Створення рецензії реалізоване через мутацію bookReviewCreateMutation, яка обгорнута у хук useToastMutation. Це забезпечує зворотний виклик із відображенням повідомлення про успіх та автоматичним оновленням списку рецензій після успішного додавання нової.

```
return (
  <>
    <BookOverviewTabs
      isAuthor={authorData.bookOne.author.id === identity?.data.id}
      bookId={id}
      currentTab="reviews"
    />

    {identity !== null && reviewsPage === 1 && (
      <ReviewInput
        username={identity?.data.user.username}
        onReviewSubmit={async text => {
          await reviewCreate({
            variables: {
              input: {
                bookId: id,
                text
              }
            }
          });
        }}
      />
    )}

    {data.reviewMany.items.length === 0 ? (
      <Stack justifyContent="center" alignItems="center" pt={8}>
        <Text>No reviews here yet!</Text>
      </Stack>
    ) : (
      data.reviewMany.items.map(item => (
        <ReviewCard
          key={`bookReview#${item.id}`}
          review={{
            id: item.id,
            username: item.author.username,
            text: item.text,
            createdAt: item.createdAt
          }}
        />
      ))
    )}

    {data.reviewMany.total > reviewsPageSize && (
      <Stack justifyContent="center" alignItems="center">
        <Pagination
          total={data.reviewMany.total}
          page={reviewsPage}
          setPage={setChaptersListPage}
          pageSize={reviewsPageSize}
          setPageSize={setChaptersListPageSize}
        />
      </Stack>
    )}
  </>
)
```

Рисунок 3.47 – Файл “review.tsx”

Зм.	Арк.	№ докум.	Підпис	Дата

Інтерфейсно, у випадку, якщо користувач автентифікований та знаходиться на першій сторінці рецензій, йому надається можливість залишити відгук через компонент ReviewInput. Усі існуючі рецензії відображаються через ReviewCard, який презентує кожну з них у форматі картки з іменем автора, текстом та датою створення.

У разі відсутності рецензій відображається інформаційне повідомлення. Якщо загальна кількість рецензій перевищує розмір сторінки, активується компонент Pagination, який дозволяє переміщатися між сторінками списку.

ReviewCreateResolver (рис. 3.48) – відповідає за створення нової рецензії до книги. На початковому етапі виконується перевірка користувача за допомогою функції ensureIdentity. Лише авторизований користувач має змогу створювати рецензії.

```

1 import { ensureIdentity } from '@app/server/helpers';
2 import { IdentityContext } from '@app/server/identity';
3 import { prisma, uuid } from '@app/server/prisma/client';
4 import { MutationReviewCreateArgs, Review } from '@generated/types';
5 import { toReviewSchema } from '../mappers/reviewMapper';
6
7 export async function ReviewCreateResolver(
8   _: any,
9   { input }: MutationReviewCreateArgs,
10  { identity }: IdentityContext
11 ): Promise<Review> {
12   ensureIdentity(identity);
13
14   if (input.text.length > 700) throw new Error('Review is too long.');
```

Рисунок 3.48 – ReviewCreateResolver

Далі виконується валідація вхідного тексту рецензії. Якщо його довжина перевищує встановлений ліміт у 700 символів, ініціюється виключення з відповідним повідомленням.

Ця перевірка запобігає зловживанню сервісом та забезпечує уніфікацію змісту рецензій.

У разі успішного проходження перевірок, за створюється новий запис у таблиці review. При цьому зберігається текст рецензії, зв'язок із відповідною книгою через bookId та ідентифікатор користувача, що створив запис.

Насамкінець, отриманий об'єкт перетворюється у GraphQL-сумісну структуру за допомогою функції toReviewSchema.

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		66

ВИСНОВОК ДО РОЗДІЛУ 3

У ході виконання роботи було реалізовано комплексний веб-додаток із чітко структурованою архітектурою, що базується на сучасних технологіях і підходах. Було розроблено шаблон проекту, який забезпечує модульність і повторне використання компонентів, що спрощує підтримку та масштабування коду.

Система авторизації та реєстрації реалізована з використанням перевірки ідентичності користувачів, що гарантує безпеку та контроль доступу до захищених ресурсів. Логіка авторизації реалізована через серверні функції, які взаємодіють із базою даних за допомогою Prisma ORM, забезпечуючи надійність і ефективність.

Навігація побудована з урахуванням маршрутизації на основі TanStack Router, що дозволяє динамічно керувати переходами між сторінками додатка без перезавантаження та керується структурою файлів. Впроваджено компоненти, які керують станом та параметрами URL, що забезпечує коректну роботу пагінації і фільтрації.

Головна сторінка та сторінка огляду книги реалізовані через комбінацію GraphQL-запитів та React-компонентів, що забезпечують отримання та відображення актуальних даних із серверної частини. Дані отримуються за допомогою Apollo Client, що забезпечує кешування та оптимізацію запитів.

Редактор та переглядач глав розроблені із застосуванням бібліотеки Slate, що надало можливість створити гнучкий текстовий редактор з підтримкою форматування, блоків і вирівнювання тексту. Логіка зміни стану редактора організована через хуки React, а обробка подій клавіатури реалізована із застосуванням бібліотеки is-hotkey. Зміни контенту синхронізуються із сервером через GraphQL-мутатори, що дозволяє зберігати стан документів.

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		67

Система відгуків реалізована через окремі GraphQL-запити та мутації, які забезпечують додавання, отримання та відображення відгуків користувачів. Логіка взаємодії з базою даних організована через Prisma, що спрощує роботу з моделями і дозволяє ефективно обробляти великі обсяги даних.

					ІАЛЦ.467200.003 ПЗ	Арк.
						68
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОГО ВЕБ-ДОДАТКА

4.1. Головна сторінка

Одразу при переході на сторінку додатку бачимо пошуковик та список робіт з пагінацією (рис. 4.1). Основні елементи навігації у веб-додатку є елементами, що знаходяться у Header компоненті, що дозволяє користувачам легко переходити у різні секції застосунку.

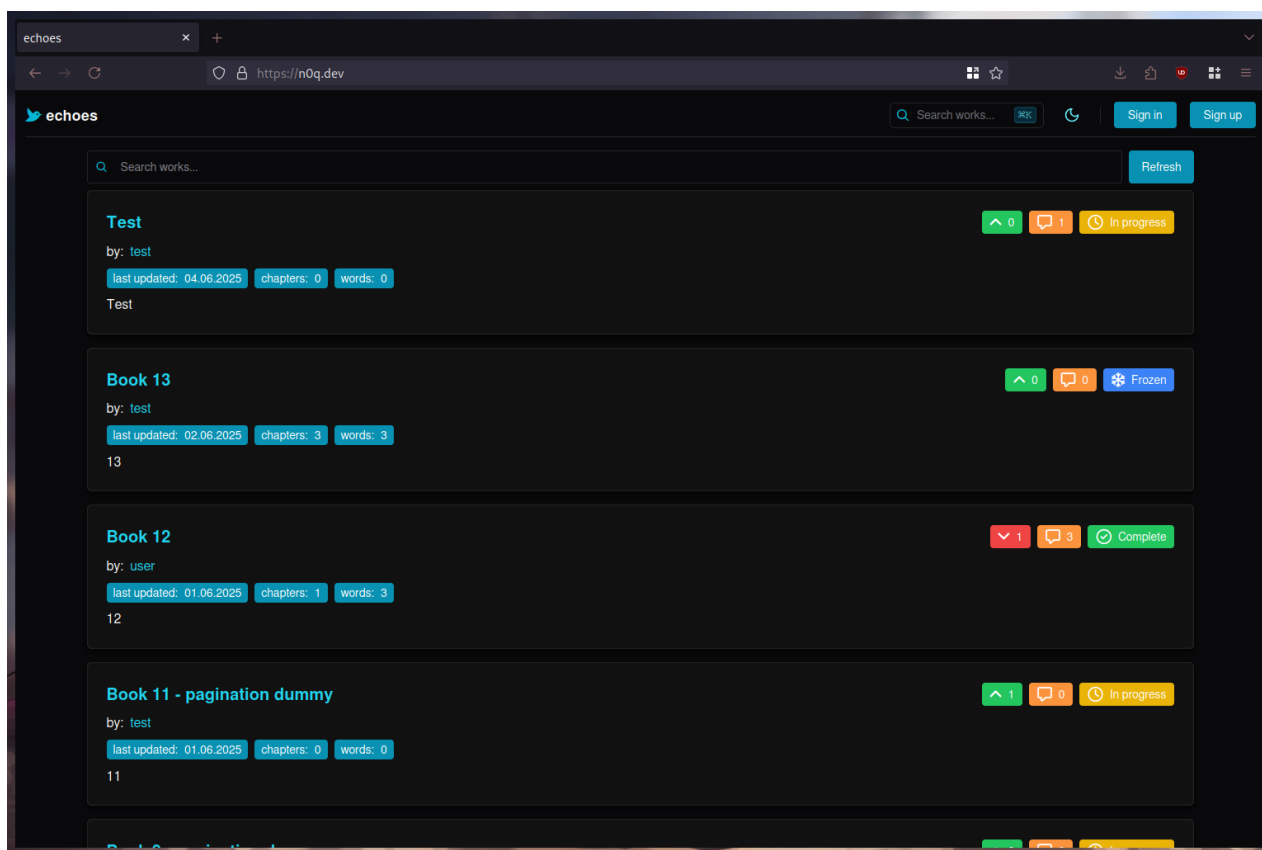


Рисунок 4.1 – Головна сторінка

4.2. Сторінки входу та реєстрації

За умовчанням, користувач має обмежений дозвіл до додатку, що обмежується тільки читацьким функціоналом. Щоб отримати повний дозвіл до

операцій, до яких входить створення книг та їх редакція, а також можливість залишення оцінок та рецензій. Тому наступним чином перейдемо до сторінки авторизації за кнопкою “Sign in”.

При переході на сторінку входу (рис. 4.2) маємо декілька опцій: ввести свої дані, чи створити новий акаунт. Як було зазначено раніше, ми також маємо змогу навігації у додатку за допомогою незмінного елемента “Header”, зокрема логотип з назвою “echoes”, перенесе нас назад на головну сторінку.

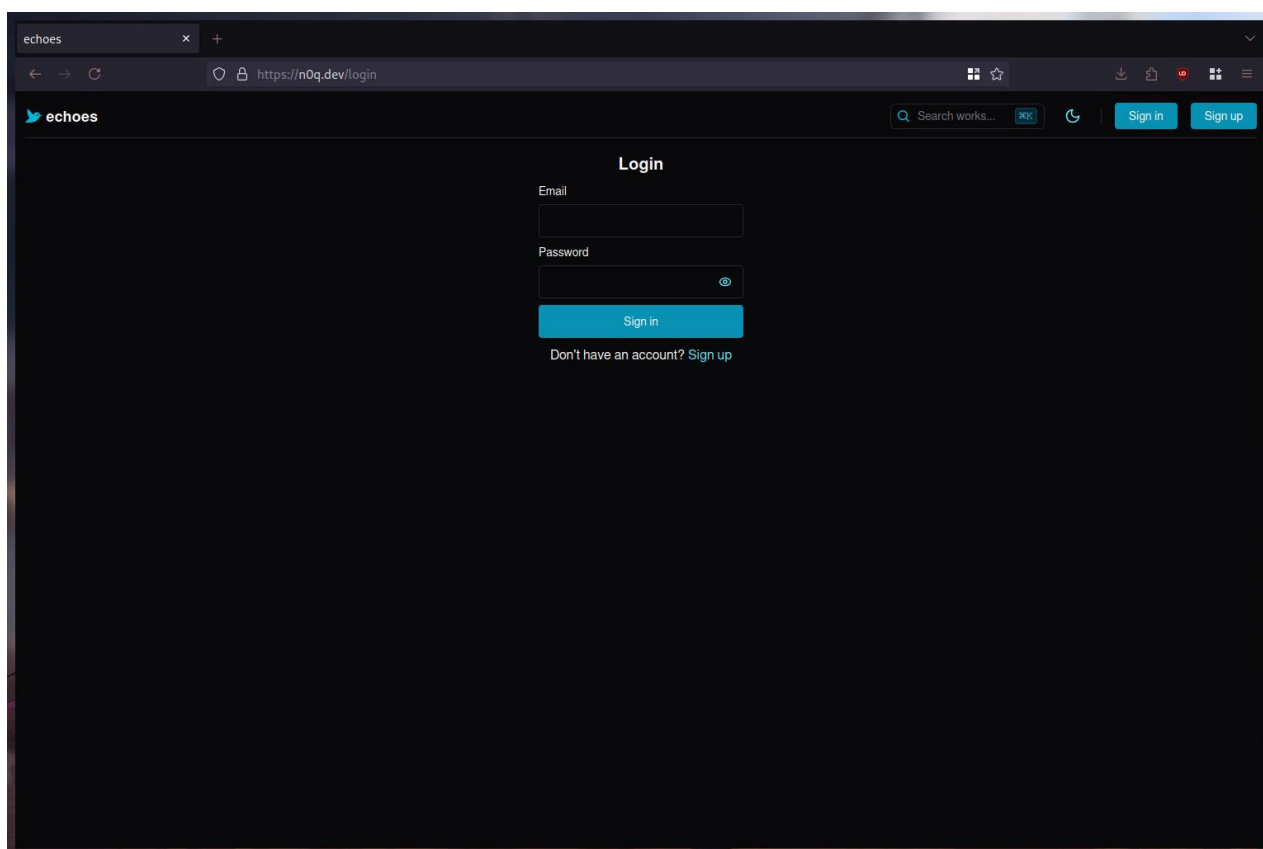


Рисунок 4.2 – Сторінка входу

Окремою опцією при вході, є опція створення нового акаунту, якщо ми користувач, який не має власного запису, ми почнемо авторизацію саме з переходу до “Sign up”.

За цим шляхом (рис. 4.3) ми маємо можливість новий акаунт, у обох формах також присутня власна валідація, яка повідомляє користувача про помилки, які можуть виникнути при створенні власного запису, такі як

обмеження на розмір полів, дублікати пошти, імені користувача, чи відмінність між полями пароллю та його поля підтвердження.

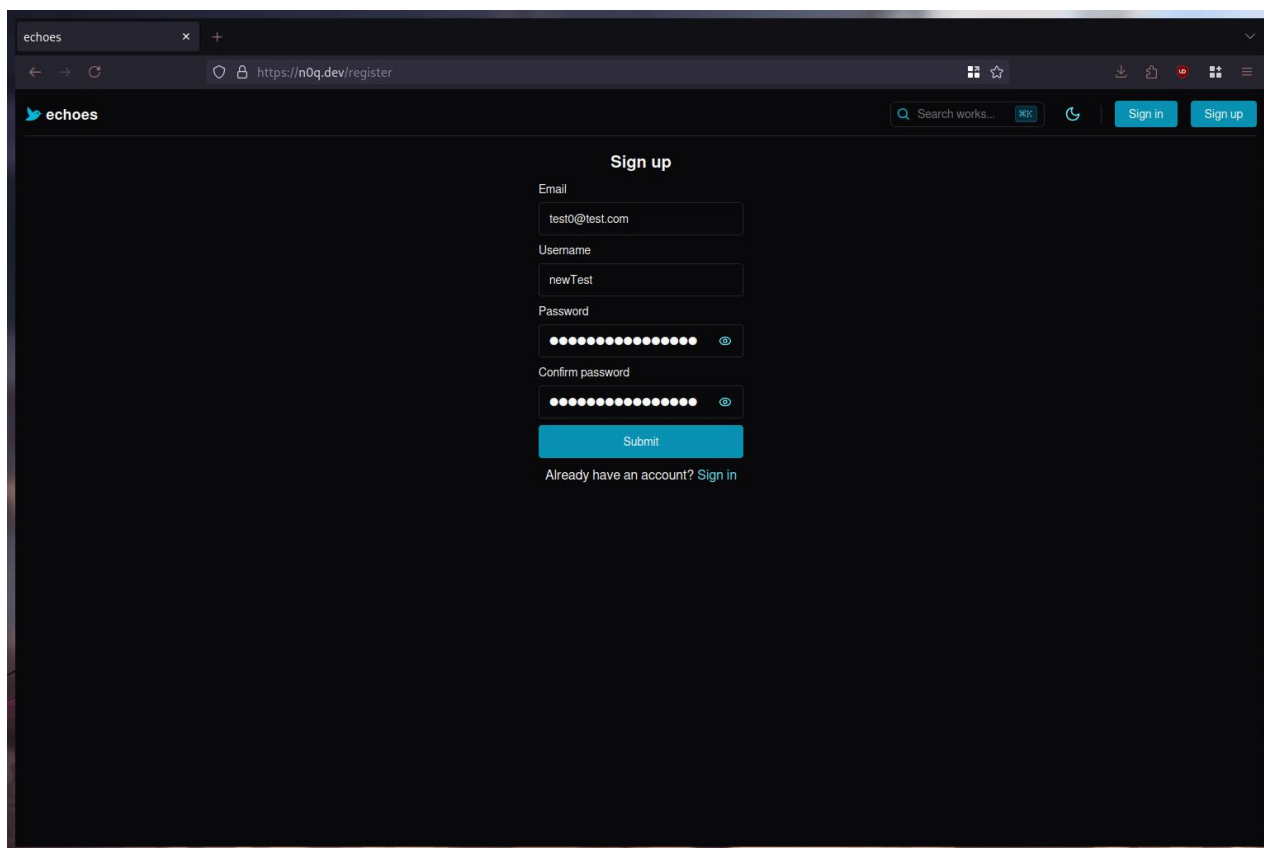


Рисунок 4.3 – Сторінка реєстрації

Після введення усіх необхідних даних, у разі успішної реєстрації користувач буде перенаправлений до сторінки входу, де він може продовжити звичайний хід авторизації.

4.3. Інтерактивні елементи та навігація

Після входу, користувач перенаправлений до головної сторінки додатку, де він буде мати вже більше опцій для інтеракції (рис. 4.4) ніж у анонімному режимі. Зокрема зверху у секції Header можемо побачити, що тепер ми маємо доступ до редактору та створення книг. При перезавантаженні сторінки, сесія

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		71

перезавантажується та зі сторони користувача залишається незмінною, що свідчить про правильну роботу системи авторизації та ідентифікації

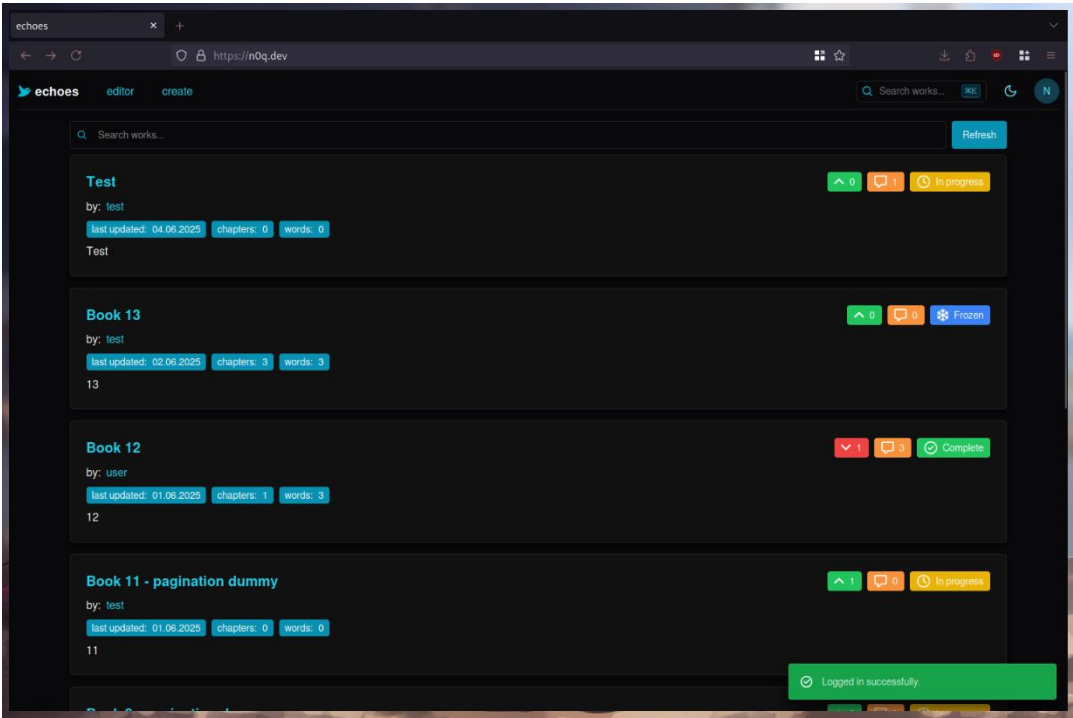


Рисунок 4.4 – Вигляд сторінки у авторизованому стані

При переході до редактору (рис. 4.5), ми побачимо сторінку для редакції набросків. Даний компонент не виконує ніякого функціоналу окрім написання та обробки тексту.

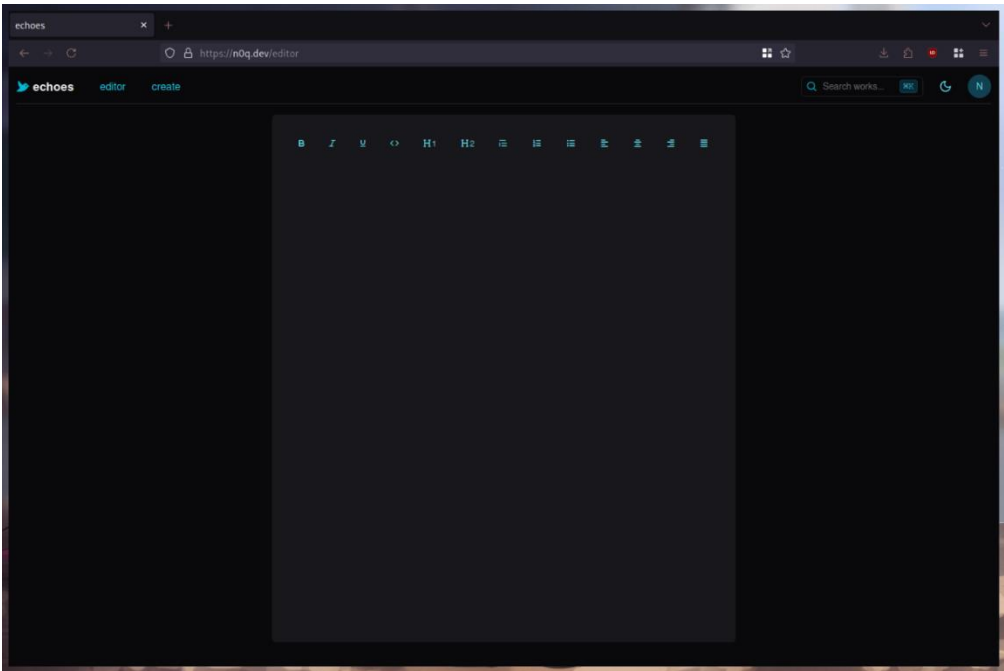


Рисунок 4.5 – Редактор набросків

Результат написання у даному сегменті нікуди не зберігається, та представляє перед користувачем можливість ознайомлення з системою редактору.

4.4. Створення та редакція книги

При переході на сторінку “create”(рис. 4.6), користувач має можливість створити книгу, для чого йому потрібно заповнити назву книги та опціонально заповнити опис книги.

Рисунок 4.6 – Сторінка створення книг

Після створення книги, користувач перенаправлений на сторінку огляду книги, де він як автор, має можливість створити нову главу, або перейти на сторінку Details (рис. 4.7), де він може змінити деталі книги: назву, опис та її

статус. Окремою опцією є повне видалення книги, що буде нести за собою видалення усіх пов’язаних відгуків та глав.

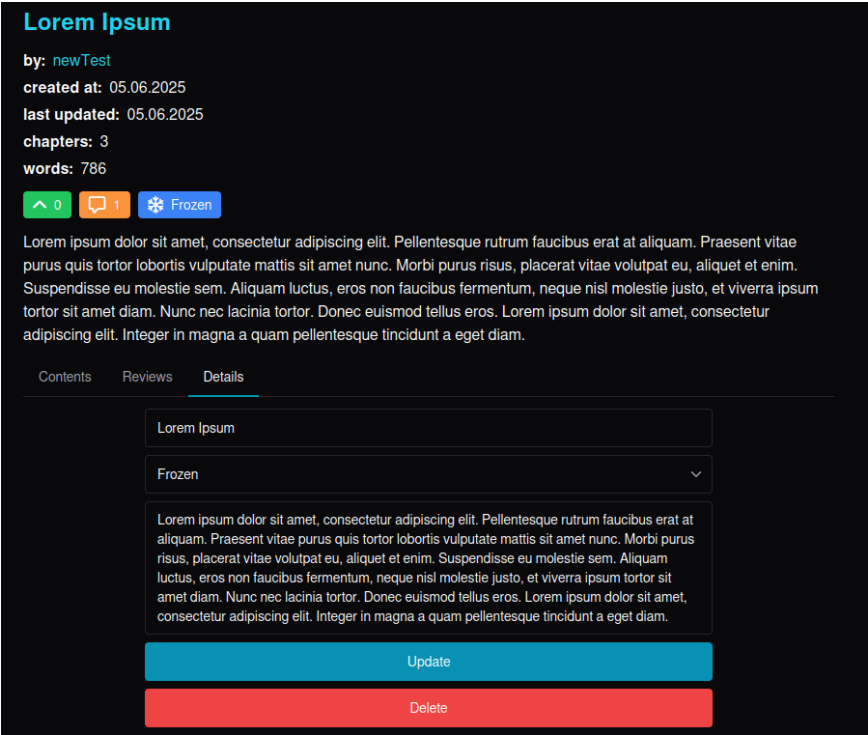


Рисунок 4.7 – Зміна деталей книги

При переході на сторінку змісту (рис. 4.8) та створення глави, бачимо сторінку редактором, та вводом назви глави, передмови та виноски.

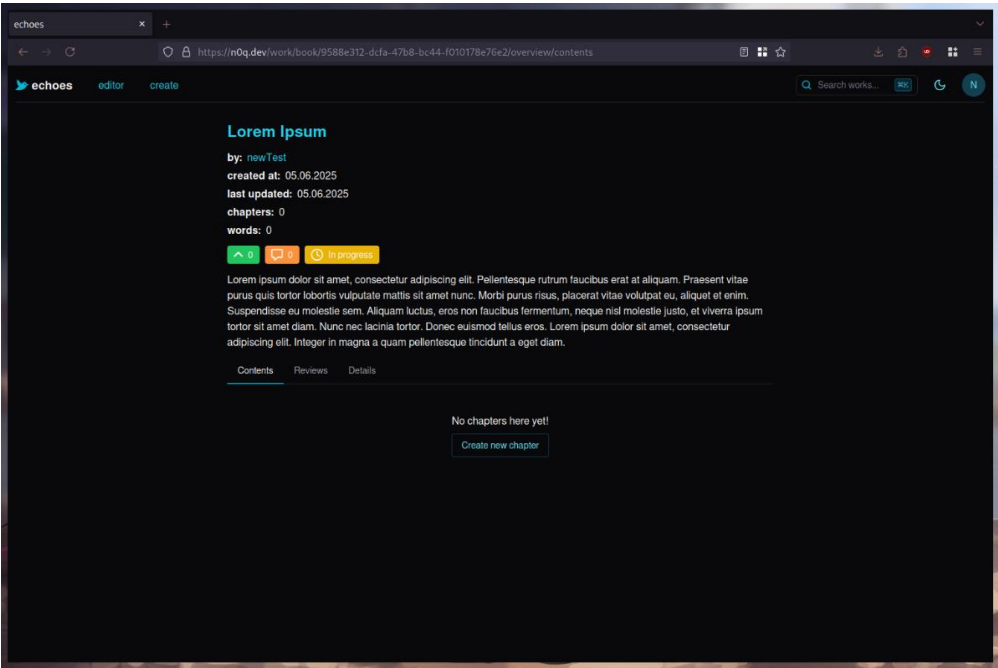


Рисунок 4.8 – Сторінка змісту книги

Редактор (рис. 4.9) пропонує багато варіантів форматування тексту, до яких входять базові операції, що можуть бути важливими для авторів.

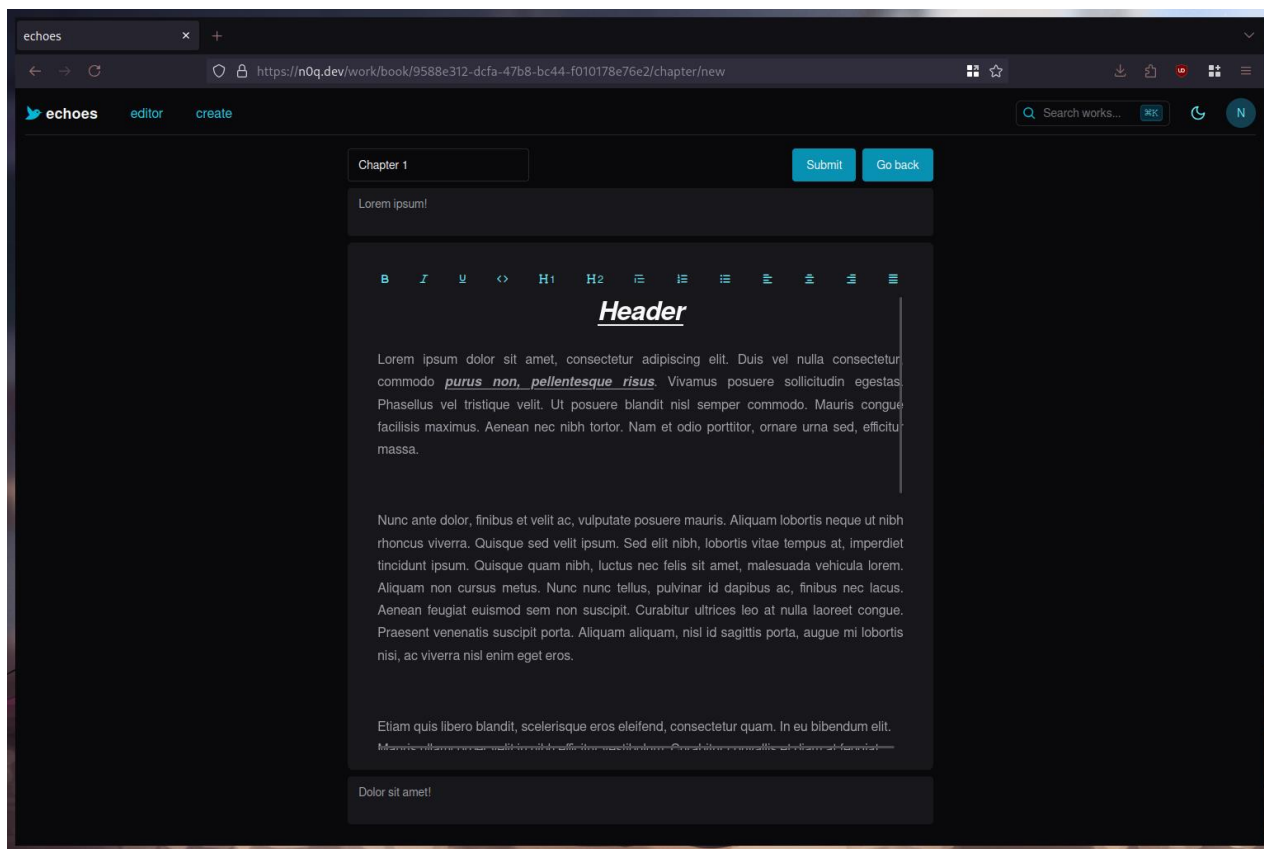


Рисунок 4.9 – Сторінка редакції та створення глав

Після заповнення даних глави та підтвердження нова глава створюється та користувач перенаправляється до сторінки змісту (рис. 4.10), де до створеної глави додають можливості зміни та видалення.

При переході до створеної глави (рис. 4.11), користувач побачить главу з усіма форматування та деталями, які були вказані користувачем, включаючи виноски та передмови.

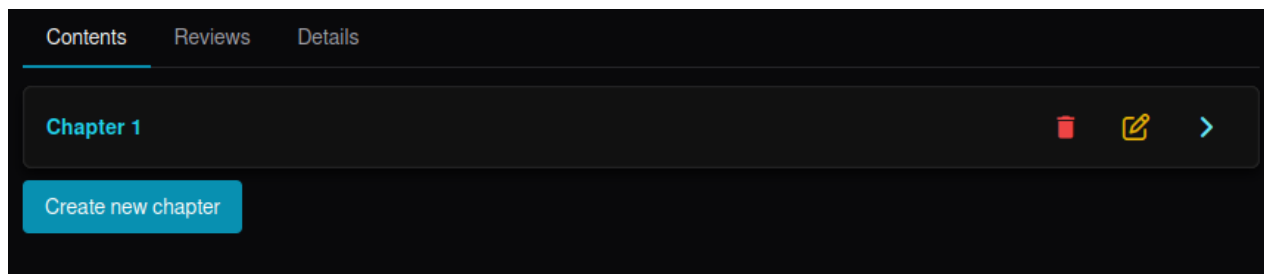


Рисунок 4.10 – Зміст зі створеною главою та опціями редагування

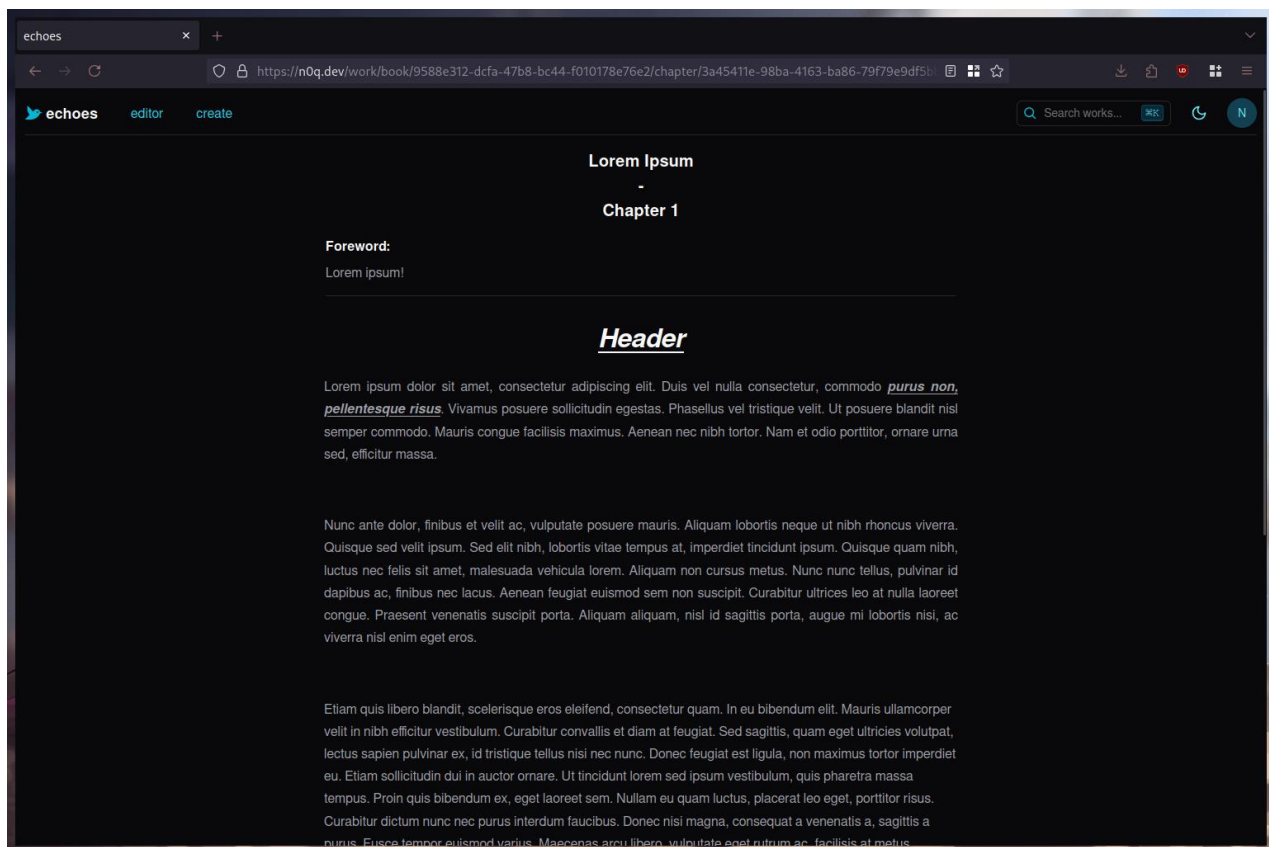


Рисунок 4.11 – Перегляд створеної глави

При досягненні кінця глави, користувач може перейти до змісту, до попередньої глави, або до наступної. Кнопки (рис. 4.12) є динамічними та є посиланнями на глави за їх унікальними ідентфікаторами, що дозволяє нам опціонально приховати, або показати кнопки з цієї панелі.

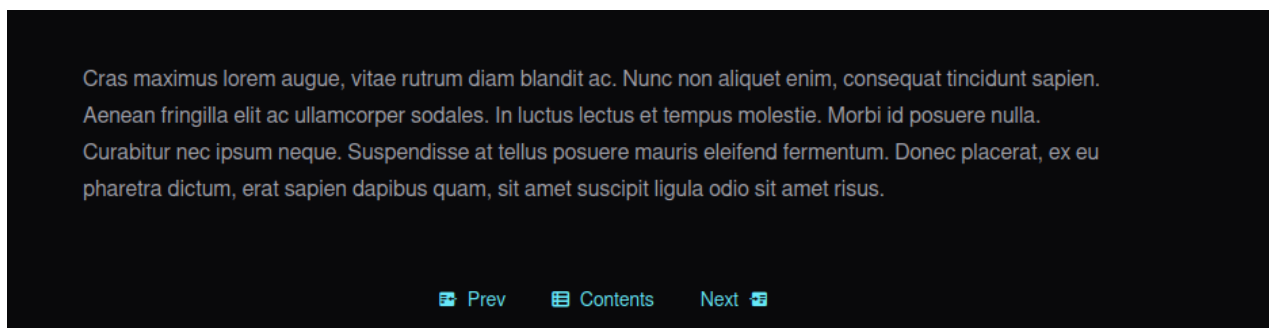


Рисунок 4.12 – Навігація по главам книги

					ІАЛП.467200.003 ПЗ	Арк.
						76
Зм.	Арк.	№ докум.	Підпис	Дата		

4.5. Система зворотнього зв'язку

Після прочитання, користувач може перейти до змісту книги, звідки він може поставити оцінку (не будучи автором книги), як за прикладом на рис. 4.13.

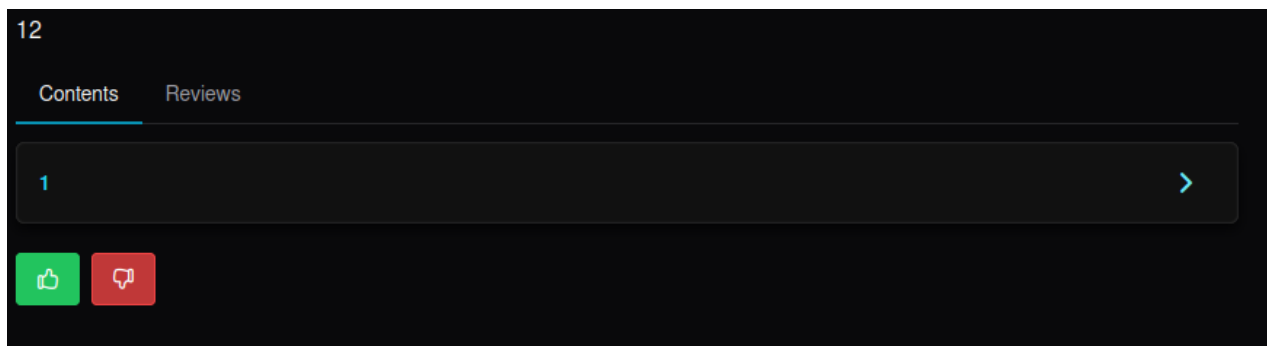


Рисунок 4.13 – Кнопки залишення оцінки

Користувач також може перейти до сторінки відгуків та залишити власну рецензію, щодо якості прочитаного твору (рис. 4.14).

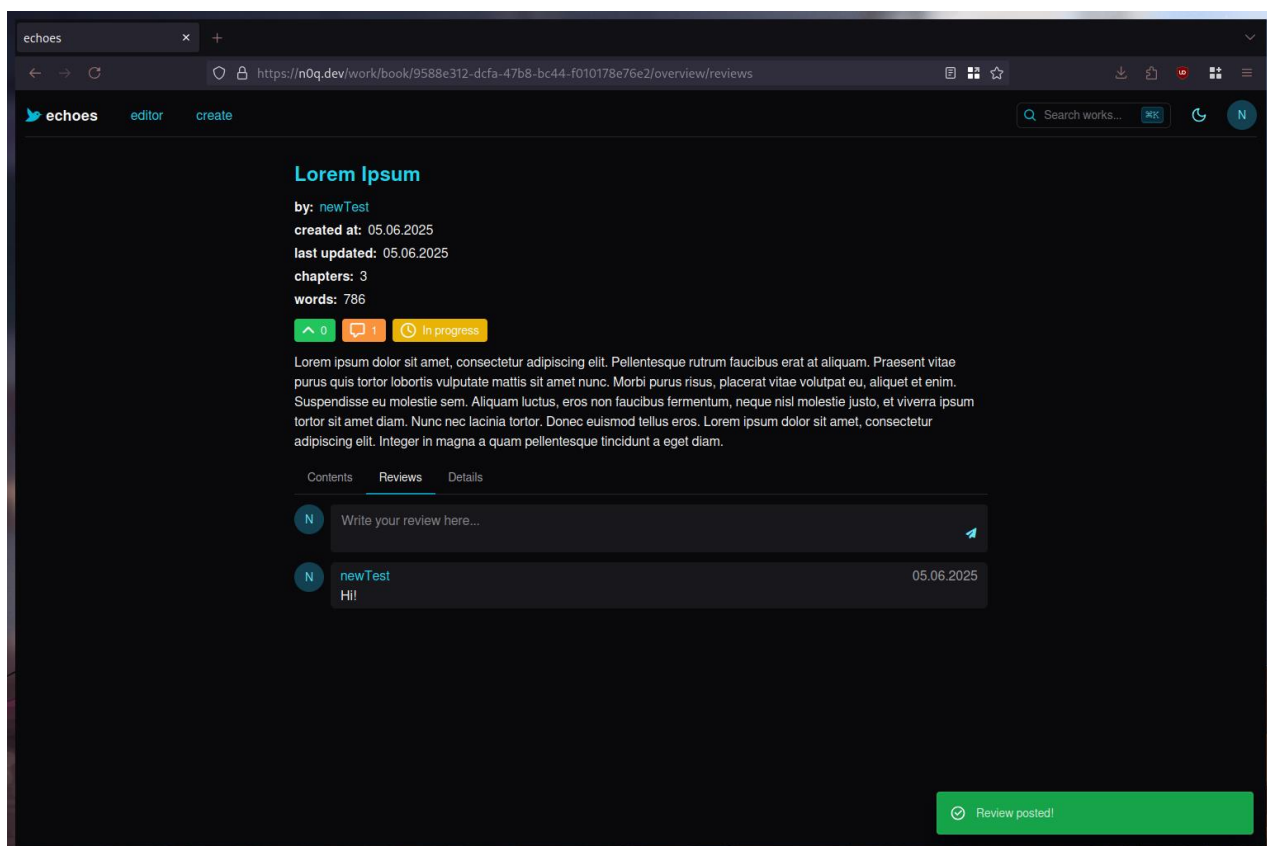


Рисунок 4.14 – Залишення відгуку

					ІАЛП.467200.003 ПЗ	Арк.
						77
Зм.	Арк.	№ докум.	Підпис	Дата		

Відгуки та книги, мають посилання на профіль їх автору, що містить обмежену інформацію про користувача (рис. 4.15).

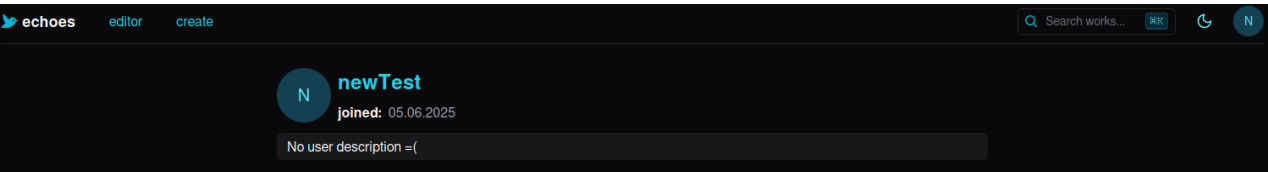


Рисунок 4.15 – Огляд профілю користувача

4.6. Вихід з облікового запису

У разі бажання користувача завершити сеанс, він може натиснути на власний профіль у секції Header, що відкриє меню, де користувач може перейти до власного профілю, або вийти з особистого профілю (рис. 4.16).

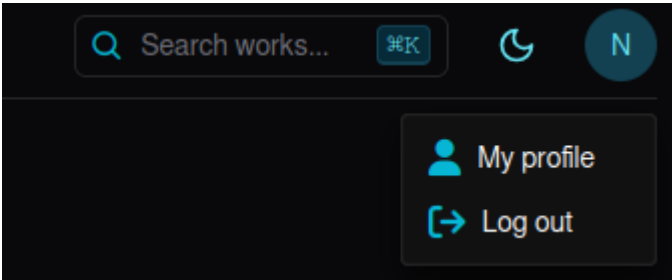


Рисунок 4.16 – Меню профілю

ВИСНОВОК ДО РОЗДІЛУ 4

У результаті дослідження функціональних можливостей веб-додатка було проаналізовано основні етапи взаємодії користувача з інтерфейсом системи та проведено оцінку реалізованого функціоналу. На головній сторінці користувачу доступні інструменти для пошуку та перегляду наявних творів, що забезпечується за допомогою пагінації та інтуїтивно зрозумілої навігації, розміщеної в компоненті Header.

Було підтверджено коректність реалізації системи аутентифікації та авторизації. Невідомий користувач має обмежений доступ до функціоналу платформи та може здійснювати лише перегляд доступних творів. Повноцінний доступ до створення, редагування, оцінювання та рецензування доступний лише після реєстрації або входу в систему.

Інтерфейс авторизації реалізований з урахуванням зручності користувача та містить вбудовану валідацію введених даних. Після успішної авторизації користувач отримує доступ до додаткових можливостей, таких як створення книг, редагування змісту та управління главами.

Редактор тексту забезпечує зручне форматування та надає користувачу всі необхідні інструменти для написання та структурування твору. Створені глави додаються до загального змісту книги з можливістю подальшого редагування або видалення.

Навігація між главами реалізована динамічно та враховує послідовність публікацій, що значно покращує користувацький досвід. Сторінки книг доповнені функціоналом оцінювання та залишення відгуків, що сприяє зворотному зв'язку між читачами та авторами.

Кожен елемент системи передбачає можливість переходу до профілю автора, де відображається обмежена, але інформативна інформація про

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		79

користувача. Також передбачено зручне завершення сесії через меню профілю у верхній навігаційній панелі.

					ІАЛЦ.467200.003 ПЗ	Арк.
						80
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У результаті виконання дипломної роботи було спроектовано, реалізовано та проаналізовано веб-додаток, який забезпечує функціонал для створення, редагування, публікації та взаємодії з літературними творами. В процесі розробки було розглянуто сучасні технології клієнтської та серверної частин застосунку, зокрема було використано GraphQL як інструмент комунікації між фронтендом і бекендом, Chakra UI — як гнучку бібліотеку інтерфейсних компонентів, а також TanStack Router — для організації маршрутизації на клієнті.

Розроблена система передбачає авторизацію користувачів з подальшим розмежуванням прав доступу до функціоналу, підтримує створення книг та їх структурування у вигляді глав із можливістю форматування тексту. Користувачі мають змогу залишати оцінки та відгуки до прочитаних творів, що сприяє розвитку інтерактивності платформи та покращенню взаємодії між авторами і читачами.

Під час реалізації було враховано питання збереження сесій, обробки помилок, валідації форм та безпеки зберігання даних. Значна увага приділялася зручності користувацького інтерфейсу, адаптивності додатку та загальній узгодженості дизайну.

Отриманий результат підтвердив ефективність обраних архітектурних рішень та інструментів, а також продемонстрував, що реалізована система здатна вирішувати поставлені задачі в рамках розробки платформи для публікації авторських творів. Робота може бути розширена у майбутньому шляхом інтеграції нових функцій, таких як система рекомендацій, приватні повідомлення або модерація контенту.

Загалом, поставлені цілі дипломної роботи були досягнуті, а розроблена система відповідає сучасним вимогам до якості веб-додатків.

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		81

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amazon Kindle: Your free personal library you can take anywhere. Amazon.com. URL: <https://read.amazon.com/landing> (date of access: 24.05.2025).
2. Chakra UI. Chakra UI. URL: <https://www.chakra-ui.com/> (date of access: 24.05.2025).
3. Draft2Digital. Draft2Digital. URL: <https://draft2digital.com/> (date of access: 24.05.2025).
4. GitHub. GraphQL (The Guild). URL: <https://the-guild.dev/graphql/codegen> (date of access: 24.05.2025).
5. Gumroad. Earn your first dollar online with Gumroad. URL: <https://gumroad.com/> (date of access: 24.05.2025).
6. Home | Royal Road. Royal Road. URL: <https://www.royalroad.com> (date of access: 24.05.2025).
7. JavaScript With Syntax For Types. TypeScript: JavaScript With Syntax For Types. URL: <https://www.typescriptlang.org> (date of access: 24.05.2025).
8. MariaDB Foundation - MariaDB.org. MariaDB.org. URL: <https://mariadb.org/> (date of access: 24.05.2025).
9. Material UI: React components that implement Material Design. MUI: The React component library you always wanted. URL: <https://mui.com/material-ui/> (date of access: 24.05.2025).
10. Self Publishing | Amazon Kindle Direct Publishing. Self Publishing | Amazon Kindle Direct Publishing. URL: https://kdp.amazon.com/en_US (date of access: 24.05.2025).
11. Streamlining APIs, Databases, & Microservices | Apollo GraphQL. Streamlining APIs, Databases, & Microservices | Apollo GraphQL. URL: <https://www.apollographql.com/> (date of access: 24.05.2025).

					ІАЛП.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		82

12. TanStack | High Quality Open-Source Software for Web Developers. TanStack | High Quality Open-Source Software for Web Developers. URL: <https://tanstack.com/> (date of access: 24.05.2025).
13. Wattpad - Where stories live. Wattpad - Where stories live. URL: <https://www.wattpad.com/> (date of access: 24.05.2025).
14. GitHub - chakra-ui/chakra-ui: Chakra UI is a component system for building SaaS products with speed. GitHub Chakra-UI. URL: <https://github.com/chakra-ui/chakra-ui> (date of access: 24.05.2025).

					ІАЛШ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		83

ДОДАТОК А

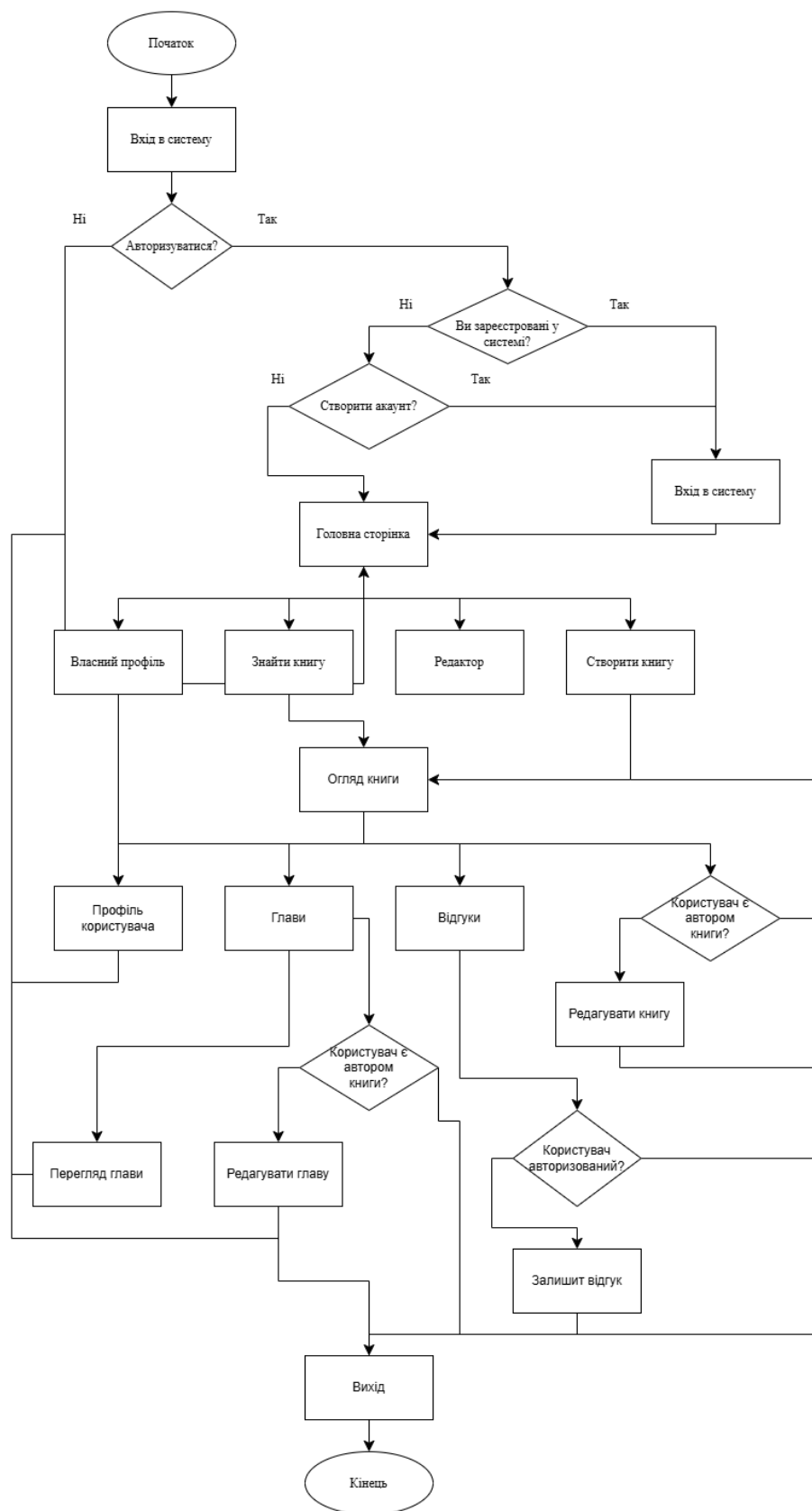
Платформа для написання та публікації робіт для письменників

Схема алгоритму роботи системи

ІАЛЦ.467200.004 ДА

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 ДА			
		№ докум.	Підпис	Дата				
Розробив	Денисенко Б.Р.				Платформа для написання та публікації робіт для письменників Схема алгоритму роботи системи	Літ.	Аркуш	Аркушів
Перевірив	Нікольський С.С.						1	1
Реценз.	Радченко К.О.					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Н. Контр.	Нікольський С.С.							
Затвердив								

ДОДАТОК Б

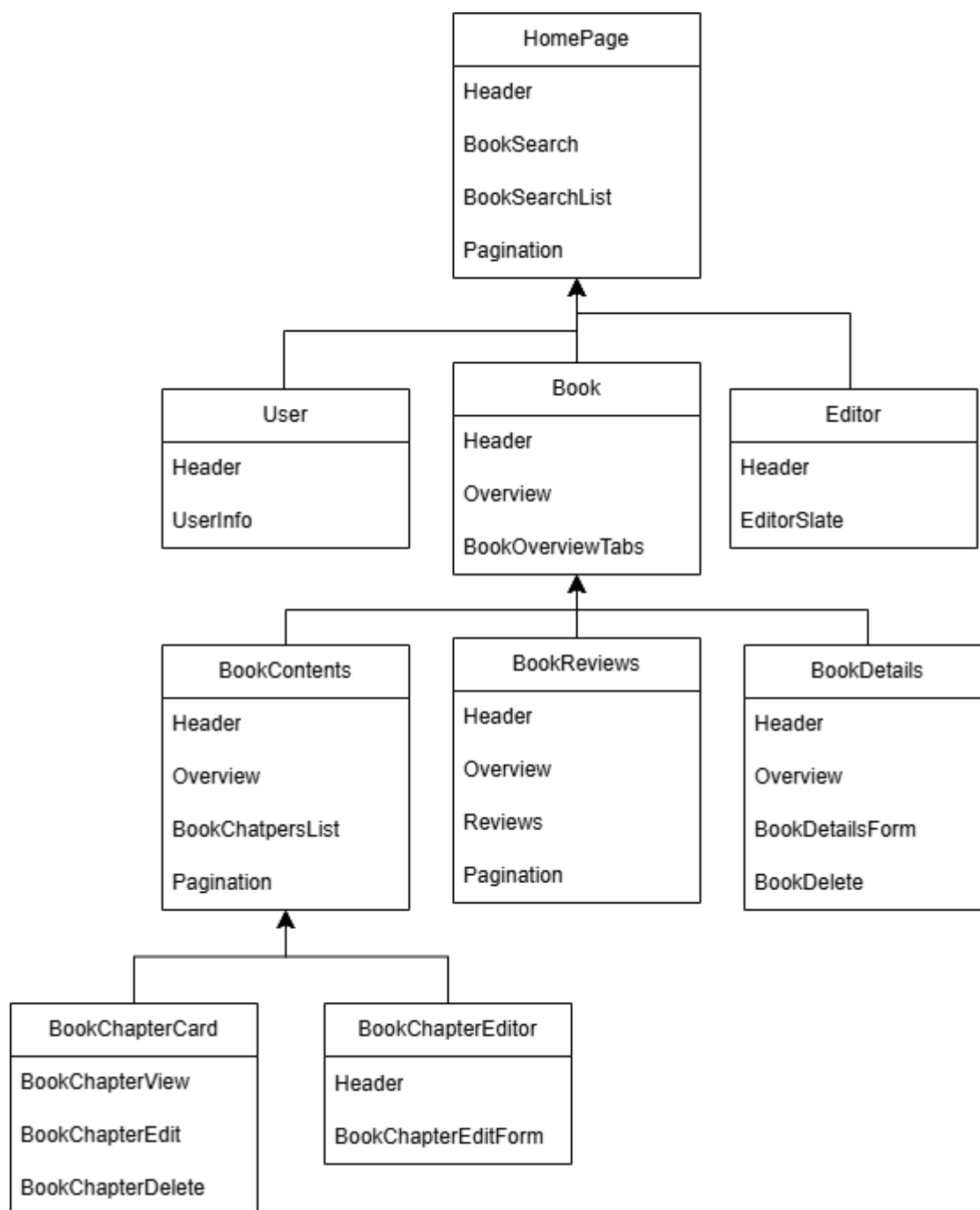
Платформа для написання та публікації робіт для письменників

Структурна схема веб-застосунку

ІАЛЦ.467200.005 ДБ

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 ДБ			
		№ докум.	Підпис	Дата	Платформа для написання та публікації робіт для письменників Структурна схема веб-застосунку			
Розробив	Денисенко Б.Р.							
Перевірив	Нікольський С.С.							
Реценз.	Радченко К.О.							
Н. Контр.	Нікольський С.С.							
Затвердив					Літ. Аркуш Аркушів 1 1 НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11			

ДОДАТОК В

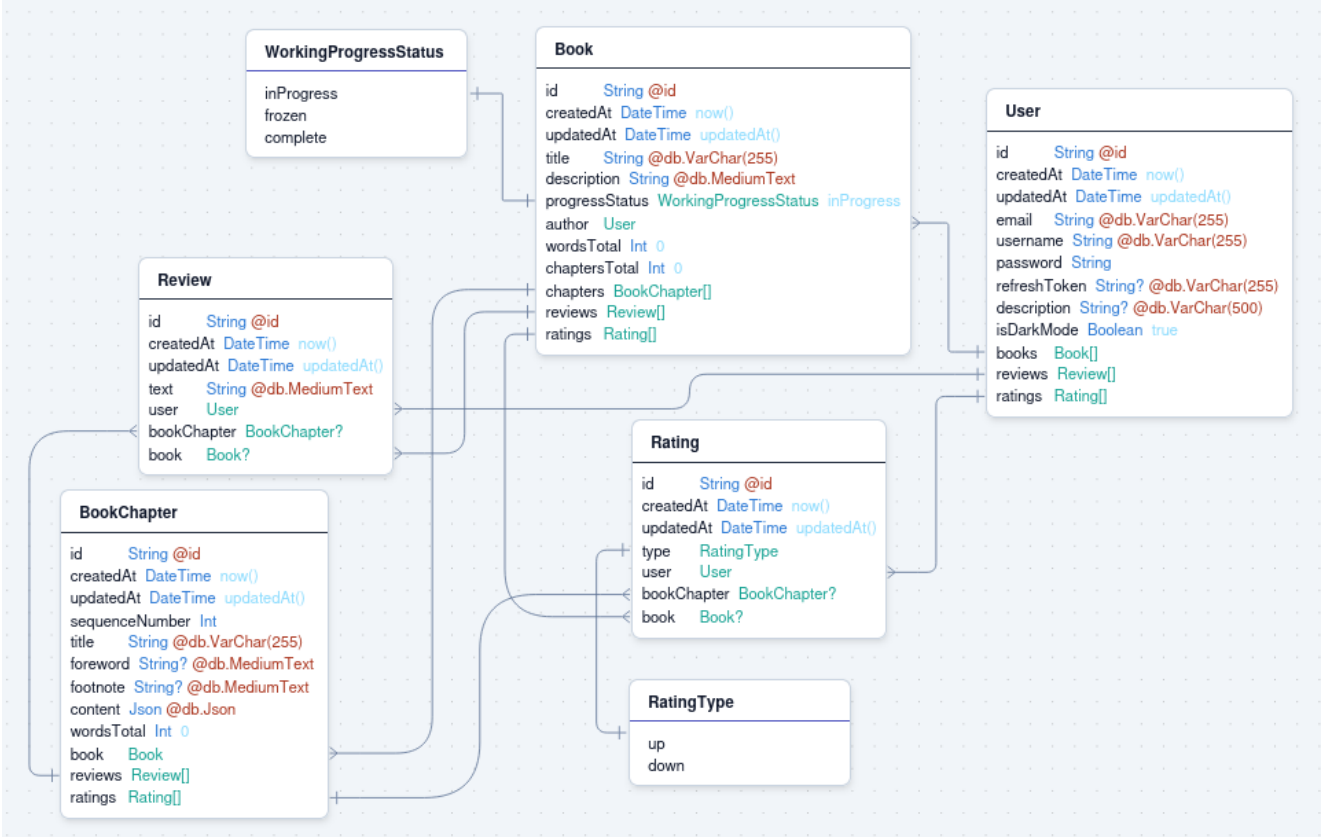
Платформа для написання та публікації робіт для письменників

Схема бази даних

ІАЛЦ.467200.006 ДВ

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 ДВ			
		№ докум.	Підпис	Дата				
Розробив	Денисенко Б.Р.				Платформа для написання та публікації робіт для письменників Схема бази даних	Літ.	Аркуш	Аркушів
Перевірив	Нікольський С.С.						1	1
Реценз.	Радченко К.О.					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Н. Контр.	Нікольський С.С.							
Затвердив								

Додаток Г

Платформа для написання та публікації робіт для письменників

Текст програмного коду

ІАЛЦ.467200.007 ДГ

Аркушів 114

Київ 2025 р

```
-- tsr.config.json --
{
  "routesDirectory": "./app/client/pages",
  "generatedRouteTree": "./app/client/routeTree.gen.ts",
  "routeFileIgnorePrefix": "-",
  "quoteStyle": "single"
}

-- package.json --
{
  "name": "ln",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "VITE_MODE=dev nodemon -w app/server -x tsx app/server/main.ts",
    "start": "NODE_ENV=production VITE_MODE=prod tsx app/server/main.ts",
    "build": "vite build",
    "genssl": "openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout ./certs/selfsigned.key -out ./certs/selfsigned.crt",
    "gql-types": "graphql-codegen --config codegen.ts",
    "checkup": "opinionated checkup",
    "prisma": "prisma",
    "chakra": "chakra"
  },
  "dependencies": {
    "@apollo/client": "^3.12.2",
    "@apollo/server": "^4.11.2",
    "@chakra-ui/react": "^3.15.0",
    "@chakra-ui/toast": "^7.0.2",
    "@coolbytes/editorjs-link": "^1.0.0",
    "@deftomat/opinionated": "^1.1.3",
    "@emotion/react": "^11.13.5",
    "@hookform/resolvers": "^3.9.1",
    "@prisma/client": "^6.4.1",
    "@tanstack/react-router": "^1.85.0",
    "@tanstack/router-vite-plugin": "^1.84.4",
    "apollo-link-logger": "^2.0.1",
    "bcrypt": "^5.1.1",
    "cookie-parser": "^1.4.7",
    "cors": "^2.8.5",
    "dayjs": "^1.11.13",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "graphql": "^16.9.0",
    "graphql-scalars": "^1.24.2",
    "is-hotkey": "^0.2.0",
    "jsonwebtoken": "^9.0.2",
    "next-themes": "^0.4.4",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-hook-form": "^7.54.2",
    "react-icons": "^5.5.0",
    "slate": "^0.114.0",
    "slate-history": "^0.113.1",
    "slate-react": "^0.114.2",
  }
}
```

					ІАЛЦ.467200.007 ДГ			
		№ докум.	Підпис	Дата	Платформа для написання та публікації робіт для письменників Текст програмного коду	Літ.	Арк.	Аркушів
Розробив	Денисенко Б.Р.						1	114
Перевірив	Нікольський С.С.					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Реценз.	Радченко К.О.							
Н. контр.	Нікольський С.С.							
Затвердив								

```

"tsx": "^4.3.0",
  "typescript": "^5.3.2",
  "uuid": "^11.1.0",
  "valibot": "^1.0.0-beta.10",
  "vite-express": "^0.20.0"
},
"@graphql-codegen/cli": "5.0.3",
"@graphql-codegen/client-preset": "4.5.1",
"@graphql-codegen/introspection": "4.0.3",
"@graphql-codegen/typescript": "^4.1.2",
"@graphql-codegen/typescript-resolvers": "^4.4.1",
"@tanstack/router-devtools": "^1.85.0",
"@tanstack/router-plugin": "^1.84.4",
"@types/bcrypt": "^5.0.2",
"@types/cookie-parser": "^1.4.8",
"@types/cors": "^2.8.17",
"@types/express": "^4.17.21",
"@types/is-hotkey": "^0.1.10",
"@types/jsonwebtoken": "^9.0.9",
"@types/node": "^20.9.3",
"@types/react": "^18.0.38",
"@types/react-dom": "^18.2.16",
"@vitejs/plugin-react": "^4.2.0",
"nodemon": "^3.0.1",
"prisma": "^6.4.1",
"vite": "^5.0.2",
"vite-tsconfig-paths": "^5.1.4"
},
"prettier": "@deftomat/opinionated/configs/prettier.config.mjs",
"prisma": {
  "schema": "app/server/prisma/schema.prisma"
}
}

```

```

-- .gitignore --
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

node_modules
dist
dist-ssr
certs/*
*.local
.env
__generated__/
vite.config.ts.timestamp-*.mjs

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?

```

					ІАЛЦ.467200.007 ДГ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

```

app/client/components/ui
app/client/assets

vite.config.ts.timestamp-*

dist

__generated__/

certs

-- .editorconfig --
# http://EditorConfig.org

[*]
charset = utf-8
end_of_line = lf
indent_size = 2
indent_style = space
insert_final_newline = true
trim_trailing_whitespace = true

-- tsconfig.json --
{
  "compilerOptions": {
    "target": "ESNext",
    "useDefineForClassFields": true,
    "lib": ["DOM", "DOM.Iterable", "ESNext"],
    "allowJs": false,
    "jsx": "react-jsx",
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "module": "ESNext",
    "moduleResolution": "Bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "paths": {
      "@app/*": ["/app/*"],
      "@client/*": ["/app/client/*"],
      "@server/*": ["/app/server/*"],
      "@generated/*": ["/__generated__/*"],
      "@root/*": ["/.*"]
    }
  },
  "exclude": ["/app/client/assets", "/__generated__/*", "/dist"]
}

-- index.html --
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/ln-logo.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>echoes</title>
  </head>
  <body>

```

					ІАЛЦ.467200.007 ДГ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    <div id="root"></div>
    <script type="module" src="/app/client/main.tsx"></script>
  </body>
</html>

```

```

-- .example.env --
VITE_MODE="dev"
VITE_HOST=192.168.1.2
VITE_HOST_REMOTE=123.123.123.123
VITE_DOMAIN_NAME=n0q.dev
VITE_PORT=443
VITE_APP_NAME="echoes"
ACCESS_TOKEN_SECRET="encoded string"
REFRESH_TOKEN_SECRET="encoded string"
DATABASE_URL="mysql://user@localhost:3306/project_ln"

```

```

-- vite.config.ts --
// vite.config.ts
import { defineConfig } from 'vite';
import viteReact from '@vitejs/plugin-react';
import { TanStackRouterVite } from '@tanstack/router-plugin/vite';
import tsConfigPaths from 'vite-tsconfig-paths';

// https://vitejs.dev/config/
export default defineConfig({
  build: {
    sourcemap: true
  },
  plugins: [TanStackRouterVite(), viteReact(), tsConfigPaths()],
  server: {
    allowedHosts: ['n0q.dev']
  }
});

```

```

-- codegen.ts --
import type { CodegenConfig } from '@graphql-codegen/cli';

const config: CodegenConfig = {
  overwrite: true,

  schema: './app/server/graphql/schema.graphql',
  documents: ['app/client/fragments', 'app/client/**/*.tsx'],
  generates: {
    './__generated__': {
      preset: 'client',
      presetConfig: {
        gqlTagName: 'gql'
      },
      plugins: ['typescript', 'typescript-resolvers', 'typescript-resolvers']
    },
    './__generated__/types.ts': {
      plugins: ['typescript', 'typescript-operations', 'typescript-resolvers'],
      config: {
        scalars: {
          Void: 'void'
        }
      }
    },
    './__generated__/graphql.schema.json': {
      plugins: ['introspection']
    }
  },
};

```

					ІАЛЦ.467200.007 ДГ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    ignoreNoDocuments: true
  };

export default config;

-- app/client/types.d.ts --
import { Descendant, BaseEditor, BaseRange, Range, Element } from 'slate';
import { ReactEditor } from 'slate-react';
import { HistoryEditor } from 'slate-history';

export type BlockQuoteElement = {
  type: 'block-quote';
  align?: string;
  children: Descendant[];
};

export type BulletedListElement = {
  type: 'bulleted-list';
  align?: string;
  children: Descendant[];
};

export type CheckListItemElement = {
  type: 'check-list-item';
  checked: boolean;
  children: Descendant[];
};

export type EditableVoidElement = {
  type: 'editable-void';
  children: EmptyText[];
};

export type HeadingElement = {
  type: 'heading-one';
  align?: string;
  children: Descendant[];
};

export type HeadingTwoElement = {
  type: 'heading-two';
  align?: string;
  children: Descendant[];
};

export type HeadingThreeElement = {
  type: 'heading-three';
  align?: string;
  children: Descendant[];
};

export type HeadingFourElement = {
  type: 'heading-four';
  align?: string;
  children: Descendant[];
};

export type HeadingFiveElement = {
  type: 'heading-five';
  align?: string;
  children: Descendant[];
};

export type HeadingSixElement = {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    type: 'heading-six';
    align?: string;
    children: Descendant[];
};

export type ImageElement = {
    type: 'image';
    url: string;
    children: EmptyText[];
};

export type LinkElement = { type: 'link'; url: string; children: Descendant[] };

export type ButtonElement = { type: 'button'; children: Descendant[] };

export type BadgeElement = { type: 'badge'; children: Descendant[] };

export type ListItemElement = { type: 'list-item'; children: Descendant[] };

export type NumberedListItemElement = {
    type: 'numbered-list';
    children: Descendant[];
};

export type MentionElement = {
    type: 'mention';
    character: string;
    children: CustomText[];
};

export type ParagraphElement = {
    type: 'paragraph';
    align?: string;
    children: Descendant[];
};

export type TableElement = { type: 'table'; children: TableRow[] };

export type TableCellElement = { type: 'table-cell'; children: CustomText[] };

export type TableRowElement = { type: 'table-row'; children: TableCell[] };

export type TitleElement = { type: 'title'; children: Descendant[] };

export type VideoElement = { type: 'video'; url: string; children: EmptyText[] };

export type CodeBlockElement = {
    type: 'code-block';
    language: string;
    children: Descendant[];
};

export type CodeLineElement = {
    type: 'code-line';
    children: Descendant[];
};

export type CustomElementWithAlign =
    | ParagraphElement
    | HeadingElement
    | HeadingTwoElement
    | HeadingThreeElement
    | HeadingFourElement
    | HeadingFiveElement
    | HeadingSixElement

```

					ІАЛЦ.467200.007 ДГ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

```

| BlockQuoteElement
| BulletedListElement;

type CustomElement =
| BlockQuoteElement
| BulletedListElement
| CheckListItemElement
| EditableVoidElement
| HeadingElement
| HeadingTwoElement
| HeadingThreeElement
| HeadingFourElement
| HeadingFiveElement
| HeadingSixElement
| ImageElement
| LinkElement
| ButtonElement
| BadgeElement
| ListItemElement
| NumberedListItemElement
| MentionElement
| ParagraphElement
| TableElement
| TableRowElement
| TableCellElement
| TitleElement
| VideoElement
| CodeBlockElement
| CodeLineElement;

export type CustomElementType = CustomElement['type'];

export type CustomText = {
  bold?: boolean;
  italic?: boolean;
  code?: boolean;
  underline?: boolean;
  strikethrough?: boolean;
  // MARKDOWN PREVIEW SPECIFIC LEAF
  underlined?: boolean;
  title?: boolean;
  list?: boolean;
  hr?: boolean;
  blockquote?: boolean;
  text: string;
};

export type CustomTextKey = keyof Omit<CustomText, 'text'>;

export type EmptyText = {
  text: string;
};

export type RenderElementPropsFor<T> = RenderElementProps & {
  element: T;
};

export type CustomEditor = BaseEditor &
  ReactEditor &
  HistoryEditor & {
    nodeToDecorations?: Map<Element, Range[]>;
  };

declare module 'slate' {
  interface CustomTypes {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    Editor: CustomEditor;
    Element: CustomElement;
    Text: CustomText;
    Range: BaseRange & {
      [key: string]: unknown;
    };
  }
}

-- app/client/tsconfig.json --
{
  "extends": "../../tsconfig.json",
  "compilerOptions": {
    "module": "ESNext",
    "moduleResolution": "Bundler",
    "jsx": "react-jsx"
  }
}

-- app/client/routeTree.gen.ts --
/* eslint-disable */

// @ts-nocheck

// noinspection JSUnusedGlobalSymbols

// This file was automatically generated by TanStack Router.
// You should NOT make any changes in this file as it will be overwritten.
// Additionally, you should also exclude this file from your linter and/or formatter to
// prevent it from being checked or modified.

// Import Routes

import { Route as rootRoute } from './pages/__root'
import { Route as RegisterImport } from './pages/register'
import { Route as LoginImport } from './pages/login'
import { Route as EditorImport } from './pages/editor'
import { Route as AboutImport } from './pages/about'
import { Route as IndexImport } from './pages/index'
import { Route as ProfileUsernameImport } from './pages/profile/$username'
import { Route as NewBookImport } from './pages/new/book'
import { Route as WorkBookBookIdOverviewImport } from './pages/work/book/$bookId/overview'
import { Route as WorkBookBookIdOverviewReviewsImport } from
'./pages/work/book/$bookId/overview/reviews'
import { Route as WorkBookBookIdOverviewDetailsImport } from
'./pages/work/book/$bookId/overview/details'
import { Route as WorkBookBookIdOverviewContentsImport } from
'./pages/work/book/$bookId/overview/contents'
import { Route as WorkBookBookIdChapterNewImport } from
'./pages/work/book/$bookId/chapter/new'
import { Route as WorkBookBookIdChapterChapterIdViewImport } from
'./pages/work/book/$bookId/chapter/$chapterId/view'
import { Route as WorkBookBookIdChapterChapterIdEditImport } from
'./pages/work/book/$bookId/chapter/$chapterId/edit'

// Create/Update Routes

const RegisterRoute = RegisterImport.update({
  id: '/register',
  path: '/register',
  getParentRoute: () => rootRoute,
} as any)

```

					ІАЛЦ.467200.007 ДГ	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const LoginRoute = LoginImport.update({
  id: '/login',
  path: '/login',
  getParentRoute: () => rootRoute,
} as any)

const EditorRoute = EditorImport.update({
  id: '/editor',
  path: '/editor',
  getParentRoute: () => rootRoute,
} as any)

const AboutRoute = AboutImport.update({
  id: '/about',
  path: '/about',
  getParentRoute: () => rootRoute,
} as any)

const IndexRoute = IndexImport.update({
  id: '/',
  path: '/',
  getParentRoute: () => rootRoute,
} as any)

const ProfileUsernameRoute = ProfileUsernameImport.update({
  id: '/profile/$username',
  path: '/profile/$username',
  getParentRoute: () => rootRoute,
} as any)

const NewBookRoute = NewBookImport.update({
  id: '/new/book',
  path: '/new/book',
  getParentRoute: () => rootRoute,
} as any)

const WorkBookBookIdOverviewRoute = WorkBookBookIdOverviewImport.update({
  id: '/work/book/$bookId/overview',
  path: '/work/book/$bookId/overview',
  getParentRoute: () => rootRoute,
} as any)

const WorkBookBookIdOverviewReviewsRoute =
  WorkBookBookIdOverviewReviewsImport.update({
    id: '/reviews',
    path: '/reviews',
    getParentRoute: () => WorkBookBookIdOverviewRoute,
  } as any)

const WorkBookBookIdOverviewDetailsRoute =
  WorkBookBookIdOverviewDetailsImport.update({
    id: '/details',
    path: '/details',
    getParentRoute: () => WorkBookBookIdOverviewRoute,
  } as any)

const WorkBookBookIdOverviewContentsRoute =
  WorkBookBookIdOverviewContentsImport.update({
    id: '/contents',
    path: '/contents',
    getParentRoute: () => WorkBookBookIdOverviewRoute,
  } as any)

const WorkBookBookIdChapterNewRoute = WorkBookBookIdChapterNewImport.update({
  id: '/work/book/$bookId/chapter/new',

```

					ІАЛЦ.467200.007 ДГ	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    path: '/work/book/$bookId/chapter/new',
    getParentRoute: () => rootRoute,
  } as any)

const WorkBookBookIdChapterChapterIdViewRoute =
  WorkBookBookIdChapterChapterIdViewImport.update({
    id: '/work/book/$bookId/chapter/$chapterId/view',
    path: '/work/book/$bookId/chapter/$chapterId/view',
    getParentRoute: () => rootRoute,
  } as any)

const WorkBookBookIdChapterChapterIdEditRoute =
  WorkBookBookIdChapterChapterIdEditImport.update({
    id: '/work/book/$bookId/chapter/$chapterId/edit',
    path: '/work/book/$bookId/chapter/$chapterId/edit',
    getParentRoute: () => rootRoute,
  } as any)

// Populate the FileRoutesByPath interface

declare module '@tanstack/react-router' {
  interface FileRoutesByPath {
    '/': {
      id: '/'
      path: '/'
      fullPath: '/'
      preLoaderRoute: typeof IndexImport
      parentRoute: typeof rootRoute
    }
    '/about': {
      id: '/about'
      path: '/about'
      fullPath: '/about'
      preLoaderRoute: typeof AboutImport
      parentRoute: typeof rootRoute
    }
    '/editor': {
      id: '/editor'
      path: '/editor'
      fullPath: '/editor'
      preLoaderRoute: typeof EditorImport
      parentRoute: typeof rootRoute
    }
    '/login': {
      id: '/login'
      path: '/login'
      fullPath: '/login'
      preLoaderRoute: typeof LoginImport
      parentRoute: typeof rootRoute
    }
    '/register': {
      id: '/register'
      path: '/register'
      fullPath: '/register'
      preLoaderRoute: typeof RegisterImport
      parentRoute: typeof rootRoute
    }
    '/new/book': {
      id: '/new/book'
      path: '/new/book'
      fullPath: '/new/book'
      preLoaderRoute: typeof NewBookImport
      parentRoute: typeof rootRoute
    }
    '/profile/$username': {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    id: '/profile/$username'
    path: '/profile/$username'
    fullPath: '/profile/$username'
    preLoaderRoute: typeof ProfileUsernameImport
    parentRoute: typeof rootRoute
  }
  '/work/book/$bookId/overview': {
    id: '/work/book/$bookId/overview'
    path: '/work/book/$bookId/overview'
    fullPath: '/work/book/$bookId/overview'
    preLoaderRoute: typeof WorkBookBookIdOverviewImport
    parentRoute: typeof rootRoute
  }
  '/work/book/$bookId/chapter/new': {
    id: '/work/book/$bookId/chapter/new'
    path: '/work/book/$bookId/chapter/new'
    fullPath: '/work/book/$bookId/chapter/new'
    preLoaderRoute: typeof WorkBookBookIdChapterNewImport
    parentRoute: typeof rootRoute
  }
  '/work/book/$bookId/overview/contents': {
    id: '/work/book/$bookId/overview/contents'
    path: '/contents'
    fullPath: '/work/book/$bookId/overview/contents'
    preLoaderRoute: typeof WorkBookBookIdOverviewContentsImport
    parentRoute: typeof WorkBookBookIdOverviewImport
  }
  '/work/book/$bookId/overview/details': {
    id: '/work/book/$bookId/overview/details'
    path: '/details'
    fullPath: '/work/book/$bookId/overview/details'
    preLoaderRoute: typeof WorkBookBookIdOverviewDetailsImport
    parentRoute: typeof WorkBookBookIdOverviewImport
  }
  '/work/book/$bookId/overview/reviews': {
    id: '/work/book/$bookId/overview/reviews'
    path: '/reviews'
    fullPath: '/work/book/$bookId/overview/reviews'
    preLoaderRoute: typeof WorkBookBookIdOverviewReviewsImport
    parentRoute: typeof WorkBookBookIdOverviewImport
  }
  '/work/book/$bookId/chapter/$chapterId/edit': {
    id: '/work/book/$bookId/chapter/$chapterId/edit'
    path: '/work/book/$bookId/chapter/$chapterId/edit'
    fullPath: '/work/book/$bookId/chapter/$chapterId/edit'
    preLoaderRoute: typeof WorkBookBookIdChapterChapterIdEditImport
    parentRoute: typeof rootRoute
  }
  '/work/book/$bookId/chapter/$chapterId/view': {
    id: '/work/book/$bookId/chapter/$chapterId/view'
    path: '/work/book/$bookId/chapter/$chapterId/view'
    fullPath: '/work/book/$bookId/chapter/$chapterId/view'
    preLoaderRoute: typeof WorkBookBookIdChapterChapterIdViewImport
    parentRoute: typeof rootRoute
  }
}
}
}

// Create and export the route tree

interface WorkBookBookIdOverviewRouteChildren {
  WorkBookBookIdOverviewContentsRoute: typeof WorkBookBookIdOverviewContentsRoute
  WorkBookBookIdOverviewDetailsRoute: typeof WorkBookBookIdOverviewDetailsRoute
  WorkBookBookIdOverviewReviewsRoute: typeof WorkBookBookIdOverviewReviewsRoute
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const WorkBookBookIdOverviewRouteChildren: WorkBookBookIdOverviewRouteChildren =
{
    WorkBookBookIdOverviewContentsRoute: WorkBookBookIdOverviewContentsRoute,
    WorkBookBookIdOverviewDetailsRoute: WorkBookBookIdOverviewDetailsRoute,
    WorkBookBookIdOverviewReviewsRoute: WorkBookBookIdOverviewReviewsRoute,
}

const WorkBookBookIdOverviewRouteWithChildren =
    WorkBookBookIdOverviewRoute._addFileChildren(
        WorkBookBookIdOverviewRouteChildren,
    )

export interface FileRoutesByFullPath {
    '/': typeof IndexRoute
    '/about': typeof AboutRoute
    '/editor': typeof EditorRoute
    '/login': typeof LoginRoute
    '/register': typeof RegisterRoute
    '/new/book': typeof NewBookRoute
    '/profile/$username': typeof ProfileUsernameRoute
    '/work/book/$bookId/overview': typeof WorkBookBookIdOverviewRouteWithChildren
    '/work/book/$bookId/chapter/new': typeof WorkBookBookIdChapterNewRoute
    '/work/book/$bookId/overview/contents': typeof WorkBookBookIdOverviewContentsRoute
    '/work/book/$bookId/overview/details': typeof WorkBookBookIdOverviewDetailsRoute
    '/work/book/$bookId/overview/reviews': typeof WorkBookBookIdOverviewReviewsRoute
    '/work/book/$bookId/chapter/$chapterId/edit': typeof
WorkBookBookIdChapterChapterIdEditRoute
    '/work/book/$bookId/chapter/$chapterId/view': typeof
WorkBookBookIdChapterChapterIdViewRoute
}

export interface FileRoutesByTo {
    '/': typeof IndexRoute
    '/about': typeof AboutRoute
    '/editor': typeof EditorRoute
    '/login': typeof LoginRoute
    '/register': typeof RegisterRoute
    '/new/book': typeof NewBookRoute
    '/profile/$username': typeof ProfileUsernameRoute
    '/work/book/$bookId/overview': typeof WorkBookBookIdOverviewRouteWithChildren
    '/work/book/$bookId/chapter/new': typeof WorkBookBookIdChapterNewRoute
    '/work/book/$bookId/overview/contents': typeof WorkBookBookIdOverviewContentsRoute
    '/work/book/$bookId/overview/details': typeof WorkBookBookIdOverviewDetailsRoute
    '/work/book/$bookId/overview/reviews': typeof WorkBookBookIdOverviewReviewsRoute
    '/work/book/$bookId/chapter/$chapterId/edit': typeof
WorkBookBookIdChapterChapterIdEditRoute
    '/work/book/$bookId/chapter/$chapterId/view': typeof
WorkBookBookIdChapterChapterIdViewRoute
}

export interface FileRoutesById {
    __root__: typeof rootRoute
    '/': typeof IndexRoute
    '/about': typeof AboutRoute
    '/editor': typeof EditorRoute
    '/login': typeof LoginRoute
    '/register': typeof RegisterRoute
    '/new/book': typeof NewBookRoute
    '/profile/$username': typeof ProfileUsernameRoute
    '/work/book/$bookId/overview': typeof WorkBookBookIdOverviewRouteWithChildren
    '/work/book/$bookId/chapter/new': typeof WorkBookBookIdChapterNewRoute
    '/work/book/$bookId/overview/contents': typeof WorkBookBookIdOverviewContentsRoute
    '/work/book/$bookId/overview/details': typeof WorkBookBookIdOverviewDetailsRoute
    '/work/book/$bookId/overview/reviews': typeof WorkBookBookIdOverviewReviewsRoute

```

					ІАЛЦ.467200.007 ДГ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    '/work/book/$bookId/chapter/$chapterId/edit': typeof
WorkBookBookIdChapterChapterIdEditRoute
    '/work/book/$bookId/chapter/$chapterId/view': typeof
WorkBookBookIdChapterChapterIdViewRoute
  }

```

```

export interface FileRouteTypes {
  fileRoutesByFullPath: FileRoutesByFullPath
  fullPaths:
    | '/'
    | '/about'
    | '/editor'
    | '/login'
    | '/register'
    | '/new/book'
    | '/profile/$username'
    | '/work/book/$bookId/overview'
    | '/work/book/$bookId/chapter/new'
    | '/work/book/$bookId/overview/contents'
    | '/work/book/$bookId/overview/details'
    | '/work/book/$bookId/overview/reviews'
    | '/work/book/$bookId/chapter/$chapterId/edit'
    | '/work/book/$bookId/chapter/$chapterId/view'
  fileRoutesByTo: FileRoutesByTo
  to:
    | '/'
    | '/about'
    | '/editor'
    | '/login'
    | '/register'
    | '/new/book'
    | '/profile/$username'
    | '/work/book/$bookId/overview'
    | '/work/book/$bookId/chapter/new'
    | '/work/book/$bookId/overview/contents'
    | '/work/book/$bookId/overview/details'
    | '/work/book/$bookId/overview/reviews'
    | '/work/book/$bookId/chapter/$chapterId/edit'
    | '/work/book/$bookId/chapter/$chapterId/view'
  id:
    | '__root__'
    | '/'
    | '/about'
    | '/editor'
    | '/login'
    | '/register'
    | '/new/book'
    | '/profile/$username'
    | '/work/book/$bookId/overview'
    | '/work/book/$bookId/chapter/new'
    | '/work/book/$bookId/overview/contents'
    | '/work/book/$bookId/overview/details'
    | '/work/book/$bookId/overview/reviews'
    | '/work/book/$bookId/chapter/$chapterId/edit'
    | '/work/book/$bookId/chapter/$chapterId/view'
  fileRoutesById: FileRoutesById
}

```

```

export interface RootRouteChildren {
  IndexRoute: typeof IndexRoute
  AboutRoute: typeof AboutRoute
  EditorRoute: typeof EditorRoute
  LoginRoute: typeof LoginRoute
  RegisterRoute: typeof RegisterRoute
  NewBookRoute: typeof NewBookRoute
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

```

ProfileUsernameRoute: typeof ProfileUsernameRoute
WorkBookBookIdOverviewRoute: typeof WorkBookBookIdOverviewRouteWithChildren
WorkBookBookIdChapterNewRoute: typeof WorkBookBookIdChapterNewRoute
WorkBookBookIdChapterChapterIdEditRoute: typeof WorkBookBookIdChapterChapterIdEditRoute
WorkBookBookIdChapterChapterIdViewRoute: typeof WorkBookBookIdChapterChapterIdViewRoute
}

const rootRouteChildren: RootRouteChildren = {
  IndexRoute: IndexRoute,
  AboutRoute: AboutRoute,
  EditorRoute: EditorRoute,
  LoginRoute: LoginRoute,
  RegisterRoute: RegisterRoute,
  NewBookRoute: NewBookRoute,
  ProfileUsernameRoute: ProfileUsernameRoute,
  WorkBookBookIdOverviewRoute: WorkBookBookIdOverviewRouteWithChildren,
  WorkBookBookIdChapterNewRoute: WorkBookBookIdChapterNewRoute,
  WorkBookBookIdChapterChapterIdEditRoute:
    WorkBookBookIdChapterChapterIdEditRoute,
  WorkBookBookIdChapterChapterIdViewRoute:
    WorkBookBookIdChapterChapterIdViewRoute,
}

export const routeTree = rootRoute
  ._addFileChildren(rootRouteChildren)
  ._addFileTypes<FileRouteTypes>()

/* ROUTE_MANIFEST_START
{
  "routes": {
    "__root__": {
      "filePath": "__root.tsx",
      "children": [
        "/",
        "/about",
        "/editor",
        "/login",
        "/register",
        "/new/book",
        "/profile/$username",
        "/work/book/$bookId/overview",
        "/work/book/$bookId/chapter/new",
        "/work/book/$bookId/chapter/$chapterId/edit",
        "/work/book/$bookId/chapter/$chapterId/view"
      ]
    },
    "/": {
      "filePath": "index.tsx"
    },
    "/about": {
      "filePath": "about.tsx"
    },
    "/editor": {
      "filePath": "editor.tsx"
    },
    "/login": {
      "filePath": "login.tsx"
    },
    "/register": {
      "filePath": "register.tsx"
    },
    "/new/book": {
      "filePath": "new/book.tsx"
    },
    "/profile/$username": {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    "filePath": "profile/$username.tsx"
  },
  "/work/book/$bookId/overview": {
    "filePath": "work/book/$bookId/overview.tsx",
    "children": [
      "/work/book/$bookId/overview/contents",
      "/work/book/$bookId/overview/details",
      "/work/book/$bookId/overview/reviews"
    ]
  },
  "/work/book/$bookId/chapter/new": {
    "filePath": "work/book/$bookId/chapter/new.tsx"
  },
  "/work/book/$bookId/overview/contents": {
    "filePath": "work/book/$bookId/overview/contents.tsx",
    "parent": "/work/book/$bookId/overview"
  },
  "/work/book/$bookId/overview/details": {
    "filePath": "work/book/$bookId/overview/details.tsx",
    "parent": "/work/book/$bookId/overview"
  },
  "/work/book/$bookId/overview/reviews": {
    "filePath": "work/book/$bookId/overview/reviews.tsx",
    "parent": "/work/book/$bookId/overview"
  },
  "/work/book/$bookId/chapter/$chapterId/edit": {
    "filePath": "work/book/$bookId/chapter/$chapterId/edit.tsx"
  },
  "/work/book/$bookId/chapter/$chapterId/view": {
    "filePath": "work/book/$bookId/chapter/$chapterId/view.tsx"
  }
}
}
ROUTE_MANIFEST_END */

```

```

-- app/client/vite-env.d.ts --
/// <reference types="vite/client" />

```

```

-- app/client/components/Pagination.tsx --
import { ButtonGroup, Pagination as ChakraPagination, IconButton } from '@chakra-ui/react';
import { LuChevronLeft, LuChevronRight } from 'react-icons/lu';

export function Pagination({
  total,
  page,
  setPage,
  pageSize = 5
}: {
  total: number;
  page: number;
  setPage: (value: number) => void;
  pageSize?: number;
  setPageSize?: (value: number) => void;
}) {
  return (
    <ChakraPagination.Root
      count={total}
      page={page}
      pageSize={pageSize}
      onPageChange={e => setPage(e.page)}
      defaultPage={1}
    >
      <ButtonGroup variant="ghost" size="sm">

```

					ІАЛЦ.467200.007 ДГ	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    <ChakraPagination.PrevTrigger asChild>
      <IconButton>
        <LuChevronLeft />
      </IconButton>
    </ChakraPagination.PrevTrigger>

    <ChakraPagination.Items
      render={page => (
        <IconButton variant={{ base: 'ghost', _selected: 'solid'
}}>{page.value}</IconButton>
      )}
    />

    <ChakraPagination.NextTrigger asChild>
      <IconButton>
        <LuChevronRight />
      </IconButton>
    </ChakraPagination.NextTrigger>
  </ButtonGroup>
</ChakraPagination.Root>
);
}

-- app/client/components/WorkingListCard.tsx --
import { Badge, Box, Card, Heading, Stack, Text } from '@chakra-ui/react';
import { Link } from '@tanstack/react-router';
import { toFormattedDate } from '../helpers';
import { BookOne } from '../interfaces';
import { WorkingProgressStatusBadge } from '../WorkingProgressStatusBadge';
import { WorkingRatingBadge } from '../WorkingRatingBadge';
import { WorkingReviewsBadge } from '../WorkingReviewsTotalBadge';

export function WorkingListCard({
  id,
  title,
  description,
  author,
  progressStatus,
  updatedAt,
  chaptersTotal,
  wordsTotal,
  ratingTotal,
  reviewsTotal
}: BookOne) {
  return (
    <Card.Root width="100%" variant="elevated">
      <Card.Body>
        <Stack>
          <Stack direction={'row'} justifyContent="space-between">
            <Text as="div" minWidth={0}>
              <Link to={`/work/book/${id}/overview/contents`} >
                <Heading
                  color="colorPalette.400"
                  _hover={{
                    textDecoration: 'underline',
                    textDecorationColor: 'colorPalette.700'
                  }}
                >
                  {title}
                </Heading>
              </Link>
            </Text>

```

```

        <Stack direction="row" justifyContent="center" alignItems="center"
hideBelow="md">
            <WorkingRatingBadge ratingTotal={ratingTotal} />
            <WorkingReviewsBadge reviewsTotal={reviewsTotal} />
            <WorkingProgressStatusBadge progressStatus={progressStatus} />
        </Stack>
    </Stack>

    <Stack>
        <Stack direction="row">
            <Link>by:</Link>
            <Link to={` /profile/${author.username}`}>
                <Text
                    color="colorPalette.400"
                    _hover={{
                        textDecoration: 'underline',
                        textDecorationColor: 'colorPalette.700'
                    }}
                >
                    {author.username}
                </Text>
            </Link>
        </Stack>

        <Stack maxWidth="100%" direction="row" overflow="auto" flexWrap="wrap">
            <Box>
                <Badge size="md" variant="solid">
                    <Stack direction="row">
                        <Text>last updated:</Text>
                        <Text>{toFormattedDate({ value: updatedAt })}</Text>
                    </Stack>
                </Badge>
            </Box>
            <Box>
                <Badge size="md" variant="solid">
                    <Stack direction="row">
                        <Text>chapters:</Text>
                        <Text>{chaptersTotal}</Text>
                    </Stack>
                </Badge>
            </Box>
            <Box>
                <Badge size="md" variant="solid">
                    <Stack direction="row">
                        <Text>words:</Text>
                        <Text>{wordsTotal}</Text>
                    </Stack>
                </Badge>
            </Box>
        </Stack>
    </Stack>

    <Stack direction="row" justifyContent="left" alignItems="center" hideFrom="md">
        <WorkingRatingBadge ratingTotal={ratingTotal} />
        <WorkingReviewsBadge reviewsTotal={reviewsTotal} />
        <WorkingProgressStatusBadge progressStatus={progressStatus} />
    </Stack>

    <Stack>
        <Text lineClamp={4}>{description}</Text>
    </Stack>
</Card.Body>
</Card.Root>
);

```

					ІАЛЦ.467200.007 ДГ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}

-- app/client/components/Link.tsx --
import { Text } from '@chakra-ui/react';
import { LinkProps as LinkExtended, Link as TanstackLink } from '@tanstack/react-router';
import { ReactNode } from 'react';

interface LinkProps {
  children: ReactNode;
  to: LinkExtended['to'];
  link?: LinkExtended;
}

export function Link(props: LinkProps) {
  return (
    <TanstackLink {...props}>
      <Text
        color="colorPalette.500"
        _hover={{
          textDecoration: 'underline',
          textDecorationColor: 'colorPalette.700'
        }}
      >
        {props.children}
      </Text>
    </TanstackLink>
  );
}

-- app/client/components/WorkingReviewsTotalBadge.tsx --
import { Box, Text } from '@chakra-ui/react';
import { IconBadge } from './IconBadge';

export function WorkingReviewsBadge({ reviewsTotal }: { readonly reviewsTotal: number }) {
  return (
    <Box>
      <IconBadge backgroundColor="orange.400" iconClass="nf nf-oct-comment">
        <Text>{reviewsTotal}</Text>
      </IconBadge>
    </Box>
  );
}

-- app/client/components/BookChapterForm.tsx --
import { Button } from '@app/client/components/ui/button';
import { Box, Input, Stack, Textarea } from '@chakra-ui/react';
import { valibotResolver } from '@hookform/resolvers/valibot';
import { Link } from '@tanstack/react-router';
import { useState } from 'react';
import { useForm } from 'react-hook-form';
import { Descendant } from 'slate';
import * as v from 'valibot';
import { EditorSlate } from './EditorSlate';
import { Field } from './ui/field';

interface BookChapterEditInitialValues {
  readonly title: string;
  readonly foreword?: string;
  readonly footnote?: string;
  readonly content?: Descendant[];
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

```

interface BookChapterInput {
  readonly title: string;
  readonly foreword?: string;
  readonly footnote?: string;
  readonly content: Descendant[];
}

interface BookChapterFormValues {
  readonly title: string;
  readonly foreword: string;
  readonly footnote: string;
}

export function BookChapterForm({
  bookId,
  initialValues,
  onChapterSubmit
}): {
  readonly bookId: string;
  readonly initialValues?: BookChapterEditInitialValues | null;
  readonly onChapterSubmit: (chapter: BookChapterInput) => Promise<void>;
}) {
  const [content, setContent] = useState<Descendant[]>(
    initialValues?.content ?? [
      {
        type: 'paragraph',
        children: [{ text: '' }]
      }
    ]
  );

  const schema = v.pipe(
    v.object({
      title: v.pipe(v.string(), v.nonEmpty('Title is required'), v.maxLength(60)),
      foreword: v.pipe(v.string()),
      footnote: v.pipe(v.string())
    })
  );

  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm<BookChapterFormValues>({
    defaultValues: {
      title: initialValues?.title,
      foreword: initialValues?.foreword,
      footnote: initialValues?.footnote
    },
    resolver: valibotResolver(schema)
  });

  const onSubmit = handleSubmit(async values =>
    onChapterSubmit({
      title: values.title,
      foreword: values.foreword,
      footnote: values.footnote,
      content
    })
  );

  return (
    <form onSubmit={onSubmit}>
      <Stack justifyContent="center" alignItems="center" minHeight="88vh">

```

```

        <Stack maxWidth="840px" minHeight="80vh" justifyContent="center"
alignItems="center">
        <Stack width="100%" direction="row" justifyContent="space-between"
alignItems="start">
        <Box>
            <Field invalid={errors.title != null} errorText={errors.title?.message}>
                <Input {...register('title')} placeholder="Chapter title" />
            </Field>
        </Box>
        <Stack direction="row">
            <Button onClick={async () => onSubmit()}>Submit</Button>
            <Link to={` /work/book/${bookId}/overview/contents`} >
                <Button>Go back</Button>
            </Link>
        </Stack>
    </Stack>

    <Field invalid={errors.foreword != null} errorText={errors.foreword?.message}>
        <Textarea
            color="fg.muted"
            variant="subtle"
            placeholder="Foreword"
            resize="none"
            maxLength={500}
            {...register('foreword')}
        />
    </Field>

    <EditorSlate
        initialValue={initialValues?.content}
        onChange={values => setContent(values)}
    />

    <Field invalid={errors.footnote != null} errorText={errors.footnote?.message}>
        <Textarea
            color="fg.muted"
            variant="subtle"
            placeholder="Footnote"
            resize="none"
            maxLength={500}
            {...register('footnote')}
        />
    </Field>
</Stack>
</form>
);
}

```

```

-- app/client/components/BookChapterCard.tsx --
import { gql } from '@apollo/client';
import { Box, Button, Card, Heading, Icon, Stack, Text } from '@chakra-ui/react';
import { BookChapterDeleteMutation } from '@generated/types';
import { Link } from '@tanstack/react-router';
import { bookFragment } from '../fragments';
import { useToastMutation } from '../hooks/useToastMutation';
import { bookChapterQuery } from '../pages/work/book/$bookId/chapter/$chapterId/view';
import { bookChapterListQuery } from '../pages/work/book/$bookId/overview/contents';

export function BookChapterCard({
    bookId,
    chapterId,
    title,
    isAuthor

```

					ІАЛЦ.467200.007 ДГ	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}): {
  readonly bookId: string;
  readonly chapterId: string;
  readonly title: string;
  readonly isAuthor: boolean;
}) {
  const { update: chapterDelete } = useToastMutation<BookChapterDeleteMutation>({
    mutation,
    successMsg: 'Chapter deleted successfully',
    refetchQueries: [bookChapterListQuery, bookChapterQuery]
  });
  return (
    <Card.Root width="100%" variant="elevated">
      <Card.Body paddingX={4} paddingY={2}>
        <Stack direction="row" justifyContent="space-between" alignItems="center">
          <Box>
            <Text as="div" minWidth={0}>
              <Link to={` /work/book/${bookId}/chapter/${chapterId}/view`} >
                <Heading
                  size="sm"
                  color="colorPalette.400"
                  _hover={{
                    textDecoration: 'underline',
                    textDecorationColor: 'colorPalette.700'
                  }}
                >
                  {title}
                </Heading>
              </Link>
            </Text>
          </Box>
          <Stack direction="row" alignItems="center">
            {isAuthor && (
              <>
                <Button
                  color="red.500"
                  variant="ghost"
                  padding={0}
                  _hover={{
                    backgroundColor: 'bg.error'
                  }}
                  onClick={async () => chapterDelete({ variables: { id: chapterId } })}
                >
                  <Icon fontSize="xl">
                    <i className="nf nf-md-delete" />
                  </Icon>
                </Button>
                <Link to={` /work/book/${bookId}/chapter/${chapterId}/edit`} >
                  <Button
                    variant="ghost"
                    padding={0}
                    _hover={{
                      backgroundColor: 'bg.warning'
                    }}
                  >
                    <Icon color="yellow.500" fontSize="xl">
                      <i className="nf nf-fa-edit" />
                    </Icon>
                  </Button>
                </Link>
              </>
            )}
          <Link to={` /work/book/${bookId}/chapter/${chapterId}/view`} >
            <Button variant="ghost" padding={0}>
              <Icon fontSize="xl">

```

					ІАЛЦ.467200.007 ДГ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        <i className="nf nf-fa-angle_right" />
      </Icon>
    </Button>
  </Link>
</Stack>
</Stack>
</Card.Body>
</Card.Root>
);
}

const mutation = gql`
  mutation BookChapterDelete($id: ID!) {
    bookChapterDelete(id: $id) {
      ...Book
    }
  }
  ${bookFragment}
`;

-- app/client/components/ReviewCard.tsx --
import { Avatar } from '@app/client/components/ui/avatar';
import { toFormattedDate } from '@app/client/helpers';
import { Box, Card, Stack, Text } from '@chakra-ui/react';
import { Link } from '@tanstack/react-router';

export function ReviewCard({
  review
}): {
  readonly review: {
    readonly id: string;
    readonly username: string;
    readonly createdAt: string;
    readonly text: string;
  };
} {
  return (
    <Stack direction="row" alignItems="start">
      <Link to={` /profile/${review.username}`}>
        <Avatar size="sm" fallback={review.username[0]} />
      </Link>
      <Card.Root variant="subtle" width="100%">
        <Card.Body paddingX="3" paddingY="1">
          <Stack direction="row" justifyContent="space-between" alignItems="center" gap={1}>
            <Link to={` /profile/${review.username}`}>
              <Text
                color="colorPalette.400"
                _hover={{
                  textDecoration: 'underline',
                  textDecorationColor: 'colorPalette.700'
                }}
              >
                {review.username}
              </Text>
            </Link>

            <Text color="fg.muted">{toFormattedDate({ value: review.createdAt })}</Text>
          </Stack>
          <Box width="100%" position="relative" padding={0}>
            <Text>{review.text}</Text>
            {/* <Button */}
            {/*   size="sm" */}
            {/*   position="absolute" */}
            {/*   right="2px" */}
          </Box>
        </Card.Body>
      </Card.Root>
    </Stack>
  );
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        { /* bottom="5px" */ }
        { /* padding={0} */ }
        { /* variant="ghost" */ }
        { /* // onClick={async () => onReviewSubmit(text)} */ }
        { /* ></Button> */ }
      </Box>
    </Card.Body>
  </Card.Root>
</Stack>
);
}

-- app/client/components/EditorSlate.tsx --
import { Button, Card, Stack, Text, Wrap } from '@chakra-ui/react';
import isHotkey from 'is-hotkey';
import React, { KeyboardEvent, MouseEvent, useCallback, useMemo } from 'react';
import { createEditor, Descendant, Editor, Element as SlateElement, Transforms } from 'slate';
import { withHistory } from 'slate-history';
import {
  Editable,
  RenderElementProps,
  RenderLeafProps,
  Slate,
  useSlate,
  withReact
} from 'slate-react';
import {
  CustomEditor,
  CustomElement,
  CustomElementType,
  CustomElementWithAlign,
  CustomTextKey
} from '../types';
import { Prose } from '../ui/prose';

export function EditorSlate({
  initialValue,
  onChange,
  readOnly
}): {
  readonly initialValue?: Descendant[];
  readonly onChange?: (values: Descendant[]) => void;
  readonly readOnly?: boolean;
} {
  return (
    <>
      <Card.Root
        variant={'subtle'}
        width="100%"
        flex={1}
        justifyContent="center"
        alignItems="center"
        background={readOnly ? 'none' : undefined}
      >
        <Card.Body
          width="100%"
          css={{
            '& *': {
              fontSize: 'revert',
              fontWeight: 'revert'
            }
          }}
        >

```

					ІАЛЦ.467200.007 ДГ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    <EditorViewport
      initialValue={
        initialValue ?? [
          {
            type: 'paragraph',
            children: [{ text: '' }]
          }
        ]
      }
      onChange={values => {
        if (onChange != null) onChange(values);
      }}
      readOnly={readOnly}
    />
  </Card.Body>
</Card.Root>
</>
);
}

const HOTKEYS: Record<string, CustomTextKey> = {
  'mod+b': 'bold',
  'mod+i': 'italic',
  'mod+u': 'underline',
  'mod+`': 'code'
};

const LIST_TYPES = ['numbered-list', 'bulleted-list'] as const;
const TEXT_ALIGN_TYPES = ['left', 'center', 'right', 'justify'] as const;

type AlignType = (typeof TEXT_ALIGN_TYPES)[number];
type ListType = (typeof LIST_TYPES)[number];
type CustomElementFormat = CustomElementType | AlignType | ListType;

function EditorViewport({
  initialValue,
  onChange,
  readOnly
}): {
  readonly initialValue: Descendant[];
  readonly onChange: (values: Descendant[]) => void;
  readonly readOnly?: boolean;
} {
  const renderElement = useCallback((props: RenderElementProps) => <Element {...props} />, []);
  const renderLeaf = useCallback((props: RenderLeafProps) => <Leaf {...props} />, []);
  const editor = useMemo(() => withHistory(withReact(createEditor())));

  return (
    <Slate editor={editor} initialValue={initialValue} onChange={onChange}>
      <!readOnly && (
        <Wrap justifyContent="center" alignItems="center" gap={1}>
          <MarkButton format="bold" icon="nf nf-oct-bold" />
          <MarkButton format="italic" icon="nf nf-oct-italic" />
          <MarkButton format="underline" icon="nf nf-md-format_underline" />
          <MarkButton format="code" icon="nf nf-oct-code" />
          <BlockButton format="heading-one" icon="nf nf-fa-heading" text="1" />
          <BlockButton format="heading-two" icon="nf nf-fa-heading" text="2" />
          <BlockButton format="block-quote" icon="nf nf-oct-quote" />
          <BlockButton format="numbered-list" icon="nf nf-fa-list_ol" />
          <BlockButton format="bulleted-list" icon="nf nf-fa-list_ul" />
          <BlockButton format="left" icon="nf nf-fa-align_left" />
          <BlockButton format="center" icon="nf nf-fa-align_center" />
          <BlockButton format="right" icon="nf nf-fa-align_right" />
          <BlockButton format="justify" icon="nf nf-fa-align_justify" />

```

					ІАЛЦ.467200.007 ДГ	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    </Wrap>
  })
  <Stack
    width="100%"
    maxWidth="100%"
    height="100%"
    paddingX={3}
    display="flex"
    justifyContent="center"
    alignItems="center"
    flex={1}
  >
    <Prose width="100%" minWidth="100%" height="100%" display="flex" flex={1}>
      <Editable
        className="slate-editor"
        style={{
          outline: 'none',
          minWidth: '100%',
          maxWidth: 'unset',
          maxHeight: readOnly ? undefined : '60vh',
          height: '100%',
          padding: 0,
          overflowY: 'auto',
          overflowX: 'clip'
        }}
        renderElement={renderElement}
        renderLeaf={renderLeaf}
        readOnly={readOnly}
        placeholder=""
        spellCheck
        autoFocus
        onKeyDown={(event: KeyboardEvent<HTMLDivElement>) => {
          for (const hotkey in HOTKEYS) {
            if (isHotkey(hotkey, event as any)) {
              event.preventDefault();
              const mark = HOTKEYS[hotkey];
              toggleMark(editor, mark);
            }
          }
        }}
      />
    </Prose>
  </Stack>
</Slate>
);
}

function toggleBlock(editor: CustomEditor, format: CustomElementFormat) {
  const isActive = isBlockActive(editor, format, isAlignType(format) ? 'align' : 'type');
  const isList = isListType(format);

  Transforms.unwrapNodes(editor, {
    match: n =>
      !Editor.isEditor(n) &&
      SlateElement.isElement(n) &&
      isListType(n.type) &&
      !isAlignType(format),
    split: true
  });
  let newProperties: Partial<SlateElement>;
  if (isAlignType(format)) {
    newProperties = {
      align: isActive ? undefined : format
    };
  } else {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    newProperties = {
      type: isActive ? 'paragraph' : isList ? 'list-item' : format
    };
  }
  Transforms.setNodes<SlateElement>(editor, newProperties);

  if (!isActive && isList) {
    const block = { type: format, children: [] };
    Transforms.wrapNodes(editor, block);
  }
}

function toggleMark(editor: CustomEditor, format: CustomTextKey) {
  const isActive = isMarkActive(editor, format);

  if (isActive) {
    Editor.removeMark(editor, format);
  } else {
    Editor.addMark(editor, format, true);
  }
}

function isBlockActive(
  editor: CustomEditor,
  format: CustomElementFormat,
  blockType: 'type' | 'align' = 'type'
) {
  const { selection } = editor;
  if (!selection) return false;

  const [match] = Array.from(
    Editor.nodes(editor, {
      at: Editor.unhangRange(editor, selection),
      match: n => {
        if (!Editor.isEditor(n) && SlateElement.isElement(n)) {
          if (blockType === 'align' && isAlignElement(n)) {
            return n.align === format;
          }
          return n.type === format;
        }
        return false;
      }
    })
  );

  return Boolean(match);
}

function isMarkActive(editor: CustomEditor, format: CustomTextKey) {
  const marks = Editor.marks(editor);
  return marks ? marks[format] === true : false;
}

function Element({ attributes, children, element }: RenderElementProps) {
  const style: React.CSSProperties = {};
  if (isAlignElement(element)) {
    style.textAlign = element.align as AlignType;
  }
  switch (element.type) {
    case 'block-quote':
      return (
        <blockquote style={style} {...attributes}>
          {children}
        </blockquote>
      );
  }
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    case 'bulleted-list':
      return (
        <ul style={style} {...attributes}>
          {children}
        </ul>
      );
    case 'heading-one':
      return (
        <h1 style={style} {...attributes}>
          {children}
        </h1>
      );
    case 'heading-two':
      return (
        <h2 style={style} {...attributes}>
          {children}
        </h2>
      );
    case 'list-item':
      return (
        <li style={style} {...attributes}>
          {children}
        </li>
      );
    case 'numbered-list':
      return (
        <ol style={style} {...attributes}>
          {children}
        </ol>
      );
    default:
      return (
        <p style={style} {...attributes}>
          {children}
        </p>
      );
  }
}

function Leaf({ attributes, children, leaf }: RenderLeafProps) {
  if (leaf.bold) {
    children = <strong>{children}</strong>;
  }

  if (leaf.code) {
    children = <code>{children}</code>;
  }

  if (leaf.italic) {
    children = <em style={{ fontStyle: 'italic' }}>{children}</em>;
  }

  if (leaf.underline) {
    children = <u>{children}</u>;
  }

  return <span {...attributes}>{children}</span>;
}

interface BlockButtonProps {
  format: CustomElementFormat;
  icon: string;
  text?: string;
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						27
Зм.	Арк.	№ докум.	Підпис	Дата		

```

function BlockButton({ format, icon, text }: BlockButtonProps) {
  const editor = useSlate();
  return (
    <Button
      variant={isBlockActive(editor, format) ? 'solid' : 'ghost'}
      onMouseDown={(event: MouseEvent<HTMLSpanElement>) => {
        event.preventDefault();
        toggleBlock(editor, format);
      }}
      gap={0.5}
    >
      <i className={icon} />
      <Stack alignItems="end">
        <Text fontSize="0.8rem">{text}</Text>
      </Stack>
    </Button>
  );
}

interface MarkButtonProps {
  format: CustomTextKey;
  icon: string;
  text?: string;
}

function MarkButton({ format, icon, text }: MarkButtonProps) {
  const editor = useSlate();
  return (
    <Button
      variant={isMarkActive(editor, format) ? 'solid' : 'ghost'}
      onMouseDown={(event: MouseEvent<HTMLSpanElement>) => {
        event.preventDefault();
        toggleMark(editor, format);
      }}
    >
      <i className={icon} />
      {text}
    </Button>
  );
}

function isAlignType(format: CustomElementFormat): format is AlignType {
  return TEXT_ALIGN_TYPES.includes(format as AlignType);
}

function isListType(format: CustomElementFormat): format is ListType {
  return LIST_TYPES.includes(format as ListType);
}

function isAlignElement(element: CustomElement): element is CustomElementWithAlign {
  return 'align' in element;
}

-- app/client/components/PaginatedListSkeleton.tsx --
import { Skeleton, Stack } from '@chakra-ui/react';

export function PaginatedListSkeleton({ disableSearch }: { readonly disableSearch?: boolean }) {
  return (
    <>
      <!disableSearch && (
        <Stack direction="row" justifyContent="center" alignItems="center" gap={2}>
          <Skeleton height={10} width="80%" />
          <Skeleton height={10} width="calc(20% - 6px)" />
        </Stack>
      )
    </>
  );
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        </Stack>
      ))
      <Skeleton height={20} />
      <Skeleton height={20} />
      <Skeleton height={20} />
    </>
  );
}

-- app/client/components/BookSearchList.tsx --
import { gql } from '@apollo/client';
import { Stack } from '@chakra-ui/react';
import { BooksListQuery } from '@generated/types';
import { useState } from 'react';
import { Pagination } from '../components/Pagination';
import { WorkingList } from '../components/WorkingList';
import { ListSearch } from '../components/ListSearch';
import { bookFragment } from '../fragments';
import { usePaginatedQuery } from '../hooks/usePaginatedQuery';
import { PaginatedListSkeleton } from '../PaginatedListSkeleton';

export function BookSearchList() {
  const [workingSearchValue, setWorkingSearchValue] = useState<string>('');
  const [workingsListPage, setWorkingsListPage] = useState<number>(1);
  const [workingsListPageSize, setWorkingsListPageSize] = useState<number>(8);
  const startIndex = (workingsListPage - 1) * workingsListPageSize;

  const { data } = usePaginatedQuery<BooksListQuery>({
    query: bookListQuery,
    input: {
      filter: {
        terms: workingSearchValue,
        offset: startIndex,
        size: workingsListPageSize
      }
    }
  });

  return (
    <>
      <ListSearch placeholder="Search works..." onSubmit={value =>
setWorkingSearchValue(value)} />
      {data != null ? (
        <>
          <WorkingList workingsList={([...data.bookMany.items]} />
          {data.bookMany.total > workingsListPageSize && (
            <Stack justifyContent="center" alignItems="center">
              <Pagination
                total={data.bookMany.total}
                page={workingsListPage}
                setPage={setWorkingsListPage}
                pageSize={workingsListPageSize}
                setPageSize={setWorkingsListPageSize}
              />
            </Stack>
          )}
        </>
      ) : (
        <PaginatedListSkeleton />
      )}
    </>
  );
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

```

export const bookListQuery = gql`
  query BooksList($input: BookManyInput!) {
    bookMany(input: $input) {
      items {
        ...Book
      }
      total
    }
  }
`
${bookFragment}
`;

-- app/client/components/WorkingList.tsx --
import { Card, List, Text } from '@chakra-ui/react';
import { WorkingListCard } from './WorkingListCard';
import { BookOne } from '../interfaces';

export function WorkingList({ workingsList }: { workingsList: BookOne[] }) {
  return (
    <List.Root variant="plain" gap={4}>
      {workingsList.length !== 0 ? (
        workingsList.map(workingData => (
          <List.Item key={workingData.id}>
            <WorkingListCard {...workingData} />
          </List.Item>
        ))
      ) : (
        <List.Item>
          <Card.Root
            variant="outline"
            width="100%"
            padding={4}
            justifyContent="center"
            alignItems={'center'}
          >
            <Text fontSize="md">Nothing is here yet!</Text>
          </Card.Root>
        </List.Item>
      )}
    </List.Root>
  );
}

-- app/client/components/WorkingRatingBadge.tsx --
import { BadgeProps, Box, Text } from '@chakra-ui/react';
import { IconBadge } from './IconBadge';

interface WorkingRatingParams {
  rating: string;
  color: BadgeProps['backgroundColor'];
  iconClass: string;
}

export function WorkingRatingBadge({ ratingTotal }: { ratingTotal: number }) {
  const ratingParams: WorkingRatingParams =
    ratingTotal >= 0
      ? {
          rating: ratingTotal.toString(),
          color: 'green.500',
          iconClass: 'nf nf-fa-angle_up'
        }
      : {
          rating: ratingTotal.toString().slice(1),

```

					ІАЛЦ.467200.007 ДГ	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        color: 'red.500',
        iconClass: 'nf nf-fa-angle_down'
      };
    return (
      <Box>
        <IconBadge backgroundColor={ratingParams.color} iconClass={ratingParams.iconClass}>
          <Text>{ratingParams.rating}</Text>
        </IconBadge>
      </Box>
    );
  }
}

```

```

-- app/client/components/IconBadge.tsx --
import { Badge, BadgeProps, Box, Icon, Stack } from '@chakra-ui/react';
import { ReactNode } from 'react';

export function IconBadge({
  children,
  iconClass,
  backgroundColor,
  size = 'lg'
}): {
  children?: ReactNode;
  iconClass: string;
  backgroundColor?: BadgeProps['backgroundColor'];
  size?: 'sm' | 'md' | 'lg' | 'xs';
} {
  return (
    <Box flex={1} direction="row" justifyContent="center" alignItems="center">
      <Badge variant="solid" size={size} backgroundColor={backgroundColor}>
        <Stack direction="row">
          <Icon fontSize="xl">
            <i className={iconClass} />
          </Icon>
          {children}
        </Stack>
      </Badge>
    </Box>
  );
}

```

```

-- app/client/components/RatingPanel.tsx --
import { Button } from '@app/client/components/ui/button';
import { Icon, Stack } from '@chakra-ui/react';
import { RatingType } from '@generated/types';

export function RatingPanel({
  personalRating,
  onRatingSet
}): {
  readonly personalRating: RatingType;
  readonly onRatingSet: (rating: RatingType) => Promise<void>;
} {
  return (
    <Stack direction="row">
      <Button
        size="sm"
        backgroundColor={personalRating === RatingType.Up ? 'green.500/80' : 'green.500'}
        borderColor="green.500"
        onClick={async () => {
          if (personalRating !== RatingType.Up) {
            await onRatingSet(RatingType.Up);
          }
          return;
        }}
      >

```

					ІАЛЦ.467200.007 ДГ	Арк.
						31
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
    await onRatingSet(RatingType.None);
  }}
}
>
<Icon>
  <i className="nf nf-oct-thumbsup" />
</Icon>
</Button>
<Button
  size="sm"
  backgroundColor={personalRating === RatingType.Down ? 'red.500/80' : 'red.500'}
  borderColor="red.500"
  onClick={async () => {
    if (personalRating !== RatingType.Down) {
      await onRatingSet(RatingType.Down);
      return;
    }
    await onRatingSet(RatingType.None);
  }}
>
  <Icon>
    <i className="nf nf-oct-thumbsdown" />
  </Icon>
</Button>
</Stack>
);
}

```

```

-- app/client/components/ReviewInput.tsx --
import { Avatar } from '@app/client/components/ui/avatar';
import { Button, Icon, Stack, Textarea } from '@chakra-ui/react';
import { useState } from 'react';

```

```

export function ReviewInput({
  username,
  onReviewSubmit
}): {
  readonly username: string;
  readonly onReviewSubmit: (text: string) => Promise<void>;
} {
  const [text, setText] = useState<string>('');
  return (
    <Stack direction="row" alignItems="start">
      <Avatar size="sm" fallback={username[0]} />
      <Stack width="100%" position="relative" borderRadius="md">
        <Textarea
          variant="subtle"
          placeholder="Write your review here..."
          resize="none"
          maxLength={700}
          maxLines={10}
          fontSize="md"
          autosize
          value={text}
          onChange={event => setText(event.target.value)}
        />
        <Button
          size="sm"
          position="absolute"
          right="2px"
          bottom="5px"
          padding={0}
          variant="ghost"
          onClick={async () => {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        await onReviewSubmit(text);
        setText('');
    }}
    >
    <Icon>
    <i className="nf nf-fa-send" />
    </Icon>
    </Button>
  </Stack>
</Stack>
);
}

-- app/client/components/ListSearch.tsx --
import { Button, Icon, Input, InputGroup, Stack } from '@chakra-ui/react';
import { useState } from 'react';

export function ListSearch({
  placeholder,
  onSubmit
}): {
  readonly placeholder?: string;
  readonly onSubmit: (value: string) => void;
} {
  const [searchValue, setSearchValue] = useState<string>('');
  return (
    <form
      onSubmit={event => {
        event.preventDefault();
        onSubmit(searchValue);
      }}
    >
      <Stack direction="row">
        <InputGroup
          flex="1"
          startElement={
            <Icon color="colorPalette.500">
              <i className="nf nf-oct-search" />
            </Icon>
          }
        >
          <Input
            value={searchValue}
            onChange={event => setSearchValue(event.currentTarget.value)}
            onSubmit={event => event.preventDefault()}
            placeholder={placeholder}
          />
        </InputGroup>
        <Button type="submit" border="none">
          Refresh
        </Button>
      </Stack>
    </form>
  );
}

-- app/client/components/BookOverviewTabs.tsx --
import { Tabs } from '@chakra-ui/react';
import { Link } from '@tanstack/react-router';

export function BookOverviewTabs({
  bookId,
  currentTab,

```

					ІАЛЦ.467200.007 ДГ	Арк.
						33
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    isAuthor
  }: {
    readonly bookId: string;
    readonly currentTab: 'contents' | 'reviews' | 'details';
    readonly isAuthor: boolean;
  }) {
    return (
      <Tabs.Root defaultValue="members" value={currentTab}>
        <Tabs.List>
          <Link to={` /work/book/${bookId}/overview/contents`} >
            <Tabs.Trigger value="contents">Contents</Tabs.Trigger>
          </Link>
          <Link to={` /work/book/${bookId}/overview/reviews`} >
            <Tabs.Trigger value="reviews">Reviews</Tabs.Trigger>
          </Link>
          {isAuthor && (
            <Link to={` /work/book/${bookId}/overview/details`} >
              <Tabs.Trigger value="details">Details</Tabs.Trigger>
            </Link>
          )}
        </Tabs.List>
      </Tabs.Root>
    );
  }
}

-- app/client/components/ui/toaster.tsx --
'use client';

import {
  Toaster as ChakraToaster,
  Portal,
  Spinner,
  Stack,
  Toast,
  createToaster
} from '@chakra-ui/react';

export const toaster = createToaster({
  placement: 'bottom-end',
  pauseOnPageIdle: true
});

export const Toaster = () => {
  return (
    <Portal>
      <ChakraToaster toaster={toaster} insetInline={{ mdDown: '4' }}>
        {toast => (
          <Toast.Root width={{ md: 'sm' }}>
            {toast.type === 'loading' ? (
              <Spinner size="sm" color="blue.solid" />
            ) : (
              <Toast.Indicator />
            )}
            <Stack gap="1" flex="1" maxWidth="100%">
              {toast.title && <Toast.Title>{toast.title}</Toast.Title>}
              {toast.description &&
                <Toast.Description>{toast.description}</Toast.Description>
              }
              {toast.action &&
                <Toast.ActionTrigger>{toast.action.label}</Toast.ActionTrigger>
              }
              {toast.meta?.closable && <Toast.CloseTrigger />
            }
          </Toast.Root>
        )}
      </ChakraToaster>
    </Portal>
  );
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						34
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    </Portal>
  );
};

-- app/client/components/ui/checkbox.tsx --
import { Checkbox as ChakraCheckbox } from "@chakra-ui/react"
import * as React from "react"

export interface CheckboxProps extends ChakraCheckbox.RootProps {
  icon?: React.ReactNode
  inputProps?: React.InputHTMLAttributes<HTMLInputElement>
  rootRef?: React.Ref<HTMLLabelElement>
}

export const Checkbox = React.forwardRef<HTMLInputElement, CheckboxProps>(
  function Checkbox(props, ref) {
    const { icon, children, inputProps, rootRef, ...rest } = props
    return (
      <ChakraCheckbox.Root ref={rootRef} {...rest}>
        <ChakraCheckbox.HiddenInput ref={ref} {...inputProps} />
        <ChakraCheckbox.Control>
          {icon || <ChakraCheckbox.Indicator />}
        </ChakraCheckbox.Control>
        {children != null && (
          <ChakraCheckbox.Label>{children}</ChakraCheckbox.Label>
        )}
      </ChakraCheckbox.Root>
    )
  },
)

-- app/client/components/ui/color-mode.tsx --
'use client';

import type { IconButtonProps } from '@chakra-ui/react';
import { ClientOnly, IconButton, Skeleton } from '@chakra-ui/react';
import { ThemeProvider, useTheme } from 'next-themes';
import type { ThemeProviderProps } from 'next-themes';
import * as React from 'react';
import { LuMoon, LuSun } from 'react-icons/lu';

export interface ColorModeProviderProps extends ThemeProviderProps {}

export function ColorModeProvider(props: ColorModeProviderProps) {
  return <ThemeProvider attribute="class" disableTransitionOnChange {...props} />;
}

export function useColorMode() {
  const { resolvedTheme, setTheme } = useTheme();
  const toggleColorMode = () => {
    setTheme(resolvedTheme === 'light' ? 'dark' : 'light');
  };
  return {
    colorMode: resolvedTheme,
    setColorMode: setTheme,
    toggleColorMode
  };
}

export function useColorModeValue<T>(light: T, dark: T) {
  const { colorMode } = useColorMode();
  return colorMode === 'light' ? light : dark;
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						35
Зм.	Арк.	№ докум.	Підпис	Дата		

```

export function ColorModeIcon() {
  const { colorMode } = useColorMode();
  return colorMode === 'light' ? <LuSun /> : <LuMoon />;
}

interface ColorModeButtonProps extends Omit<IconButtonProps, 'aria-label'> {}

export const ColorModeButton = React.forwardRef<HTMLButtonElement, ColorModeButtonProps>(
  function ColorModeButton(props, ref) {
    const { toggleColorMode } = useColorMode();
    return (
      <ClientOnly fallback={<Skeleton boxSize="8" />}>
        <IconButton
          onClick={toggleColorMode}
          variant="ghost"
          aria-label="Toggle color mode"
          size="sm"
          ref={ref}
          {...props}
          css={{
            _icon: {
              width: '5',
              height: '5'
            }
          }}
        >
          <ColorModeIcon />
        </IconButton>
      </ClientOnly>
    );
  }
);

```

```

-- app/client/components/ui/tooltip.tsx --
import { Tooltip as ChakraTooltip, Portal } from "@chakra-ui/react"
import * as React from "react"

```

```

export interface TooltipProps extends ChakraTooltip.RootProps {
  showArrow?: boolean
  portalled?: boolean
  portalRef?: React.RefObject<HTMLElement>
  content: React.ReactNode
  contentProps?: ChakraTooltip.ContentProps
  disabled?: boolean
}

export const Tooltip = React.forwardRef<HTMLDivElement, TooltipProps>(
  function Tooltip(props, ref) {
    const {
      showArrow,
      children,
      disabled,
      portalled,
      content,
      contentProps,
      portalRef,
      ...rest
    } = props

    if (disabled) return children

    return (
      <ChakraTooltip.Root {...rest}>

```

Зм.	Арк.	№ докум.	Підпис	Дата

ІАЛЦ.467200.007 ДГ

Арк.

36

```

    <ChakraTooltip.Trigger asChild>{children}</ChakraTooltip.Trigger>
    <Portal disabled={!portalled} container={portalRef}>
      <ChakraTooltip.Positioner>
        <ChakraTooltip.Content ref={ref} {...contentProps}>
          {showArrow && (
            <ChakraTooltip.Arrow>
              <ChakraTooltip.ArrowTip />
            </ChakraTooltip.Arrow>
          )}
          {content}
        </ChakraTooltip.Content>
      </ChakraTooltip.Positioner>
    </Portal>
  </ChakraTooltip.Root>
)
},
)

```

```

-- app/client/components/ui/prose.tsx --
"use client"

```

```

import { chakra } from "@chakra-ui/react"

export const Prose = chakra("div", {
  base: {
    color: "fg.muted",
    maxWidth: "65ch",
    fontSize: "sm",
    lineHeight: "1.7em",
    "& p": {
      marginTop: "1em",
      marginBottom: "1em",
    },
    "& blockquote": {
      marginTop: "1.285em",
      marginBottom: "1.285em",
      paddingInline: "1.285em",
      borderInlineStartWidth: "0.25em",
    },
    "& a": {
      color: "fg",
      textDecoration: "underline",
      textUnderlineOffset: "3px",
      textDecorationThickness: "2px",
      textDecorationColor: "border.muted",
      fontWeight: "500",
    },
    "& strong": {
      fontWeight: "600",
    },
    "& a strong": {
      color: "inherit",
    },
    "& h1": {
      fontSize: "2.15em",
      letterSpacing: "-0.02em",
      marginTop: "0",
      marginBottom: "0.8em",
      lineHeight: "1.2em",
    },
    "& h2": {
      fontSize: "1.4em",
      letterSpacing: "-0.02em",
      marginTop: "1.6em",
    },
  },
})

```

Зм.	Арк.	№ докум.	Підпис	Дата

ІАЛЦ.467200.007 ДГ

Арк.

37

```

marginBottom: "0.8em",
lineHeight: "1.4em",
},
"& h3": {
fontSize: "1.285em",
letterSpacing: "-0.01em",
marginTop: "1.5em",
marginBottom: "0.4em",
lineHeight: "1.5em",
},
"& h4": {
marginTop: "1.4em",
marginBottom: "0.5em",
letterSpacing: "-0.01em",
lineHeight: "1.5em",
},
"& img": {
marginTop: "1.7em",
marginBottom: "1.7em",
borderRadius: "lg",
boxShadow: "inset",
},
"& picture": {
marginTop: "1.7em",
marginBottom: "1.7em",
},
"& picture > img": {
marginTop: "0",
marginBottom: "0",
},
"& video": {
marginTop: "1.7em",
marginBottom: "1.7em",
},
"& kbd": {
fontSize: "0.85em",
borderRadius: "xs",
paddingTop: "0.15em",
paddingBottom: "0.15em",
paddingInlineEnd: "0.35em",
paddingInlineStart: "0.35em",
fontFamily: "inherit",
color: "fg.muted",
"--shadow": "colors.border",
boxShadow: "0 0 0 1px var(--shadow),0 1px 0 1px var(--shadow)",
},
"& code": {
fontSize: "0.925em",
letterSpacing: "-0.01em",
borderRadius: "md",
borderWidth: "1px",
padding: "0.25em",
},
"& pre code": {
fontSize: "inherit",
letterSpacing: "inherit",
borderWidth: "inherit",
padding: "0",
},
"& h2 code": {
fontSize: "0.9em",
},
"& h3 code": {
fontSize: "0.8em",
},

```

					ІАЛЦ.467200.007 ДГ	Арк.
						38
Зм.	Арк.	№ докум.	Підпис	Дата		

```

"& pre": {
  backgroundColor: "bg.subtle",
  marginTop: "1.6em",
  marginBottom: "1.6em",
  borderRadius: "md",
  fontSize: "0.9em",
  paddingTop: "0.65em",
  paddingBottom: "0.65em",
  paddingInlineEnd: "1em",
  paddingInlineStart: "1em",
  overflowX: "auto",
  fontWeight: "400",
},
"& ol": {
  marginTop: "1em",
  marginBottom: "1em",
  paddingInlineStart: "1.5em",
},
"& ul": {
  marginTop: "1em",
  marginBottom: "1em",
  paddingInlineStart: "1.5em",
},
"& li": {
  marginTop: "0.285em",
  marginBottom: "0.285em",
},
"& ol > li": {
  paddingInlineStart: "0.4em",
  listStyleType: "decimal",
  "&::marker": {
    color: "fg.muted",
  },
},
"& ul > li": {
  paddingInlineStart: "0.4em",
  listStyleType: "disc",
  "&::marker": {
    color: "fg.muted",
  },
},
"& > ul > li p": {
  marginTop: "0.5em",
  marginBottom: "0.5em",
},
"& > ul > li > p:first-of-type": {
  marginTop: "1em",
},
"& > ul > li > p:last-of-type": {
  marginBottom: "1em",
},
"& > ol > li > p:first-of-type": {
  marginTop: "1em",
},
"& > ol > li > p:last-of-type": {
  marginBottom: "1em",
},
"& ul ul, ul ol, ol ul, ol ol": {
  marginTop: "0.5em",
  marginBottom: "0.5em",
},
"& dl": {
  marginTop: "1em",
  marginBottom: "1em",
},

```

					ІАЛЦ.467200.007 ДГ	Арк.
						39
Зм.	Арк.	№ докум.	Підпис	Дата		

```

"& dt": {
  fontWeight: "600",
  marginTop: "1em",
},
"& dd": {
  marginTop: "0.285em",
  paddingInlineStart: "1.5em",
},
"& hr": {
  marginTop: "2.25em",
  marginBottom: "2.25em",
},
"& :is(h1,h2,h3,h4,h5,hr) + *": {
  marginTop: "0",
},
"& table": {
  width: "100%",
  tableLayout: "auto",
  textAlign: "start",
  lineHeight: "1.5em",
  marginTop: "2em",
  marginBottom: "2em",
},
"& thead": {
  borderBottomWidth: "1px",
  color: "fg",
},
"& tbody tr": {
  borderBottomWidth: "1px",
  borderBottomColor: "border",
},
"& thead th": {
  paddingInlineEnd: "1em",
  paddingBottom: "0.65em",
  paddingInlineStart: "1em",
  fontWeight: "medium",
  textAlign: "start",
},
"& thead th:first-of-type": {
  paddingInlineStart: "0",
},
"& thead th:last-of-type": {
  paddingInlineEnd: "0",
},
"& tbody td, tfoot td": {
  paddingTop: "0.65em",
  paddingInlineEnd: "1em",
  paddingBottom: "0.65em",
  paddingInlineStart: "1em",
},
"& tbody td:first-of-type, tfoot td:first-of-type": {
  paddingInlineStart: "0",
},
"& tbody td:last-of-type, tfoot td:last-of-type": {
  paddingInlineEnd: "0",
},
"& figure": {
  marginTop: "1.625em",
  marginBottom: "1.625em",
},
"& figure > *": {
  marginTop: "0",
  marginBottom: "0",
},
"& figcaption": {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						40
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        fontSize: "0.85em",
        lineHeight: "1.25em",
        marginTop: "0.85em",
        color: "fg.muted",
      },
      "& h1, h2, h3, h4": {
        color: "fg",
        fontWeight: "600",
      },
    },
  },
  variants: {
    size: {
      md: {
        fontSize: "sm",
      },
      lg: {
        fontSize: "md",
      },
    },
  },
  defaultVariants: {
    size: "md",
  },
})

-- app/client/components/ui/slider.tsx --
import { Slider as ChakraSlider, For, HStack } from "@chakra-ui/react"
import * as React from "react"

export interface SliderProps extends ChakraSlider.RootProps {
  marks?: Array<number | { value: number; label: React.ReactNode }>
  label?: React.ReactNode
  showValue?: boolean
}

export const Slider = React.forwardRef<HTMLDivElement, SliderProps>(
  function Slider(props, ref) {
    const { marks: marksProp, label, showValue, ...rest } = props
    const value = props.defaultValue ?? props.value

    const marks = marksProp?.map((mark) => {
      if (typeof mark === "number") return { value: mark, label: undefined }
      return mark
    })

    const hasMarkLabel = !!marks?.some((mark) => mark.label)

    return (
      <ChakraSlider.Root ref={ref} thumbAlignment="center" {...rest}>
        {label && !showValue && (
          <ChakraSlider.Label>{label}</ChakraSlider.Label>
        )}
        {label && showValue && (
          <HStack justify="space-between">
            <ChakraSlider.Label>{label}</ChakraSlider.Label>
            <ChakraSlider.ValueText />
          </HStack>
        )}
        <ChakraSlider.Control data-has-mark-label={hasMarkLabel || undefined}>
          <ChakraSlider.Track>
            <ChakraSlider.Range />
          </ChakraSlider.Track>
          <SliderThumbs value={value} />
          <SliderMarks marks={marks} />
        </ChakraSlider.Control>
      </ChakraSlider.Root>
    )
  },
  ref
)

```

					ІАЛЦ.467200.007 ДГ	Арк.
						41
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        </ChakraSlider.Control>
      </ChakraSlider.Root>
    )
  },
)

function SliderThumbs(props: { value?: number[] }) {
  const { value } = props
  return (
    <For each={value}>
      {(_, index) => (
        <ChakraSlider.Thumb key={index} index={index}>
          <ChakraSlider.HiddenInput />
        </ChakraSlider.Thumb>
      )}
    </For>
  )
}

interface SliderMarksProps {
  marks?: Array<number | { value: number; label: React.ReactNode }>
}

const SliderMarks = React.forwardRef<HTMLDivElement, SliderMarksProps>(
  function SliderMarks(props, ref) {
    const { marks } = props
    if (!marks?.length) return null

    return (
      <ChakraSlider.MarkerGroup ref={ref}>
        {marks.map((mark, index) => {
          const value = typeof mark === "number" ? mark : mark.value
          const label = typeof mark === "number" ? undefined : mark.label
          return (
            <ChakraSlider.Marker key={index} value={value}>
              <ChakraSlider.MarkerIndicator />
              {label}
            </ChakraSlider.Marker>
          )
        })}
      </ChakraSlider.MarkerGroup>
    )
  },
)

-- app/client/components/ui/close-button.tsx --
import type { ButtonProps } from "@chakra-ui/react"
import { IconButton as ChakraIconButton } from "@chakra-ui/react"
import * as React from "react"
import { LuX } from "react-icons/lu"

export type CloseButtonProps = ButtonProps

export const CloseButton = React.forwardRef<
  HTMLButtonElement,
  CloseButtonProps
>(function CloseButton(props, ref) {
  return (
    <ChakraIconButton variant="ghost" aria-label="Close" ref={ref} {...props}>
      {props.children ?? <LuX />}
    </ChakraIconButton>
  )
})

```

```

-- app/client/components/ui/button.tsx --
import type { ButtonProps as ChakraButtonProps } from "@chakra-ui/react"
import {
  AbsoluteCenter,
  Button as ChakraButton,
  Span,
  Spinner,
} from "@chakra-ui/react"
import * as React from "react"

interface ButtonLoadingProps {
  loading?: boolean
  loadingText?: React.ReactNode
}

export interface ButtonProps extends ChakraButtonProps, ButtonLoadingProps {}

export const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  function Button(props, ref) {
    const { loading, disabled, loadingText, children, ...rest } = props
    return (
      <ChakraButton disabled={loading || disabled} ref={ref} {...rest}>
        {loading && !loadingText ? (
          <>
            <AbsoluteCenter display="inline-flex">
              <Spinner size="inherit" color="inherit" />
            </AbsoluteCenter>
            <Span opacity={0}>{children}</Span>
          </>
        ) : loading && loadingText ? (
          <>
            <Spinner size="inherit" color="inherit" />
            {loadingText}
          </>
        ) : (
          children
        )}
      </ChakraButton>
    )
  },
  ref
)

-- app/client/components/ui/drawer.tsx --
import { Drawer as ChakraDrawer, Portal } from "@chakra-ui/react"
import { CloseButton } from "../close-button"
import * as React from "react"

interface DrawerContentProps extends ChakraDrawer.ContentProps {
  portalled?: boolean
  portalRef?: React.RefObject<HTMLElement>
  offset?: ChakraDrawer.ContentProps["padding"]
}

export const DrawerContent = React.forwardRef<
  HTMLDivElement,
  DrawerContentProps
>(function DrawerContent(props, ref) {
  const { children, portalled = true, portalRef, offset, ...rest } = props
  return (
    <Portal disabled={!portalled} container={portalRef}>
      <ChakraDrawer.Positioner padding={offset}>
        <ChakraDrawer.Content ref={ref} {...rest} asChild={false}>
          {children}
        </ChakraDrawer.Content>
      </ChakraDrawer.Positioner>
    </Portal>
  )
})

```

```

        </ChakraDrawer.Content>
      </ChakraDrawer.Positioner>
    </Portal>
  )
})

export const DrawerCloseTrigger = React.forwardRef<
  HTMLButtonElement,
  ChakraDrawer.CloseTriggerProps
>(function DrawerCloseTrigger(props, ref) {
  return (
    <ChakraDrawer.CloseTrigger
      position="absolute"
      top="2"
      insetEnd="2"
      {...props}
      asChild
    >
      <CloseButton size="sm" ref={ref} />
    </ChakraDrawer.CloseTrigger>
  )
})

export const DrawerTrigger = ChakraDrawer.Trigger
export const DrawerRoot = ChakraDrawer.Root
export const DrawerFooter = ChakraDrawer.Footer
export const DrawerHeader = ChakraDrawer.Header
export const DrawerBody = ChakraDrawer.Body
export const DrawerBackdrop = ChakraDrawer.Backdrop
export const DrawerDescription = ChakraDrawer.Description
export const DrawerTitle = ChakraDrawer.Title
export const DrawerActionTrigger = ChakraDrawer.ActionTrigger

-- app/client/components/ui/provider.tsx --
'use client';

import { ChakraProvider, createSystem, defaultConfig, Theme } from '@chakra-ui/react';
import { ColorModeProvider, type ColorModeProviderProps } from './color-mode';

export function Provider(props: ColorModeProviderProps) {
  const system = createSystem(defaultConfig, {
    theme: {
      tokens: {}
    }
  });
  return (
    <ChakraProvider value={system}>
      <Theme colorPalette="cyan">
        <ColorModeProvider defaultTheme="dark">{props.children}</ColorModeProvider>
      </Theme>
    </ChakraProvider>
  );
}

-- app/client/components/ui/radio.tsx --
import { RadioGroup as ChakraRadioGroup } from "@chakra-ui/react"
import * as React from "react"

export interface RadioProps extends ChakraRadioGroup.ItemProps {
  rootRef?: React.Ref<HTMLDivElement>
  inputProps?: React.InputHTMLAttributes<HTMLInputElement>
}

```

```

export const Radio = React.forwardRef<HTMLInputElement, RadioProps>(
  function Radio(props, ref) {
    const { children, inputProps, rootRef, ...rest } = props
    return (
      <ChakraRadioGroup.Item ref={rootRef} {...rest}>
        <ChakraRadioGroup.ItemHiddenInput ref={ref} {...inputProps} />
        <ChakraRadioGroup.ItemIndicator />
        {children && (
          <ChakraRadioGroup.ItemText>{children}</ChakraRadioGroup.ItemText>
        )}
      </ChakraRadioGroup.Item>
    )
  },
)

```

```

export const RadioGroup = ChakraRadioGroup.Root

```

```

-- app/client/components/ui/popover.tsx --
import { Popover as ChakraPopover, Portal } from "@chakra-ui/react"
import { CloseButton } from "../close-button"
import * as React from "react"

```

```

interface PopoverContentProps extends ChakraPopover.ContentProps {
  portalled?: boolean
  portalRef?: React.RefObject<HTMLElement>
}

```

```

export const PopoverContent = React.forwardRef<
  HTMLDivElement,
  PopoverContentProps
>(function PopoverContent(props, ref) {
  const { portalled = true, portalRef, ...rest } = props
  return (
    <Portal disabled={!portalled} container={portalRef}>
      <ChakraPopover.Positioner>
        <ChakraPopover.Content ref={ref} {...rest} />
      </ChakraPopover.Positioner>
    </Portal>
  )
})

```

```

export const PopoverArrow = React.forwardRef<
  HTMLDivElement,
  ChakraPopover.ArrowProps
>(function PopoverArrow(props, ref) {
  return (
    <ChakraPopover.Arrow {...props} ref={ref}>
      <ChakraPopover.ArrowTip />
    </ChakraPopover.Arrow>
  )
})

```

```

export const PopoverCloseTrigger = React.forwardRef<
  HTMLButtonElement,
  ChakraPopover.CloseTriggerProps
>(function PopoverCloseTrigger(props, ref) {
  return (
    <ChakraPopover.CloseTrigger
      position="absolute"
      top="1"
      insetEnd="1"
      {...props}
      asChild
      ref={ref}
    />
  )
})

```

					ІАЛЦ.467200.007 ДГ	Арк.
						45
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    >
    <CloseButton size="sm" />
  </ChakraPopover.CloseTrigger>
)
})

export const PopoverTitle = ChakraPopover.Title
export const PopoverDescription = ChakraPopover.Description
export const PopoverFooter = ChakraPopover.Footer
export const PopoverHeader = ChakraPopover.Header
export const PopoverRoot = ChakraPopover.Root
export const PopoverBody = ChakraPopover.Body
export const PopoverTrigger = ChakraPopover.Trigger

-- app/client/components/ui/input-group.tsx --
import type { BoxProps, InputElementProps } from "@chakra-ui/react"
import { Group, InputElement } from "@chakra-ui/react"
import * as React from "react"

export interface InputGroupProps extends BoxProps {
  startElementProps?: InputElementProps
  endElementProps?: InputElementProps
  startElement?: React.ReactNode
  endElement?: React.ReactNode
  children: React.ReactElement
  startOffset?: InputElementProps["paddingStart"]
  endOffset?: InputElementProps["paddingEnd"]
}

export const InputGroup = React.forwardRef<HTMLDivElement, InputGroupProps>(
  function InputGroup(props, ref) {
    const {
      startElement,
      startElementProps,
      endElement,
      endElementProps,
      children,
      startOffset = "6px",
      endOffset = "6px",
      ...rest
    } = props

    return (
      <Group ref={ref} {...rest}>
        {startElement && (
          <InputElement pointerEvents="none" {...startElementProps}>
            {startElement}
          </InputElement>
        )}
        {React.cloneElement(children, {
          ...(startElement && {
            ps: `calc(var(--input-height) - ${startOffset})`,
          }),
          ...(endElement && { pe: `calc(var(--input-height) - ${endOffset})` }),
          ...children.props,
        )}}
        {endElement && (
          <InputElement placement="end" {...endElementProps}>
            {endElement}
          </InputElement>
        )}
      </Group>
    )
  },
)

```

					ІАЛЦ.467200.007 ДГ	Арк.
						46
Зм.	Арк.	№ докум.	Підпис	Дата		

```

)

-- app/client/components/ui/password-input.tsx --
'use client';

import type { ButtonProps, GroupProps, InputProps, StackProps } from '@chakra-ui/react';
import {
  Box,
  HStack,
  IconButton,
  Input,
  InputGroup,
  Stack,
  mergeRefs,
  useControllableState
} from '@chakra-ui/react';
import * as React from 'react';
import { LuEye, LuEyeOff } from 'react-icons/lu';

export interface PasswordVisibilityProps {
  defaultVisible?: boolean;
  visible?: boolean;
  onVisibleChange?: (visible: boolean) => void;
  visibilityIcon?: { on: React.ReactNode; off: React.ReactNode };
}

export interface PasswordInputProps extends InputProps, PasswordVisibilityProps {
  rootProps?: GroupProps;
}

export const PasswordInput = React.forwardRef<HTMLInputElement, PasswordInputProps>(
  function PasswordInput(props, ref) {
    const {
      rootProps,
      defaultVisible,
      visible: visibleProp,
      onVisibleChange,
      visibilityIcon = { on: <LuEye />, off: <LuEyeOff /> },
      ...rest
    } = props;

    const [visible, setVisible] = useControllableState({
      value: visibleProp,
      defaultValue: defaultVisible || false,
      onChange: onVisibleChange
    });

    const inputRef = React.useRef<HTMLInputElement>(null);

    return (
      <InputGroup
        endElement={
          <VisibilityTrigger
            disabled={rest.disabled}
            onPointerDown={e => {
              if (rest.disabled) return;
              if (e.button !== 0) return;
              e.preventDefault();
              setVisible(!visible);
            }}
          >
            {visible ? visibilityIcon.off : visibilityIcon.on}
          </VisibilityTrigger>
        }
      >

```

					ІАЛЦ.467200.007 ДГ	Арк.
						47
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        {...rootProps}
      >
        <Input {...rest} ref={mergeRefs(ref, inputRef)} type={visible ? 'text' : 'password'}
      />
    </InputGroup>
  );
}
);

const VisibilityTrigger = React.forwardRef<HTMLButtonElement, ButtonProps>(
  function VisibilityTrigger(props, ref) {
    return (
      <IconButton
        tabIndex={-1}
        ref={ref}
        me="-2"
        aspectRatio="square"
        size="sm"
        variant="ghost"
        height="calc(100% - {spacing.2})"
        aria-label="Toggle password visibility"
        {...props}
      />
    );
  }
);

interface PasswordStrengthMeterProps extends StackProps {
  max?: number;
  value: number;
}

export const PasswordStrengthMeter = React.forwardRef<HTMLDivElement,
PasswordStrengthMeterProps>(
  function PasswordStrengthMeter(props, ref) {
    const { max = 4, value, ...rest } = props;

    const percent = (value / max) * 100;
    const { label, colorPalette } = getColorPalette(percent);

    return (
      <Stack align="flex-end" gap="1" ref={ref} {...rest}>
        <HStack width="full" ref={ref} {...rest}>
          {Array.from({ length: max }).map((_, index) => (
            <Box
              key={index}
              height="1"
              flex="1"
              rounded="sm"
              data-selected={index < value ? '' : undefined}
              layerStyle="fill.subtle"
              colorPalette="gray"
              _selected={{
                colorPalette,
                layerStyle: 'fill.solid'
              }}
            />
          ))}
        </HStack>
        {label && <HStack textStyle="xs">{label}</HStack>}
      </Stack>
    );
  }
);

```

					ІАЛЦ.467200.007 ДГ	Арк.
						48
Зм.	Арк.	№ докум.	Підпис	Дата		

```

function getColorPalette(percent: number) {
  switch (true) {
    case percent < 33:
      return { label: 'Low', colorPalette: 'red' };
    case percent < 66:
      return { label: 'Medium', colorPalette: 'orange' };
    default:
      return { label: 'High', colorPalette: 'green' };
  }
}

-- app/client/components/ui/field.tsx --
import { Field as ChakraField } from '@chakra-ui/react';
import * as React from 'react';

export interface FieldProps extends Omit<ChakraField.RootProps, 'label'> {
  label?: React.ReactNode;
  helperText?: React.ReactNode;
  errorText?: React.ReactNode;
  optionalText?: React.ReactNode;
  type?: 'text' | 'password';
}

export const Field = React.forwardRef<HTMLDivElement, FieldProps>(function Field(props, ref)
{
  const { label, children, helperText, errorText, optionalText, ...rest } = props;
  return (
    <ChakraField.Root ref={ref} {...rest}>
      {label && (
        <ChakraField.Label>
          {label}
        <ChakraField.RequiredIndicator fallback={optionalText} />
        </ChakraField.Label>
      )}
      {children}
      {helperText && <ChakraField.HelperText>{helperText}</ChakraField.HelperText>}
      {errorText && <ChakraField.ErrorText>{errorText}</ChakraField.ErrorText>}
    </ChakraField.Root>
  );
});

-- app/client/components/ui/dialog.tsx --
import { Dialog as ChakraDialog, Portal } from "@chakra-ui/react"
import { CloseButton } from "../close-button"
import * as React from "react"

interface DialogContentProps extends ChakraDialog.ContentProps {
  portalled?: boolean
  portalRef?: React.RefObject<HTMLElement>
  backdrop?: boolean
}

export const DialogContent = React.forwardRef<
  HTMLDivElement,
  DialogContentProps
>(function DialogContent(props, ref) {
  const {
    children,
    portalled = true,
    portalRef,
    backdrop = true,
    ...rest
  } = props

```

					ІАЛЦ.467200.007 ДГ	Арк.
						49
Зм.	Арк.	№ докум.	Підпис	Дата		

```

return (
  <Portal disabled={!portalled} container={portalRef}>
    {backdrop && <ChakraDialog.Backdrop />}
    <ChakraDialog.Positioner>
      <ChakraDialog.Content ref={ref} {...rest} asChild={false}>
        {children}
      </ChakraDialog.Content>
    </ChakraDialog.Positioner>
  </Portal>
)
})

export const DialogCloseTrigger = React.forwardRef<
  HTMLButtonElement,
  ChakraDialog.CloseTriggerProps
>(function DialogCloseTrigger(props, ref) {
  return (
    <ChakraDialog.CloseTrigger
      position="absolute"
      top="2"
      insetEnd="2"
      {...props}
      asChild
    >
      <CloseButton size="sm" ref={ref}>
        {props.children}
      </CloseButton>
    </ChakraDialog.CloseTrigger>
  )
})

export const DialogRoot = ChakraDialog.Root
export const DialogFooter = ChakraDialog.Footer
export const DialogHeader = ChakraDialog.Header
export const DialogBody = ChakraDialog.Body
export const DialogBackdrop = ChakraDialog.Backdrop
export const DialogTitle = ChakraDialog.Title
export const DialogDescription = ChakraDialog.Description
export const DialogTrigger = ChakraDialog.Trigger
export const DialogActionTrigger = ChakraDialog.ActionTrigger

-- app/client/components/ui/link.tsx --
import { Link as ChakraLink } from '@chakra-ui/react';
import { createLink, LinkComponent } from '@tanstack/react-router';
import * as React from 'react';

interface ChakraLinkProps extends Omit<React.ComponentPropsWithoutRef<typeof ChakraLink>,
  'href'> {}

const ChakraLinkComponent = React.forwardRef<HTMLAnchorElement, ChakraLinkProps>((props,
ref) => {
  return <ChakraLink ref={ref} {...props} />;
});

const CreatedLinkComponent = createLink(ChakraLinkComponent);

export const Link: LinkComponent<typeof ChakraLinkComponent> = props => {
  return (
    <CreatedLinkComponent
      textDecoration={'underline'}
      _hover={{ textDecoration: 'none' }}
      _focus={{ textDecoration: 'none' }}
      preload={'intent'}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						50
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        {...props}
      />
    );
  };
};

-- app/client/components/ui/avatar.tsx --
"use client"

import type { GroupProps, SlotRecipeProps } from "@chakra-ui/react"
import { Avatar as ChakraAvatar, Group } from "@chakra-ui/react"
import * as React from "react"

type ImageProps = React.ImgHTMLAttributes<HTMLImageElement>

export interface AvatarProps extends ChakraAvatar.RootProps {
  name?: string
  src?: string
  srcSet?: string
  loading?: ImageProps["loading"]
  icon?: React.ReactElement
  fallback?: React.ReactNode
}

export const Avatar = React.forwardRef<HTMLDivElement, AvatarProps>(
  function Avatar(props, ref) {
    const { name, src, srcSet, loading, icon, fallback, children, ...rest } =
      props
    return (
      <ChakraAvatar.Root ref={ref} {...rest}>
        <AvatarFallback name={name} icon={icon}>
          {fallback}
        </AvatarFallback>
        <ChakraAvatar.Image src={src} srcSet={srcSet} loading={loading} />
        {children}
      </ChakraAvatar.Root>
    )
  },
  ref
)

interface AvatarFallbackProps extends ChakraAvatar.FallbackProps {
  name?: string
  icon?: React.ReactElement
}

const AvatarFallback = React.forwardRef<HTMLDivElement, AvatarFallbackProps>(
  function AvatarFallback(props, ref) {
    const { name, icon, children, ...rest } = props
    return (
      <ChakraAvatar.Fallback ref={ref} {...rest}>
        {children}
        {name != null && children == null && <>{getInitials(name)}</>}
        {name == null && children == null && (
          <ChakraAvatar.Icon asChild={!icon}>{icon}</ChakraAvatar.Icon>
        )}
      </ChakraAvatar.Fallback>
    )
  },
  ref
)

function getInitials(name: string) {
  const names = name.trim().split(" ")
  const firstName = names[0] != null ? names[0] : ""
  const lastName = names.length > 1 ? names[names.length - 1] : ""
  return firstName && lastName
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						51
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    ? `${firstName.charAt(0)}${lastName.charAt(0)}`
    : firstName.charAt(0)
  }
}

interface AvatarGroupProps extends GroupProps, SlotRecipeProps<"avatar"> {}

export const AvatarGroup = React.forwardRef<HTMLDivElement, AvatarGroupProps>(
  function AvatarGroup(props, ref) {
    const { size, variant, borderless, ...rest } = props
    return (
      <ChakraAvatar.PropsProvider value={{ size, variant, borderless }}>
        <Group gap="0" spaceX="-3" ref={ref} {...rest} />
      </ChakraAvatar.PropsProvider>
    )
  },
)

-- app/client/components/Editor.tsx --
import { Card, Heading, SystemStyleObject } from '@chakra-ui/react';
import EditorJS, { EditorConfig } from '@editorjs/editorjs';
import Header from '@editorjs/header';
import List from '@editorjs/list';
import Quote from '@editorjs/quote';
import { useEffect, useRef } from 'react';

// @ts-ignore
import Link from '@coolbytes/editorjs-link';

export function Editor({ initialData }: { initialData?: EditorConfig['data'] }) {
  const ejInstance = useRef<EditorJS | undefined | null>();

  const initEditor = () => {
    const editor = new EditorJS({
      holder: 'editorjs',
      onReady: () => {
        ejInstance.current = editor;
      },
      autofocus: true,
      data: initialData,
      // onChange: async () => {
      //   const content = await editor.saver.save();
      // },
      tools: {
        header: Header,
        list: List,
        link: Link,
        quote: Quote
      }
    });
  };

  useEffect(() => {
    if (ejInstance.current === null) {
      initEditor();
    }

    return () => {
      ejInstance?.current?.destroy();
      ejInstance.current = null;
    };
  }, []);

  return (
    <>

```

					ІАЛЦ.467200.007 ДГ	Арк.
						52
Зм.	Арк.	№ докум.	Підпис	Дата		

```

<Card.Root
  variant={'subtle'}
  width="100%"
  maxWidth="840px"
  minHeight="sm"
  justifyContent="center"
  alignItems="center"
>
  <Card.Header>
    <Heading textAlign="center">Editor</Heading>
  </Card.Header>
  <Card.Body
    width="90%"
    css={{
      '& *': {
        fontSize: 'revert',
        fontWeight: 'revert'
      },
      ...editorJsStyles
    }}
  >
    <div className="editorjs" id="editorjs"></div>
  </Card.Body>
</Card.Root>
</>
);
}

```

```

const editorJsStyles:
  | SystemStyleObject
  | Omit<Array<SystemStyleObject | undefined>, keyof any[]>
  | undefined = {
  '& .ce-block--selected .ce-block__content, \
  & .ce-inline-toolbar, \
  & .codex-editor--narrow .ce-toolbox, \
  & .ce-conversion-toolbar, \
  & .ce-settings, \
  & .ce-settings__button, \
  & .ce-toolbar__settings-btn, \
  & .cdx-button, \
  & .ce-popover': {
    background: 'colorPalette.500',
    border: 'none'
  },

  '& .ce-inline-tool, \
  & .ce-conversion-toolbar__label, \
  & .ce-toolbox__button, \
  & .cdx-settings-button, \
  & .ce-toolbar__plus': {
    color: 'colorPalette.500'
  },

  '& ::selection': {
    background: 'colorPalette.500'
  },

  '& .cdx-settings-button: hover, \
  & .ce-settings__button: hover, \
  & .ce-toolbox__button--active, \
  & .ce-toolbox__button: hover, \
  & .cdx-button: hover, \
  & .ce-inline-toolbar__dropdown: hover, \
  & .ce-inline-tool: hover, \
  & .ce-popover__item: hover, \

```

					ІАЛЦ.467200.007 ДГ	Арк.
						53
Зм.	Арк.	№ докум.	Підпис	Дата		

```

& .ce-toolbar__settings-btn:hover': {
  background: 'colorPalette.500',
  color: 'fg'
},

'& .cdx-notify--error': {
  background: 'red !important'
},

'& .cdx-notify__cross::after, & .cdx-notify__cross::before': {
  background: 'white'
},

'& .ce-toolbar__plus:hover, \
& .ce-popover__container': {
  backgroundColor: 'bg.emphasized',
  color: 'colorPalette.500',
  border: 'none'
},

'& .ce-popover-item:hover:not(.ce-popover-item--no-hover)': {
  backgroundColor: 'colorPalette.500',
  color: 'fg'
},

'& .cdx-search-field': {
  background: 'bg.muted',
  color: 'fg',
  border: 'none'
},

'& .ce-popover-item__title, \
& .ce-popover-item__icon': {
  color: 'fg'
},

'& .ce-popover-item-separator': {
  color: 'bg.muted'
},

'& .ce-toolbar__settings-btn': {
  color: 'white'
},

'& .ce-toolbar__settings-btn:hover': {
  color: 'black'
}
};

-- app/client/components/WorkingProgressStatusBadge.tsx --
import { Badge, Box, Icon, Stack, Text } from '@chakra-ui/react';
import { WorkingProgressStatus } from '@generated/types';
import {
  assertUnreachable,
  toBgColorByProgressStatus,
  toWorkingProgressStatusLabel
} from '../helpers';

export function WorkingProgressStatusBadge({
  progressStatus
}): {
  progressStatus: WorkingProgressStatus;
}) {
  return (

```

					ІАЛЦ.467200.007 ДГ	Арк.
						54
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    <Box>
      <Badge
        variant="solid"
        backgroundColor={toBgColorByProgressStatus(progressStatus)}
        size={'lg'}
      >
        <Stack direction="row">
          <Icon fontSize="xl">
            <i className={` ${toWorkingProgressStatusIcon(progressStatus)} `} />
          </Icon>
          <Text>{toWorkingProgressStatusLabel(progressStatus)}</Text>
        </Stack>
      </Badge>
    </Box>
  );
}

function toWorkingProgressStatusIcon(value: WorkingProgressStatus): string {
  if (value === WorkingProgressStatus.InProgress) return 'nf nf-fa-clock';
  if (value === WorkingProgressStatus.Frozen) return 'nf nf-fa-snowflake';
  if (value === WorkingProgressStatus.Complete) return 'nf nf-fa-circle_check';
  assertUnreachable(value);
}

-- app/client/pages/__root.tsx --
import { Icon, Kbd, Menu, Separator, Stack, Text } from '@chakra-ui/react';
import { createRootRouteWithContext, Link, Outlet, useNavigate } from '@tanstack/react-router';
import isHotkey from 'is-hotkey';
import React from 'react';
import { Avatar } from '../components/ui/avatar';
import { Button } from '../components/ui/button';
import { ColorModeButton } from '../components/ui/color-mode';
import { Tooltip } from '../components/ui/tooltip';
import { useAuth } from '../hooks/auth';
import { useIdentity } from '../hooks/identity';
import { appName } from '../main';

const TanStackRouterDevtools =
  process.env.NODE_ENV === 'production'
    ? () => null // Render nothing in production
    : React.lazy(() =>
        // Lazy load in development
        import('@tanstack/router-devtools').then(res => ({
          default: res.TanStackRouterDevtools
            // For Embedded Mode
            // default: res.TanStackRouterDevtoolsPanel
        })))
    );

interface RouterContext {
  isAuthenticated: boolean;
}

export const Route = createRootRouteWithContext<RouterContext>(() => ({
  component: () => {
    const { logout } = useAuth();
    const identity = useIdentity();
    const navigate = useNavigate();

    return (
      <Stack
        onKeyDown={e => {
          if (isHotkey('mod+k', e)) {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						55
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        navigate({ to: '/' });
    }
}}
padding={3}
>
<Stack flexDirection="row" justifyContent="space-between" alignItems="center">
  <Stack flexDirection="row" justifyContent="center" alignItems="center" gap={10}>
    <Link to="/">
      <Stack direction="row" justifyContent="center" alignItems="center" gap={1}>
        <Icon fontSize={24} color="colorPalette.500">
          <i className="nf nf-md-bird" />
        </Icon>
        <Text fontSize="lg" fontWeight="bold">
          {appName}
        </Text>
      </Stack>
    </Link>

    {/* <Stack hideBelow="md"> */}
    {/*   <Link to="/about"> */}
    {/*     <Icon color="colorPalette.500" fontSize="md"> */}
    {/*       <i className="nf nf-oct-question" /> */}
    {/*     </Icon> */}
    {/*   </Link> */}
    {/* </Stack> */}

    {identity != null && (
      <>
        <Stack direction="row" alignItems="center">
          <Link to="/editor">
            <Text
              color="colorPalette.400"
              _hover={{
                textDecoration: 'underline',
                textDecorationColor: 'colorPalette.700'
              }}
            >
              editor
            </Text>
          </Link>
        </Stack>
        <Stack direction="row" alignItems="center">
          <Link to="/new/book">
            <Text
              color="colorPalette.400"
              _hover={{
                textDecoration: 'underline',
                textDecorationColor: 'colorPalette.700'
              }}
            >
              create
            </Text>
          </Link>
        </Stack>
      </>
    )}
  </Stack>

  <Stack flexDirection="row" alignItems="center" gap={4}>
    <Stack
      hideBelow="md"
      direction="row"
      alignItems="center"
      paddingY={1}
      paddingX={2}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						56
Зм.	Арк.	№ докум.	Підпис	Дата		

```

gap={6}
border="sm"
borderColor="border"
borderRadius="md"
onClick={() => navigate({ to: '/' })}
>
  <Stack direction="row" alignItems="center">
    <Icon size="sm" color="colorPalette.500">
      <i className="nf nf-oct-search" />
    </Icon>

    <Text fontSize="sm" color="fg.subtle">
      Search works...{' '}
    </Text>
  </Stack>

  <Kbd>⌘K</Kbd>
</Stack>

<ColorModeButton />

{identity != null ? (
  <Menu.Root>
    <Menu.Trigger css={{ '&:hover': { cursor: 'pointer' } }}>
      <Avatar size="sm" fallback={identity.data.user.username[0]} />
    </Menu.Trigger>
    <Menu.Positioner mt={2}>
      <Menu.Content backgroundColor={'bg.subtle'}>
        <Link to={` /profile/${identity.data.user.username}`}>
          <Menu.Item value="/">
            <Tooltip content="My profile">
              <Stack direction="row">
                <Icon color="colorPalette.500" fontSize="xl">
                  <i className="nf nf-fa-user" />
                </Icon>
                My profile
              </Stack>
            </Tooltip>
          </Menu.Item>
        </Link>

        <Menu.Item value="logout" onClick={async () => logout()}>
          <Tooltip content="Log out">
            <Stack direction="row">
              <Icon color="colorPalette.500" fontSize="xl">
                <i className="nf nf-fa-sign_out" />
              </Icon>
              Log out
            </Stack>
          </Tooltip>
        </Menu.Item>
      </Menu.Content>
    </Menu.Positioner>
  </Menu.Root>
) : (
  <>
    <Separator orientation="vertical" height={5} />
    <Link to="/login">
      <Button height={8}>Sign in</Button>
    </Link>
    <Link to="/register">
      <Button height={8}>Sign up</Button>
    </Link>
  </>
)}

```

```

        </Stack>
      </Stack>
      <Separator style={{ padding: '4px' }} />
      <Outlet />
      <Stack hidden>
        <TanStackRouterDevtools />
      </Stack>
    </Stack>
  );
}
});

-- app/client/pages/work/book/$bookId/overview/contents.tsx --
import { gql, useMutation, useQuery } from '@apollo/client';
import { BookChapterCard } from '@app/client/components/BookChapterCard';
import { BookOverviewTabs } from '@app/client/components/BookOverviewTabs';
import { PaginatedListSkeleton } from '@app/client/components/PaginatedListSkeleton';
import { Pagination } from '@app/client/components/Pagination';
import { RatingPanel } from '@app/client/components/RatingPanel';
import { Button } from '@app/client/components/ui/button';
import { bookChapterFragment } from '@app/client/fragments';
import { useIdentity } from '@app/client/hooks/identity';
import { usePaginatedQuery } from '@app/client/hooks/usePaginatedQuery';
import { Box, Skeleton, Stack, Text } from '@chakra-ui/react';
import { BookChapterListQuery, BookOverviewRatingQuery } from '@generated/types';
import { createFileRoute, Link } from '@tanstack/react-router';
import { useState } from 'react';

export const Route = createFileRoute('/work/book/$bookId/overview/contents')({
  component: RouteComponent
});

function RouteComponent() {
  const { bookId: id } = Route.useParams();

  const identity = useIdentity();

  const { data } = useQuery<BookOverviewRatingQuery>(bookRatingQuery, {
    variables: { id }
  });

  const [ratingUpdate] = useMutation<BookOverviewRatingQuery>(bookRatingMutation, {
    refetchQueries: [bookRatingQuery]
  });

  return (
    <>
      {data != null ? (
        <>
          <BookOverviewTabs
            isAuthor={data?.bookOne.author.id === identity?.data.id}
            bookId={id}
            currentTab="contents"
          />

          <BookChapterList
            id={data.bookOne.id}
            isAuthor={identity?.data.id === data.bookOne.author.id}
          />

          {identity != null &&
            identity?.data.id !== data.bookOne.author.id &&
            data.bookOne.chaptersTotal > 0 && (
              <RatingPanel

```

					ІАЛЦ.467200.007 ДГ	Арк.
						58
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        personalRating={data.bookOne.personalRating}
        onRatingSet={async ratingType => {
          await ratingUpdate({
            variables: {
              input: {
                bookId: data.bookOne.id,
                ratingType
              }
            }
          });
        }}
      </>
    </>
  ) : (
    <Skeleton height="200px" />
  )
</>
);
}

function BookChapterList({ id, isAuthor }: { readonly id: string; readonly isAuthor: boolean
}) {
  const [chaptersListPage, setChaptersListPage] = useState<number>(1);
  const [chaptersListPageSize, setChaptersListPageSize] = useState<number>(10);
  const startIndex = (chaptersListPage - 1) * chaptersListPageSize;

  const { data } = usePaginatedQuery<BookChapterListQuery>({
    query: bookChapterListQuery,
    input: {
      bookId: id,
      filter: {
        offset: startIndex,
        size: chaptersListPageSize
      }
    }
  });

  return (
    <Stack>
      {data != null ? (
        <>
          <Stack>
            {data.bookChapterMany.items.length > 0 ? (
              <>
                {data.bookChapterMany.items.map(item => (
                  <Stack key={`bookChapterList-${item.id}`}>
                    <BookChapterCard
                      bookId={id}
                      chapterId={item.id}
                      title={item.title}
                      isAuthor={isAuthor}
                    />
                  </Stack>
                ))}
              </>
            ) : (
              <Stack justifyContent="center" alignItems="center" pt={8}>
                <Text>No chapters here yet!</Text>
                {isAuthor && (
                  <Link to={`/work/book/${id}/chapter/new`}>
                    <Button variant="outline" size="sm">
                      Create new chapter
                    </Button>
                  </Link>
                )}
              </Stack>
            )}
            </>
          </Stack>
        ) : (
          <Stack justifyContent="center" alignItems="center" pt={8}>
            <Text>No chapters here yet!</Text>
            {isAuthor && (
              <Link to={`/work/book/${id}/chapter/new`}>
                <Button variant="outline" size="sm">
                  Create new chapter
                </Button>
              </Link>
            )}
          </Stack>
        )}
      ) : (
        <Stack justifyContent="center" alignItems="center" pt={8}>
          <Text>No chapters here yet!</Text>
          {isAuthor && (
            <Link to={`/work/book/${id}/chapter/new`}>
              <Button variant="outline" size="sm">
                Create new chapter
              </Button>
            </Link>
          )}
        </Stack>
      )}
    </Stack>
  );
}

```

```

    })
  </Stack>
  })
</Stack>
{data.bookChapterMany.total > chaptersListPageSize && (
  <Stack justifyContent="center" alignItems="center">
    <Pagination
      total={data.bookChapterMany.total}
      page={chaptersListPage}
      setPage={setChaptersListPage}
      pageSize={chaptersListPageSize}
      setPageSize={setChaptersListPageSize}
    />
  </Stack>
)}

<Stack direction="row">
  <>
    {isAuthor && data.bookChapterMany.items.length !== 0 && (
      <Box>
        <Link to={`/work/book/${id}/chapter/new`} >
          <Button size="sm">Create new chapter</Button>
        </Link>
      </Box>
    )}
  </>
</Stack>
</>
) : (
  <PaginatedListSkeleton disableSearch />
)
</Stack>
);
}

export const bookChapterListQuery = gql`
  query BookChapterList($input: BookChapterManyInput!) {
    bookChapterMany(input: $input) {
      items {
        ...BookChapter
      }
      total
    }
  }
`
${bookChapterFragment}
`;

const bookRatingQuery = gql`
  query BookOverviewRating($id: ID!) {
    bookOne(id: $id) {
      id
      ratingTotal
      personalRating
      chaptersTotal
      author {
        id
      }
    }
  }
`
`;

const bookRatingMutation = gql`
  mutation BookOverviewRatingUpdate($input: RatingSetInput!) {
    ratingSet(input: $input)
  }
`

```

					ІАЛЦ.467200.007 ДГ	Арк.
						60
Зм.	Арк.	№ докум.	Підпис	Дата		

```
`;

-- app/client/pages/work/book/$bookId/overview/reviews.tsx --
import { gql, useQuery } from '@apollo/client';
import { BookOverviewTabs } from '@app/client/components/BookOverviewTabs';
import { Pagination } from '@app/client/components/Pagination';
import { reviewFragment } from '@app/client/fragments';
import { useIdentity } from '@app/client/hooks/identity';
import { usePaginatedQuery } from '@app/client/hooks/usePaginatedQuery';
import { useToastMutation } from '@app/client/hooks/useToastMutation';
import { Skeleton, Stack, Text } from '@chakra-ui/react';
import { BookReviewCreateMutation, BookReviewsDataQuery, BookReviewsQuery } from '@generated/types';
import { createFileRoute } from '@tanstack/react-router';
import { useState } from 'react';
import { bookQuery } from '../overview';
import { ReviewInput } from '@app/client/components/ReviewInput';
import { ReviewCard } from '@app/client/components/ReviewCard';

export const Route = createFileRoute('/work/book/$bookId/overview/reviews')({
  component: RouteComponent
});

function RouteComponent() {
  const { bookId: id } = Route.useParams();
  const identity = useIdentity();

  const [reviewsPage, setReviewsPage] = useState<number>(1);
  const [reviewsPageSize, setReviewsPageSize] = useState<number>(10);
  const startIndex = (reviewsPage - 1) * reviewsPageSize;

  const { data } = usePaginatedQuery<BookReviewsQuery>({
    query: bookReviewsQuery,
    input: {
      bookId: id,
      filter: {
        offset: startIndex,
        size: reviewsPageSize
      }
    }
  });

  const { data: authorData } = useQuery<BookReviewsDataQuery>(bookAuthorQuery, {
    variables: { id }
  });

  const { update: reviewCreate } = useToastMutation<BookReviewCreateMutation>({
    mutation: bookReviewCreateMutation,
    successMsg: 'Review posted!',
    refetchQueries: [bookReviewsQuery, bookQuery]
  });

  if (data == null || authorData == null) return <Skeleton />;

  return (
    <>
      <BookOverviewTabs
        isAuthor={authorData.bookOne.author.id === identity?.data.id}
        bookId={id}
        currentTab="reviews"
      />

      {identity != null && reviewsPage === 1 && (
        <ReviewInput

```

					ІАЛЦ.467200.007 ДГ	Арк.
						61
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        username={identity?.data.user.username}
        onReviewSubmit={async text => {
          await reviewCreate({
            variables: {
              input: {
                bookId: id,
                text
              }
            }
          });
        }}
      />
    )}

    {data.reviewMany.items.length === 0 ? (
      <Stack justifyContent="center" alignItems="center" pt={8}>
        <Text>No reviews here yet!</Text>
      </Stack>
    ) : (
      data.reviewMany.items.map(item => (
        <ReviewCard
          key={`bookReview#${item.id}`}
          review={{
            id: item.id,
            username: item.author.username,
            text: item.text,
            createdAt: item.createdAt
          }}
        />
      ))
    )}

    {data.reviewMany.total > reviewsPageSize && (
      <Stack justifyContent="center" alignItems="center">
        <Pagination
          total={data.reviewMany.total}
          page={reviewsPage}
          setPage={setChaptersListPage}
          pageSize={reviewsPageSize}
          setPageSize={setChaptersListPageSize}
        />
      </Stack>
    )}
  </>
);
}

const bookAuthorQuery = gql`
  query BookReviewsData($id: ID!) {
    bookOne(id: $id) {
      id
      author {
        id
      }
    }
  }
`;

const bookReviewsQuery = gql`
  query BookReviews($input: ReviewManyInput!) {
    reviewMany(input: $input) {
      items {
        ...Review
      }
      total
    }
  }
`;

```

					ІАЛЦ.467200.007 ДГ	Арк.
						62
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
  }
  ${reviewFragment}
`;

const bookReviewCreateMutation = gql`
  mutation BookReviewCreate($input: ReviewCreateInput!) {
    reviewCreate(input: $input) {
      ...Review
    }
  }
  ${reviewFragment}
`;

-- app/client/pages/work/book/$bookId/overview/details.tsx --
import { gql, useQuery } from '@apollo/client';
import { BookOverviewTabs } from '@app/client/components/BookOverviewTabs';
import { bookListQuery } from '@app/client/components/BookSearchList';
import { bookFragment } from '@app/client/fragments';
import { useIdentity } from '@app/client/hooks/identity';
import { useToastMutation } from '@app/client/hooks/useToastMutation';
import { Button, Input, NativeSelect, Skeleton, Stack, Textarea } from '@chakra-ui/react';
import { BookDetailsQuery, WorkingProgressStatus } from '@generated/types';
import { useNavigate } from '@tanstack/react-router';
import { createFileRoute } from '@tanstack/react-router';
import { useState } from 'react';

export const Route = createFileRoute('/work/book/$bookId/overview/details')({
  component: RouteComponent
});

function RouteComponent() {
  const { bookId: id } = Route.useParams();

  const identity = useIdentity();

  const navigate = useNavigate();

  const { data } = useQuery<BookDetailsQuery>(bookQuery, {
    variables: { id }
  });

  const { update: bookUpdate } = useToastMutation({
    mutation: bookUpdateMutation,
    successMsg: 'Book updated successfully'
  });

  const { update: bookDelete } = useToastMutation({
    mutation: bookDeleteMutation,
    successMsg: 'Book deleted successfully',
    refetchQueries: [bookListQuery]
  });

  const [title, setTitle] = useState<string | undefined>(data?.bookOne.title);
  const [description, setDescription] = useState<string |
undefined>(data?.bookOne.description);

  const [progress, setProgress] = useState<WorkingProgressStatus | undefined>(
    data?.bookOne.progressStatus
  );

  if (data == null) return <Skeleton />;

  return (

```

					ІАЛЦ.467200.007 ДГ	Арк.
						63
Зм.	Арк.	№ докум.	Підпис	Дата		

```

<>
<BookOverviewTabs
  isAuthor={data.bookOne.author.id === identity?.data.id}
  bookId={id}
  currentTab="details"
/>
<Stack justifyContent="center" alignItems="center">
  <Stack width="70%">
    <Input
      value={title}
      onChange={event => setTitle(event.currentTarget.value)}
      placeholder="title"
      maxLength={60}
    />

    <NativeSelect.Root>
      <NativeSelect.Field
        value={progress}
        onChange={event => {
          const updatedStatus = event.currentTarget.value;
          ensureProgressStatus(updatedStatus);
          setProgress(updatedStatus);
        }}
      >
        <option value={WorkingProgressStatus.InProgress}>In progress</option>
        <option value={WorkingProgressStatus.Frozen}>Frozen</option>
        <option value={WorkingProgressStatus.Complete}>Complete</option>
      </NativeSelect.Field>
      <NativeSelect.Indicator />
    </NativeSelect.Root>
    <Textarea
      value={description}
      onChange={event => setDescription(event.currentTarget.value)}
      placeholder="description"
      maxLength={500}
      autosize
    />

    <Button
      width="100%"
      onClick={async () => {
        await bookUpdate({
          variables: {
            input: {
              id,
              title,
              description,
              progressStatus: progress
            }
          }
        });
      }}
    >
      Update
    </Button>

    <Button
      width="100%"
      backgroundColor="red.500"
      onClick={async () => {
        await bookDelete({ variables: { id } });
        navigate({ to: '/' });
      }}
    >
      Delete

```

					ІАЛЦ.467200.007 ДГ	Арк.
						64
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        </Button>
      </Stack>
    </Stack>
  </>
);
}

function ensureProgressStatus(value: string): asserts value is WorkingProgressStatus {
  if (value === WorkingProgressStatus.Complete) return;
  if (value === WorkingProgressStatus.InProgress) return;
  if (value === WorkingProgressStatus.Frozen) return;

  throw new Error('Invalid WorkingPrgressStatus');
}

const bookQuery = gql`
  query BookDetails($id: ID!) {
    bookOne(id: $id) {
      ...Book
    }
  }
`
${bookFragment}
`;

const bookUpdateMutation = gql`
  mutation BookDetailsUpdate($input: BookUpdateInput!) {
    bookUpdate(input: $input) {
      ...Book
    }
  }
`
${bookFragment}
`;

const bookDeleteMutation = gql`
  mutation BookDetailsDelete($id: ID!) {
    bookDelete(id: $id)
  }
`
`;

-- app/client/pages/work/book/$bookId/overview.tsx --
import { gql, useQuery } from '@apollo/client';
import { WorkingProgressStatusBadge } from
'@app/client/components/WorkingProgressStatusBadge';
import { WorkingRatingBadge } from '@app/client/components/WorkingRatingBadge';
import { WorkingReviewsBadge } from '@app/client/components/WorkingReviewsTotalBadge';
import { bookFragment } from '@app/client/fragments';
import { toFormattedDate } from '@app/client/helpers';
import { BookOne } from '@app/client/interfaces';
import { Heading, HStack, Skeleton, SkeletonText, Stack, Text } from '@chakra-ui/react';
import { BookOverviewQuery } from '@generated/types';
import { createFileRoute, Link, Outlet } from '@tanstack/react-router';

export const Route = createFileRoute('/work/book/$bookId/overview')({
  component: RouteComponent
});

function RouteComponent() {
  const { bookId: id } = Route.useParams();
  const { data: bookData } = useQuery<BookOverviewQuery>(bookQuery, {
    variables: { id }
  });

  return (
    <Stack width="100%" justifyContent="center" alignItems="center" padding={3}>

```

					ІАЛЦ.467200.007 ДГ	Арк.
						65
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    {bookData != null ? <BookOverview book={bookData.bookOne} /> : <BookOverviewSkeleton
/>}
</Stack>
);
}

function BookOverview({ book }: { readonly book: BookOne }) {
  return (
    <Stack width={{ base: '100%', md: '80%' }} maxWidth="840px" gap={3}>
      <Stack direction="row" justifyContent="space-between" alignItems="center">
        <Heading color="colorPalette.400" size="2xl" overflow="auto">
          {book.title}
        </Heading>
      </Stack>

      <Stack direction="column" gap={1}>
        <Stack direction="row">
          <Link>
            <Text fontWeight="bold">by:</Text>
          </Link>
          <Link to={` /profile/${book.author.username}`}>
            <Text
              color="colorPalette.400"
              _hover={{
                textDecoration: 'underline',
                textDecorationColor: 'colorPalette.700'
              }}
            >
              {book.author.username}
            </Text>
          </Link>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">created at:</Text>
          <Text>{toFormattedDate({ value: book.createdAt })}</Text>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">last updated:</Text>
          <Text>{toFormattedDate({ value: book.updatedAt })}</Text>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">chapters:</Text>
          <Text>{book.chaptersTotal}</Text>
        </Stack>

        <Stack direction="row">
          <Text fontWeight="bold">words:</Text>
          <Text>{book.wordsTotal}</Text>
        </Stack>

        <Stack direction="row" justifyContent="left" alignItems="center">
          <WorkingRatingBadge ratingTotal={book.ratingTotal} />
          <WorkingReviewsBadge reviewsTotal={book.reviewsTotal} />
          <WorkingProgressStatusBadge progressStatus={book.progressStatus} />
        </Stack>

        <Text>{book.description}</Text>

        <Outlet />
      </Stack>
    );
  };
}

```

Зм.	Арк.	№ докум.	Підпис	Дата

ІАЛЦ.467200.007 ДГ

Арк.

66

```

}

function BookOverviewSkeleton() {
  return (
    <Stack width={{ base: '100%', md: '80%' }} maxWidth="840px">
      <Stack gap="6" maxWidth="100%">
        <HStack width="full">
          <SkeletonText noOfLines={5} />
        </HStack>
        <Skeleton height="200px" />
      </Stack>
    </Stack>
  );
}

export const bookQuery = gql`
  query BookOverview($id: ID!) {
    bookOne(id: $id) {
      ...Book
    }
  }
`
${bookFragment}
`;

-- app/client/pages/work/book/$bookId/chapter/$chapterId/edit.tsx --
import { gql, useQuery } from '@apollo/client';
import { BookChapterForm } from '@app/client/components/BookChapterForm';
import { useIdentity } from '@app/client/hooks/identity';
import { useToastMutation } from '@app/client/hooks/useToastMutation';
import { Skeleton } from '@chakra-ui/react';
import { BookChapterEditQuery, BookChapterNewCreateMutation } from '@generated/types';
import { createFileRoute, redirect, useNavigate } from '@tanstack/react-router';

export const Route = createFileRoute('/work/book/$bookId/chapter/$chapterId/edit')({
  component: RouteComponent,
  beforeLoad: ({ context }) => {
    if (!context.isAuthenticated) throw redirect({ to: '/' });
  }
});

function RouteComponent() {
  const { bookId, chapterId } = Route.useParams();

  const identity = useIdentity();
  const navigate = useNavigate();

  const { data } = useQuery<BookChapterEditQuery>(query, {
    variables: { id: chapterId }
  });

  const { update: chapterUpdate } = useToastMutation<BookChapterNewCreateMutation>({
    mutation,
    successMsg: 'Chapter updated successfully'
  });

  if (data == null) return <Skeleton />;
  if (identity?.data.user.id !== data.bookChapterOne.book.author.id)
    throw redirect({ to: `/work/book/${bookId}/overview` });

  return (
    <BookChapterForm
      bookId={bookId}
      initialValues={{
        title: data.bookChapterOne.title,

```

					ІАЛЦ.467200.007 ДГ	Арк.
						67
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        foreword: data.bookChapterOne.foreword != null ? data.bookChapterOne.foreword :
undefined,
        footnote: data.bookChapterOne.footnote != null ? data.bookChapterOne.footnote :
undefined,
        content: data.bookChapterOne.content
    }}
    onChapterSubmit={async chapter => {
        await chapterUpdate({
            variables: {
                input: {
                    id: chapterId,
                    ...chapter
                }
            }
        });

        navigate({ to: `/work/book/${bookId}/overview/contents` });
    }}
    />
  );
}

const query = gql`
  query BookChapterEdit($id: ID!) {
    bookChapterOne(id: $id) {
      id
      title
      foreword
      footnote
      content
      previousChapterId
      nextChapterId
      book {
        id
        updatedAt
        wordsTotal
        chaptersTotal
        author {
          id
        }
      }
    }
  }
`;

const mutation = gql`
  mutation BookChapterEditUpdate($input: BookChapterUpdateInput!) {
    bookChapterUpdate(input: $input) {
      id
      title
      foreword
      footnote
      content
      previousChapterId
      nextChapterId
      book {
        id
        updatedAt
        wordsTotal
        chaptersTotal
        author {
          id
        }
      }
    }
  }
`;

```

Зм.	Арк.	№ докум.	Підпис	Дата

ІАЛЦ.467200.007 ДГ

Арк.

68

```

    }
  `;

-- app/client/pages/work/book/$bookId/chapter/$chapterId/view.tsx --
import { gql, useQuery } from '@apollo/client';
import { EditorSlate } from '@app/client/components/EditorSlate';
import { bookChapterFragment } from '@app/client/fragments';
import { Button, Heading, Icon, Separator, Skeleton, Stack, Text } from '@chakra-ui/react';
import { BookChapterViewNavigationQuery, BookChapterViewQuery } from '@generated/types';
import { createFileRoute, Link } from '@tanstack/react-router';

export const Route = createFileRoute('/work/book/$bookId/chapter/$chapterId/view')({
  component: RouteComponent
});

function RouteComponent() {
  const { bookId, chapterId } = Route.useParams();

  return (
    <Stack justifyContent="center" alignItems="center">
      <BookChapterView bookId={bookId} chapterId={chapterId} />
    </Stack>
  );
}

function BookChapterView({
  bookId,
  chapterId
}): {
  readonly bookId: string;
  readonly chapterId: string;
} {
  const { data } = useQuery<BookChapterViewQuery>(bookChapterQuery, {
    variables: { id: chapterId },
    fetchPolicy: 'cache-and-network'
  });

  const { data: navigationData } = useQuery<BookChapterViewNavigationQuery>(
    bookChapterNavigationQuery,
    {
      variables: { id: chapterId },
      fetchPolicy: 'no-cache'
    }
  );

  if (data == null) return <Skeleton />;

  return (
    <Stack
      maxWidth="840px"
      width="100%"
      minHeight="80vh"
      justifyContent="center"
      alignItems="center"
    >
      <Heading as="div" maxWidth="100%">
        <Text textAlign="center" minWidth={0}>
          {data.bookChapterOne.book.title}
        </Text>
        <Text textAlign="center">-</Text>
        <Text textAlign="center" minWidth={0}>
          {data.bookChapterOne.title}
        </Text>
      </Heading>
    >

```

					ІАЛЦ.467200.007 ДГ	Арк.
						69
Зм.	Арк.	№ докум.	Підпис	Дата		

```

{data.bookChapterOne.foreword && (
  <>
    <Stack width="91%" justifyContent="left" alignItems="start" paddingY={2}>
      <Text fontWeight="bold">Foreword:</Text>
      <Text color="fg.muted">{data.bookChapterOne.foreword}</Text>
    </Stack>
    <Separator width="91%" />
  </>
)}
<EditorSlate initialValue={data.bookChapterOne.content} readOnly />
{data.bookChapterOne.footnote && (
  <>
    <Separator width="91%" />
    <Stack width="91%" justifyContent="left" alignItems="start" paddingY={2}>
      <Text fontWeight="bold">Footnote:</Text>
      <Text color="fg.muted">{data.bookChapterOne.footnote}</Text>
    </Stack>
  </>
)}

<ChapterNavigation
  bookId={bookId}
  previousChapterId={navigationData?.bookChapterOne.previousChapterId}
  nextChapterId={navigationData?.bookChapterOne.nextChapterId}
/>
</Stack>
);
}

```

```

function ChapterNavigation({
  bookId,
  previousChapterId,
  nextChapterId
}): {
  readonly bookId: string;
  readonly previousChapterId?: string | null;
  readonly nextChapterId?: string | null;
} {
  return (
    <Stack direction="row" gap={0}>
      {previousChapterId && (
        <Link to={` /work/book/${bookId}/chapter/${previousChapterId}/view`} >
          <Button variant="ghost" borderRightRadius={0}>
            <Icon>
              <i className="nf nf-md-page_previous" />
            </Icon>
            <Text>Prev</Text>
          </Button>
        </Link>
      )}
      <Link to={` /work/book/${bookId}/overview/contents`} >
        <Button
          variant="ghost"
          borderLeftRadius={previousChapterId ? 0 : undefined}
          borderRightRadius={nextChapterId ? 0 : undefined}
        >
          <Icon>
            <i className="nf nf-fa-table_list" />
          </Icon>
          <Text>Contents</Text>
        </Button>
      </Link>
      {nextChapterId && (
        <Link to={` /work/book/${bookId}/chapter/${nextChapterId}/view`} >
          <Button variant="ghost" borderLeftRadius={0}>

```

					ІАЛЦ.467200.007 ДГ	Арк.
						70
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        <Text>Next</Text>
        <Icon>
          <i className="nf nf-md-page_next" />
        </Icon>
      </Button>
    </Link>
  </Stack>
);
}

export const bookChapterQuery = gql`
  query BookChapterView($id: ID!) {
    bookChapterOne(id: $id) {
      id
      content
      ...BookChapter
      book {
        title
        chaptersTotal
      }
    }
  }
`
${bookChapterFragment}
`;

const bookChapterNavigationQuery = gql`
  query BookChapterViewNavigation($id: ID!) {
    bookChapterOne(id: $id) {
      id
      previousChapterId
      nextChapterId
    }
  }
`;

-- app/client/pages/work/book/$bookId/chapter/new.tsx --
import { gql, useQuery } from '@apollo/client';
import { BookChapterForm } from '@app/client/components/BookChapterForm';
import { useIdentity } from '@app/client/hooks/identity';
import { useToastMutation } from '@app/client/hooks/useToastMutation';
import { Skeleton } from '@chakra-ui/react';
import { BookChapterNewCreateMutation, BookChapterNewQuery } from '@generated/types';
import { createFileRoute, redirect, useNavigate } from '@tanstack/react-router';
import { bookChapterListQuery } from '../overview/contents';

export const Route = createFileRoute('/work/book/$bookId/chapter/new')({
  component: RouteComponent,
  beforeLoad: ({ context }) => {
    if (!context.isAuthenticated) throw redirect({ to: '/' });
  }
});

function RouteComponent() {
  const { bookId: id } = Route.useParams();

  const identity = useIdentity();
  const navigate = useNavigate();

  const { data } = useQuery<BookChapterNewQuery>(query, {
    variables: { id }
  });

  const { update: chapterCreate } = useToastMutation<BookChapterNewCreateMutation>({

```

					ІАЛЦ.467200.007 ДГ	Арк.
						71
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    mutation,
    successMsg: 'Chapter created successfully',
    refetchQueries: [bookChapterListQuery]
  });

  if (data == null) return <Skeleton />;
  if (identity?.data.user.id !== data.bookOne.author.id)
    throw redirect({ to: `/work/book/${id}/overview/contents` });

  return (
    <BookChapterForm
      bookId={id}
      onChapterSubmit={async chapter => {
        await chapterCreate({
          variables: {
            input: {
              bookId: id,
              ...chapter
            }
          }
        });
      }}
      navigate={{ to: `/work/book/${id}/overview/contents` }}
    />
  );
}

const query = gql`
  query BookChapterNew($id: ID!) {
    bookOne(id: $id) {
      id
      author {
        id
      }
    }
  }
`;

const mutation = gql`
  mutation BookChapterNewCreate($input: BookChapterCreateInput!) {
    bookChapterCreate(input: $input) {
      id
      title
      foreword
      footnote
      content
      previousChapterId
      nextChapterId
      book {
        id
        updatedAt
        wordsTotal
        chaptersTotal
        author {
          id
        }
      }
    }
  }
`;

-- app/client/pages/login.tsx --
import { Heading, Input, Stack, Text } from '@chakra-ui/react';

```

					ІАЛЦ.467200.007 ДГ	Арк.
						72
Зм.	Арк.	№ докум.	Підпис	Дата		

```

import { Button } from '@client/components/ui/button';
import { Field } from '@client/components/ui/field';
import { valibotResolver } from '@hookform/resolvers/valibot';
import { createFileRoute, redirect } from '@tanstack/react-router';
import { useForm } from 'react-hook-form';
import * as v from 'valibot';
import { Link } from '../components/ui/link';
import { PasswordInput } from '../components/ui/password-input';
import { useAuth } from '../hooks/auth';

export const Route = createFileRoute('/login')({
  component: LoginComponent,
  beforeLoad: ({ context }) => {
    if (context.isAuthenticated) throw redirect({ to: '/' });
  }
});

function LoginComponent() {
  return (
    <Stack flexDirection={'column'} justifyContent={'center'} alignItems={'center'}>
      <Heading>Login</Heading>
      <LoginForm />
      <Text>
        Don't have an account?{' '}
        <Link to="/register" textDecoration="none">
          Sign up
        </Link>
      </Text>
    </Stack>
  );
}

interface FormValues {
  email: string;
  password: string;
}

function LoginForm() {
  const schema = v.pipe(
    v.object({
      email: v.pipe(v.string(), v.nonEmpty('Email is required'), v.email('Invalid email')),
      password: v.pipe(v.string(), v.nonEmpty('Password is required'))
    })
  );

  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm<FormValues>({ resolver: valibotResolver(schema) });

  const { login } = useAuth();

  return (
    <form
      onSubmit={handleSubmit(async values => {
        const session = await login({
          email: values.email,
          password: values.password
        });
        if (session !== null) window.history.back();
      })}
    >
      <Stack>

```

```

        <Field label="Email" invalid={errors.email != null}
errorText={errors.email?.message}>
        <Input {...register('email')} />
    </Field>
    <Field
        label="Password"
        invalid={errors.password != null}
        errorText={errors.password?.message}
    >
        <PasswordInput {...register('password')} />
    </Field>
    <Button type="submit">Sign in</Button>
</Stack>
</form>
);
}

-- app/client/pages/about.tsx --
import { Text, Icon, Stack } from '@chakra-ui/react';
import { createFileRoute } from '@tanstack/react-router';

export const Route = createFileRoute('/about')({
  component: About
});

function About() {
  return (
    <Stack flexDirection={'row'} alignItems={'center'}>
      <Text>Hello from About!</Text>
      <Text>And NerdFonts!</Text>
      <Icon>
        <i className="nf nf-fa-heart" />
      </Icon>
      <Icon>
        <i className="nf nf-custom-vim" />
      </Icon>
    </Stack>
  );
}

-- app/client/pages/editor.tsx --
import { Stack } from '@chakra-ui/react';
import { createFileRoute } from '@tanstack/react-router';
import { EditorSlate } from '../components/EditorSlate';

export const Route = createFileRoute('/editor')({
  component: RouteComponent
});

function RouteComponent() {
  return (
    <Stack justifyContent="center" alignItems="center" minHeight="88vh">
      <Stack maxWidth="840px" justifyContent="center" alignItems="center" minHeight="88vh">
        <EditorSlate />
      </Stack>
    </Stack>
  );
}

-- app/client/pages/new/book.tsx --
import { gql } from '@apollo/client';
import { bookListQuery } from '@app/client/components/BookSearchList';

```

					ІАЛЦ.467200.007 ДГ	Арк.
						74
Зм.	Арк.	№ докум.	Підпис	Дата		

```

import { useToastMutation } from '@app/client/hooks/useToastMutation';
import { Heading, Input, Stack, Textarea } from '@chakra-ui/react';
import { Button } from '@client/components/ui/button';
import { Field } from '@client/components/ui/field';
import { CreateNewBookMutation } from '@generated/types';
import { valibotResolver } from '@hookform/resolvers/valibot';
import { createFileRoute, redirect, useNavigate } from '@tanstack/react-router';
import { useForm } from 'react-hook-form';
import * as v from 'valibot';

export const Route = createFileRoute('/new/book')({
  component: RouteComponent,
  beforeLoad: ({ context }) => {
    if (!context.isAuthenticated) throw redirect({ to: '/' });
  }
});

function RouteComponent() {
  return (
    <Stack width="100%" flexDirection={'column'} justifyContent={'center'}
alignItems={'center'}>
      <Heading>New book</Heading>
      <NewBookForm />
    </Stack>
  );
}

interface FormValues {
  title: string;
  description: string;
}

function NewBookForm() {
  const navigate = useNavigate();
  const { update: bookCreate } = useToastMutation<CreateNewBookMutation>({
    mutation: bookCreateMutation,
    successMsg: 'Book created successfully!',
    refetchQueries: [bookListQuery]
  });

  const schema = v.pipe(
    v.object({
      title: v.pipe(v.string(), v.nonEmpty('Title is required'), v.maxLength(60)),
      description: v.pipe(v.string(), v.nonEmpty('Description is required'),
v.maxLength(900))
    })
  );

  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm<FormValues>({ resolver: valibotResolver(schema) });

  return (
    <form
      style={{ width: '80%', maxWidth: '840px' }}
      onSubmit={handleSubmit(async values => {
        const { data } = await bookCreate({
          variables: {
            input: {
              title: values.title,
              description: values.description
            }
          }
        })
      })
    >

```

					ІАЛЦ.467200.007 ДГ	Арк.
						75
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    });
    if (data != null)
      navigate({
        to: `/work/book/${data.bookCreate.id}/overview/contents`
      });
  }}}
  >
  <Stack>
    <Field label="Title" invalid={errors.title != null}
errorText={errors.title?.message}>
      <Input {...register('title')} />
    </Field>
    <Field
      label="Description"
      invalid={errors.description != null}
      errorText={errors.description?.message}
    >
      <Textarea width="100%" height={'sm'} {...register('description')} />
    </Field>
    <Button type="submit">Create new book</Button>
  </Stack>
</form>
);
}

const bookCreateMutation = gql`
  mutation CreateNewBook($input: BookCreateInput!) {
    bookCreate(input: $input) {
      id
      title
      description
    }
  }
`;

-- app/client/pages/profile/$username.tsx --
import { Avatar } from '@app/client/components/ui/avatar';
import { toFormattedDate } from '@app/client/helpers';
import { useUserOneQuery } from '@app/client/hooks/useUserOneQuery';
import { Card, Heading, HStack, SkeletonCircle, SkeletonText, Stack, Text } from '@chakra-
ui/react';
import { createFileRoute } from '@tanstack/react-router';

export const Route = createFileRoute('/profile/$username')({
  component: RouteComponent
});

function RouteComponent() {
  const { username } = Route.useParams();

  const { data } = useUserOneQuery({
    input: {
      username
    }
  });

  if (data == null) return <UserOverviewSkeleton />;

  return (
    <Stack width="100%" justifyContent="center" alignItems="center" padding={3}>
      <UserOverview user={{ ...data.userOne }} />
    </Stack>
  );
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						76
Зм.	Арк.	№ докум.	Підпис	Дата		

```

function UserOverview({
  user
}): {
  readonly user: {
    readonly id: string;
    readonly username: string;
    readonly description?: string | null;
    readonly createdAt: string;
  };
}

return (
  <Stack width={{ base: '100%', md: '80%' }} maxWidth="840px" gap={3}>
    <Stack direction="row" alignItems="start">
      <Avatar size="2x1" fallback={user.username[0]} />
      <Stack width="100%">
        <Heading color="colorPalette.400" size="2x1" overflow="auto">
          {user.username}
        </Heading>

        <Stack direction="row">
          <Text fontWeight="bold">joined:</Text>
          <Text color="fg.muted">{toFormattedDate({ value: user.createdAt })}</Text>
        </Stack>
      </Stack>
    </Stack>
    <Card.Root variant="subtle" width="100%">
      <Card.Body paddingX="3" paddingY="1">
        <Text>{user.description ?? 'No user description =('}</Text>
      </Card.Body>
    </Card.Root>
  </Stack>
);
}

function UserOverviewSkeleton() {
  return (
    <Stack width="100%" justifyContent="center" alignItems="center" padding={3}>
      <Stack width={{ base: '100%', md: '80%' }} maxWidth="840px">
        <SkeletonCircle width={70} height={70} />
        <Stack gap="6" maxWidth="100%">
          <HStack width="full">
            <SkeletonText noOfLines={5} />
          </HStack>
        </Stack>
      </Stack>
    </Stack>
  );
}

-- app/client/pages/index.tsx --
import { Stack } from '@chakra-ui/react';
import { createFileRoute } from '@tanstack/react-router';
import { BookSearchList } from '../components/BookSearchList';

export const Route = createFileRoute('/')({
  component: RouteComponent
});

function RouteComponent() {
  return (
    <Stack justifyContent="center" alignItems="center">
      <Stack width="90%">
        <BookSearchList />
      </Stack>
    </Stack>
  );
}

```

```

        </Stack>
    </Stack>
  );
}

-- app/client/pages/register.tsx --
import { gql } from '@apollo/client';
import { Heading, Input, Stack, Text } from '@chakra-ui/react';
import { Button } from '@client/components/ui/button';
import { Field } from '@client/components/ui/field';
import { PasswordInput } from '@client/components/ui/password-input';
import { AuthSignUpMutation } from '@generated/types';
import { valibotResolver } from '@hookform/resolvers/valibot';
import { createFileRoute, redirect, useNavigate } from '@tanstack/react-router';
import { useForm } from 'react-hook-form';
import * as v from 'valibot';
import { Link } from '../../components/ui/link';
import { useToastMutation } from '../../hooks/useToastMutation';

export const Route = createFileRoute('/register')({
  component: RegisterComponent,
  beforeLoad: ({ context }) => {
    if (context.isAuthenticated) throw redirect({ to: '/' });
  }
});

function RegisterComponent() {
  return (
    <Stack flexDirection={'column'} justifyContent={'center'} alignItems={'center'}>
      <Heading>Sign up</Heading>
      <SignUpForm />
      <Text>
        Already have an account?{' '}
        <Link to="/login" textDecorations="none">
          Sign in
        </Link>
      </Text>
    </Stack>
  );
}

interface FormValues {
  email: string;
  username: string;
  password: string;
  confirm_password: string;
}

function SignUpForm() {
  const navigate = useNavigate();
  const { update: signUp } = useToastMutation<AuthSignUpMutation>({
    mutation,
    successMsg: 'Account created successfully'
  });

  const schema = v.pipe(
    v.object({
      email: v.pipe(v.string(), v.nonEmpty('Email is required'), v.email('Invalid email')),
      username: v.pipe(v.string(), v.nonEmpty('Username is required')),
      password: v.pipe(
        v.string(),
        v.minLength(8, 'Your password is too short.'),
        v.maxLength(45, 'Your password is too long.'),
        v.regex(/[a-z]/, 'Your password must contain a lowercase letter.'),

```

					ІАЛЦ.467200.007 ДГ	Арк.
						78
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        v.regex(/[A-Z]/, 'Your password must contain a uppercase letter.'),
        v.regex(/[0-9]/, 'Your password must contain a number.'),
    ),
    confirm_password: v.pipe(v.string(), v.nonEmpty('Confirm your password'))
  }},
  v.forward(
    v.check(input => input.password === input.confirm_password, 'Passwords do not match'),
    ['confirm_password']
  )
);

const {
  register,
  handleSubmit,
  formState: { errors }
} = useForm<FormValues>({ resolver: valibotResolver(schema) });

const onSubmit = handleSubmit(async values => {
  await signUp({
    variables: {
      input: {
        email: values.email,
        username: values.username,
        password: values.password
      }
    }
  });
  navigate({
    to: '/login'
  });
});

return (
  <form onSubmit={onSubmit}>
    <Stack>
      <Field label="Email" invalid={errors.email != null}
errorText={errors.email?.message}>
        <Input {...register('email')} />
      </Field>
      <Field
        label="Username"
        invalid={errors.username != null}
        errorText={errors.username?.message}
      >
        <Input {...register('username')} />
      </Field>
      <Field
        label="Password"
        invalid={errors.password != null}
        errorText={errors.password?.message}
      >
        <PasswordInput {...register('password')} />
      </Field>
      <Field
        label="Confirm password"
        invalid={errors.confirm_password != null}
        errorText={errors.confirm_password?.message}
      >
        <PasswordInput {...register('confirm_password')} />
      </Field>
      <Button type="submit">Submit</Button>
    </Stack>
  </form>
);
}

```

```

const mutation = gql`
  mutation AuthSignUp($input: SignUpInput!) {
    signUp(input: $input)
  }
`;

-- app/client/fragments.tsx --
import { gql } from '@apollo/client';

export const sessionFragment = gql`
  fragment Session on Session {
    userId
    accessToken
  }
`;

export const userFragment = gql`
  fragment User on User {
    id
    username
    description
    createdAt
    updatedAt
  }
`;

export const bookFragment = gql`
  fragment Book on Book {
    id
    title
    description
    author {
      id
      username
    }
    createdAt
    updatedAt
    progressStatus
    wordsTotal
    chaptersTotal
    ratingTotal
    reviewsTotal
  }
`;

export const bookChapterFragment = gql`
  fragment BookChapter on BookChapter {
    id
    title
    createdAt
    updatedAt
    foreword
    footnote
    sequenceNumber
    reviewsTotal
  }
`;

export const reviewFragment = gql`
  fragment Review on Review {
    id
    author {
      id

```

					ІАЛЦ.467200.007 ДГ	Арк.
						80
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        username
      }
      text
      ratingTotal
      createdAt
    }
  `;

-- app/client/main.tsx --
import { ApolloClient, ApolloProvider, HttpLink, InMemoryCache, from } from
'@apollo/client';
import { loadDevMessages, loadErrorMessages } from '@apollo/client/dev';
import { onError } from '@apollo/client/link/error';
import { Box, Center, Spinner, Stack } from '@chakra-ui/react';
import '@client/assets/styles/index.css';
import '@client/assets/styles/nerd-fonts-icons.css';
import { Provider as ChakraProvider } from '@client/components/ui/provider';
import { Toaster } from '@client/components/ui/toaster';
import { RouterProvider, createRouter } from '@tanstack/react-router';
import { StrictMode, useEffect, useState } from 'react';
import ReactDOM from 'react-dom/client';
import { AuthProvider, useRestAuth } from '../hooks/auth';
import { IdentityProvider, useIdentityQuery } from '../hooks/identity';

// TanStackRouter - Import the generated route tree
import { Session } from '@generated/types';
import { routeTree } from './routeTree.gen';

// Create a new router instance
export const router = createRouter({
  routeTree,
  context: {
    isAuthenticated: false
  }
});

// Register the router instance for type safety
declare module '@tanstack/react-router' {
  interface Register {
    router: typeof router;
  }
}

// Environment
const mode = import.meta.env.VITE_MODE || 'prod';
const isProd = mode === 'prod';

export const appName = import.meta.env.VITE_APP_NAME || 'project-1n';

const host = import.meta.env.VITE_DOMAIN_NAME || import.meta.env.VITE_HOST || 'localhost';
const port = import.meta.env.VITE_PORT || '3000';
export const url = `https://${host}:${port}`;
const graphqlUri = `${url}/graphql`;

function App() {
  const [authHeaders, setAuthHeaders] = useState<Session | null>(null);
  const [initialLoading, setInitialLoading] = useState<boolean>(true);
  const { refreshToken } = useRestAuth();

  useEffect(() => {
    const getPreviousSession = async () => {
      const session = await refreshToken();
      if (session !== undefined) setAuthHeaders(session);
      setInitialLoading(false);
    };
  });
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						81
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    });
    if (initialLoading) getPreviousSession();
  }, []));

  if (initialLoading)
    return (
      <StrictMode>
        <ChakraProvider>
          <Box pos="absolute" inset="0">
            <Center h="full">
              <Spinner color="colorPalette.500" />
            </Center>
          </Box>
        </ChakraProvider>
      </StrictMode>
    );

  const httpLink = new HttpLink({
    uri: graphqlUri,
    credentials: 'include',
    headers: authHeaders !== null ? { authorization: JSON.stringify(authHeaders) } :
undefined
  });

  const errorLink = onError(({ graphqlErrors, networkError, operation, forward }) => {
    if (graphqlErrors !== null && graphqlErrors.length > 0)
      graphqlErrors.map(async error => {
        if (error.message.includes('jwt expired')) {
          const session = await refreshToken();
          if (session !== undefined) setAuthHeaders(session);
          return forward(operation);
        }
        if (!isProd)
          console.error(
            `[GraphQLError]: Message: ${error.message}, Location:
${JSON.stringify(error.locations)}, Path: ${error.path}`
          );
      });

    if (networkError !== null && graphqlErrors == null) console.error(networkError);
  });

  if (!isProd) {
    loadDevMessages();
    loadErrorMessage();
  }

  const client = new ApolloClient({
    uri: graphqlUri,
    link: from([errorLink, httpLink]),
    cache: new InMemoryCache(),
    connectToDevTools: !isProd,
    credentials: 'include'
  });

  return (
    <StrictMode>
      <AuthProvider
        value={{
          headers: authHeaders,
          setHeaders: (session: Session | null) => {
            setAuthHeaders(session);
          }
        }}
      >

```

					ІАЛЦ.467200.007 ДГ	Арк.
						82
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    <ApolloProvider client={client}>
      <ChakraProvider>
        <Toaster />
        <Stack maxWidth="100vw">
          <RouterProviderWrapper authHeaders={authHeaders} />
        </Stack>
      </ChakraProvider>
    </ApolloProvider>
  </AuthProvider>
</StrictMode>
);
}

function RouterProviderWrapper({ authHeaders }: { authHeaders: Session | null }) {
  const userId = authHeaders?.userId;
  const { data, loading, refetch } = useIdentityQuery({ skip: userId == null });

  if (loading)
    return (
      <Box pos="absolute" inset="0">
        <Center h="full">
          <Spinner color="colorPalette.500" />
        </Center>
      </Box>
    );

  return (
    <IdentityProvider
      value={userId != null && data != null ? { data: data.identity, refetch } : null}
    >
      <RouterProvider router={router} context={{ isAuthenticated: authHeaders != null }} />
    </IdentityProvider>
  );
}

const rootElement = document.getElementById('root')!;
if (!rootElement.innerHTML) {
  const root = ReactDOM.createRoot(rootElement);
  root.render(<App />);
}

-- app/client/helpers.tsx --
import { WorkingProgressStatus } from '@generated/types';
import dayjs from 'dayjs';

export function assertUnreachable(value: never): never {
  throw new Error(`Unhandled scenario: ${value}`);
}

export function toFormattedDate({ value }: { value: string }): string {
  const date = new Date(value);
  const day = dayjs(date).format('DD.MM.YYYY');
  return day;
}

export function toBgColorByProgressStatus(value: WorkingProgressStatus): string {
  if (value === WorkingProgressStatus.InProgress) return 'yellow.500';
  if (value === WorkingProgressStatus.Frozen) return 'blue.500';
  if (value === WorkingProgressStatus.Complete) return 'green.500';
  assertUnreachable(value);
}

export function toFgColorByProgressStatus(value: WorkingProgressStatus): string {
  if (value === WorkingProgressStatus.InProgress) return 'fg.warning';

```

					ІАЛЦ.467200.007 ДГ	Арк.
						83
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    if (value === WorkingProgressStatus.Frozen) return 'fg.info';
    if (value === WorkingProgressStatus.Complete) return 'fg.success';
    assertUnreachable(value);
  }

export function toWorkingProgressStatusLabel(value: WorkingProgressStatus): string {
  if (value === WorkingProgressStatus.InProgress) return 'In progress';
  if (value === WorkingProgressStatus.Frozen) return 'Frozen';
  if (value === WorkingProgressStatus.Complete) return 'Complete';
  assertUnreachable(value);
}

-- app/client/interfaces.tsx --
import { WorkingProgressStatus } from '@generated/types';

export interface BookOne {
  readonly id: string;
  readonly title: string;
  readonly description: string;
  readonly author: {
    readonly id: string;
    readonly username: string;
  };
  readonly createdAt: string;
  readonly updatedAt: string;
  readonly progressStatus: WorkingProgressStatus;
  readonly wordsTotal: number;
  readonly chaptersTotal: number;
  readonly ratingTotal: number;
  readonly reviewsTotal: number;
}

-- app/client/hooks/usePaginatedQuery.tsx --
import {
  ApolloError,
  ApolloQueryResult,
  MaybeMasked,
  OperationVariables,
  useQuery
} from '@apollo/client';
import { DocumentNode } from 'graphql';

interface UsePaginatedQueryProps {
  query: DocumentNode;
  input: {
    [key: string]: any; // for the custom inputs
    filter: {
      terms?: string | null;
      offset?: number | null;
      size?: number | null;
    };
  };
}

interface UsePaginatedQueryReturn<T> {
  data?: MaybeMasked<T> | undefined;
  loading: boolean;
  error: ApolloError | undefined;
  refetch: (
    variables?: Partial<OperationVariables> | undefined
  ) => Promise<ApolloQueryResult<MaybeMasked<T>>>>;
}

```

```

export function usePaginatedQuery<T>(props: UsePaginatedQueryProps):
UsePaginatedQueryReturn<T> {
  const { data, loading, error, refetch } = useQuery<T>(props.query, {
    variables: { input: props.input }
  });
  return {
    data,
    loading,
    error,
    refetch
  };
}

-- app/client/hooks/auth.tsx --
import { gql } from '@apollo/client';
import { Session } from '@generated/types';
import { createContext, useContext } from 'react';
import { sessionFragment } from '../fragments';
import { useToastMutation } from '../hooks/useToastMutation';
import { url } from '../main';

interface AuthContextProps {
  headers: Session | null;
  setHeaders: (session: Session | null) => void;
}

export const AuthContext = createContext<AuthContextProps | null>(null);
export const AuthProvider = AuthContext.Provider;

interface UseAuthOutput extends AuthContextProps {
  isAuthenticated: boolean;
  login: (value: { email: string; password: string }) => Promise<Session | null |
undefined>;
  logout: () => Promise<void>;
}

export function useAuth(): UseAuthOutput {
  const authContext = useContext(AuthContext);
  const { update: login } = useToastMutation<{ login: Session }>({
    mutation: loginMutation,
    successMsg: 'Logged in successfully.'
  });

  const { update: logout } = useToastMutation({
    mutation: logoutMutation,
    successMsg: 'Logged out successfully.'
  });

  if (authContext == null) throw new Error('useAuth must be used within an AuthProvider');

  return {
    ...authContext,
    isAuthenticated: authContext.headers != null,
    login: async ({ email, password }: { email: string; password: string }) => {
      const { data } = await login({
        variables: {
          input: {
            email,
            password
          }
        }
      });
    };
    logout: async () => {
      const { data } = await logout({
        variables: {
          input: {
            email,
            password
          }
        }
      });
    };
  };
  if (data != null) {
    authContext.setHeaders(data.login);
  }
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						85
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return data.login;
    }
    return data;
  },
  logout: async () => {
    await logout();
    authContext.setHeaders(null);
  }
};
}

interface UseRestAuthOutput {
  refreshToken: () => Promise<Session | null | undefined>;
}

export function useRestAuth(): UseRestAuthOutput {
  return {
    refreshToken: async () => {
      try {
        const response = await fetch(`${url}/api/refreshAccessToken`, {
          credentials: 'include'
        });
        if (!response.ok) {
          throw new Error(`[error]: ${response.status}`);
        }
        const session = await response.json();
        return session;
      } catch (error) {
        console.error(error);
      }
    }
  };
}

const loginMutation = gql`
  mutation AuthLogin($input: LoginInput!) {
    login(input: $input) {
      ...Session
    }
  }

  ${sessionFragment}
`;

const logoutMutation = gql`
  mutation AuthLogout {
    logout
  }
`;

-- app/client/hooks/identity.tsx --
import { ApolloError, ApolloQueryResult, gql, OperationVariables, useQuery } from
'@apollo/client';
import { Identity, IdentityRootQuery } from '@generated/types';
import { createContext, useContext } from 'react';

export interface IdentityContext {
  data: Identity;
  refetch: () => void;
}

export const IdentityContext = createContext<IdentityContext | null>(null);
export const IdentityProvider = IdentityContext.Provider;

```

					ІАЛЦ.467200.007 ДГ	Арк.
						86
Зм.	Арк.	№ докум.	Підпис	Дата		

```

export function useIdentity(): IdentityContext | null {
  const identityContext = useContext(IdentityContext);
  if (identityContext == null) return null;
  return identityContext;
}

export function useIdentityQuery({ skip }: { skip?: boolean }): {
  data: IdentityRootQuery | undefined;
  loading: boolean;
  error: ApolloError | undefined;
  refetch: (
    variables?: Partial<OperationVariables> | undefined
  ) => Promise<ApolloQueryResult<IdentityRootQuery>>;
} {
  const { data, loading, error, refetch } = useQuery<IdentityRootQuery>(query, {
    skip
  });
  return {
    data,
    loading,
    error,
    refetch
  };
}

const query = gql`
  query IdentityRoot {
    identity {
      id
      email
      isDarkTheme
      user {
        id
        username
        createdAt
        updatedAt
      }
    }
  }
`;

-- app/client/hooks/useToastMutation.tsx --
import {
  ApolloCache,
  DefaultContext,
  DocumentNode,
  FetchResult,
  InternalRefetchQueriesInclude,
  MaybeMasked,
  MutationFunctionOptions,
  OperationVariables,
  Unmasked,
  useMutation
} from '@apollo/client';
import { toaster } from '../components/ui/toaster';

interface UseToastMutationProps<T> {
  mutation: DocumentNode;
  refetchQueries?:
    | InternalRefetchQueriesInclude
    | ((result: FetchResult<Unmasked<NoInfer<T>>>)) => InternalRefetchQueriesInclude)
    | undefined;
  successMsg?: string;
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						87
Зм.	Арк.	№ докум.	Підпис	Дата		

```

interface UseToastMutationReturn<T> {
  update: (
    options?:
      | MutationFunctionOptions<T, OperationVariables, DefaultContext, ApolloCache<any>>
      | undefined
  ) => Promise<FetchResult<MaybeMasked<T>>>;
}

export function useToastMutation<T>(props: UseToastMutationProps<T>):
UseToastMutationReturn<T> {
  const [update] = useMutation<T>(props.mutation, {
    onCompleted: () => toaster.create({ description: props.successMsg, type: 'success' }),
    onError: error => {
      toaster.create({ description: `${error.message}`, type: 'error' });
      console.error(error);
    },
    refetchQueries: props.refetchQueries
  });

  return { update };
}

-- app/client/hooks/useUserOneQuery.tsx --
import { ApolloError, ApolloQueryResult, gql, OperationVariables, useQuery } from
'@apollo/client';
import { UserOneQuery } from '@generated/types';
import { userFragment } from '../fragments';

export function useUserOneQuery({
  input,
  skip
}): {
  readonly input: { readonly id: string } | { readonly username: string };
  readonly skip?: boolean;
}): {
  data: UserOneQuery | undefined;
  loading: boolean;
  error: ApolloError | undefined;
  refetch: (
    variables?: Partial<OperationVariables> | undefined
  ) => Promise<ApolloQueryResult<UserOneQuery>>;
} {
  const { data, loading, error, refetch } = useQuery<UserOneQuery>(query, {
    variables: { input },
    skip
  });
  return {
    data,
    loading,
    error,
    refetch
  };
}

const query = gql`
  query UserOne($input: UserOneInput!) {
    userOne(input: $input) {
      ...User
    }
  }
  ${userFragment}
`;

```

					ІАЛЦ.467200.007 ДГ	Арк.
						88
Зм.	Арк.	№ докум.	Підпис	Дата		

```

-- app/common.ts --
export function call<T>(fn: () => T) {
  return fn();
}

-- app/server/prisma/schema.prisma --
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id String @id

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  email    String @unique @db.VarChar(255)
  username String @unique @db.VarChar(255)
  password String

  refreshToken String? @unique @db.VarChar(255)

  description String? @db.VarChar(500)

  isDarkMode Boolean @default(true)

  books Book[]

  reviews Review[]
  ratings Rating[]
}

enum WorkingProgressStatus {
  inProgress
  frozen
  complete
}

model Book {
  id String @id

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  title          String @db.VarChar(255)
  description     String @db.MediumText
  progressStatus WorkingProgressStatus @default(inProgress)

  author User @relation(fields: [authorId], references: [id])
  authorId String

  wordsTotal Int @default(0)
  chaptersTotal Int @default(0)

  chapters BookChapter[]

  reviews Review[]
  ratings Rating[]
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						89
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}

model BookChapter {
  id String @id

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  sequenceNumber Int

  title String @db.VarChar(255)

  foreword String? @db.MediumText
  footnote String? @db.MediumText

  content Json @db.Json

  wordsTotal Int @default(0)

  book Book @relation(fields: [bookId], references: [id], onDelete: Cascade)
  bookId String @db.VarChar(255)

  reviews Review[]
  ratings Rating[]

  @@unique([bookId, sequenceNumber])
}

model Review {
  id String @id

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  text String @db.MediumText

  user User @relation(fields: [userId], references: [id])
  userId String @db.VarChar(255)

  bookChapter BookChapter? @relation(fields: [bookChapterId], references: [id], onDelete: Cascade)
  bookChapterId String? @db.VarChar(255)

  book Book? @relation(fields: [bookId], references: [id], onDelete: Cascade)
  bookId String?
}

enum RatingType {
  up
  down
}

model Rating {
  id String @id

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  type RatingType

  user User @relation(fields: [userId], references: [id])
  userId String @db.VarChar(255)

  bookChapter BookChapter? @relation(fields: [bookChapterId], references: [id])
  bookChapterId String? @db.VarChar(255)
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						90
Зм.	Арк.	№ докум.	Підпис	Дата		

```

book Book? @relation(fields: [bookId], references: [id])
bookId String?

@@unique([userId, bookId])
@@unique([userId, bookChapterId])
}

-- app/server/prisma/client.ts --
import { PrismaClient } from '@prisma/client';
import { v4 } from 'uuid';

export const prisma = new PrismaClient();

export function uuid() {
  return v4();
}

-- app/server/.env --
# Environment variables declared in this file are automatically made available to Prisma.
# See the documentation for more detail: https://pris.ly/d/prisma-schema#accessing-
environment-variables-from-the-schema

# Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL
Server, MongoDB and CockroachDB.
# See the documentation for all the connection string options: https://pris.ly/d/connection-
strings

DATABASE_URL="postgresql://johndoe:randompassword@localhost:5432/mydb?schema=public"

-- app/server/helpers.ts --
import { RatingType as RatingTypeSchema } from '@generated/types';
import { Descendant, Text } from 'slate';
import { IdentityContext } from './identity';

export function ensureIdentity(
  identity: IdentityContext['identity']
): asserts identity is NonNullable<IdentityContext['identity']> {
  if (identity == null) throw new Error('Unauthorized.');
```

```

}

export function countWords(descendants: Descendant[]): number {
  let wordCount = 0;

  function traverse(nodes: Descendant[]) {
    for (const node of nodes) {
      if (Text.isText(node)) {
        const words = node.text.trim().split(/\s+/).filter(Boolean);
        wordCount += words.length;
      } else if ('children' in node) {
        traverse(node.children);
      }
    }
  }

  traverse(descendants);
  return wordCount;
}

export function ensureRatingType(value: string): asserts value is RatingTypeSchema {
  if (value === RatingTypeSchema.Up) return;
  if (value === RatingTypeSchema.Down) return;

```

```

    throw new Error('Invalid rating type');
}

-- app/server/graphql/schema.graphql --
scalar Void
scalar JSON

type Query {
  identity: Identity!
  userOne(input: UserOneInput!): User!
  bookOne(id: ID!): Book!
  bookMany(input: BookManyInput!): BookManyOutput!
  bookChapterOne(id: ID!): BookChapter!
  bookChapterMany(input: BookChapterManyInput!): BookChapterManyOutput!
  reviewMany(input: ReviewManyInput!): ReviewManyOutput!
}

type Mutation {
  signUp(input: SignUpInput!): Void
  login(input: LoginInput!): Session!
  logout: Void
  bookCreate(input: BookCreateInput!): Book!
  bookUpdate(input: BookUpdateInput!): Book!
  bookDelete(id: ID!): Void
  bookChapterCreate(input: BookChapterCreateInput!): BookChapter!
  bookChapterUpdate(input: BookChapterUpdateInput!): BookChapter!
  bookChapterDelete(id: ID!): Book!
  reviewCreate(input: ReviewCreateInput!): Review!
  reviewUpdate(input: ReviewUpdateInput!): Review!
  reviewDelete(id: ID!): Void
  ratingSet(input: RatingSetInput!): Void
}

input SignUpInput {
  email: String!
  username: String!
  password: String!
}

input LoginInput {
  email: String!
  password: String!
}

type Session {
  userId: ID!
  accessToken: String!
}

type Identity {
  id: ID!
  email: String!
  isDarkTheme: Boolean!
  user: User!
}

type User {
  id: ID!
  username: String!
  description: String
  createdAt: String!
  updatedAt: String!
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						92
Зм.	Арк.	№ докум.	Підпис	Дата		

```

input UserOneInput {
  id: ID
  username: String
}

input PaginationFilterInput {
  terms: String
  offset: Int
  size: Int
}

input BookCreateInput {
  title: String!
  description: String!
}

input BookManyInput {
  filter: PaginationFilterInput!
}

enum WorkingProgressStatus {
  inProgress
  frozen
  complete
}

type BookManyOutput {
  items: [Book!]!
  total: Int!
}

type Book {
  id: ID!
  title: String!
  description: String!
  author: User!
  createdAt: String!
  updatedAt: String!
  progressStatus: WorkingProgressStatus!
  wordsTotal: Int!
  chaptersTotal: Int!
  ratingTotal: Int!
  reviewsTotal: Int!
  personalRating: RatingType!
}

input BookUpdateInput {
  id: ID!
  title: String
  description: String
  progressStatus: WorkingProgressStatus
}

type BookChapter {
  id: ID!
  title: String!
  createdAt: String!
  updatedAt: String!
  foreword: String
  footnote: String
  content: JSON!
  sequenceNumber: Int!
  reviewsTotal: Int!
  previousChapterId: String
  nextChapterId: String
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						93
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    book: Book!
    personalRating: RatingType!
}

input BookChapterCreateInput {
  bookId: ID!
  title: String!
  foreword: String
  footnote: String
  content: JSON!
}

input BookChapterUpdateInput {
  id: ID!
  title: String!
  foreword: String
  footnote: String
  content: JSON!
}

input BookChapterManyInput {
  bookId: String!
  filter: PaginationFilterInput!
}

type BookChapterManyOutput {
  items: [BookChapter!]!
  total: Int!
}

type Review {
  id: ID!
  author: User!
  text: String!
  ratingTotal: Int!
  createdAt: String!
  updatedAt: String!
}

input ReviewCreateInput {
  bookId: ID
  chapterId: ID
  text: String!
}

input ReviewUpdateInput {
  id: ID!
  text: String!
}

input ReviewManyInput {
  bookId: String
  bookChapterId: String
  filter: PaginationFilterInput!
}

type ReviewManyOutput {
  items: [Review!]!
  total: Int!
}

input RatingSetInput {
  bookId: String
  bookChapterId: String
  ratingType: RatingType!
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						94
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}

enum RatingType {
  up
  down
  none
}

-- app/server/graphql/resolvers/reviewCreateResolver.ts --
import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma, uuid } from '@app/server/prisma/client';
import { MutationReviewCreateArgs, Review } from '@generated/types';
import { toReviewSchema } from './mappers/reviewMapper';

export async function ReviewCreateResolver(
  _: any,
  { input }: MutationReviewCreateArgs,
  { identity }: IdentityContext
): Promise<Review> {
  ensureIdentity(identity);

  if (input.text.length > 700) throw new Error('Review is too long.');
```

```

  const model = await prisma.review.create({
    data: {
      id: uuid(),
      text: input.text,
      bookId: input.bookId,
      userId: identity.id
    }
  });

  return toReviewSchema(model);
}

-- app/server/graphql/resolvers/bookOneResolver.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { Book, QueryBookOneArgs } from '@generated/types';
import { toBookSchema } from './mappers/bookMapper';

export async function BookOneResolver(
  _: any,
  { id }: QueryBookOneArgs,
  { req, res, identity }: IdentityContext
): Promise<Book> {
  const model = await prisma.book.findUniqueOrThrow({
    where: {
      id
    }
  });
  return toBookSchema(model, { req, res, identity });
}

-- app/server/graphql/resolvers/bookChapterUpdateResolver.ts --
import { countWords, ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { updateBookStats } from '@app/server/service/book';
import { BookChapter, MutationBookChapterUpdateArgs } from '@generated/types';
import { toBookChapterSchema } from './mappers/bookChapterMapper';
```

					ІАЛЦ.467200.007 ДГ	Арк.
						95
Зм.	Арк.	№ докум.	Підпис	Дата		

```

export async function BookChapterUpdateResolver(
  _: any,
  { input }: MutationBookChapterUpdateArgs,
  { req, res, identity }: IdentityContext
): Promise<BookChapter> {
  ensureIdentity(identity);

  const model = await prisma.bookChapter.findUniqueOrThrow({
    where: { id: input.id },
    select: {
      book: {
        select: {
          authorId: true
        }
      }
    }
  });

  if (identity.id !== model.book.authorId) throw new Error('Operation is not permitted.');
```

```

  if (input.title.length > 60) throw new Error('Title is too long.');
```

```

  if (input.foreword != null && input.foreword.length > 500)
    throw new Error('Foreword is too long.');
```

```

  if (input.footnote != null && input.footnote.length > 500)
    throw new Error('Footnote is too long.');
```

```

  const modelUpdated = await prisma.bookChapter.update({
    where: { id: input.id },
    data: {
      title: input.title,
      foreword: input.foreword,
      footnote: input.footnote,
      content: input.content,
      wordsTotal: countWords(input.content)
    }
  });

  await updateBookStats(modelUpdated.bookId);
  return toBookChapterSchema(modelUpdated, { req, res, identity });
}

-- app/server/graphql/resolvers/reviewManyResolver.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { QueryReviewManyArgs, ReviewManyOutput } from '@generated/types';
import { toReviewSchema } from '../mappers/reviewMapper';

export async function ReviewManyResolver(
  _: any,
  { input }: QueryReviewManyArgs,
  { req, res, identity }: IdentityContext
): Promise<ReviewManyOutput> {
  const { bookId, filter } = input;

  const items = await prisma.review.findMany({
    where: { bookId },
    skip: filter.offset ?? 0,
    take: filter.size ?? 10,
    orderBy: {
      createdAt: 'desc'
    }
  });
}

```

```

return {
  items: await Promise.all(items.map(value => toReviewSchema(value))),
  total: await prisma.review.count({ where: { bookId } })
};
}

-- app/server/graphql/resolvers/bookChapterCreateResolver.ts --
import { countWords, ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma, uuid } from '@app/server/prisma/client';
import { updateBookStats } from '@app/server/service/book';
import { BookChapter, MutationBookChapterCreateArgs } from '@generated/types';
import { toBookChapterSchema } from './mappers/bookChapterMapper';

export async function BookChapterCreateResolver(
  _: any,
  { input }: MutationBookChapterCreateArgs,
  { req, res, identity }: IdentityContext
): Promise<BookChapter> {
  ensureIdentity(identity);

  if (input.title.length > 60) throw new Error('Title is too long.');
```

```

  if (input.foreword != null && input.foreword.length > 500)
    throw new Error('Foreword is too long.');
```

```

  if (input.footnote != null && input.footnote.length > 500)
    throw new Error('Footnote is too long.');
```

```

  const chapterCount = await prisma.bookChapter.count({ where: { bookId: input.bookId } });

  const model = await prisma.bookChapter.create({
    data: {
      id: uuid(),
      bookId: input.bookId,
      title: input.title,
      foreword: input.foreword,
      footnote: input.footnote,
      content: input.content,
      wordsTotal: countWords(input.content),
      sequenceNumber: chapterCount + 1
    }
  });

  await updateBookStats(input.bookId);
  return toBookChapterSchema(model, { req, res, identity });
}

-- app/server/graphql/resolvers/reviewUpdateResolver.ts --

-- app/server/graphql/resolvers/bookCreateResolver.ts --
import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma, uuid } from '@app/server/prisma/client';
import { Book, MutationBookCreateArgs } from '@generated/types';
import { toBookSchema } from './mappers/bookMapper';

export async function BookCreateResolver(
  _: any,
  { input }: MutationBookCreateArgs,
  { req, res, identity }: IdentityContext
): Promise<Book> {
  ensureIdentity(identity);

```

```

if (input.title.length > 60) throw new Error('Title is too long.');
```

```

if (input.description.length > 900) throw new Error('Description is too long.');
```

```

const model = await prisma.book.create({
  data: {
    id: uuid(),
    title: input.title,
    description: input.description,
    authorId: identity.id
  }
});
return toBookSchema(model, { req, res, identity });
}

-- app/server/graphql/resolvers/identityResolver.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { Identity } from '@generated/types';
import { toIdentitySchema } from './mappers/identityMapper';

export async function IdentityResolver(
  _: any,
  __: any,
  { identity }: IdentityContext
): Promise<Identity> {
  if (identity == null) throw new Error('Unauthorized.');
```

```

  const model = await prisma.user.findUniqueOrThrow({ where: { id: identity.id } });
  return toIdentitySchema(model);
}

-- app/server/graphql/resolvers/reviewDeleteResolver.ts --

-- app/server/graphql/resolvers/bookUpdateResolver.ts --
import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { Book, MutationBookUpdateArgs } from '@generated/types';
import { toBookSchema } from './mappers/bookMapper';

export async function BookUpdateResolver(
  _: any,
  { input }: MutationBookUpdateArgs,
  { req, res, identity }: IdentityContext
): Promise<Book> {
  ensureIdentity(identity);

  const model = await prisma.book.findUniqueOrThrow({
    where: { id: input.id }
  });

  if (identity.id !== model.authorId) throw new Error('Operation is not permitted.');
```

```

  if (input.title !== null && input.title.length > 60) throw new Error('Title is too long.');
```

```

  if (input.description !== null && input.description.length > 900)
    throw new Error('Description is too long.');
```

```

  const modelUpdated = await prisma.book.update({
    where: { id: input.id },
    data: {
      title: input.title !== null ? input.title : undefined,
      description: input.description !== null ? input.description : undefined,
      progressStatus: input.progressStatus !== null ? input.progressStatus : undefined
    }
  });
  return toBookSchema(modelUpdated, { req, res, identity });
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						98
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
  });

  return toBookSchema(modelUpdated, { req, res, identity });
}

-- app/server/graphql/resolvers/userOneResolver.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { QueryUserOneArgs, User } from '@generated/types';
import { toUserSchema } from './mappers/userMapper';

export async function UserOneResolver(
  _: any,
  { input }: QueryUserOneArgs,
  { identity }: IdentityContext
): Promise<User> {
  if (input.id == null && input.username == null) throw new Error('Invalid user input.');
```

```

  if (input.id != null && input.username != null) throw new Error('Invalid user input.');
```

```

  const model = await prisma.user.findUniqueOrThrow({
    where: {
      id: input.id != null ? input.id : undefined,
      username: input.username != null ? input.username : undefined
    }
  });
  return toUserSchema(model);
}

-- app/server/graphql/resolvers/ratingSetResolver.ts --
import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma, uuid } from '@app/server/prisma/client';
import { MutationRatingSetArgs, RatingType } from '@generated/types';

export async function RatingSetResolver(
  _: any,
  { input }: MutationRatingSetArgs,
  { identity }: IdentityContext
): Promise<void> {
  ensureIdentity(identity);
  if (input.bookId != null) {
    const rating = await prisma.rating.findUnique({
      where: {
        userId_bookId: {
          userId: identity.id,
          bookId: input.bookId
        }
      }
    });
    if (rating != null && input.ratingType !== RatingType.None) {
      await prisma.rating.update({
        where: { id: rating.id },
        data: {
          type: input.ratingType
        }
      });
    }
    return;
  }

  if (input.ratingType === RatingType.None) {
    await prisma.rating.delete({

```

					ІАЛЦ.467200.007 ДГ	Арк.
						99
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        where: {
          userId_bookId: {
            userId: identity.id,
            bookId: input.bookId
          }
        }
      });
    return;
  }

  await prisma.rating.create({
    data: {
      id: uuid(),
      userId: identity.id,
      bookId: input.bookId,
      type: input.ratingType
    }
  });

  return;
}

if (input.bookChapterId != null) {
  const rating = await prisma.rating.findUnique({
    where: {
      userId_bookChapterId: {
        userId: identity.id,
        bookChapterId: input.bookChapterId
      }
    }
  });
}

if (rating != null && input.ratingType !== RatingType.None) {
  await prisma.rating.update({
    where: { id: rating.id },
    data: {
      type: input.ratingType
    }
  });
  return;
}

if (input.ratingType === RatingType.None) {
  await prisma.rating.delete({
    where: {
      userId_bookChapterId: {
        userId: identity.id,
        bookChapterId: input.bookChapterId
      }
    }
  });
  return;
}

await prisma.rating.create({
  data: {
    id: uuid(),
    userId: identity.id,
    bookChapterId: input.bookChapterId,
    type: input.ratingType
  }
});

return;
}

```

					ІАЛЦ.467200.007 ДГ	Арк.
						100
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    throw new Error('Invalid Rating id input.');
```

```

}
```

```

-- app/server/graphql/resolvers/bookDeleteResolver.ts --
```

```

import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { MutationBookDeleteArgs } from '@generated/types';
```

```

export async function BookDeleteResolver(
```

```
  _: any,
  { id }: MutationBookDeleteArgs,
  { identity }: IdentityContext
```

```
): Promise<void> {
  ensureIdentity(identity);
```

```

  const model = await prisma.book.findUniqueOrThrow({
    where: { id }
  });
```

```

  if (identity.id !== model.authorId) throw new Error('Operation is not permitted.');
```

```

  await prisma.book.delete({
    where: { id }
  });
```

```

}
```

```

-- app/server/graphql/resolvers/bookChapterManyResolver.ts --
```

```

import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { BookChapterManyOutput, QueryBookChapterManyArgs } from '@generated/types';
import { Prisma } from '@prisma/client';
import { toBookChapterSchema } from '../mappers/bookChapterMapper';
```

```

export async function BookChapterManyResolver(
```

```
  _: any,
  { input }: QueryBookChapterManyArgs,
  { req, res, identity }: IdentityContext
```

```
): Promise<BookChapterManyOutput> {
  const { bookId, filter } = input;
  const terms = filter.terms?.toLowerCase();
```

```

  const bookChapterWhere: Prisma.BookChapterWhereInput = {
    AND: [{ bookId }, { OR: terms !== null ? [{ title: { contains: terms } }] : undefined }];
  };
```

```

  const items = await prisma.bookChapter.findMany({
    where: bookChapterWhere,
    skip: filter.offset ?? 0,
    take: filter.size ?? 10,
    orderBy: { sequenceNumber: 'asc' }
  });
```

```

  return {
    items: await Promise.all(
      items.map(value => toBookChapterSchema(value, { req, res, identity })),
    ),
    total: items.length
  };
}
```

```

-- app/server/graphql/resolvers/bookManyResolver.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { BookManyOutput, QueryBookManyArgs } from '@generated/types';
import { Prisma } from '@prisma/client';
import { toBookSchema } from '../mappers/bookMapper';

export async function BookManyResolver(
  _: any,
  { input }: QueryBookManyArgs,
  { req, res, identity }: IdentityContext
): Promise<BookManyOutput> {
  const { filter } = input;
  const terms = filter.terms?.toLowerCase();

  const bookWhere: Prisma.BookWhereInput = {
    OR:
      terms != null ? [{ title: { contains: terms } }, { author: { username: terms } }] :
undefined
  };

  const items = await prisma.book.findMany({
    where: bookWhere,
    skip: filter.offset ?? 0,
    take: filter.size ?? 10,
    orderBy: {
      createdAt: 'desc'
    }
  });

  return {
    items: await Promise.all(items.map(value => toBookSchema(value, { req, res, identity
}))),
    total: await prisma.book.count({ where: bookWhere })
  };
}

-- app/server/graphql/resolvers/bookChapterDelete.ts --
import { ensureIdentity } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { updateBookStats } from '@app/server/service/book';
import { Book, MutationBookChapterDeleteArgs } from '@generated/types';
import { toBookSchema } from '../mappers/bookMapper';

export async function BookChapterDeleteResolver(
  _: any,
  { id }: MutationBookChapterDeleteArgs,
  { req, res, identity }: IdentityContext
): Promise<Book> {
  ensureIdentity(identity);
  const model = await prisma.bookChapter.findUniqueOrThrow({
    where: { id },
    select: {
      sequenceNumber: true,
      book: { select: { id: true, authorId: true } }
    }
  });
  if (identity.id !== model.book.authorId) throw new Error('Operation is not permitted.');
```

```

  await prisma.bookChapter.delete({ where: { id } });

  await prisma.bookChapter.updateMany({
    where: {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						102
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        bookId: model.book.id,
        sequenceNumber: {
            gt: model.sequenceNumber
        }
    },
    data: {
        sequenceNumber: {
            decrement: 1
        }
    }
});

await updateBookStats(model.book.id);
const book = await prisma.book.findUniqueOrThrow({ where: { id: model.book.id } });
return toBookSchema(book, { req, res, identity });
}

-- app/server/graphql/resolvers/bookChapterOneResolver.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { BookChapter, QueryBookChapterOneArgs } from '@generated/types';
import { toBookChapterSchema } from './mappers/bookChapterMapper';

export async function BookChapterOneResolver(
    _: any,
    { id }: QueryBookChapterOneArgs,
    { req, res, identity }: IdentityContext
): Promise<BookChapter> {
    const model = await prisma.bookChapter.findUniqueOrThrow({
        where: {
            id
        }
    });
    return toBookChapterSchema(model, { req, res, identity });
}

-- app/server/graphql/resolvers/mappers/bookChapterMapper.ts --
import { call } from '@app/common';
import { ensureRatingType } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { BookChapter as BookChapterSchema, RatingType } from '@generated/types';
import { BookChapter } from '@prisma/client';
import { toBookSchema } from './bookMapper';

export async function toBookChapterSchema(
    model: BookChapter,
    { req, res, identity }: IdentityContext
): Promise<BookChapterSchema> {
    return {
        id: model.id,
        title: model.title,
        createdAt: model.createdAt.toISOString(),
        updatedAt: model.updatedAt.toISOString(),
        foreword: model.foreword,
        footnote: model.footnote,
        content: model.content,
        sequenceNumber: model.sequenceNumber,
        previousChapterId: await call(async () => {
            const isFirst = model.sequenceNumber === 1;
            if (isFirst) return null;
            const previousChapter = await prisma.bookChapter.findUniqueOrThrow({
                where: {

```

					ІАЛЦ.467200.007 ДГ	Арк.
						103
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        bookId_sequenceNumber: {
            bookId: model.bookId,
            sequenceNumber: model.sequenceNumber - 1
        }
    },
    select: {
        id: true
    }
});
return previousChapter.id;
}),
nextChapterId: await call(async () => {
    const book = await prisma.book.findUniqueOrThrow({
        where: { id: model.bookId },
        select: { chaptersTotal: true }
    });

    const isLast = model.sequenceNumber === book.chaptersTotal;
    if (isLast) return null;

    const nextChapter = await prisma.bookChapter.findUniqueOrThrow({
        where: {
            bookId_sequenceNumber: {
                bookId: model.bookId,
                sequenceNumber: model.sequenceNumber + 1
            }
        },
        select: { id: true }
    });
    return nextChapter.id;
}),
reviewsTotal: await call(async () => {
    const reviewsCount = await prisma.review.count({
        where: {
            bookChapterId: model.id
        }
    });
    return reviewsCount;
}),
book: await call(async () => {
    const book = await prisma.book.findUniqueOrThrow({
        where: {
            id: model.bookId
        }
    });
    return toBookSchema(book, { req, res, identity });
}),
personalRating: await call(async () => {
    if (identity == null) return RatingType.None;
    const rating = await prisma.rating.findUnique({
        where: {
            userId_bookChapterId: {
                userId: identity.id,
                bookChapterId: model.id
            }
        }
    });
    if (rating == null) return RatingType.None;
    ensureRatingType(rating.type);
    return rating.type;
})
});
}

```

```

-- app/server/graphql/resolvers/mappers/userMapper.ts --
import { User as UserSchema } from '@generated/types';
import { User } from '@prisma/client';

export async function toUserSchema(model: User): Promise<UserSchema> {
  return {
    id: model.id,
    username: model.username,
    createdAt: model.createdAt.toISOString(),
    updatedAt: model.updatedAt.toISOString()
  };
}

-- app/server/graphql/resolvers/mappers/identityMapper.ts --
import { Identity as IdentitySchema } from '@generated/types';
import { User } from '@prisma/client';
import { toUserSchema } from './userMapper';

export async function toIdentitySchema(model: User): Promise<IdentitySchema> {
  return {
    id: model.id,
    email: model.email,
    isDarkTheme: true,
    user: await toUserSchema(model)
  };
}

-- app/server/graphql/resolvers/mappers/bookMapper.ts --
import { assertUnreachable } from '@app/client/helpers';
import { call } from '@app/common';
import { ensureRatingType } from '@app/server/helpers';
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import {
  Book as BookSchema,
  RatingType as RatingTypeSchema,
  WorkingProgressStatus as WorkingProgressStatusSchema
} from '@generated/types';
import { Book, RatingType, WorkingProgressStatus } from '@prisma/client';
import { toUserSchema } from './userMapper';

export async function toBookSchema(
  model: Book,
  { identity }: IdentityContext
): Promise<BookSchema> {
  return {
    id: model.id,
    title: model.title,
    description: model.description,
    author: await call(async () => {
      const userModel = await prisma.user.findUniqueOrThrow({ where: { id: model.authorId }
    });
    return toUserSchema(userModel);
  }),
    createdAt: model.createdAt.toISOString(),
    updatedAt: model.updatedAt.toISOString(),
    progressStatus: call(() => {
      ensureWorkingProgressStatus(model.progressStatus);
      return model.progressStatus;
    }),
    wordsTotal: model.wordsTotal,
    chaptersTotal: await call(async () => {
      const chaptersCount = await prisma.bookChapter.count({

```

					ІАЛЦ.467200.007 ДГ	Арк.
						105
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        where: {
          bookId: model.id
        }
      });
      return chaptersCount;
    }
  },
  ratingTotal: await call(async () => {
    const ratingsUpCount = await prisma.rating.count({
      where: {
        bookId: model.id,
        type: RatingType.up
      }
    });

    const ratingsDownCount = await prisma.rating.count({
      where: {
        bookId: model.id,
        type: RatingType.down
      }
    });

    return ratingsUpCount - ratingsDownCount;
  }),
  reviewsTotal: await call(async () => {
    const reviewsCount = await prisma.review.count({
      where: {
        bookId: model.id
      }
    });
    return reviewsCount;
  }),
  personalRating: await call(async () => {
    if (identity == null) return RatingTypeSchema.None;
    const rating = await prisma.rating.findUnique({
      where: {
        userId_bookId: {
          userId: identity.id,
          bookId: model.id
        }
      }
    });
    if (rating == null) return RatingTypeSchema.None;
    ensureRatingType(rating.type);
    return rating.type;
  })
};
}

function ensureWorkingProgressStatus(
  value: WorkingProgressStatus
): asserts value is WorkingProgressStatusSchema {
  if (value === WorkingProgressStatus.inProgress) return;
  if (value === WorkingProgressStatus.frozen) return;
  if (value === WorkingProgressStatus.complete) return;
  assertUnreachable(value);
}

-- app/server/graphql/resolvers/mappers/reviewMapper.ts --
import { Review } from '@prisma/client';
import { Review as ReviewSchema } from '@generated/types';
import { toUserSchema } from './userMapper';
import { call } from '@app/common';
import { prisma } from '@app/server/prisma/client';

```

```

export async function toReviewSchema(model: Review): Promise<ReviewSchema> {
  return {
    id: model.id,
    author: await call(async () => {
      const userModel = await prisma.user.findUniqueOrThrow({
        where: {
          id: model.userId
        }
      });
      return toUserSchema(userModel);
    }),
    text: model.text,
    createdAt: model.createdAt.toISOString(),
    updatedAt: model.updatedAt.toISOString(),
    // todo: ratings implementation
    ratingTotal: await call(async () => {
      return 0;
    })
  };
}

```

```

-- app/server/graphql/resolvers/auth/logout.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';

```

```

export async function LogoutResolver(
  _: any,
  __: any,
  { req, res, identity }: IdentityContext
): Promise<void> {
  const cookie: { jwt?: string | null } = req.cookies;
  if (cookie == null || cookie.jwt == null) return;

  const refreshToken = cookie.jwt;

  const user = await prisma.user.findUnique({
    where: { refreshToken },
    select: { id: true }
  });

  if (user == null) {
    res.clearCookie('jwt', { httpOnly: true });
    return;
  }

  await prisma.user.update({
    where: {
      id: user.id
    },
    data: { refreshToken: null }
  });

  res.clearCookie('jwt', { httpOnly: true });
}

```

```

-- app/server/graphql/resolvers/auth/login.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma } from '@app/server/prisma/client';
import { MutationLoginArgs, Session } from '@generated/types';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { config } from '@app/server/main';

```

					ІАЛЦ.467200.007 ДГ	Арк.
						107
Зм.	Арк.	№ докум.	Підпис	Дата		

```

export async function LoginResolver(
  _: any,
  { input }: MutationLoginArgs,
  { req, res, identity }: IdentityContext
): Promise<Session> {
  const error = 'Invalid password or email';

  const user = await prisma.user.findUnique({
    where: { email: input.email }
  });
  if (user == null) throw new Error(error);

  const match = await bcrypt.compare(input.password, user.password);
  if (!match) throw new Error(error);

  const accessToken = jwt.sign({ userId: user.id }, config.accessToken.secret, {
    expiresIn: config.accessToken.expiresIn
  });
  const refreshToken = jwt.sign({ userId: user.id }, config.refreshToken.secret, {
    expiresIn: config.refreshToken.expiresIn
  });

  await prisma.user.update({
    where: { id: user.id },
    data: { refreshToken }
  });

  res.cookie('jwt', refreshToken, {
    httpOnly: true,
    sameSite: 'none',
    secure: true,
    maxAge: config.refreshToken.maxAge
  });

  return {
    userId: user.id,
    accessToken
  };
}

-- app/server/graphql/resolvers/auth/signUp.ts --
import { IdentityContext } from '@app/server/identity';
import { prisma, uuid } from '@app/server/prisma/client';
import { MutationSignUpArgs } from '@generated/types';
import bcrypt from 'bcrypt';

export async function SignUpResolver(
  _: any,
  { input }: MutationSignUpArgs,
  { req, res, identity }: IdentityContext
): Promise<void> {
  const existingUsername = await prisma.user.findUnique({
    where: { username: input.username },
    select: { id: true }
  });
  if (existingUsername != null) throw new Error('User with this username already exists.');
```

```

  const existingEmail = await prisma.user.findUnique({
    where: { email: input.email },
    select: { id: true }
  });
  if (existingEmail != null) throw new Error('Email is already in use.');
```

```

  const hashedPassword = await bcrypt.hash(input.password, 10);
```

					ІАЛЦ.467200.007 ДГ	Арк.
						108
Зм.	Арк.	№ докум.	Підпис	Дата		

```

await prisma.user.create({
  data: {
    id: uuid(),
    email: input.email,
    username: input.username,
    password: hashedPassword
  }
});
}

-- app/server/graphql/graphql.ts --
/* gql imports */
import { MutationResolvers, QueryResolvers } from '@generated/types';

/* Query */
import { BookChapterManyResolver } from './resolvers/bookChapterManyResolver';
import { BookChapterOneResolver } from './resolvers/bookChapterOneResolver';
import { BookManyResolver } from './resolvers/bookManyResolver';
import { BookOneResolver } from './resolvers/bookOneResolver';
import { IdentityResolver } from './resolvers/identityResolver';
import { ReviewManyResolver } from './resolvers/reviewManyResolver';
import { UserOneResolver } from './resolvers/userOneResolver';

/* Mutation */
import { LoginResolver } from './resolvers/auth/login';
import { LogoutResolver } from './resolvers/auth/logout';
import { SignUpResolver } from './resolvers/auth/signUp';
import { BookUpdateResolver } from './resolvers/bookUpdateResolver';
import { BookDeleteResolver } from './resolvers/bookDeleteResolver';
import { BookChapterCreateResolver } from './resolvers/bookChapterCreateResolver';
import { BookChapterDeleteResolver } from './resolvers/bookChapterDelete';
import { BookChapterUpdateResolver } from './resolvers/bookChapterUpdateResolver';
import { BookCreateResolver } from './resolvers/bookCreateResolver';
import { RatingSetResolver } from './resolvers/ratingSetResolver';
import { ReviewCreateResolver } from './resolvers/reviewCreateResolver';

export const Query: QueryResolvers = {
  bookOne: BookOneResolver,
  bookMany: BookManyResolver,
  bookChapterOne: BookChapterOneResolver,
  bookChapterMany: BookChapterManyResolver,
  identity: IdentityResolver,
  reviewMany: ReviewManyResolver,
  userOne: UserOneResolver
};

export const Mutation: MutationResolvers = {
  signUp: SignUpResolver,
  login: LoginResolver,
  logout: LogoutResolver,
  bookCreate: BookCreateResolver,
  bookUpdate: BookUpdateResolver,
  bookDelete: BookDeleteResolver,
  bookChapterCreate: BookChapterCreateResolver,
  bookChapterDelete: BookChapterDeleteResolver,
  bookChapterUpdate: BookChapterUpdateResolver,
  ratingSet: RatingSetResolver,
  reviewCreate: ReviewCreateResolver
};

export const Subscription = {};

```

```

-- app/server/express/auth/refreshToken.ts --
import { config } from '@app/server/main';
import { prisma } from '@app/server/prisma/client';
import { Session } from '@generated/types';
import 'dotenv/config';
import { Request, Response } from 'express';
import jwt from 'jsonwebtoken';

export async function refreshTokenController(req: Request, res: Response) {
  try {
    const cookie: { jwt?: string | null } = req.cookies;

    if (cookie == null || cookie.jwt == null) {
      res.sendStatus(401);
      return;
    }

    const refreshToken = cookie.jwt;

    const user = await prisma.user.findUnique({
      where: { refreshToken },
      select: { id: true }
    });

    if (user == null) {
      res.statusMessage = 'User not found.';
      res.status(401).end();
      return;
    }

    const decoded = jwt.verify(refreshToken, config.refreshToken.secret) as {
      userId: string;
      iat: number;
      exp: number;
    };

    if (user.id !== decoded.userId) {
      res.statusMessage = 'Invalid token.';
      res.status(401).end();
      return;
    }

    const accessToken = jwt.sign({ userId: user.id }, config.accessToken.secret, {
      expiresIn: config.accessToken.expiresIn
    });

    const session: Session = { userId: user.id, accessToken };
    res.json(session);
  } catch (error) {
    console.error(error);
  }
}

-- app/server/service/book.ts --
import { prisma } from '@app/server/prisma/client';

export async function updateBookStats(id: string) {
  const bookWordsTotal = await prisma.bookChapter.aggregate({
    where: {
      bookId: id
    },
    _sum: {
      wordsTotal: true
    }
  });

```

					ІАЛЦ.467200.007 ДГ	Арк.
						110
Зм.	Арк.	№ докум.	Підпис	Дата		

```

});

const chaptersTotal = await prisma.bookChapter.count({
  where: {
    bookId: id
  }
});

return prisma.book.update({
  where: { id },
  data: {
    wordsTotal: bookWordsTotal._sum.wordsTotal !== null ? bookWordsTotal._sum.wordsTotal :
0,
    chaptersTotal
  }
});
}

-- app/server/config.ts --
import cors from 'cors';
import 'dotenv/config';

interface AppConfig {
  readonly isProd: boolean;
  readonly host: string;
  readonly port: number;
  readonly limit: { readonly limit: string };
  readonly origin: string[];
  readonly corsOptions: cors.CorsOptions;
  readonly greeting: string;
  readonly accessToken: TokenConfig;
  readonly refreshToken: RefreshTokenConfig;
}

interface TokenConfig {
  readonly secret: string;
  readonly expiresIn: number;
}

interface RefreshTokenConfig extends TokenConfig {
  readonly maxAge: number;
}

export function toAppConfig(): AppConfig {
  const MODE = process.env.VITE_MODE || 'prod';
  const isProd = MODE === 'prod';

  const host = process.env.VITE_HOST || 'localhost';
  const port = parseInt(process.env.VITE_PORT || '3000');

  const hostRemote = process.env.VITE_HOST_REMOTE || 'localhost';
  const domainName = process.env.VITE_DOMAIN_NAME || 'localhost';

  const urlIp = `http://${hostRemote}:${port}`;
  const urlDomain = `http://${domainName}:${port}`;
  const urlRemote = `https://${host}:${port}`;
  const urlLocal = host !== 'localhost' ? `https://localhost:${port}` : undefined;

  const limit = { limit: '50mb' };

  const origin = urlLocal !== null ? [urlLocal, urlRemote, urlIp, urlDomain] : [urlRemote];
  const corsOptions: cors.CorsOptions = {
    origin,
    credentials: true
  }

```

					ІАЛЦ.467200.007 ДГ	Арк.
						111
Зм.	Арк.	№ докум.	Підпис	Дата		

```

};

const accessTokenExpiresIn = 24 * 60 * 60;
const refreshTokenExpiresIn = 15 * 24 * 60 * 60;

const accessTokenSecret = process.env.ACCESS_TOKEN_SECRET;
const refreshTokenSecret = process.env.REFRESH_TOKEN_SECRET;
if (accessTokenSecret == null || refreshTokenSecret == null) throw new Error('No secrets found');

const accessToken: TokenConfig = {
  secret: accessTokenSecret,
  expiresIn: accessTokenExpiresIn
};

const refreshToken: RefreshTokenConfig = {
  secret: refreshTokenSecret,
  expiresIn: refreshTokenExpiresIn,
  maxAge: refreshTokenExpiresIn * 1000
};

const greeting = `\\n\\n\\n\\x1b[35m-server is listening at...-
\\x1b[0m\\n\\n${urlRemote}\\n${urlLocal}\\n\\n\\n`;

return {
  isProd,
  host,
  port,
  limit,
  origin,
  corsOptions,
  accessToken,
  refreshToken,
  greeting
};
}

-- app/server/main.ts --
import { ApolloServer } from '@apollo/server';
import { expressMiddleware } from '@apollo/server/express4';
import { ApolloServerPluginDrainHttpServer } from '@apollo/server/plugin/drainHttpServer';
import cookies from 'cookie-parser';
import cors from 'cors';
import 'dotenv/config';
import express from 'express';
import fs from 'fs';
import { GraphQLJSON } from 'graphql-scalars';
import https from 'https';
import path from 'path';
import ViteExpress from 'vite-express';
import { toAppConfig } from './config';
import { refreshTokenController } from './express/auth/refreshToken';
import { Mutation, Query } from './graphql/graphql';
import { getIdentityContext, IdentityContext } from './identity';

export const config = toAppConfig();

const __dirname = import.meta.dirname;
const key = fs.readFileSync(path.join(__dirname + '../../../certs/selfsigned.key'));
const cert = fs.readFileSync(path.join(__dirname + '../../../certs/selfsigned.crt'));

const app = express();
const server = https.createServer({ key, cert }, app);

```

					ІАЛЦ.467200.007 ДГ	Арк.
						112
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const typeDefs = fs.readFileSync(path.join(__dirname, 'graphql', 'schema.graphql'), 'utf8');

const resolvers = {
  JSON: GraphQLJSON,
  Query,
  Mutation
  // Subscription,
};

const apollo = new ApolloServer<IdentityContext>({
  typeDefs,
  resolvers,
  introspection: !config.isProd,
  plugins: [ApolloServerPluginDrainHttpServer({ httpServer: server })],
  formatError: formattedError => {
    process.stdout.write('\n```\n');
    console.error(formattedError);
    process.stdout.write('\n```\n');
    return formattedError;
  }
});

await apollo.start();

app.use(cors<cors.CorsRequest>(config.corsOptions));
app.use(express.json(config.limit));
app.use(cookies());
app.get('/api/refreshAccessToken', refreshTokenController);
app.use(
  '/graphql',
  expressMiddleware(apollo, {
    context: getIdentityContext
  })
);

await new Promise<void>(resolve =>
  server.listen({ hostname: config.host, port: config.port }, resolve)
);

process.stdout.write(config.greeting);

ViteExpress.bind(app, server);

-- app/server/identity.ts --
import { Session } from '@generated/types';
import 'dotenv/config';
import { Request, Response } from 'express';
import { GraphQLError } from 'graphql';
import jwt from 'jsonwebtoken';
import { config } from './main';
import { prisma } from './prisma/client';

export interface IdentityContext {
  req: Request;
  res: Response;
  identity?: {
    id: string;
    email: string;
    username: string;
  };
}

export async function getIdentityContext({
  req,

```

					ІАЛЦ.467200.007 ДГ	Арк.
						113
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    res
  }: {
    req: Request;
    res: Response;
  }): Promise<IdentityContext> {
    try {
      const authHeaders = req.headers.authorization;
      if (authHeaders == null) return { req, res };

      const session: Session = JSON.parse(authHeaders);
      const { accessToken } = session;

      const decoded = jwt.verify(accessToken, config.accessToken.secret) as {
        userId: string;
        iat: number;
        exp: number;
      };

      const identity = await prisma.user.findUnique({
        where: { id: decoded.userId },
        select: {
          id: true,
          email: true,
          username: true
        }
      });
      if (identity == null) throw new Error("Couldn't find the user.");
      return { req, res, identity };
    } catch (error) {
      if (error instanceof Error)
        throw new GraphQLError(error.message, {
          extensions: {
            code: 'UNAUTHENTICATED',
            http: { status: 401 }
          }
        });
      throw new GraphQLError(typeof error === 'string' ? error : 'Unknown error', {
        extensions: {
          code: 'UNAUTHENTICATED',
          http: { status: 401 }
        }
      });
    }
  }
}

```

Зм.	Арк.	№ докум.	Підпис	Дата

ІАЛЦ.467200.007 ДГ

Арк.

114