

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Веб-платформа для читання коміксів»

Виконав студент групи ІПЗ-21-2
ШАРОБОКОВ Іван Віталійович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Консультант:
МОРОЗОВ Андрій Васильович

Рецензент:
ТУРЕНКО Єлизавета Єгорівна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ШАРОБОКОВ Іван Віталійович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Веб-платформа для читання коміксів»**,

затверджена Наказом закладу вищої освіти №61/с від **14.02.2025**.

Вихідні дані для роботи: **розробка веб-платформи для читання коміксів, що забезпечує інтерактивний перегляд, мультимедійні можливості та соціальну взаємодію між користувачами.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А. В.	27.02.2025р.	27.02.2025р.
2	Морозов А. В.	27.02.2025р.	27.02.2025р.
3	Морозов А. В.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота складається з вебдодатку платформи для читання коміксів та пояснювальної записки. Пояснювальна записка до випускної роботи містить 96 сторінок, 52 ілюстрації, 2 таблиці, 7 додатків та 30 літературних джерел.

Метою роботи є створення онлайн веб-платформи для читання коміксів з можливістю музичного супроводу та спільного читання з іншими користувачами.

Архітектура вебплатформи побудована на основі клієнт-серверної моделі із застосуванням сучасних вебтехнологій, що забезпечують швидкий обмін даними, адаптивний інтерфейс та стабільну синхронізацію аудіо і зображень. Реалізовано функціональність створення кімнат, у яких кілька користувачів можуть одночасно та синхронізовано переглядати один і той самий комікс із музичним супроводом.

Додаток має інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам переглядати комікси, додавати аудіотреки, створювати або приєднуватися до кімнат спільного перегляду з можливістю синхронізованого читання та спілкування між учасниками у реальному часі.

Результати досліджень доцільно використати для застосунків, що прагнуть впровадити не лише візуальне, а й слухове сприйняття контенту. Також отримані дослідження можна використати для впровадження функціоналу взаємодії користувачів у режимі реального часу.

КЛЮЧОВІ СЛОВА: ВЕБ-ПЛАТФОРМА, КОМІКС, АУДІОСУПРОВІД, КІМНАТА.

					ІПЗ.КР.Б – 121 – 25 – ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата								
Розроб.		Шаробоків І. В.			Розробка веб-платформи для читання коміксів			Літ.	Арк.	Аркушів		
										3	96	
Керівник		Морозов А. В.						Житомирська політехніка, група ІПЗ-21-2				
Н. контр.		Туренко Є. Є.										
Зав. каф.		Вакалюк Т. А.										

ABSTRACT

The bachelor's thesis consists of a web application – a platform for reading comics and an explanatory note. The explanatory note to the final thesis contains 96 pages, 52 illustrations, 2 tables, 7 appendices, and 30 literary sources.

The goal of the work is to develop an online web platform for reading comics with the possibility of musical accompaniment and shared reading with other users.

The architecture of the web platform is based on a client-server model using modern web technologies that ensure fast data exchange, a responsive interface, and stable synchronization of audio and images. The functionality of creating rooms has been implemented, where multiple users can simultaneously and synchronously view the same comic with musical accompaniment.

The app has an intuitive interface that allows users to read comics, add audio tracks, create or join shared reading rooms with the ability to read synchronously and communicate with other participants in real time.

The research results are advisable to apply in applications that aim to incorporate not only visual but also auditory perception of content. The findings can also be used to implement real-time user interaction functionality.

KEYWORDS: WEB PLATFORM, COMIC, MUSICAL ACCOMPANIMENT, ROOM.

		Шаробок І. В.			ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ПЛАТФОРМИ ДЛЯ ЧИТАННЯ КОМІКСІВ.....	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури веб-орієнтованої платформи для читання коміксів ..	16
1.4. Обґрунтування вибору інструментальних засобів	24
Висновки до першого розділу.....	27
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ЧИТАННЯ.....	28
КОМІКСІВ.....	28
2.1. Вимоги до системи.....	28
2.2. Розробка бази даних системи	34
2.3. Проектування та реалізація алгоритмів роботи системи	36
2.4. Реалізація додатку веб-платформи для читання коміксів	50
Висновки до другого розділу	53
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ПЛАТФОРМИ ДЛЯ ЧИТАННЯ КОМІКСІВ	54
3.1. Порядок встановлення та налаштування параметрів системи.....	54
3.2. Структура інтерфейсу веб-платформи для читання коміксів	56
3.3. Тестування роботи програмного комплексу.....	69
Висновки до третього розділу.....	73
ВИСНОВКИ	75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	77
ДОДАТКИ.....	80

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ORM – об’єктно-реляційне відображення

MVC – схема розділення даних додатків і керуючої логіки на три окремі компоненти Model-View-Controller

CRUD – основні операції, що використовуються для управління даними (Create, Read, Update, Delete)

SSR – рендеринг на стороні серверу

CSR – рендеринг на стороні клієнту

JWT – стандарт передачі токенів для аутентифікації

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. Веб-платформа для читання коміксів є надзвичайно актуальною в умовах стрімкого розвитку цифрових медіа, зростання популярності візуального контенту та переходу користувачів до онлайн-форматів споживання інформації. Комікси як форма мистецтва та розваги привертають увагу широкої аудиторії – від підлітків до дорослих читачів. Традиційні друковані видання поступово поступаються місцем цифровим платформам, що забезпечують зручний доступ, інтерактивні можливості та персоналізацію користувацького досвіду. Створення сучасної веб-платформи для читання коміксів дозволяє не лише популяризувати цей жанр, але й підтримувати авторів, надаючи їм нові канали дистрибуції. Таке рішення відповідає поточним запитам ринку та сприяє розвитку культурного середовища у цифрову епоху.

Метою випускної роботи є розробка зручного та функціонального онлайн-середовища, яке забезпечить користувачам доступ до широкого каталогу коміксів, можливість персоналізованого читання, збереження улюблених коміксів, взаємодію з авторами та іншими читачами. Платформа має на меті популяризувати цифрові комікси, спростити процес їх пошуку та читання, підтримати розвиток незалежних авторів, а також привнести в сферу візуального контенту новий унікальний підхід до сприйняття інформації інтегруючи музичне супроводження та взаємодію читачів між собою в режимі реального часу.

Завдання роботи.

Встановлена мета обумовлює наступні завдання:

- проведення аналізу існуючих рішень для онлайн-читання коміксів, їх функціональних можливостей та підходів до взаємодії з користувачами;
- визначення вимог до системи та розробка загальної архітектури веб-платформи;
- обґрунтування та вибір технологічного стеку для реалізації клієнтської та серверної частини системи;
- проєктування та реалізація бази даних для зберігання інформації про

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

- комікси, користувачів, кімнати спільного перегляду та аудіотреки;
- розробка функціоналу читання коміксів із можливістю музичного супроводу та синхронізованого перегляду у реальному часі;
 - реалізація можливості створення та приєднання до кімнат, а також спілкування між користувачами під час спільного читання;
 - тестування веб-платформи, оцінка її зручності, стабільності та ефективності роботи.

Об’єктом дослідження є процес читання цифрових коміксів у вебсередовищі з можливістю мультимедійного супроводу та взаємодії користувачів в режимі реального часу.

Предметом дослідження є використання веб-орієнтованих технологій для розробки веб-платформи для читання коміксів.

Методи дослідження. У роботі використано методи системного аналізу для вивчення вимог до веб-платформи та визначення її функціональних компонентів, методи об’єктно-орієнтованого проєктування для моделювання структури системи, методи вебпрограмування для реалізації інтерфейсу й синхронної взаємодії користувачів, а також методи UI/UX-дизайну для створення зручного мультимедійного середовища.

Практичне значення. Розроблений вебзастосунок може бути використаний у сфері цифрових медіа, онлайн-читання та розповсюдження коміксів – на платформах видавництв, творчих спільнот або стартапів як інструмент для інтерактивного представлення візуального контенту. Завдяки персоналізованому читанню, музичному супроводу та синхронному перегляду, платформа сприяє залученню аудиторії й створенню спільнот.

Структура та обсяг проекту. Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 30 найменувань (74 стор.), 7 додатків (77 стор.), 2 таблиць та 52 ілюстрацій по тексту, з них 1 оформлена на окремій сторінці. Повний обсяг роботи становить 96 сторінок, з них 64 сторінок основного тексту.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ПЛАТФОРМИ ДЛЯ ЧИТАННЯ КОМІКСІВ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення сучасної онлайн-платформи для читання коміксів, яка поєднує в собі мультимедійний досвід, функції взаємодії користувачів та можливість спільного читання з іншими учасниками, що надає приємний інтерактивний досвід користувачу. Платформа розробляється із використанням сучасних технологій, що забезпечать її масштабованість, високу швидкодію та зручність у використанні.

Онлайн-платформа для читання коміксів – це вебзастосунок, який дозволяє користувачам переглядати ліцензійні та незалежно опубліковані комікси у зручному форматі, з можливістю автоматичного перегортання сторінок, аудіосупроводу, а також синхронного читання в режимі спільного перегляду з іншими користувачами. Проєкт передбачає інтерактивний підхід до споживання візуального контенту, роблячи досвід читання більш захопливим і соціально насиченим.

Ключовими модулями програмного комплексу є:

1. Пошуковий модуль, що надає можливість за пошуком, сортуванням та фільтрацією швидко знайти користувачу саме той комікс який він бажає.
2. Модуль автоматичного читання, який забезпечує плавне відображення сторінок коміксів за допомогою заданих таймерів та інтервалів, оптимізованих під різні типи контенту.
3. Музичний модуль, що дозволяє авторам або адміністраторам додавати саундтреки до коміксів, які синхронізуються з процесом читання для створення повноцінного емоційного ефекту.
4. Модуль спільного перегляду, який дозволяє користувачам створювати "кімнату" для синхронізованого читання, спілкуючись через чат і реагуючи на події в реальному часі.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

5. Модуль користувачів та адміністративного управління, який дає змогу налаштовувати профілі, переглядати історію читання, коментувати контент, а адміністраторам – модерувати публікації та стежити за дотриманням політики платформи.

Усі ключові елементи системи реалізуються з використанням стеку технологій JavaScript, Node.js, React, Next.js, для зберігання даних – Redis та PostgreSQL у зв’язці з ORM-бібліотекою Prisma, а для реального часу – Socket.IO. Система аутентифікації та авторизації реалізується за допомогою NextAuth, що забезпечує безпечну аутентифікацію та авторизацію користувачів.

Основні етапи розробки включають:

1. Проектування дизайну інтерфейсу:
 - 1.1. створення загальної візуальної концепції платформи.
 - 1.2. дизайн інтуїтивно зрозумілих UI-компонентів.
2. Побудова архітектури фронтенду та бекенду:
 - 2.1. розділення логіки на клієнтську та серверну частини.
 - 2.2. налаштування REST та WebSocket API.
 - 2.3. інтеграція з базою даних та системою управління станом.
3. Реалізація ключових функцій:
 - 3.1. створення системи читання з підтримкою таймерів.
 - 3.2. імплементація музичного плеєра з можливістю синхронізації.
 - 3.3. розробка механізмів реального часу для спільного перегляду.
 - 3.4. персоналізований досвід користувача з історією читання та вподобаннями.
4. Системи контролю доступу та автентифікації:
 - 4.1. рольова модель доступу до ресурсів.
 - 4.2. підтримка авторизації через OAuth.
5. Тестування та оптимізація:
 - 5.1. юніт- і інтеграційне тестування.
 - 5.2. UI/UX тестування з фокус-групами.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

5.3. профілювання швидкодії та навантаження.

У результаті розробки буде створена повноцінна платформа для читання коміксів, яка дозволяє користувачам взаємодіяти, підтримувати авторів та насолоджуватись контентом у сучасному мультимедійному форматі. Завдяки використанню сучасного стеку технологій, система буде оптимізованою для масштабування та подальшого розвитку з урахуванням нових функцій та розширень.

1.2. Аналіз аналогів програмного продукту

Сьогодні існує чимало онлайн-платформ та додатків для читання коміксів, які задовольняють різні смаки – від шанувальників манги та вебтунів до поціновувачів класичних американських графічних романів. Деякі з них спеціалізуються на оригінальному контенті, інші пропонують офіційні переклади популярних серій. Нижче наведено огляд 5 популярних платформ для читання коміксів, де коротко описано і також виділено основні переваги та недоліки кожної.

1. Webtoon

Webtoon - Один із найпопулярніших сайтів і додатків для читання вертикальних кольорових коміксів (манхва), переважно корейського походження.

До переваг даного програмного продукту можна віднести велику бібліотеку безкоштовних коміксів, зручний мобільний інтерфейс, а також професійну якість артів та перекладів.

До недоліків даного програмного продукту можна віднести те, що частина контенту є платною або має обмежений доступ за часом, а орієнтацію більшості коміксів на жанри романтики та драми.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

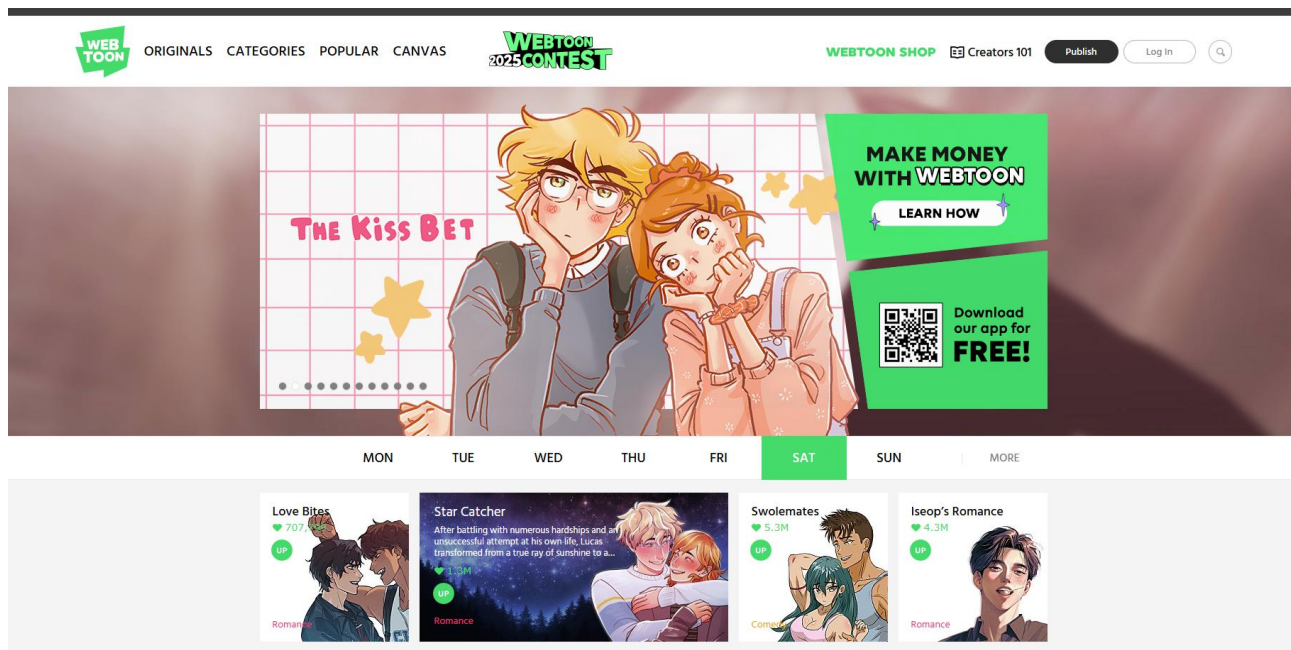


Рис. 1.1. Інтерфейс Webtoon [1]

2. Тарас

Тарас – платформа для читання вебкоміксів та коротких новел, з можливістю для авторів самостійно публікувати роботи.

До переваг даного програмного продукту можна віднести підтримку незалежних авторів, наявність як безкоштовного, так і платного контенту, зручну навігацію та систему рекомендацій, великий вибір вебкоміксів і вебновел на різні теми, зокрема фентезі, романтику, драму, трилери тощо. Крім того, користувачі мають змогу донатити авторам, купувати епізоди або залишати чайові, що додатково мотивує творців.

До недоліків даного програмного продукту можна віднести те, що частина найкращого контенту доступна лише за монети, а інтерфейс може здаватися менш сучасним порівняно з комерційними продуктами. Також іноді переклад від авторів буває нестабільним.

		Шаробок І. В.			ІІЗ.КР.Б – 121 – 25 – ІЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

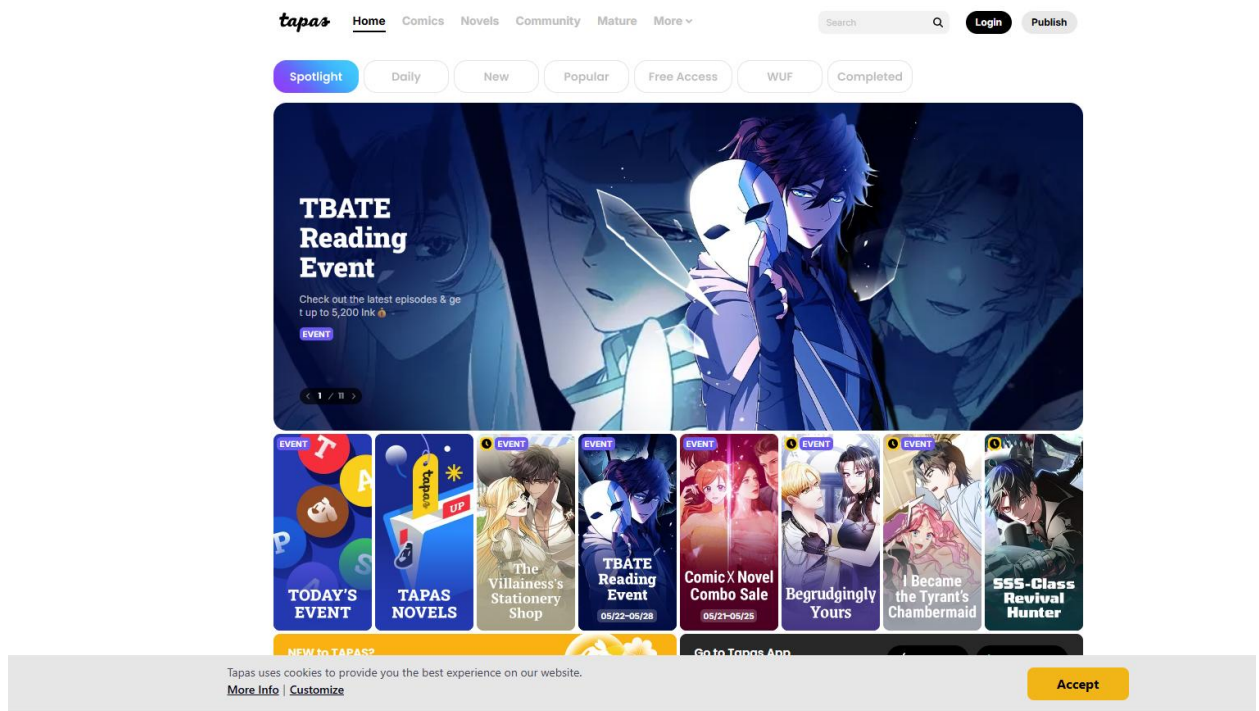


Рис. 1.2. Інтерфейс Tapas [2]

3. MangaPlus

MangaPlus – Офіційна платформа для читання манги англійською мовою від одного з найбільших японських видавництв.

До переваг даного програмного продукту можна віднести наявність офіційних перекладів, безкоштовний доступ до нових розділів, велику кількість популярних серій, а також зручний інтерфейс мобільного додатку.

До недоліків даного програмного продукту можна віднести те, що доступними є лише кілька останніх і перших розділів, обмежений каталог поза серіями Shueisha, відсутність офлайн-режиму та повноцінної системи коментарів чи спільноти.

		Шаробоків І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

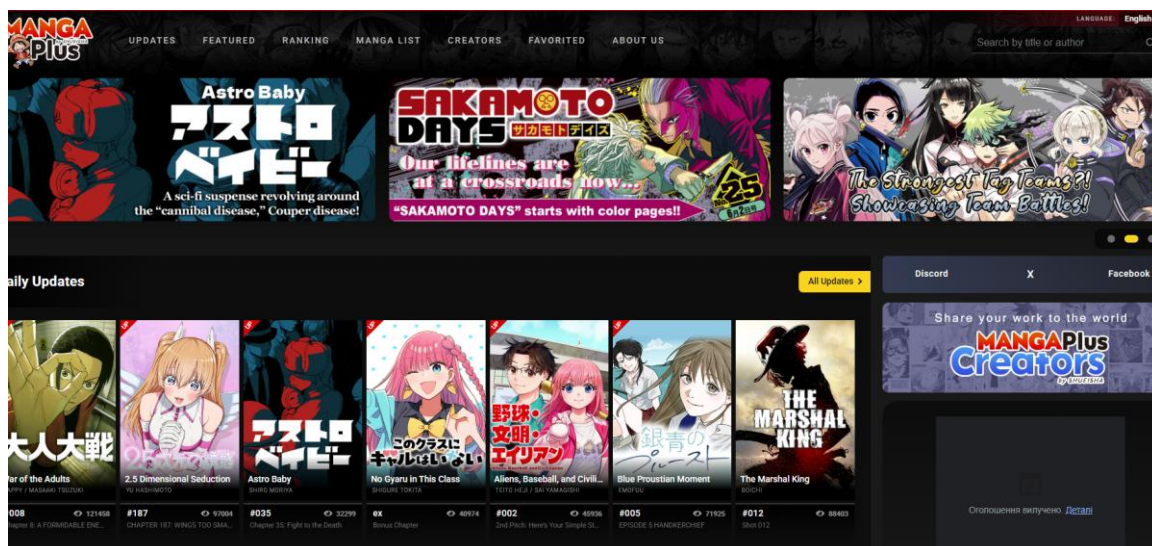


Рис. 1.3. Інтерфейс MangaPlus [3]

4. ComiXology

ComiXology - Одна з найбільших платформ для купівлі та читання коміксів від Marvel, DC, Image та ін.

До переваг даного програмного продукту можна віднести широкий вибір американських коміксів, підтримку смарт-зуму та зручної навігації, а також можливість купувати окремі випуски або оформлювати підписку через сервіс ComiXology Unlimited.

До недоліків даного програмного продукту можна віднести те, що майже весь контент є платним, а сам сервіс не підтримується в усіх країнах.

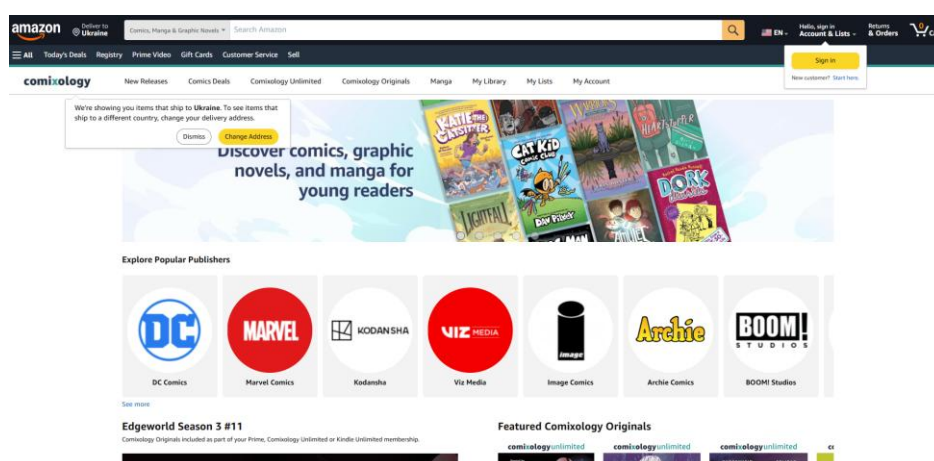


Рис. 1.4. Інтерфейс Comixology [4]

		Шаробоків І. В.			ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

5. WebComics

WebComics – платформа з великим набором кольорових вебкоміксів, орієнтованих на молодь.

До переваг даного програмного продукту можна віднести щоденні оновлення, наявність інтерактивного ком’юніті з можливістю залишати коментарі та ставити рейтинги, а також можливість отримувати безкоштовні монети за активність користувача.

До недоліків даного програмного продукту можна віднести часту агресивну монетизацію та іноді низьку якість перекладу.

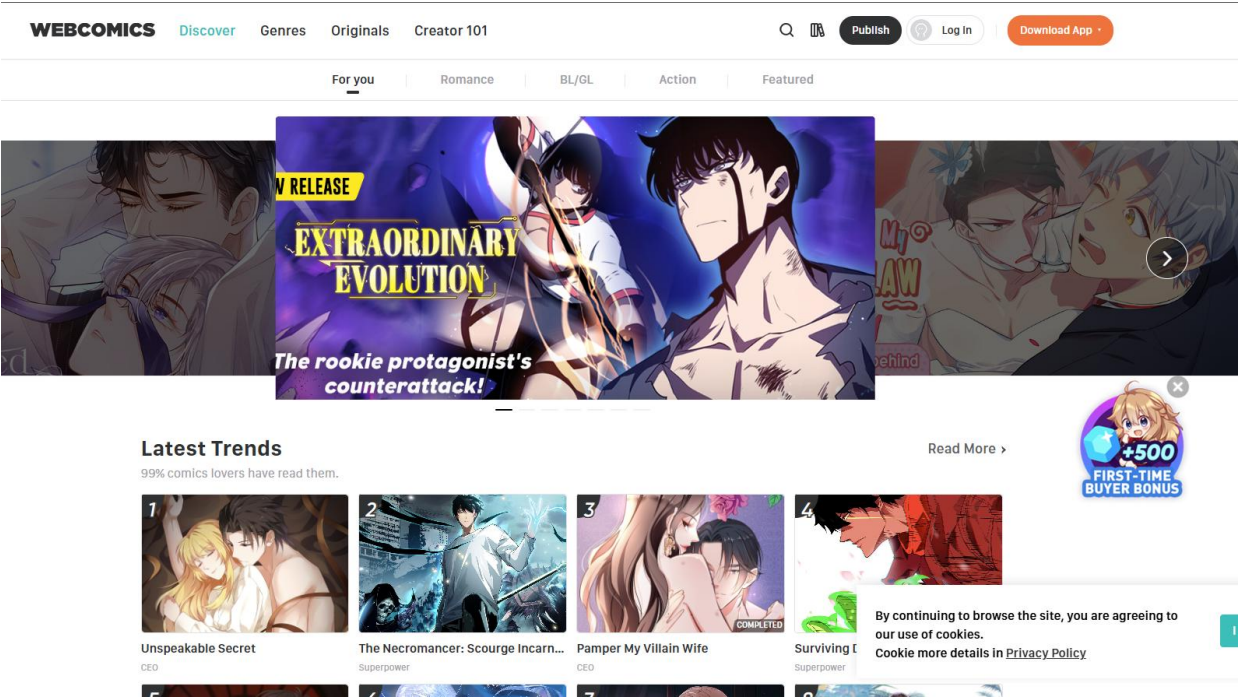


Рис. 1.5. Інтерфейс WebComics [5]

Провівши аналіз існуючих аналогів, можемо скласти їх структуровану та детальну порівняльну характеристику (табл. 1.1.).

Таблиця 1.1

Порівняльна характеристика аналогів

Функція	WebComics	ComiXology	MANGA Plus	Webtoon	Tapas
Ліценція	+ -	+	+	+	+
Коментарі	+	-	-	+	+

Продовження таблиці 1.1

Власна публікація	-	-	-	+	+
Монетизація авторів	+	-	-	+	+
Безкоштовний доступ	+-	+-	+	+	-
Офлайн-доступ	+-	+	-	+	-

Враховуючи інформацію отриману при дослідженні було прийнято рішення створити платформу яка буде відрізнятися серед існуючих платформ для читання коміксів завдяки унікальному поєднанню соціального, мультимедійного та доступного підходу. На відміну від таких популярних сервісів, як Webtoon, Tapas, MANGA Plus, ComiXology чи WebComics, які здебільшого зосереджені на односторонньому читанні та монетизації, платформа буде пропонувати новий формат взаємодії. Вона дозволяє користувачам не лише читати, а й робити це разом у реальному часі, з музичним супроводом і автоматичним перегортанням, що перетворює комікс на емоційний, живий досвід. При цьому збережено ключові цінності – безкоштовний доступ до контенту, підтримка творчих ініціатив та інтеграція з чатами для комунікації. У результаті проєкт поєднує функції читання, соціальної взаємодії та аудіо-візуального занурення, що робить його унікальним на ринку і відкриває новий напрямок у розвитку цифрових коміксів.

1.3. Вибір архітектури веб-орієнтованої платформи для читання коміксів

Система контролю версій

Для контролю та зберігання версій програми, особливо під час командної розробки, бажано використовувати Git – систему контролю версій. Існує кілька

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

популярних сервісів, що забезпечують роботу з Git, серед яких GitHub, GitLab та Bitbucket – три провідні платформи, кожна з яких має свої унікальні переваги. GitHub є найпопулярнішою платформою для хостингу Git-репозиторіїв, яка належить компанії Microsoft. Вона підтримує GitHub Actions – вбудовану CI/CD систему, має велику спільноту користувачів і пропонує зручні інструменти для співпраці, такі як Pull Requests і Issues. Серед інших переваг – підтримка GitHub Copilot, інтеграція з VS Code, інструментами Projects і Codespaces.

GitLab вирізняється відкритим вихідним кодом і можливістю хостингу власного сервера, що робить його популярним у корпоративному середовищі. Він має потужну вбудовану CI/CD систему – GitLab CI, та підтримує DevOps повного циклу: від планування та розробки коду до моніторингу. Система управління доступом дозволяє налаштовувати гнучкі права, створювати групи та сабгрупи, а також реалізовувати автоматичне рев'ю коду.

Bitbucket, який належить Atlassian, тісно інтегрується з іншими інструментами цієї компанії, такими як Jira, Trello та Confluence, що зручно для команд, які вже користуються екосистемою Atlassian. Платформа підтримувала як Git, так і Mercurial (до 2020 року), і має власну CI/CD систему – Bitbucket Pipelines. Також надає можливість створення приватних репозиторіїв з обмеженням на кількість користувачів, що може бути корисним для малих команд.

Frontend бібліотеки

Для побудови frontend-частини веб-додатку можна використовувати сучасні бібліотеки та фреймворки, серед яких найбільш популярними є React, Vue.js, Angular та Svelte. Кожен із них має свої особливості та переваги.

React – це бібліотека, розроблена компанією Meta (Facebook), яка часто використовується як фреймворк. Вона має велику екосистему (Next.js, Redux, React Router), компонентну архітектуру та синтаксис JSX. React ідеально підходить для створення SPA (односторінкових застосунків) та інтерактивних інтерфейсів користувача.

Vue.js – легкий у навчанні фреймворк, побудований на основі HTML та

		Шаробоків І. В.			ІІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

JavaScript. Він підтримує двостороннє зв'язування даних (MVVM) та добре підходить як для невеликих, так і для великих проєктів. Vue є альтернативою React з простішою кривою навчання.

Angular – це повноцінний фреймворк, створений компанією Google. Він має вбудовану підтримку інжекції залежностей (DI), використовує TypeScript і бібліотеку RxJS для реактивного програмування. Angular складніший у вивченні, однак дуже потужний і часто застосовується в enterprise-проєктах.

Svelte – сучасний фреймворк, який компілюється у чистий JavaScript і працює без використання віртуального DOM. Завдяки цьому він забезпечує високу продуктивність і менше навантаження на браузер. Svelte вимагає менше коду і швидко набирає популярності, попри те, що є відносно новим.

Backend фреймворки

Для створення серверної логіки вебзастосунків використовуються сучасні backend-фреймворки. Вони дозволяють реалізувати API, працювати з базами даних, авторизацією, маршрутизацією тощо. Серед найпопулярніших – Express.js, Django, Flask, Laravel, Ruby on Rails та ASP.NET Core. Кожен із них має свої особливості, підходи до архітектури та переваги.

Express.js – це мінімалістичний фреймворк на JavaScript, побудований поверх Node.js. Має велику спільноту та підтримує численні *middleware*, що дозволяють швидко додавати функціональність. Добре інтегрується з frontend-бібліотеками, такими як React, Vue чи Angular, і часто використовується для створення REST API та мікросервісів.

Django – повнофункціональний фреймворк на Python, який пропонує все необхідне «з коробки»: ORM, авторизацію, панель адміністратора, маршрутизацію. Забезпечує високий рівень безпеки та швидку розробку. Ідеально підходить для CRUD-застосунків, MVP та CMS.

Flask – легший мікрофреймворк на Python, який надає більше свободи, але вимагає більше ручної реалізації. Зручний для створення API, мікросервісів або невеликих проєктів. Добрий вибір для швидкого прототипування.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

Laravel – популярний PHP-фреймворк з елегантним синтаксисом і зручною архітектурою. Має вбудовані засоби для роботи з базами даних, поштою, авторизацією. Часто використовується для створення MVP, SaaS-платформ та класичних вебзастосунків.

Ruby on Rails – фреймворк на мові Ruby, який спирається на принцип «конвенція замість конфігурації». Забезпечує швидкий старт, пропонує багато інструментів одразу після встановлення. Особливо популярний серед стартапів і для створення прототипів.

ASP.NET Core – кросплатформений фреймворк на C# від Microsoft. Відзначається високою продуктивністю та гнучкістю. Підходить для корпоративних рішень, особливо в екосистемі Microsoft (Azure, SQL Server) і застосовується для розробки масштабованих і захищених вебдодатків.

Fullstack фреймворки

Для створення повноцінних вебзастосунків, що об'єднують клієнтську та серверну частину, використовуються сучасні **fullstack-фреймворки**. Вони дозволяють працювати з рендерингом сторінок, API, базами даних та інтерфейсами користувача в одному середовищі. Серед найпоширеніших – Next.js, Nuxt.js, Remix, Meteor та Blitz.js.

Next.js – фреймворк на основі React, який підтримує серверний рендеринг (SSR), статичну генерацію (SSG), маршрутизацію API та інші повноцінні можливості fullstack-розробки. Широко використовується для створення SEO-оптимізованих сайтів, прогресивних вебзастосунків (PWA) та складних SPA.

Nuxt.js – аналог Next.js, але побудований на базі Vue. Підтримує SSR, SSG, маршрутизацію на основі структури файлів і має гнучку архітектуру для створення масштабованих вебзастосунків. Добре підходить як для невеликих проєктів, так і для корпоративного рівня.

Remix – сучасний фреймворк, побудований на React, який забезпечує глибоку інтеграцію клієнта і сервера. Особливу увагу приділяє формам, маршрутам і продуктивності. Remix орієнтований на створення швидких, UX-орієнтованих вебзастосунків.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

Meteor – повноцінний стек, що включає як клієнтську, так і серверну частину, інтеграцію з MongoDB та підтримку реального часу через WebSockets. Дозволяє швидко запускати повнофункціональні додатки з мінімальними налаштуваннями.

Blitz.js – фреймворк на основі Next.js, орієнтований на розробку SaaS-рішень та CRUD-застосунків. Має підхід "все з коробки", включаючи генерацію коду, авторизацію, підключення до бази даних та управління станом – що робить його зручним для повноцінної розробки без необхідності збирати окремі компоненти вручну.

Основні архітектури веб-додатків

Для розробки вебзастосунків існують різні **архітектурні підходи**, кожен з яких має свої переваги, недоліки та сфери застосування. Серед найпоширеніших – монолітна, мікросервісна, безсерверна (serverless), Jamstack, SPA та PWA архітектури. Вибір залежить від розміру проєкту, команди, вимог до масштабованості та швидкості розробки.

Монолітна архітектура – найпростіший варіант на старті проєкту. Весь застосунок розгортається як єдина система, що спрощує розробку та деплой на початкових етапах. Проте зі зростанням системи стає важче масштабувати окремі частини, підтримувати код та впроваджувати нові функції. Прикладом можуть бути класичні проєкти на Ruby on Rails або Django.

Мікросервісна архітектура – підхід, при якому вся логіка застосунку розділяється на незалежні сервіси, що взаємодіють між собою. Це дозволяє масштабувати окремі частини системи, використовувати різні мови та технології. Водночас додається складність у налаштуванні DevOps-процесів, моніторингу та взаємодії між сервісами. Цей підхід використовують великі компанії, як-от Netflix, Amazon чи Uber.

Serverless (безсерверна архітектура) – модель, в якій розробник не керує інфраструктурою. Усі функції працюють у хмарі, автоматично масштабуються та активуються подіями (наприклад, запитами до API). Перевагами є спрощене розгортання й масштабування, але існують обмеження: тайм-аути, cold start,

		Шаробоків І. В.			ІІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

складності з налагодженням. Типові приклади – AWS Lambda, Google Cloud Functions.

Jamstack – сучасна архітектура, яка поєднує статичні сторінки, API та контент з headless CMS. Забезпечує високу продуктивність, безпеку та легке масштабування. Чудово підходить для сайтів, які рідко змінюються, або для маркетингових сторінок. Однак для складних інтерактивних додатків потребує додаткових рішень. Поширені інструменти – Next.js у режимі SSG, Gatsby, Hugo.

SPA (Single Page Application) – підхід, коли вся логіка виконується на клієнті. Додаток працює як єдина сторінка, що динамічно змінює контент без перезавантаження. Це забезпечує плавну взаємодію, але створює труднощі з SEO. Часто використовується разом з SSR або SSG для оптимізації пошукової видимості. Приклади: Gmail, Trello, Figma.

PWA (Progressive Web App) – формат вебзастосунку, який поєднує можливості браузера та досвід, близький до нативних додатків. Працює в офлайн-режимі, підтримує push-сповіщення, кешування, можливість встановлення на пристрій. При цьому має обмеження в доступі до деяких нативних функцій. Типові приклади – Twitter Lite, Starbucks PWA.

Найбільш використовувані архітектурні патерни для побудови веб-додатку

У веброзробці використовуються різні **архітектурні патерни**, які визначають структуру застосунку, спосіб організації коду, взаємодію між компонентами та рівнями. Вибір архітектури залежить від масштабу проєкту, вимог до масштабованості, тестування, командної роботи та технологічного стеку. Найпоширенішими є MVC, MVVM, Layered, Hexagonal, Clean Architecture, Event-Driven, CQRS та Micro-frontend.

MVC (Model–View–Controller) – класична архітектура, яка розділяє логіку застосунку, інтерфейс користувача та обробку даних на окремі компоненти. Завдяки цьому покращується підтримка коду та масштабованість. Широко використовується у фреймворках як Django, Ruby on Rails або ASP.NET.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

MVVM (Model–View–ViewModel) – підхід, ідеальний для двостороннього зв'язування даних між моделлю та інтерфейсом. Дає змогу створювати реактивні UI без потреби вручну оновлювати стан. Часто використовується у фронтенд-фреймворках, таких як Angular або Vue.

Layered Architecture (шарова архітектура) – поділяє систему на окремі рівні: UI, сервісну логіку, доступ до даних та базу даних. Ця структура зрозуміла для командної роботи та легко підтримується. Проте для невеликих проєктів може здаватися занадто громіздкою.

Hexagonal Architecture (Ports and Adapters) – ізолює бізнес-логіку від зовнішніх залежностей (інтерфейсів, бази, API) через адаптери. Завдяки цьому стає простішим тестування, заміна технологій і досягнення незалежності від середовища. Добре підходить для створення масштабованих систем із чистою логікою.

Clean Architecture – архітектурний підхід, популяризований Робертом Мартіном (Uncle Bob), що базується на ізоляції бізнес-логіки від інфраструктури, UI і зовнішніх бібліотек. Код організований у вигляді кілець, де найвнутрішніші кола містять доменну логіку. Такий підхід вимагає більше початкового планування, але значно полегшує розвиток системи в майбутньому.

Event-Driven Architecture – система, в якій компоненти взаємодіють між собою через події в асинхронному режимі. Підходить для складних або розподілених систем, зокрема мікросервісів. Використовуються брокери подій, такі як Kafka, RabbitMQ, Webhooks. Водночас ускладнюється дебаг і моніторинг.

CQRS (Command Query Responsibility Segregation) – розділяє логіку застосунку на окремі частини для читання (Query) та запису (Command). Це дає змогу масштабувати їх незалежно та оптимізувати під конкретні задачі. Проте реалізація складніша та вимагає синхронізації між джерелами даних.

Micro-frontend – архітектура, при якій фронтенд поділяється на окремі незалежні модулі або мікрозастосунки. Кожна команда може розробляти, тестувати та деплоїти свою частину окремо. Це забезпечує гнучкість, але вимагає налаштування інструментів інтеграції, оркестрації та єдиної системи дизайну.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

Архітектура додатку побудована за принципами Clean Architecture з поділом на окремі шари:

Підхід Clean Architecture дозволяє розмежувати відповідальність, полегшує тестування, супровід і масштабування проєкту. Основні складові включають фронтенд, базу даних, аутентифікацію та реальний час через WebSocket.

Фронтенд реалізований на основі Next.js та React. Відповідає за рендеринг інтерфейсу, поєднуючи серверний рендеринг (SSR) та клієнтські компоненти. Інтерактивність забезпечується за допомогою React-компонентів, управління станом – через контекст, а валідація форм і даних виконується за допомогою бібліотеки Zod. Така структура дозволяє створювати швидкий, динамічний і зручний інтерфейс користувача.

База даних побудована на PostgreSQL та Redis. PostgreSQL зберігає основну інформацію: дані користувачів, комікси, кімнати, повідомлення в чаті, музичні треки, рейтинги тощо. Таблиці оптимізовані для складних запитів, фільтрації та зв'язків між сутностями. Redis використовується як швидке in-memory сховище, що дає змогу ефективно реалізувати модуль спільного перегляду в кімнатах.

Аутентифікація здійснюється за допомогою NextAuth.js. Система підтримує вхід через email та OAuth-провайдерів, захищає доступ до сторінок і API, працює із сесіями. Для взаємодії з базою даних використовується Prisma-адаптер, що забезпечує зручну інтеграцію з ORM.

WebSocket-функціональність реалізована через Socket.IO. Вона відповідає за синхронізацію спільного перегляду коміксів, музики та чату в режимі реального часу. Комунікація між користувачами в кімнатах відбувається асинхронно, забезпечуючи плавний і інтерактивний досвід.

Архітектура застосунку побудована за принципами Clean Architecture з чітким поділом на шари, що дозволяє досягти високої масштабованості, модульності та розширюваності. Такий підхід особливо ефективний для складних інтерактивних вебзастосунків із мультимедійним функціоналом.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

Presentation Layer відповідає за відображення інтерфейсу користувача. Реалізований на базі Next.js та React, він забезпечує серверний рендеринг, клієнтські компоненти, інтерактивність через React Context і валідацію за допомогою Zod. Цей шар забезпечує зручну та швидку взаємодію з користувачем.

Application Layer реалізує бізнес-процеси та use cases. Саме тут розміщуються сервіси, що координують дії між інтерфейсом, доменною логікою та зовнішніми інструментами. Це центральний шар для прикладної логіки.

Domain Layer містить основні сутності та бізнес-правила. Усі критичні обчислення, перевірки та модельні операції виконуються на цьому рівні, ізольовано від інфраструктурних залежностей. Це ядро системи, незалежне від технологій.

Infrastructure Layer включає конкретні реалізації та залежності: Prisma для роботи з базою даних PostgreSQL, Socket.IO для реального часу, NextAuth для аутентифікації, Redis як in-memory сховище. Цей шар взаємодіє з зовнішніми сервісами, не порушуючи чистоти бізнес-логіки.

Завдяки такому розділенню система залишається стабільною, легко підтримується, тестується та масштабовується. Такий підхід ідеально підходить для складних, мультимедійних застосунків з активною взаємодією між користувачами.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для реалізації веб-проєкту платформи для читання коміксів із підтримкою автоматичного читання, музичного супроводу, спільного перегляду в реальному часі та чатів, використовуватиметься сучасний технологічний стек, який забезпечить масштабованість, продуктивність, зручність розробки та майбутню підтримку. Основними технологіями проєкту є JavaScript, Node.js, React, Next.js, PostgreSQL, Redis, Socket.IO та Prisma. У цій структурі будуть використані перевірені архітектури, проєктні патерни, фреймворки і технології взаємодії фронтенду з бекендом через API та WebSocket-з'єднання.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

Фреймворки та бібліотеки

В веб-додатках на стороні клієнту використовується мова JavaScript для побудови будь-якої інтерактивності на сайті [6]. З боку фронтенду буде використовуватися React – популярна бібліотека для створення компонентного інтерфейсу користувача. React забезпечує швидке оновлення UI, хорошу продуктивність, підтримку великої кількості екосистемних бібліотек. Як основу для фронтенду було обрано Next.js – фреймворк на React, який підтримує серверний рендеринг (SSR), статичну генерацію сторінок (SSG), API-роути та інші повноцінні можливості для побудови full-stack застосунку [15-19].

Для серверної сторони планується використовувати Node.js – середовище виконання JavaScript-коду на сервері [7]. Node.js використовує npm – пакетний менеджер для швидкого та зручного завантаження публічних бібліотек чи фреймворків до проєкту [21]. Одним з найкращих фреймворків на Node.js для побудови веб-сайтів на сьогоднішній день є Next.js, тому оберемо його. Він надає змогу писати високопродуктивний та високофункціональний веб-додаток на JavaScript. Для асинхронної взаємодії у реальному часі (наприклад, у спільному читанні, чатах, синхронізації аудіо та сторінок) буде задіяна бібліотека Socket.IO. Вона забезпечить WebSocket-з'єднання між користувачами в межах однієї групи або кімнати

Бази даних та ORM

Основною базою даних проєкту є PostgreSQL – надійна реляційна СУБД з підтримкою транзакцій, типів JSONB (для зберігання метаданих, наприклад, параметрів читання), а також розширень для повнотекстового пошуку (наприклад, по опису або назвах коміксів) [22].

Для зручної взаємодії з базою використовуватиметься Prisma ORM. Prisma – це сучасна ORM для Node.js, яка дозволяє типізовано та безпечно працювати з PostgreSQL. Вона підтримує генерацію моделей, міграції, референційні зв'язки та оптимізовані запити. Також Prisma чудово інтегрується з TypeScript, що дозволяє мати чітку модельну архітектуру даних.

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		25

Також використовується додаткова БД: Redis – високошвидкісна нереляційна база даних, яка зберігає дані в оперативній пам’яті [23-25]. Redis використовуватиметься для реалізації функціоналу кімнат для спільного перегляду коміксів у реальному часі. Завдяки своїй швидкодії, підтримці структур даних (наприклад, списків, множин, хешів) і можливості публікації/підписки (Pub/Sub), Redis ідеально підходить для зберігання тимчасових станів сеансів читання, синхронізації дій користувачів та управління підключеннями в рамках спільних кімнат.

Авторизація та аутентифікація

Для реалізації авторизації та аутентифікації в проєкті використовується NextAuth.js – сучасне, гнучке рішення для Next.js, яке дозволяє легко інтегрувати систему входу в застосунок. NextAuth підтримує як класичну авторизацію через email/пароль (credentials), так і OAuth-провайдерів (Google, GitHub, Discord тощо), що забезпечує зручний вхід для користувачів.

Завдяки глибокій інтеграції з Next.js, NextAuth дозволяє:

1. зберігати сесії за допомогою JWT або в базі даних (наприклад, через Prisma).
2. захищати API-роути та сторінки за допомогою middleware.
3. підтримувати серверний рендеринг (SSR) із контекстом користувача.
4. легко налаштувати рольову модель доступу, наприклад: адміністратор та користувач.

Також NextAuth добре працює з Prisma як адаптер бази даних, що дозволяє зберігати користувачів, сесії та облікові записи в PostgreSQL без необхідності ручного створення таблиць.

Це рішення гарантує безпечну, масштабовану та гнучку систему аутентифікації, придатну як для невеликих застосунків, так і для великих мультирольових платформ.

WebSocket та мультимедіа

За синхронізацію користувачів у кімнатах відповідатиме Socket.IO, що за

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

безпечить:

1. оновлення сторінок коміксу в реальному часі.
2. синхронне програвання музики.
3. чат з повідомленнями в кімнаті.
4. контроль прав користувачів (ведучий, спостерігачі).

Для музичного плеєра буде реалізовано підтримку HTML5 Audio API, з можливістю інтеграції сторонніх джерел музики, наприклад, через URL-стріми або локальні треки, дозволені для використання.

Висновки до першого розділу

У результаті проведеного аналізу напрямків використання інформаційних технологій для реалізації онлайн-платформи, орієнтованої на читання коміксів з інтерактивним функціоналом, було сформульовано ключові вимоги до майбутньої системи. Зокрема, було визначено потребу в автоматизованому читанні, музичному супроводі, чаті, а також можливості спільного перегляду з синхронізацією сторінок у реальному часі. Під час аналізу аналогічних рішень виявлено, що жодна з існуючих платформ не поєднує усі необхідні функції в єдиному продукті, особливо у контексті мультимедійності та соціальної взаємодії між читачами.

На основі цього обґрунтовано вибір архітектурного підходу – Clean Architecture, що забезпечує модульність, масштабованість і розділення відповідальностей. Визначено інструментальний стек для реалізації функціоналу: Next.js та React – для клієнтської та серверної частин, PostgreSQL – основна база даних, Prisma – ORM для PostgreSQL, Socket.IO – обмін даними в реальному часі, Redis – зберігання інформації в кімнатах спільного перегляду, а також NextAuth – для аутентифікації користувачів.

Таким чином сформовано міцне технологічне підґрунтя для розробки інноваційної онлайн-платформи з сучасними мультимедійними можливостями та соціальними елементами, що є безкоштовною для всіх користувачів.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ЧИТАННЯ КОМІКСІВ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Бізнес вимоги

Проект спрямований на забезпечення швидкого та безкоштовного доступу до читання коміксів, шляхом створення інтуїтивної онлайн-платформи. Основна мета полягає в тому, щоб надати користувачам зручні можливості для перегляду, навігації та пошуку коміксів, а також запропонувати персоналізовані рекомендації відповідно до їхніх уподобань. Реалізація проекту передбачає створення веб-платформи з адаптивним інтерфейсом, який буде зручним для користувачів різного віку та типів пристроїв. Однією з ключових функцій стане інтеграція рекомендаційного модуля, що дозволить автоматично підбирати комікси за інтересами користувача, тим самим підвищуючи їхню залученість і тривалість взаємодії з платформою.

Вимоги користувачів

Платформа передбачає можливість створення та редагування особистого профілю користувача, зокрема налаштування улюблених жанрів і вподобань. На основі історії перегляду та оцінок користувач отримуватиме персоналізовані рекомендації. Важливою складовою стане зручна взаємодія з іншими користувачами, що реалізується через вбудовані коментарі та чати для обговорення коміксів. Також буде доступна функція читання в реальному часі, яка дозволить одночасно переглядати нові розділи разом з іншими читачами у режимі «спільного читання». Система зберігатиме історію прочитаних коміксів, закладки та обрані серії для зручного доступу в майбутньому. Користувачі також отримуватимуть автоматизовані нагадування про вихід нових випусків улюблених серій або про нові комікси від авторів, на яких вони підписані. Крім того, платформа надає можливість авторам додавати власні комікси через спеціальний інтерфейс. Платформа передбачає можливість завантаження, редагування та керування коміксами

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		28

для користувачів. Уся взаємодія з платформою супроводжується дотриманням вимог конфіденційності облікових даних і стандартів захисту персональних даних.

Вимоги персоналу платформи (адміністратори)

Адміністратори платформи мають доступ до розширених інструментів керування контентом і взаємодії користувачів. Зокрема, вони можуть редагувати, видаляти або схвалювати комікси, а також здійснювати повний контроль над кімнатами для спільного перегляду, включаючи створення, закриття та управління учасниками. До обов'язків адміністратора входить модерування коментарів – з можливістю видалення некоректних повідомлень або блокування користувачів за порушення правил. Також адміністраторам надається доступ до аналітики платформи, що дозволяє відстежувати активність, поведінку користувачів і якість контенту. Окрема увага приділяється забезпеченню дотримання авторських прав, а також контролю якості матеріалів, що публікуються, через спеціалізовані внутрішні інструменти.

Характеристика об'єкта комп'ютеризації

Користувач платформи матиме змогу зручно переглядати та читати комікси завдяки простому, інтуїтивно зрозумілому інтерфейсу та системі персоналізованих рекомендацій. Платформа забезпечить швидкий доступ до контенту, впорядкованого за жанрами, авторами та популярністю, що сприятиме комфортному користувацькому досвіду.

Функціональні вимоги

Платформа повинна забезпечувати авторизацію користувачів, надаючи можливість реєстрації та входу в систему. Важливим елементом є персоналізовані рекомендації – система аналізуватиме вподобання користувачів і пропонуватиме комікси на основі їхньої історії переглядів та оцінок. Користувачі матимуть змогу залишати коментарі до коміксів або окремих розділів, а також автоматично зберігати прогрес читання для подальшого зручного продовження. Автори через особистий кабінет зможуть завантажувати, редагувати та видаляти власні комікси. Для адміністраторів передбачені спеціальні інструменти, що дозволяють

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		29

модерувати контент, керувати користувачами та переглядати загальну статистику платформи.

Нефункціональні вимоги

Система повинна забезпечувати високу продуктивність, зокрема швидке завантаження сторінок і коміксів, при цьому час відгуку не має перевищувати 2 секунди за стандартного навантаження. Платформа повинна бути доступною цілодобово, 7 днів на тиждень, із рівнем доступності не менше 99,5%, гарантуючи безперервний доступ до контенту для користувачів. Одним із ключових аспектів є масштабованість – система має витримувати навантаження щонайменше у 1000 активних користувачів одночасно без зниження якості обслуговування. Особливу увагу необхідно приділити безпеці, забезпечивши захист персональних даних відповідно до вимог GDPR, включаючи шифрування інформації та захист від несанкціонованого доступу. Інтерфейс платформи повинен бути кросплатформним і стабільно функціонувати на різних пристроях та браузерах, таких як комп'ютери, планшети й мобільні телефони. Крім того, важливою є зручність використання – інтерфейс має бути інтуїтивно зрозумілим та адаптивним, що дозволить працювати з платформою користувачам із різним рівнем технічної підготовки.

Варіанти використання

Аналіз вимог користувачів дав змогу визначити варіанти використання веб-платформи для читання коміксів. На рисунку 1.4 представлено діаграму використання програмної системи, яка відображає основні функціональні процеси та взаємодію користувачів із системою.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

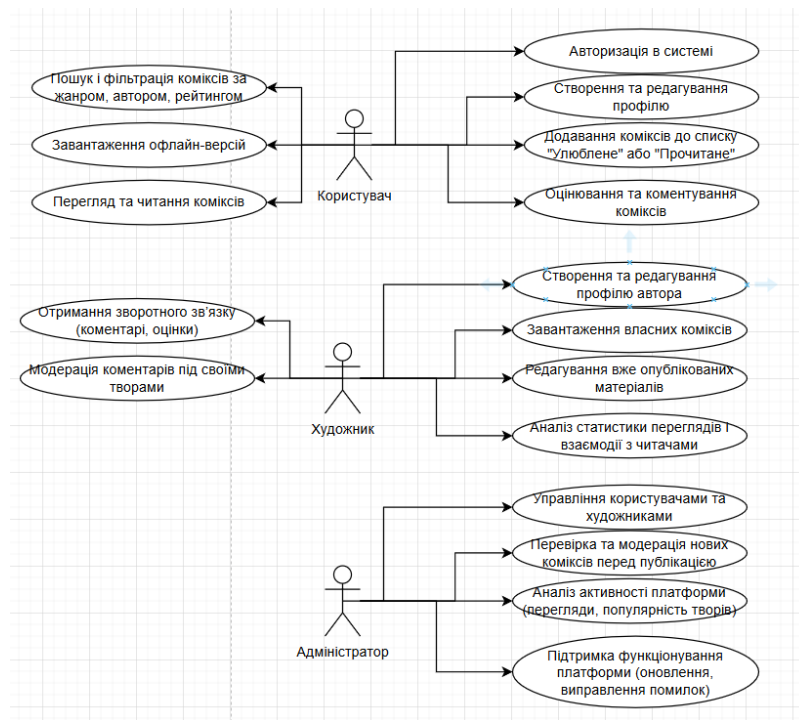


Рис. 2.1. Варіанти використання платформи для читання коміксів

Вимоги до часових характеристик

У контексті проєктування онлайн-платформи для читання коміксів з підтримкою автоматичного прокручування сторінок, музичного супроводу та спільного перегляду, особливу увагу необхідно приділити часовим характеристикам реакції системи на дії користувача. Висока швидкодія та мінімальні затримки критично важливі для забезпечення комфортного користувацького досвіду, особливо в режимах реального часу.

Основні вимоги до часових характеристик платформи

Завантаження сторінки коміксу повинно відбуватись не довше ніж за 300–500 мс, навіть при переході до нової сторінки у режимі спільного перегляду, а при попередньому кешуванні – майже миттєво, тобто менше ніж за 100 мс. Синхронізація дій між користувачами в кімнаті, таких як гортання сторінок, відтворення музики, пауза або спілкування в чаті, має відбуватись із мережею затримкою не більше 100–150 мс, щоб забезпечити ефект спільної присутності без помітного лагу. Відгук інтерфейсу на дії користувача, включаючи натискання кнопок, навігацію та запуск треків, має становити менше 100 мс відповідно до сучасних рекомендацій UX/UI. Процес авторизації користувача через NextAuth має

тривати менше ніж 1 секунду для OAuth-провайдерів та не більше 2 секунд у разі використання облікових даних з валідацією. Чат у кімнатах перегляду повинен забезпечувати миттєве відображення повідомлень із загальною мережею затримкою не більше 100 мс, щоб уникнути розривів у комунікації. Крім того, інтерактивні анімації та переходи між розділами платформи мають бути плавними й завершуватись у межах 200–300 мс, щоб не створювати відчуття затримки.

Ці вимоги забезпечують не лише зручність, а й технічну придатність платформи до спільного перегляду коміксів в реальному часі. Для досягнення таких характеристик використовуються WebSocket-з'єднання (через Socket.IO), in-memory БД Redis, кешування даних, оптимізовані API-запити, клієнтська передзагрузка та розподілена система доставки контенту Cloudflare.

Основні сутності системи

У системі онлайн-платформи для читання коміксів з мультимедійними функціями передбачено набір ключових сутностей, кожна з яких виконує конкретну роль у забезпеченні функціональності, зручності використання та розширюваності продукту. Ці сутності мають власну структуру (поля) та набір методів, що описують поведінку елементів системи.

Першою основною сутністю є користувач (User) – представлення особи, яка взаємодіє з платформою. Користувач має такі основні поля, як ідентифікатор, ім'я, email, роль (читач, автор або адміністратор), аватар, біографію та дату реєстрації. Його методи включають login() для авторизації, logout() для завершення сесії, updateProfile() – оновлення інформації про себе, subscribeToComic() – оформлення підписки на комікс, та donateToAuthor(amount) – надсилення донатів авторам, що дозволяє підтримати їхню творчість.

Сутність Comic (Комікс) представляє головний тип контенту платформи. Вона містить такі поля, як назва, опис, жанр, автор, список сторінок (зображень), теги та пов'язані аудіотреки. Комікс має методи addPage(image) для додавання сторінок, attachMusic(trackId) для зв'язування з музичним супроводом, getPage(pageNumber) для отримання конкретної сторінки, а також rate(score) – для оцінювання користувачами.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		32

Для реалізації мультимедійної складової передбачена сутність Track (Музичний трек). До її структури входять ідентифікатор, назва, URL для відтворення, тривалість, автор треку та комікс, з яким трек пов'язаний. Методи play() та pause() відповідають за управління відтворенням, attachToPage(pageNumber) дозволяє прив'язати трек до конкретної сторінки коміксу, а delete() – видалити його з системи.

Функціональність одночасного перегляду коміксів кількома користувачами реалізується через сутність Room (Кімната спільного перегляду). Кімната має ID, посилання на комікс, ID ведучого, список учасників, поточну сторінку перегляду та активний трек. Методи createRoom() та joinRoom(userId) дозволяють створити або приєднатися до кімнати, syncPage(pageNumber) синхронізує поточну сторінку для всіх учасників, а closeRoom() – завершує сесію.

Обмін повідомленнями в межах кімнати реалізується через сутність Chat / Message (Повідомлення). Повідомлення містять ідентифікатор, ID відправника, ID кімнати, мітку часу та текст. Основні методи: sendMessage() – надсилання повідомлення, getMessages(roomId) – отримання історії чату в конкретній кімнаті.

Користувачі мають змогу залишати оцінки до коміксів. Сутність Rating містить ID користувача, коміксу, значення оцінки, а також дату. Серед методів – addRating(score) для виставлення оцінки.

Сутність ComicList описує взаємодію користувача з коміксом: улюблені, плановані до читання, прочитані, покинуті. Містить тип списку, користувача та комікс. Методи: addComic(comicId, type) – додавання коміксу до списку, removeComic(comicId) – видалення, getList(type) – отримання всіх коміксів певного списку.

Сутність Chapter є складовою частиною коміксу. Вона містить назву розділу, номер тому, номер розділу, пов'язаний комікс, список сторінок, коментарі та вподобання. Розділ можна однозначно ідентифікувати за комбінацією номеру тому, номеру розділу та ID коміксу. Методи включають addPage(image) – додавання сторінки, getPages() – отримання списку всіх сторінок розділу, addComment(text) – створення коментаря, likeChapter() – вподобання розділу.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		33

Сутність Page представляє окрему сторінку розділу коміксу. Містить номер сторінки, зображення та посилання на розділ. Основні методи: getImage() – отримання зображення сторінки, delete() – видалення сторінки адміністратором або автором.

Сутність Review зберігає текстовий огляд коміксу від користувача. Містить текст, дату публікації, автора рецензії та пов’язаний комікс. Основні методи: addReview(text) – додавання рецензії, editReview() – редагування, deleteReview() – видалення рецензії.

Сутність ChapterComment – це текстовий коментар до конкретного розділу. Містить текст, автора, дату та прив’язку до розділу. Методи: addComment(text), editComment(), deleteComment().

Сутність ChapterLike фіксує взаємодію «вподобання» між користувачем і розділом. Вказує користувача та розділ. Методи: likeChapter(), unlikeChapter().

Сутність LastChapter зберігає останній прочитаний розділ певного коміксу конкретним користувачем. Це дає змогу швидко продовжити читання з місця зупинки. Методи: saveProgress(chapterId), getLastChapter(comicId).

Ця структура забезпечує гнучкість, масштабованість і зручність у реалізації основної функціональності платформи, дозволяючи легко розширювати систему, підтримувати інтеграцію з мультимедіа та взаємодію між користувачами.

Модель сутностей дозволяє формувати розширювану, підтримувану і масштабовану платформу, у якій чітко розділені відповідальності між частинами системи.

2.2. Розробка бази даних системи

У процесі розробки сучасної онлайн-платформи для читання коміксів із мультимедійними можливостями, важливою складовою є проектування фізичної структури бази даних, яка забезпечить ефективне зберігання, доступ та взаємодію між усіма об’єктами системи. Така платформа повинна підтримувати функції обліку користувачів, каталогізації коміксів, перегляду розділів, збереження історії читання, а також додавання відгуків.

		Шаробоків І. В.			ІІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

Серед основних функцій системи також передбачено можливість мультимедійного супроводу під час перегляду коміксів. Зокрема, для кожного розділу може бути додано фонову музику або звукові ефекти, які покликані створити глибший емоційний зв'язок із читачем та занурення в сюжет. З технічної точки зору, це реалізовано через прив'язку аудіотреків до окремих розділів у базі даних, з можливістю синхронізованого відтворення під час навігації.

Ключовою інновацією платформи є підтримка режиму спільного перегляду, що реалізується через так звані кімнати (rooms). Це окремі віртуальні сесії, в яких декілька користувачів можуть одночасно читати один і той самий комікс, синхронізуючи положення перегляду (розділ та скрол) і обмінюючись повідомленнями в реальному часі через вбудований чат. Такий функціонал орієнтований на створення соціального досвіду спільного читання, що особливо актуально для фанатських спільнот, стрімерів або друзів, які хочуть обговорити події сюжету в процесі.

Для реалізації цих вимог була використана ORM-бібліотека Prisma у зв'язці з PostgreSQL як основною системою керування базами даних. Схема визначає структуру таблиць, поля, типи даних, унікальні ключі, зовнішні зв'язки та обмеження, що забезпечують цілісність і узгодженість даних. Кожна сутність у моделі відповідає окремій таблиці, яка містить відповідні атрибути та пов'язана з іншими сутностями за допомогою реляційних зв'язків.

У результаті розробки фізичної схеми було сформовано базову модель даних, що включає сутності User, Comic, Chapter, Page, Track, Review, ComicList, Rating, ChapterComment, ChapterLike та LastChapter. Усі ці об'єкти утворюють узгоджену систему, що дозволяє ефективно реалізувати необхідну функціональність – від базової авторизації до складного багаторівневого перегляду коміксів із мультимедійним супроводом (див. додаток Е).

З метою забезпечення високої швидкодії та масштабованості функціоналу спільного перегляду, було прийнято рішення використовувати Redis – високопродуктивне in-memory сховище даних, яке ідеально підходить для обробки

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

тимчасових, часто змінюваних даних із низькою затримкою. У Redis зберігається актуальна інформація про всі активні кімнати, включаючи:

- roomId – унікальний ідентифікатор кімнати,
- host – ідентифікатор ведучого користувача (організатора кімнати),
- chapterNumber – номер розділу, який переглядається,
- comicId – ідентифікатор вибраного коміксу,
- scroll – координати або значення позиції прокрутки сторінки.

Окрім цього, у Redis зберігаються повідомлення з чату, які передаються у форматі pub/sub або за допомогою списків, що дозволяє в режимі реального часу здійснювати обмін повідомленнями між учасниками кімнати. Такий підхід забезпечує мінімальну затримку та дозволяє зберігати історію повідомлень на час сесії. При потребі чат може бути розширено функціоналом реакцій, емодзі, цитування або приватних повідомлень.

Таким чином, завдяки грамотному поєднанню PostgreSQL для основних структурованих даних та Redis для тимчасових сесійних і потокових даних, вдалося побудувати гнучку, продуктивну та масштабовану архітектуру. Це дозволяє платформі не лише обслуговувати індивідуальних користувачів, але й підтримувати динамічні спільні сесії із синхронізацією перегляду, звуковим супроводом та живим спілкуванням.

У перспективі можливе розширення системи за рахунок інтеграції відео- або голосового зв'язку, статистики активності в кімнатах, а також застосування машинного навчання для рекомендацій коміксів, базуючись на інтересах учасників.

2.3. Проектування та реалізація алгоритмів роботи системи

Для забезпечення цілісної та злагодженої роботи програмного комплексу онлайн-платформи для читання коміксів з мультимедійною підтримкою та інтерактивними функціями, необхідно визначити узагальнені алгоритми функціонування системи та описати основні сценарії взаємодії її компонентів.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

Цей розділ присвячений аналізу логіки роботи ключових функцій додатку – від завантаження контенту на стороні сервера до реакції на дії користувача в інтерфейсі. Розглядаються механізми авторизації, завантаження та перегляду коміксів, синхронізації читання в режимі реального часу, програвання фонові музики, залишення відгуків і збереження історії прочитаного.

Опис алгоритмів включає як послідовність дій на рівні клієнта і сервера, так і логіку взаємодії між окремими модулями – базою даних, API, frontend-інтерфейсом, а також сервісами, що відповідають за обмін повідомленнями та спільний перегляд. Такий підхід дозволяє забезпечити узгоджену поведінку системи в режимі багатокористувацького середовища та гарантувати стабільність роботи основних сервісів навіть при великому навантаженні.

Процес реєстрації та авторизації

Процес взаємодії користувача із системою починається з реєстрації або авторизації (зображено на рисунку 2.2). Якщо користувач новий, він вводить особисті дані, після чого система перевіряє їх коректність і створює обліковий запис. Для зареєстрованих користувачів передбачена авторизація шляхом введення логіна та пароля, які система перевіряє на валідність. У разі успішної авторизації користувач отримує доступ до функціоналу платформи.

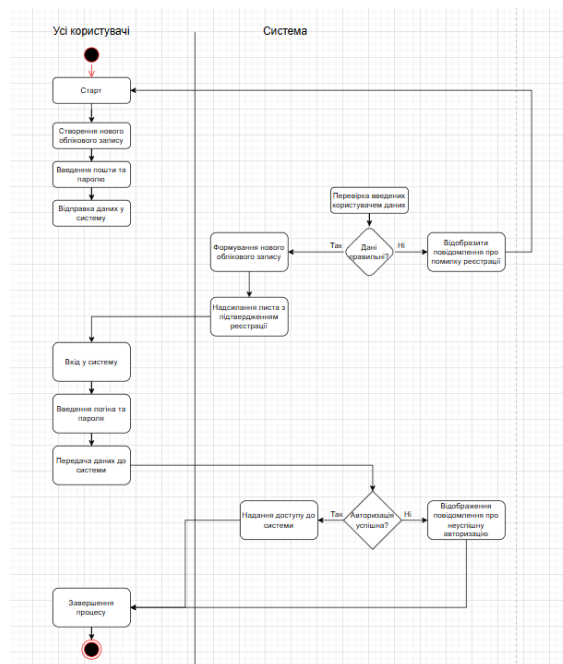


Рис. 2.2. Діаграма активності реєстрації та авторизації

Процес перегляду бібліотеки коміксів

Процес взаємодії користувача із системою зображено на рисунку 2.3. Після запуску платформи користувач одразу отримує доступ до бібліотеки коміксів без необхідності проходження авторизації. Користувач може переглядати загальний перелік доступних коміксів, а також скористатися функціями фільтрації або пошуку для зручнішого вибору. Після вибору конкретного коміксу система завантажує відповідний вміст і відображає його користувачеві. У разі недоступності коміксу система повідомляє про помилку. Після завершення перегляду користувач може повернутися до бібліотеки або завершити роботу з платформою.

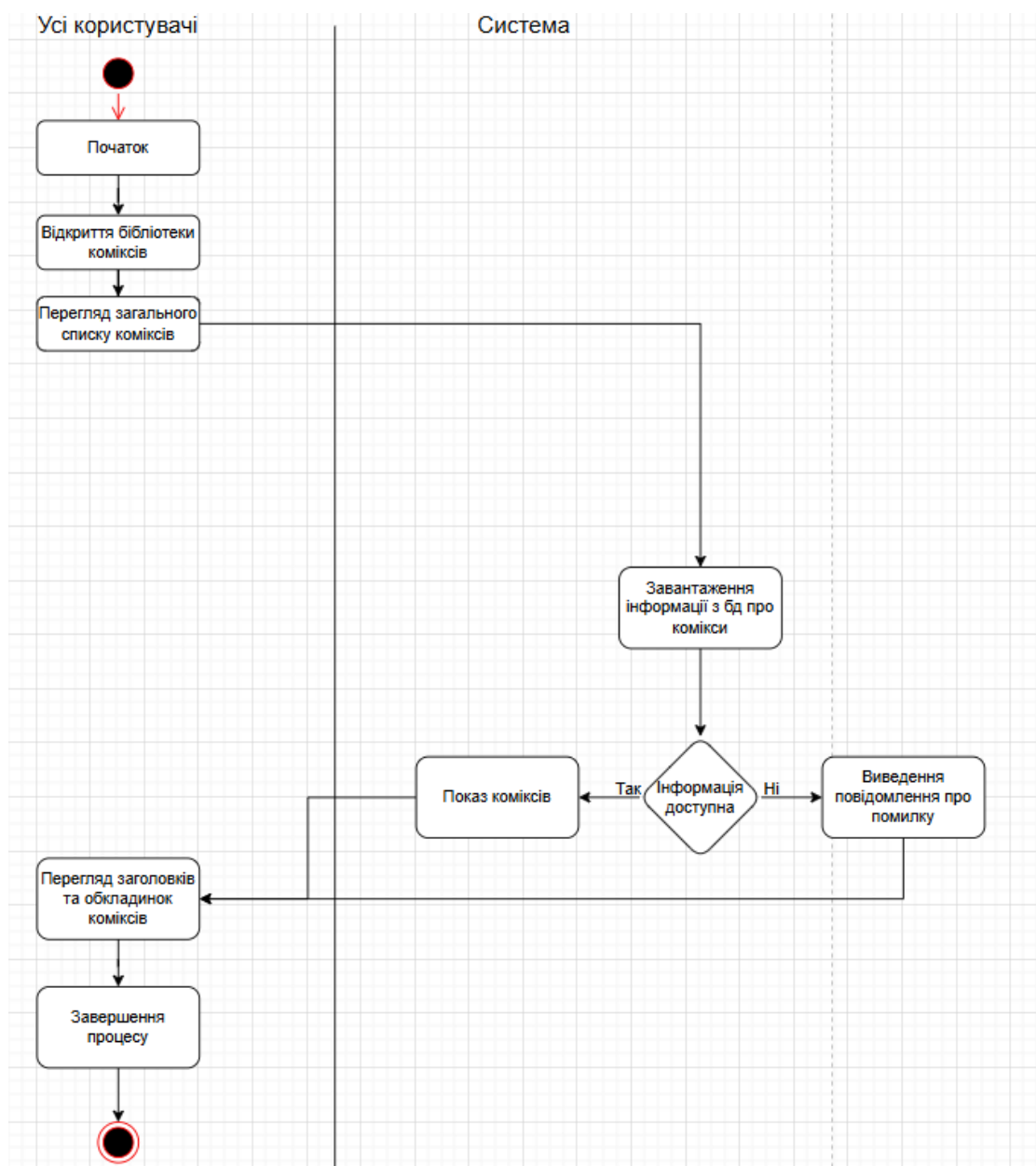


Рис. 2.3. Діаграма активності перегляду бібліотеки коміксів

Процес вибору та читання коміксу

Процес вибору та читання коміксу зображено на рисунку 2.4. Після завантаження бібліотеки користувач має змогу переглянути список доступних коміксів. Для зручного пошуку реалізовано можливість фільтрації за жанром, автором або назвою. Після вибору бажаного коміксу система завантажує відповідний вміст. Користувач може перегортати сторінки, читати комікс у зручному форматі та, за потреби, повернутись до бібліотеки для вибору іншого коміксу або завершити перегляд.

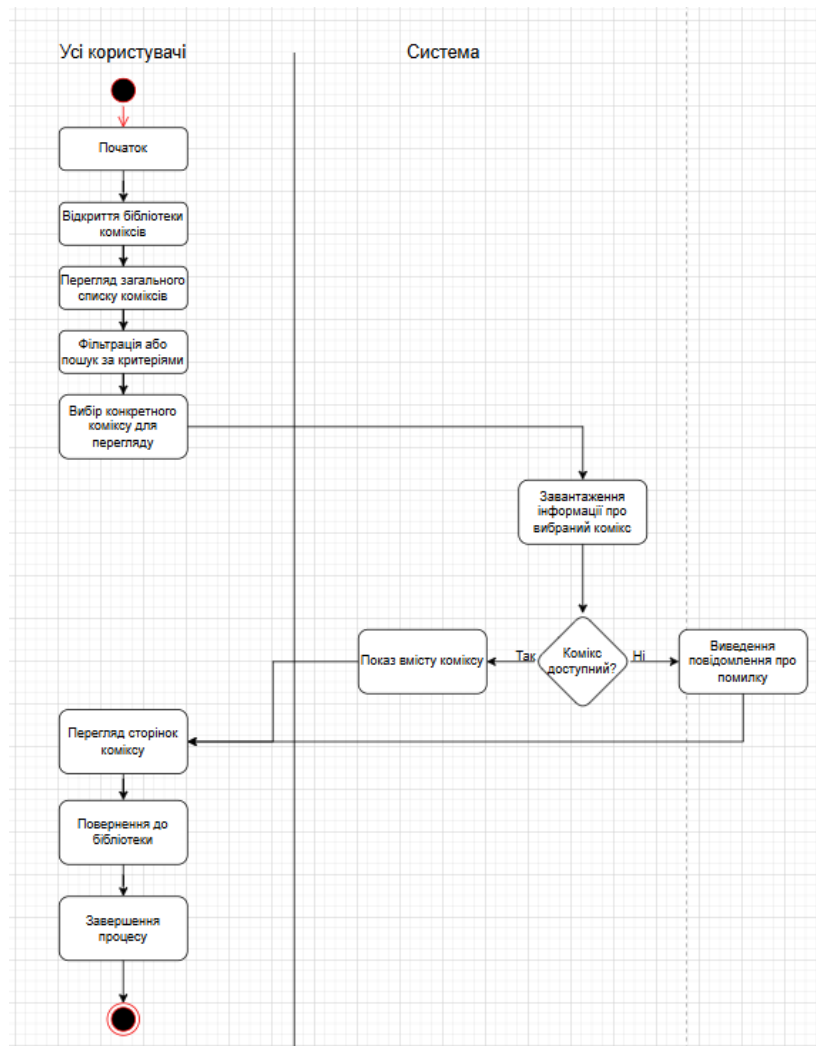


Рис. 2.4. Діаграма активності вибору і читання коміксу

Процес збереження прогресу прочитаного коміксу

Процес збереження прогресу прочитаного коміксу зображено на рисунку 2.5. Коли користувач завершує читання сторінки коміксу, система автоматично фіксує поточну сторінку як останню прочитану. Цей прогрес зберігається у

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

профілі користувача, щоб під час наступного входу відкрити комікс із місця, де він зупинився.

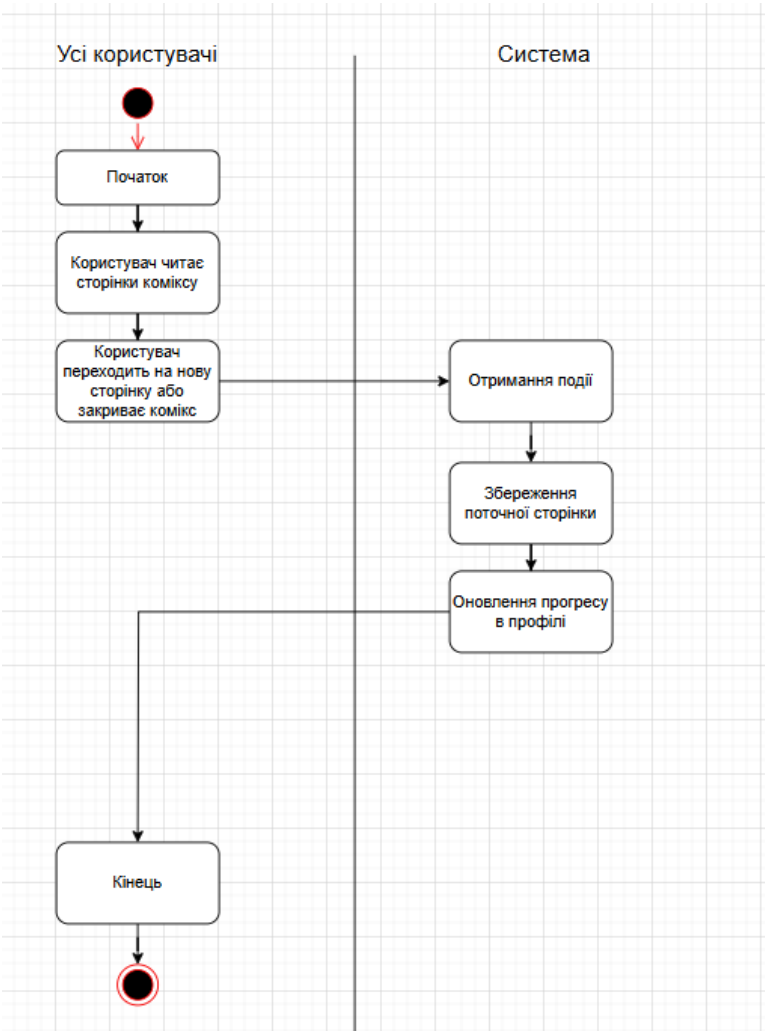


Рис. 2.5. Діаграма активності збереження прогресу прочитаного коміксу

Об’єктно-орієнтована модель системи

Діаграма класів представляє платформу для читання коміксів з музичним супровідом та кімнатами для спільного перегляду. Класи та їх зв’язки відображають основні функціональні можливості сервісу, взаємодію користувачів із контентом, авторів із власними творами, а також інтеграцію мультимедійних елементів, зокрема звукового супроводу, що синхронізується з перебігом читання. Система забезпечує персоналізований досвід користувача, підтримує керування бібліотекою коміксів, додавання та редагування музичних треків, а також адаптацію інтерфейсу під різні пристрої та формати перегляду.

На рисунку 2.6 зображено діаграму класів платформи для читання коміксів. Вона демонструє основні класи, їхні атрибути, методи та зв'язки між ними. Кожен клас відповідає за окремий аспект функціональності платформи: управління користувачами, читання та створення коміксів, інтеграцію музичного супроводу, комунікацію в чатах, а також організацію спільного перегляду. Зв'язки між класами забезпечують ефективну взаємодію користувачів та адміністратора, а також автоматизують ключові процеси системи з урахуванням мультимедійних можливостей.

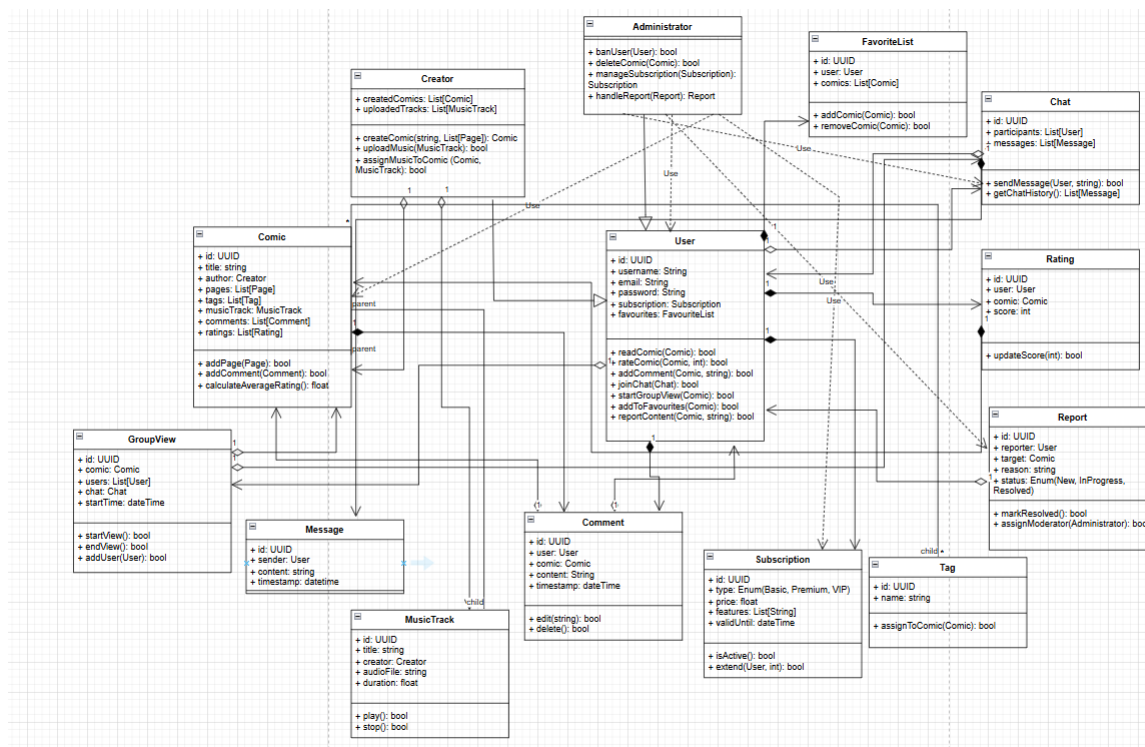


Рис. 2.6 – Діаграма класів платформи для читання коміксів.

Архітектурні рішення та дизайн-патерни

Для розробки проекту «онлайн-платформа для читання коміксів» можна використати різні дизайн-патерни, щоб забезпечити зрозумілу архітектуру, спростити підтримку та розширення системи. Ось основні патерни, які підходять для цього проекту:

Factory Method

Factory Method є одним із найпоширеніших патернів проєктування для створення об'єктів, коли конкретний тип об'єкта визначається під час виконання

програми. Для нашої системи цей патерн підходить для створення різних типів повідомлень, таких як:

1. Текстове повідомлення – текстове повідомлення в чаті.
2. Повідомлення з зображенням – відправка зображення в чат.
3. Аудіо-повідомлення – відправка аудіо-повідомлення в чат.

Фабричний метод дозволяє абстрагувати процес створення повідомлень, що значно полегшує розширення функціоналу. Якщо в майбутньому виникне необхідність додати нові типи повідомлень, це можна зробити без зміни існуючого коду, завдяки додаванню нового підкласу, який реалізує потрібний тип. Діаграму класів дизайн-патерну «Фабричний метод» зображено на рисунку 2.7.

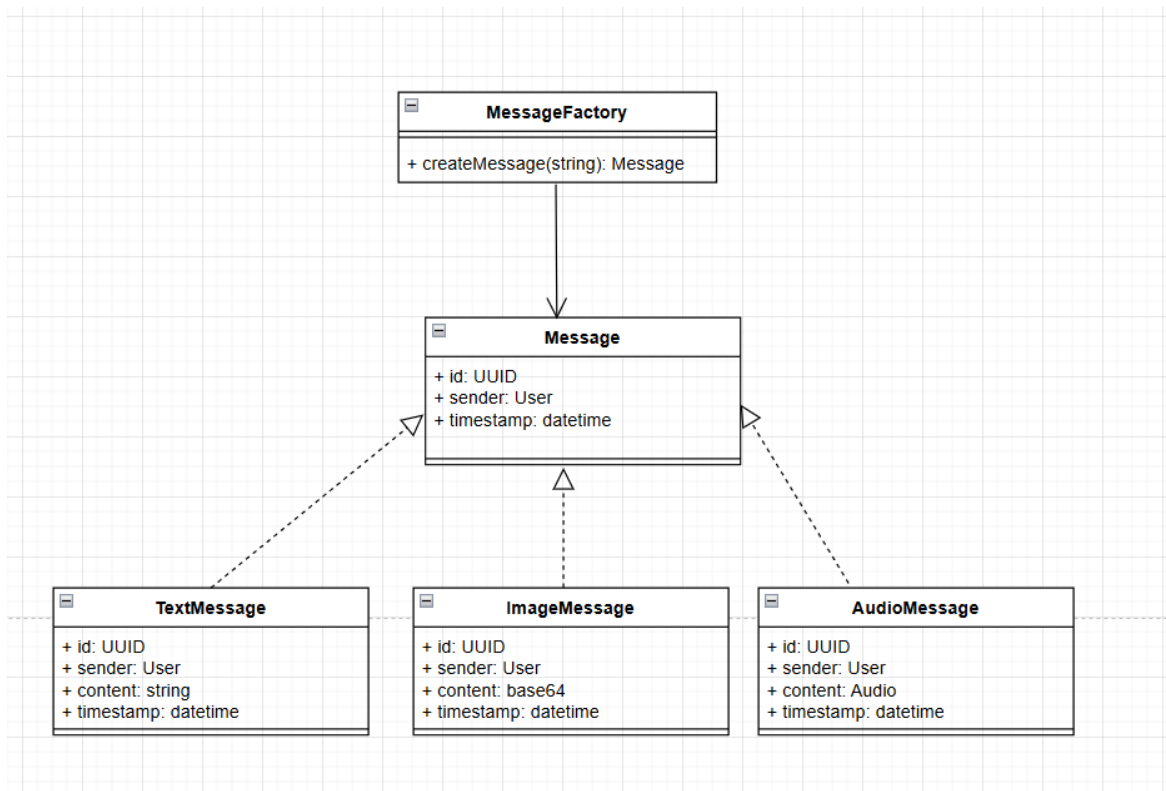


Рис. 2.7. – Діаграма класів дизайн-патерну «Factory Method» для створення різних типів повідомлень

Strategy

Патерн проєктування Strategy є ефективним інструментом для реалізації динамічної зміни поведінки об'єкта без необхідності змінювати його внутрішній код. У контексті нашої платформи для читання коміксів, цей патерн ідеально

підходить для реалізації різних типів підписок, дозволяючи гнучко керувати функціональністю залежно від обраного тарифного плану.

Основні переваги використання Strategy для підписок:

- 1. Гнучка структура – легко додавати нові типи підписок (наприклад, стандартна, преміум) без внесення змін до вже існуючих класів.
- 2. Можливість розширення – додавання нових можливостей до певного тарифу відбувається через створення нової стратегії, без ризику порушити логіку інших типів підписок.
- 3. Інкапсульована логіка – кожна стратегія реалізує власну версію функціоналу, що робить код чистішим і легшим у підтримці та тестуванні.
- 4. Поліморфізм – всі типи підписок реалізують спільний інтерфейс, тому платформа працює з ними універсально, не залежно від конкретної реалізації.

Таким чином, застосування патерну Strategy забезпечує масштабовану, підтримувану та модульну архітектуру для обробки різних типів підписок на платформі. Діаграму класів дизайн-патерну «Strategy» зображено на рисунку 2.8.

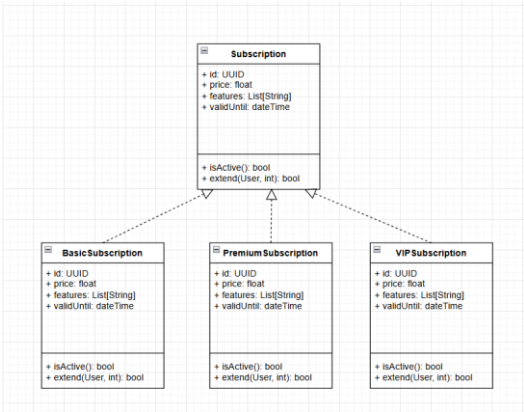


Рис. 2.8 – Діаграма класів дизайн-патерну «Strategy» для створення різних типів підписок

Observer

Observer це шаблон проєктування, який дозволяє об’єктам (спостерігачам) автоматично отримувати сповіщення про зміни стану іншого об’єкта (суб’єкта). Він встановлює залежність «один до багатьох»: коли суб’єкт змінюється – всі підписані на нього спостерігачі повідомляються про це.

Основні переваги:

1. Автоматичне оновлення – спостерігачі не потребують ручної перевірки змін, вони отримують сповіщення автоматично.
2. Слабкий зв'язок – суб'єкт не знає деталей реалізації спостерігачів, що підвищує гнучкість системи.
3. Добра масштабованість – легко додавати нових спостерігачів без зміни існуючого коду.
4. Зручна реалізація подій – ідеально підходить для систем, що працюють на основі подій (наприклад, чат-сповіщення, оновлення нових коміксів, статус друзів онлайн).

Можемо використати цей патерн для реалізування функції сповіщень користувачам при виході нового розділу коміксу.

Діаграму класів дизайн-патерну «Strategy» зображено на рисунку 2.9.

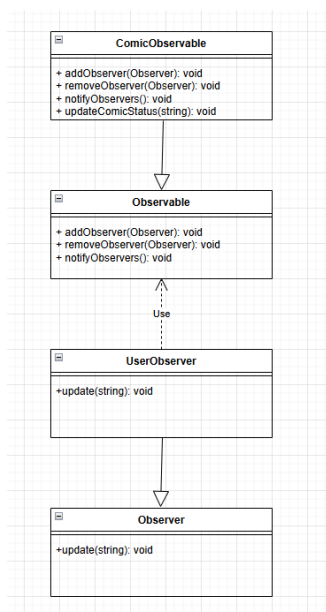


Рис. 2.9 – Діаграма класів дизайн-патерну «Observer» для сповіщення користувачів про вихід нового розділу

Комунікації і послідовність взаємодії об'єктів системи

Цей розділ висвітлює процеси взаємодії між компонентами системи під час користування платфоною для читання коміксів. Представлено три діаграми послідовностей, які ілюструють основні сценарії роботи системи: спільний перегляд

коміксу, відтворення коміксу з музичним супроводом та обмін повідомлення в чаті під час читання.

Діаграма послідовності “Спільний перегляд коміксу”

Діаграма послідовності "Спільний перегляд коміксу" на рисунку 2.10 ілюструє взаємодію між двома користувачами та кількома компонентами системи під час ініціації та проведення сесії спільного перегляду з музичним супроводом. Усе починається з того, що Користувач 1 обирає комікс у системі коміксів. Система, у свою чергу, надсилає запит до сервера синхронізації, який отримує відповідні дані про комікс, доставляє їх назад до системи та повертає відповідь Користувачу 1.

Після цього Користувач 1 ініціює створення сесії перегляду. Система надсилає запрошення Користувачу 2, який підтверджує участь. Після підтвердження система розпочинає завантаження матеріалів: комікс надходить із музичного сервісу та бази даних коміксів, а потім завантажується на сервер синхронізації.

Коли обидва користувачі беруть участь, вони отримують дані коміксу. Користувач 1 обирає музичний плейліст, після чого система отримує дані плейліста й активує музичний плеєр. Плеєр повертає поточний статус, і перегляд може розпочатися з одночасним відтворенням музики.

Якщо Користувач 2 перезавантажує сторінку, він надсилає запит на відновлення стану. Система коміксів через сервер синхронізації отримує статус поточної сторінки та передає його Користувачу 2. Це дозволяє зберігати синхронізацію навіть при втраті з'єднання або оновленні сторінки.

На завершення Користувач 2 ініціює завершення перегляду. Система отримує запит, завершує сесію та через сервер синхронізації підтверджує завершення для обох користувачів.

Загалом, ця послідовність демонструє, як за допомогою інтеграції фронтенду, системи коміксів, сервера синхронізації, музичного сервісу, бази даних і плеєра забезпечується повноцінна спільна взаємодія користувачів у реальному часі з мультимедійною підтримкою.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		45

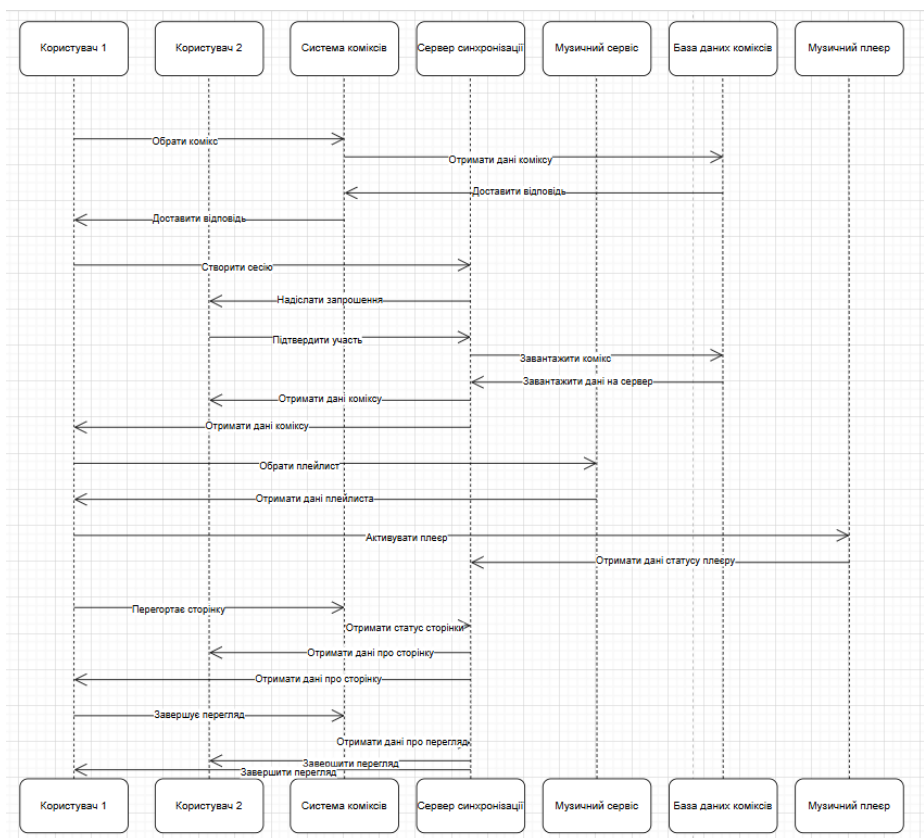


Рис. 2.10 – Діаграма послідовності “Спільний перегляд коміксу”

Діаграма послідовності “Відтворення коміксу з музичним супроводом”

Діаграма послідовності “Відтворення коміксу з музичним супроводом” на рисунку 2.11 описує процес, у якому користувач взаємодіє з системою для перегляду коміксу з паралельним музичним супроводом. Уся взаємодія відбувається між кількома компонентами: користувачем, системою коміксів, базою даних коміксів, музичним сервісом та музичним плеєром.

Усе починається з того, що користувач обирає комікс. Система коміксів надсилає запит до бази даних коміксів для отримання відповідних даних, після чого повертає отриману інформацію назад користувачу.

Далі користувач обирає плейліст. Система коміксів надсилає запит на отримання плейліста до музичного сервісу. Отримані музичні дані повертаються до системи, яка після цього надсилає запит до музичного плеєра для початку відтворення. Плеєр активується, і відтворення музики разом із переглядом коміксу розпочинається.

Цей сценарій ілюструє взаємодію фронтенду з базою даних і зовнішніми мультимедійними сервісами, забезпечуючи інтегрований мультимедійний досвід. Він демонструє, як система координує завантаження та запуск візуального й аудіоконтенту, щоб забезпечити синхронізоване сприйняття для користувача.

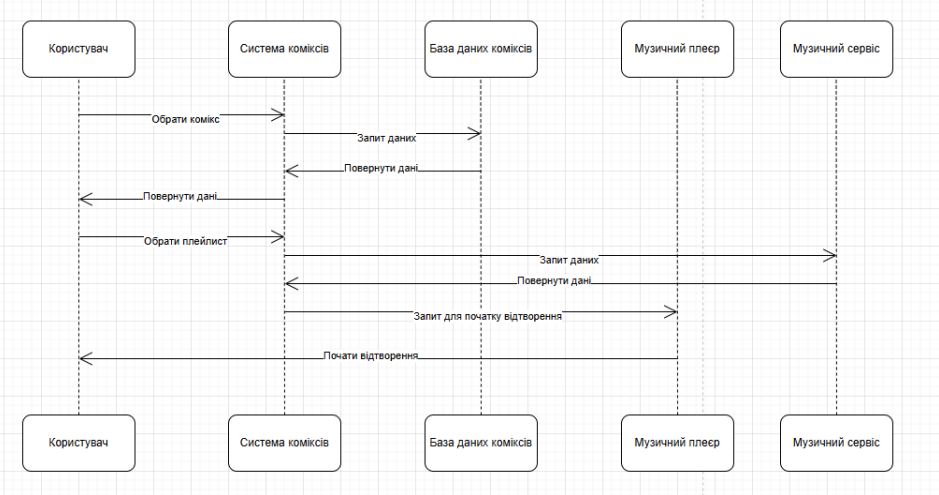


Рис. 2.11 – Діаграма послідовності “Відтворення коміксу з музичним супроводом”

Діаграма послідовності “Обмін повідомленнями в чаті під час читання”

Діаграма послідовності “Обмін повідомленнями в чаті під час читання” на рисунку 2.12 описує процес спільного читання коміксу двома користувачами з можливістю обміну повідомленнями через вбудований чат. У ній показано послідовну взаємодію між користувачами, системою читання, чат-сервером та базою даних.

Усе починається з того, що Користувач 1 обирає комікс, після чого система читання надсилає запит і отримує необхідні дані, які повертаються користувачу. Далі Користувач 1 ініціює спільне читання, надсилаючи запрошення до Користувача 2 через систему. Користувач 2 отримує запрошення, приймає участь і система повертає йому відповідні дані для синхронізації перегляду.

Після початку спільного сеансу читання Користувач 1 надсилає повідомлення. Система перенаправляє це повідомлення до чат-сервера, який своєю чергою зберігає його у базі даних. Після успішного збереження повідомлення чат-

сервер повертає підтвердження, і система надсилає його другому користувачу. Таким чином, Користувач 2 отримує повідомлення в режимі реального часу.

На завершення сеансу будь-який з користувачів може ініціювати завершення сесії. Система читання опрацьовує цю дію та надсилає підтвердження обом сторонам.

Цей сценарій ілюструє узгоджену роботу між системою читання та чат-сервером, що дозволяє забезпечити інтерактивне спілкування користувачів під час перегляду, з надійним збереженням історії повідомлень у базі даних.

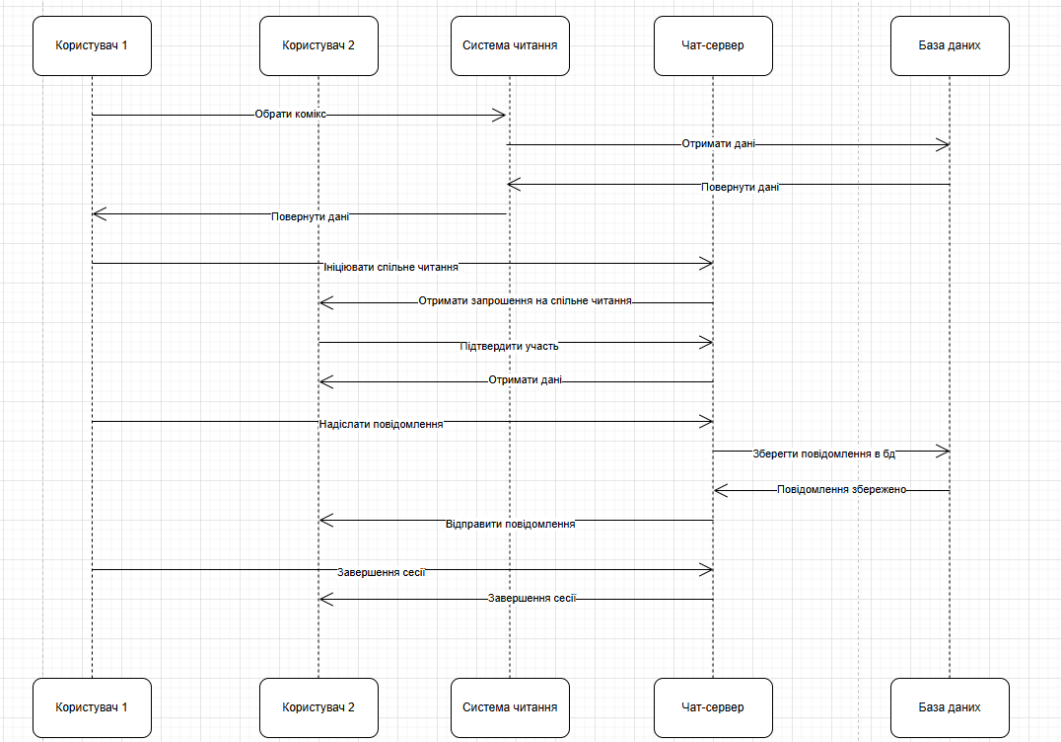


Рис. 2.12 – Діаграма послідовності “Обмін повідомленнями в чаті під час читання”

Діаграма кооперації об’єктів системи

На рисунку 2.13 представлено діаграму кооперації об’єктів системи для платформи читання коміксів із музичним супроводом, чатом і функцією спільного перегляду. Діаграма демонструє взаємодію основних об’єктів, таких як User, Comic, MusicTrack, Chat, GroupView, Subscription, Message, Comment та інших під час виконання ключових функцій системи. Користувач взаємодіє з об’єктом Comic для перегляду вмісту, слухає відповідні MusicTrack, надсилає

Message у рамках Chat, залишає Comment та Rating, формує власний FavoriteList і може організовувати або приєднуватися до GroupView для спільного читання. Creator створює нові комікси та завантажує музичні треки, які пов'язуються з коміксами. Administrator відповідає за управління користувачами, підписками (Subscription), контентом і обробку скарг (Report), забезпечуючи належне функціонування платформи.

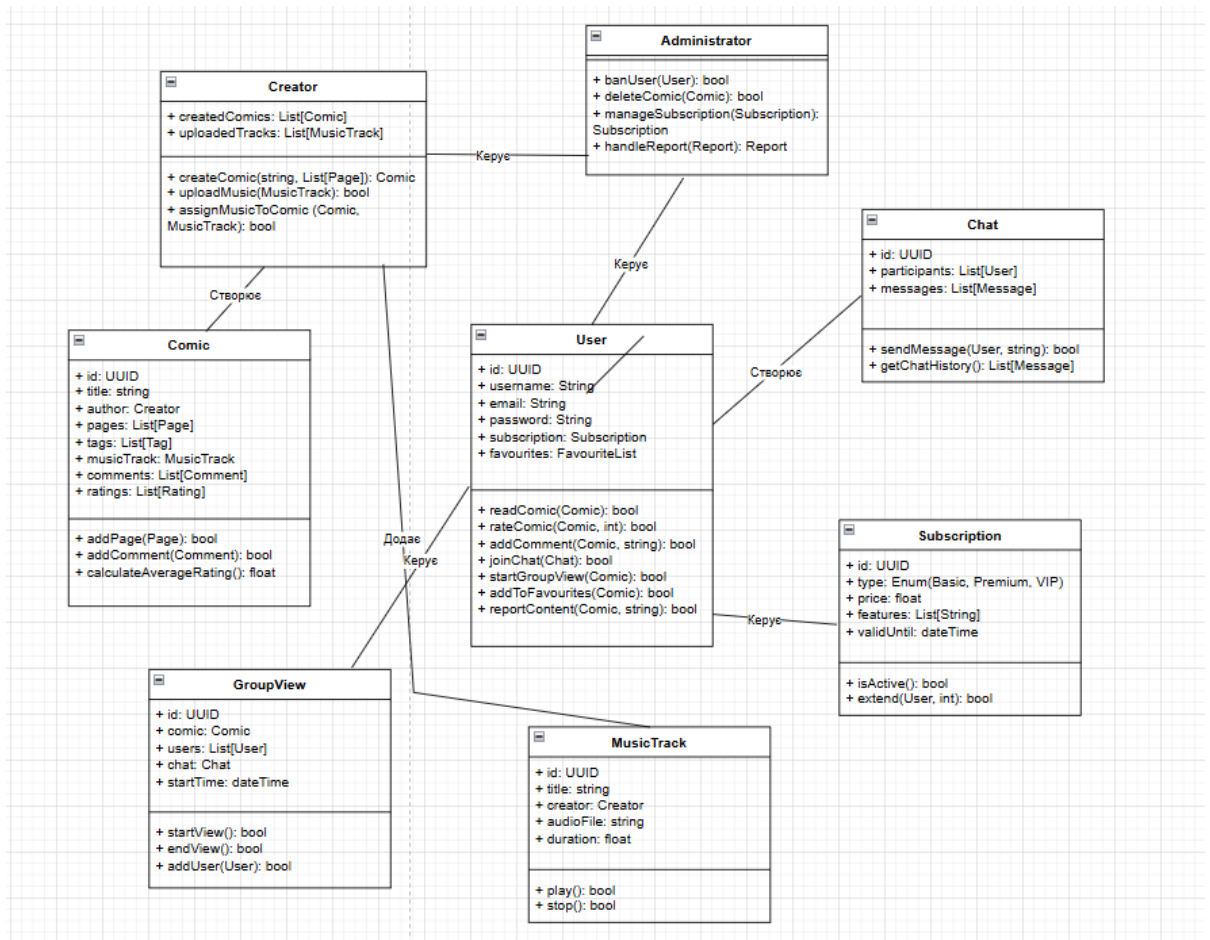


Рис. 2.13 – Діаграма кооперації “Платформи читання коміксів”

Девайс користувача

Користувач взаємодіє з платформою через веб-браузер, що надсилає HTTPS-запити до веб-сервера. Інтерфейс користувача реалізується у вигляді веб-сторінок, які дозволяють переглядати комікси, здійснювати пошук, коментувати та управляти підписками.

Веб-сервер

Центральний елемент обробки запитів, який включає декілька ключових

компонентів:

1. Веб-інтерфейс – відповідає за відображення сторінок і обробку дій користувача.
2. Інтерфейс бази даних – забезпечує доступ до даних, зокрема до інформації про комікси, користувачів, оцінки, коментарі та підписки.
3. Інтерфейс файлового сховища – забезпечує завантаження та віддачу зображень, аудіофайлів та іншого контенту, пов'язаного з коміксами.

Сервер бази даних

Для зберігання структурованої інформації використовується PostgreSQL, що містить дані про облікові записи користувачів, метадані коміксів, списки улюбленого, чати, групові перегляди, рейтинги, теги тощо. Для зберігання тимчасових (сесійних) даних використовується база даних Redis. На основі Redis реалізується модуль спільного читання.

Сервер файлового сховища

Файли зображень, обкладинки, музичний супровід до коміксів тощо зберігаються у зовнішньому файловому сховищі, до якого веб-сервер звертається через захищений інтерфейс.

Архітектура системи забезпечує: ефективну обробку запитів користувачів у реальному часі, надійне збереження даних та мультимедійних ресурсів, гнучке масштабування з можливістю розширення функціональності, стабільну роботу платформи навіть за умови великої кількості одночасних користувачів. Ця модель є базовою для побудови повноцінної системи доступу до інтерактивного контенту з підтримкою мультимедіа, комунікації та персоналізації.

2.4. Реалізація додатку веб-платформи для читання коміксів

Функція автоматичного читання коміксів

Ця функція дозволяє користувачу запустити автоматичний перегляд коміксу зі встановленою швидкістю зміни сторінок. На рівні клієнта це реалізується за допомогою механізму таймерів, які керують переходом між сторінками. Кожна сторінка коміксу – це окреме зображення, яке підвантажується з сервера.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		50

Під час запуску автоматичного режиму інтерфейс починає виводити сторінки одну за одною через заданий інтервал. Цей інтервал можна змінювати, наприклад, за допомогою повзунка або меню налаштувань. У випадку складних сцен чи збільшення зображення – таймер призупиняється.

На серверному рівні система лише один раз передає всі дані (URL сторінок), які кешуються на стороні клієнта для уникнення додаткових запитів. З метою оптимізації використовуються попереднє завантаження наступної сторінки (preloading) та lazy-loading попередніх.

Музичний плеєр, синхронізований із читанням

Цей функціонал дозволяє користувачу запускати музику під час читання коміксу. Крім того, плеєр може бути прив'язаний до окремих розділів. Наприклад, кожний розділ може мати свій власний музичний супровід. Для реалізації прототипу плеєру використовувалася документація з W3Schools [20].

Механізм реалізації передбачає:

1. Зберігання метаданих про композиції (назва, URL, тривалість, розділ коміксу).
2. Ініціалізацію Audio API у браузері з можливістю керування потоком (play, pause, stop).
3. Автоматичну зміну треків при переході до іншого розділу.
4. Налаштування гучності та циклу відтворення.

Дані про треки зберігаються в окремій таблиці бази даних і завантажуються асинхронно. Також передбачена можливість кешування треків у браузері для покращення продуктивності.

Кімнати для спільного перегляду (спільний режим)

Однією з найважливіших функцій є підтримка кімнат – групового читання одного й того ж коміксу декількома користувачами одночасно. Для реалізації цього сценарію використовується бібліотека Socket.io, яка забезпечує двостороннє з'єднання між клієнтом і сервером, та база даних Redis, що надає можливість збереження та швидкого доступу до інформації про кімнату.

		Шарбоков І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		51

Алгоритм роботи виглядає так:

1. Один із користувачів створює кімнату, що генерує унікальний ідентифікатор.
2. Інші користувачі приєднуються до кімнати, вводячи цей ідентифікатор.
3. При зміні сторінки організатором або ведучим – сигнал надсилається всім учасникам кімнати через WebSocket.
4. Кожен клієнт отримує команду й синхронізує свою поточну сторінку.

Для збереження стабільності використовується окремий namespace для кожної кімнати, механізми reconnection у разі обриву з'єднання, а також пінгування для перевірки активності клієнтів.

Збереження історії перегляду

Ще одна із функцій – це збереження останньої прочитаної сторінки та розділу. При кожному переході до нової сторінки відбувається запис до бази даних з userId, chapterId і timestamp. Важливо, що цей запис не відбувається надто часто – з метою зниження навантаження застосовується механізм debounce або локальне кешування з періодичним автооновленням (наприклад, кожні 15 секунд або при виході з розділу).

При повторному вході користувача в комікс система автоматично перенаправляє його на останню прочитану сторінку на основі цієї інформації.

Каталог коміксів

Модуль каталогу коміксів дозволяє користувачу шукати, сортувати та фільтрувати комікси для швидкого й зручного пошуку потрібного контенту. Пошук здійснюється за назвою, описом, авторами або тегами з використанням повнотекстового пошуку на сервері. Сортування доступне за популярністю, датою, рейтингом або алфавітом у вибраному напрямку. Фільтри допомагають відбирати комікси за жанром, статусом, типом, мовою, автором та іншими параметрами, які можна комбінувати. Всі запити обробляються сервером, а результати кешуються для швидкості. Інтерфейс оновлюється динамічно, підтримує пагінацію та адаптується під різні пристрої. Крім того, система зберігає історію пошуку і налаштувань користувача для зручності повторного використання.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

Висновки до другого розділу

У цьому розділі було розглянуто архітектуру платформи для читання коміксів з музичним супроводом, чатом та функцією спільного перегляду, а також взаємодію її основних компонентів. Діаграма класів відобразила структуру системи, ключові об'єкти – такі як User, Comic, Chapter, Page, Track, Review, ComicList, Rating, ChapterComment, ChapterLike, LastChapter – та зв'язки між ними. Діаграми послідовностей ілюстрували основні процеси платформи: перегляд коміксу з музикою, участь у чаті та спільний перегляд з іншими користувачами. Діаграма кооперації показала взаємодію об'єктів під час реалізації основних функцій. Використання шаблонів проєктування забезпечило гнучкість, розширюваність та ефективну побудову архітектури системи. У результаті детального опрацювання механізмів реалізації найскладніших функціональних компонентів онлайн-платформи для читання коміксів було визначено основні технологічні підходи та логіку їх інтеграції у загальну архітектуру програмного комплексу. Всі складові системи – автоматичне читання, музичний плеєр, кімнати для спільного перегляду, система донатів та збереження історії перегляду – розроблено з урахуванням високої продуктивності, масштабованості та зручності для кінцевого користувача.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		53

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ПЛАТФОРМИ ДЛЯ ЧИТАННЯ КОМІКСІВ

3.1. Порядок встановлення та налаштування параметрів системи

Встановлення та конфігурування платформи передбачає розгортання всіх її компонентів, а також налаштування як окремих елементів, так і всієї системи загалом.

Для наочного представлення архітектури платформи було розроблено діаграму розгортання (рис. 3.1).

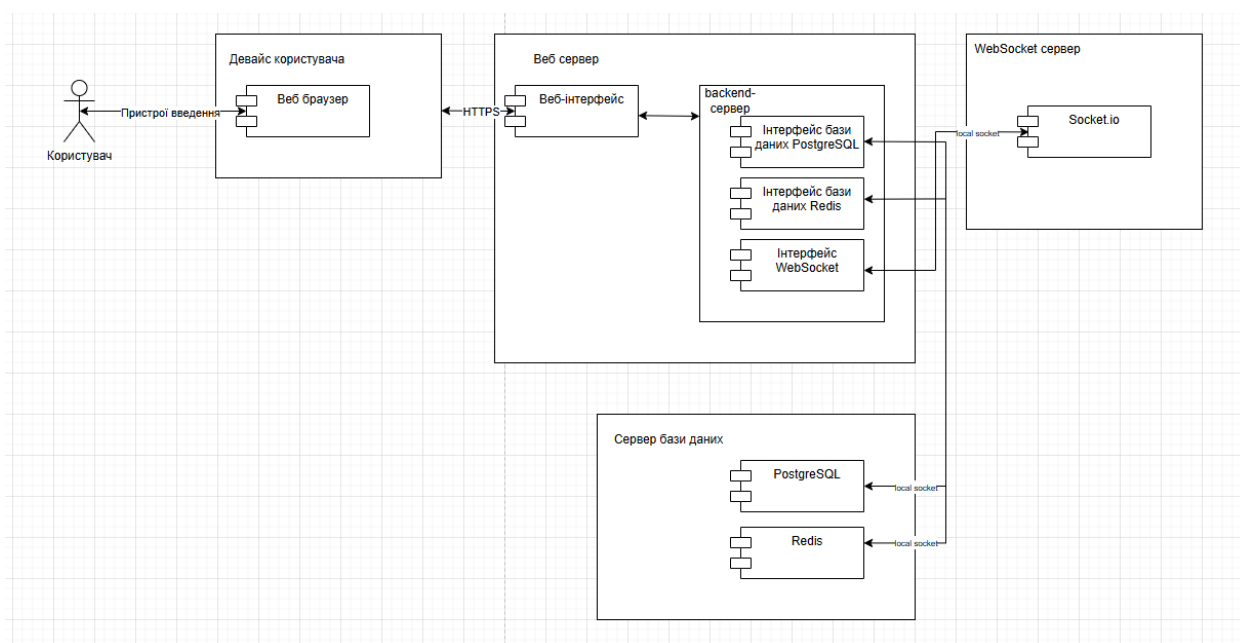


Рис. 3.1 – Діаграма розгортання веб-платформи для читання коміксів

Опис процесу розгортання.

Взаємодія компонентів – користувачі взаємодіють з фронтендом через браузер, який надсилає HTTPS-запити до веб-сервера, що, у свою чергу, передає їх до бекенда для обробки бізнес-логіки. Бекенд здійснює доступ до баз даних через відповідні інтерфейси PostgreSQL і Redis, а для забезпечення режиму реального часу взаємодіє з WebSocket-сервером через локальний сокет.

Backend-сервер – розгортання серверної частини додатку, яка обробляє запити користувачів, надає REST API, управляє даними коміксів, користувачів, музики, а також взаємодіє з базами даних PostgreSQL та Redis. Через WebSocket

інтерфейс здійснюється передача повідомлень у режимі реального часу для функцій спільного читання коміксів.

Сервер баз даних – система містить дві ключові СУБД: PostgreSQL, яка слугує основною реляційною базою даних для зберігання структурованих даних, таких як користувачі, комікси, розділи, уподобання та історія переглядів, і Redis, що використовується для зберігання тимчасових або сесійних даних, а також для синхронізації стану між користувачами в режимі спільного читання.

Websocket сервер – спеціалізований сервер, розгорнутий окремо, обробляє двосторонні з'єднання за допомогою Socket.io та використовується для передавання повідомлень між користувачами під час групового читання коміксів, синхронізації дій, таких як перегортання сторінок чи вмикання музики, а також для забезпечення оновлень у реальному часі без необхідності перезавантаження сторінки.

Взаємодія компонентів системи зображується за допомогою діаграм компонентів, які відображають залежності між ключовими елементами програмного забезпечення. Така діаграма показує структурні особливості системи, включно з модулями користувача, логікою керування контентом, обробкою мультимедіа, чатом і системою підписок. Вона дозволяє візуально відобразити архітектуру платформи та показати, як взаємодіють її частини.

На рисунку 3.2. представлено діаграму компонентів веб-платформи для читання коміксів з музичним супроводом, що включає компоненти для користувачів, творців контенту, адміністраторів, а також модулі керування коміксами, музичними треками, чатом, спільним переглядом та підписками. Зв'язки між компонентами демонструють основні процеси платформи: створення, читання, оцінювання, коментування та модерацію контенту.

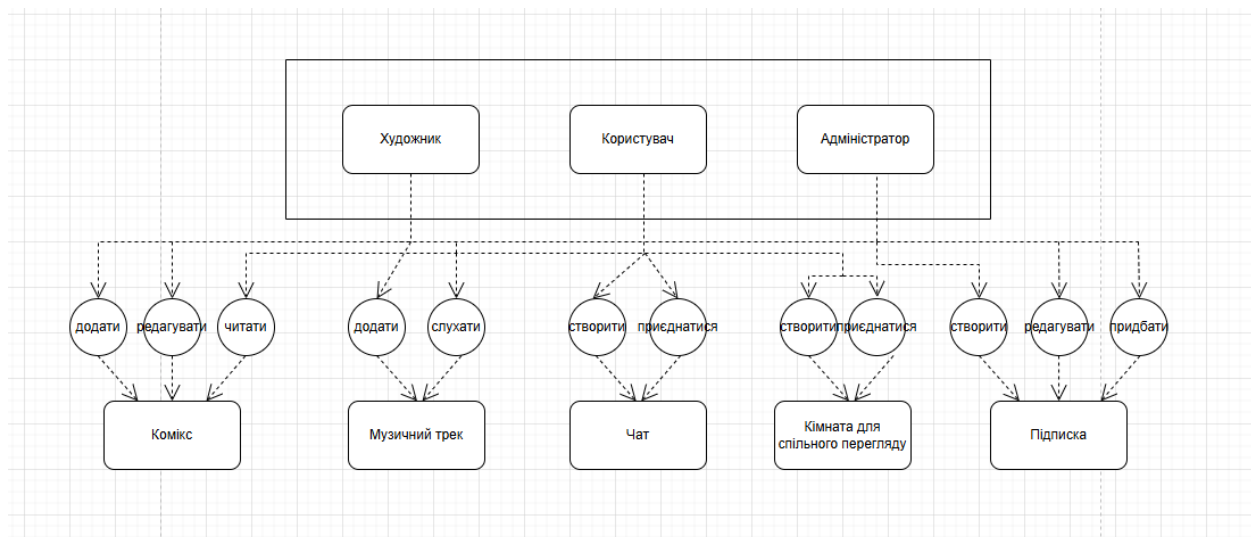


Рис. 3.2. – Діаграма компонентів системи для обробки даних

3.2. Структура інтерфейсу веб-платформи для читання коміксів

Взаємодія користувачів відбувається через вебдодаток. На головній сторінці користувач може списки різних видів останніх доданих коміксів(Новели, Манхви, Манги). Також користувач може побачити загалом останні додані комікси. Якщо ж користувач є авторизованим, то він буде бачити додатково список коміксів, які користувач може продовжити читати.

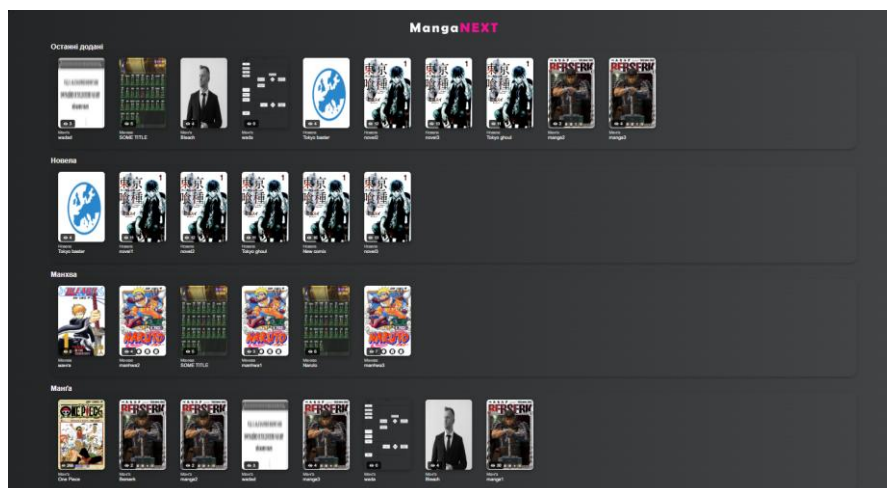


Рис. 3.3 – Головна сторінка додатку

В шапці сайту ми можемо бачити, такий функціонал, як: перехід на головну сторінку, перехід на сторінку з каталогом, пошук конкретного коміксу, та кнопка авторизації. Якщо користувач авторизований, в шапці він матиме такий

додатковий функціонал: додати новий комікс, перехід на сторінку власного профілю та вихід з акаунту.



Рис. 3.4 – Шапка додатку

В самому низу сторінки є посилання на політику конфіденційності, публічну оферту, та інформацію щодо DMCA – правовласникам. Також вказана пошта для звернень, при виникненні питань щодо авторських прав.



Рис. 3.5 – Підвал додатку

Далі при наведення на комікс, ми можемо бачити додатку інформацію щодо нього: назва, кількість переглядів, кількість вподобань, кількість людей що добавили його в список, опис.



Рис. 3.6 – Додаткова інформація про комікс

При натисканні на сам комікс попадаємо на саму сторінку з коміксом, в якому бачимо усю можливу інформацію про нього, кнопку почати читати чи продовжити, можливість додавати його у власний список (читаю, у планах, кинуті і

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

т. д.) та можливість переглядати, виставляти оцінку коміксу. Також бачимо чотири вкладки: загальна, розділи, відгуки, плейлист.

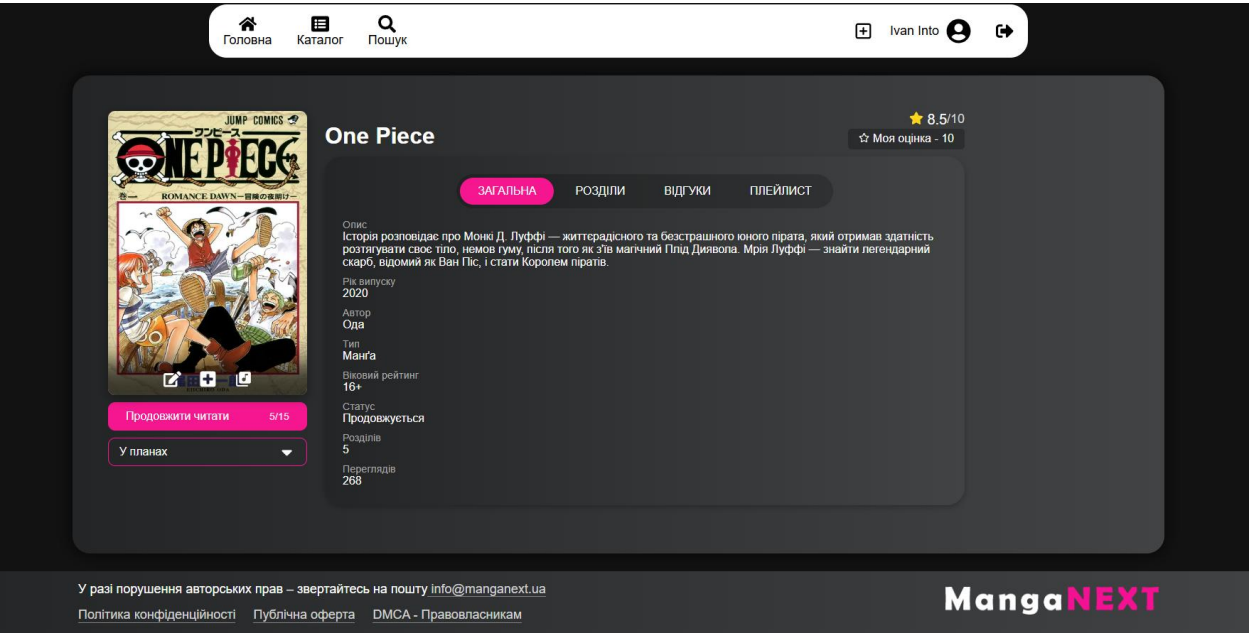


Рис. 3.7. Сторінка коміксу. Вкладка “загальна”

Вкладки можемо переключити. При переключенні на вкладку “розділи” отримуємо список усіх розділів коміксу та можливість їх пошуку.

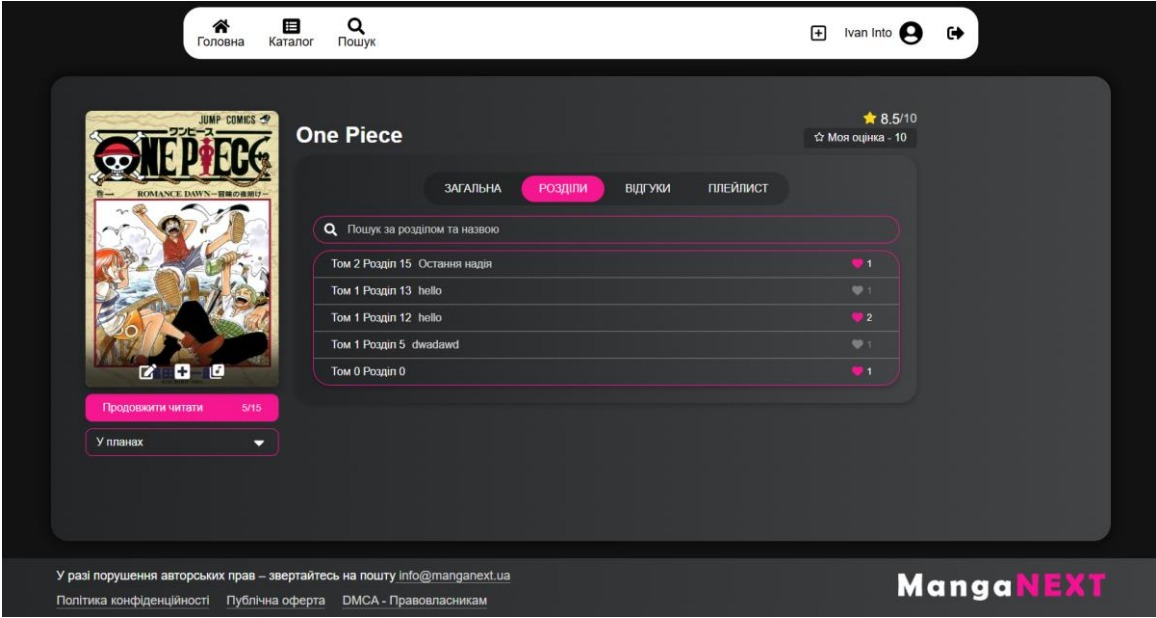


Рис. 3.8. Сторінка коміксу. Вкладка “розділи”

У вкладці “відгуки” бачимо усі відгуки до коміксу та можливість надіслання власного відгуку.

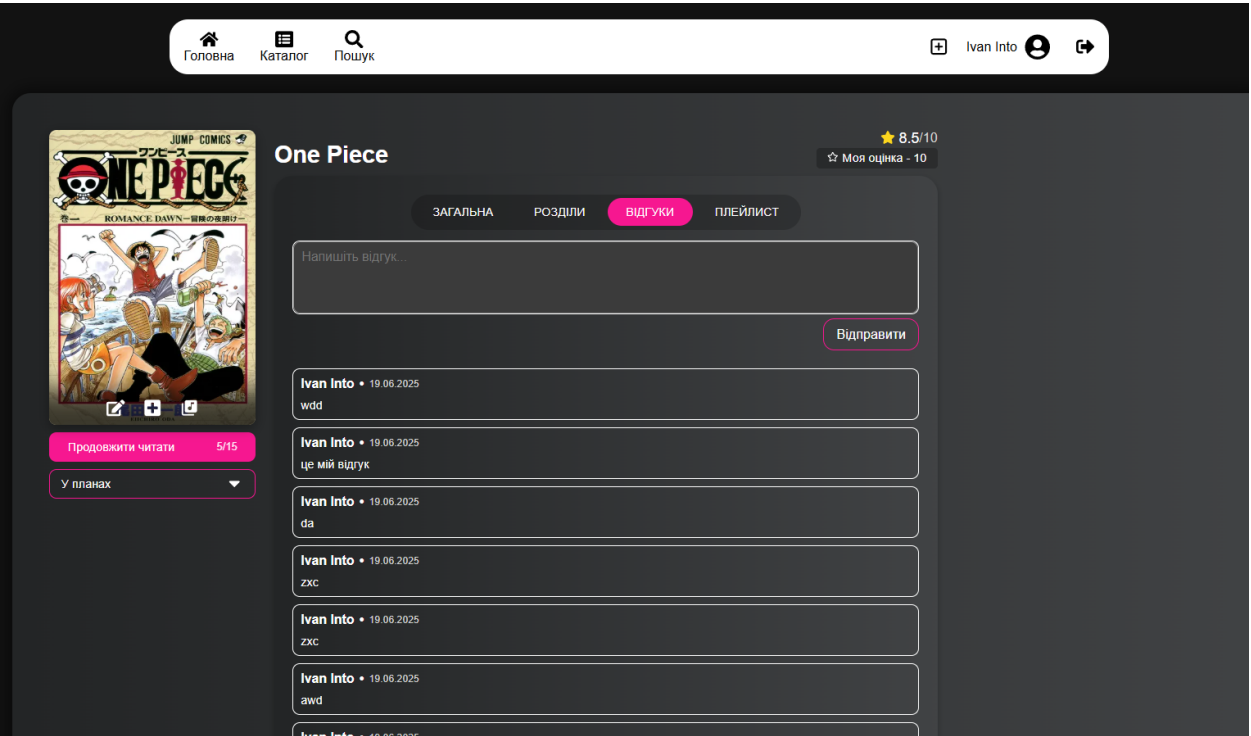


Рис. 3.9. Сторінка коміксу. Вкладка “відгуки”

У вкладці “плейлист” можемо прослухати та побачити список усіх аудіотреків які автор коміксу завантажив для прослуховування під час читання.

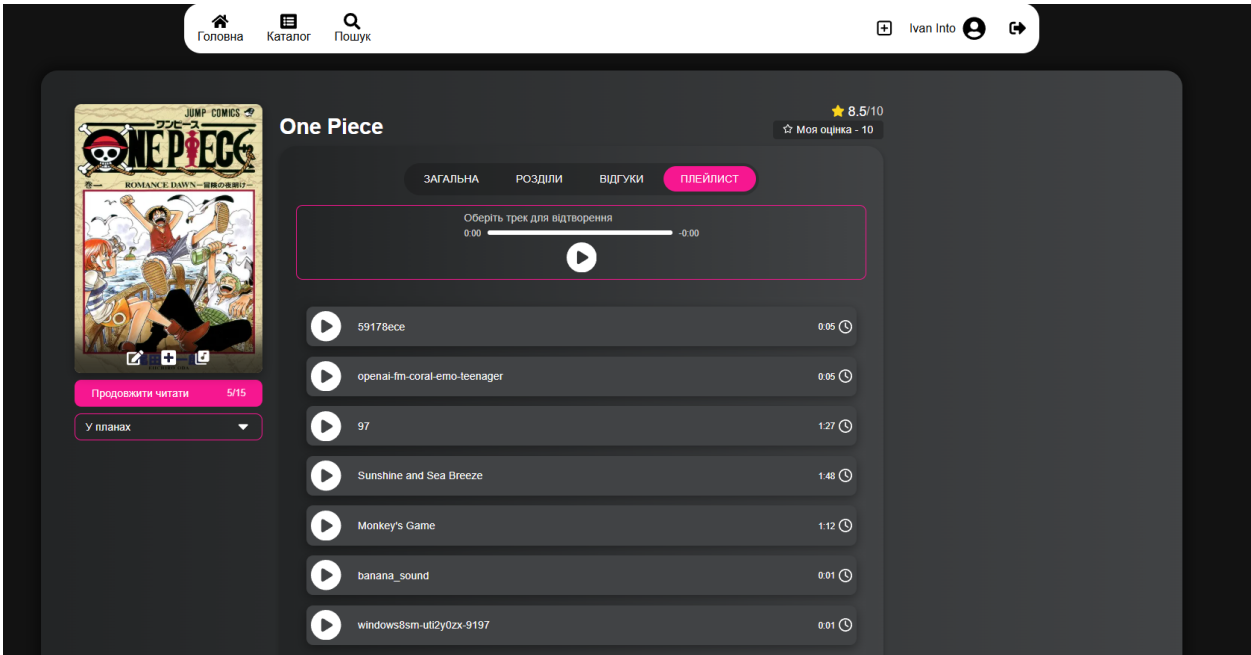


Рис. 3.10. Сторінка коміксу. Вкладка “плейлист”

Якщо користувач авторизований він може виставити оцінку коміксу від 1 до 10.

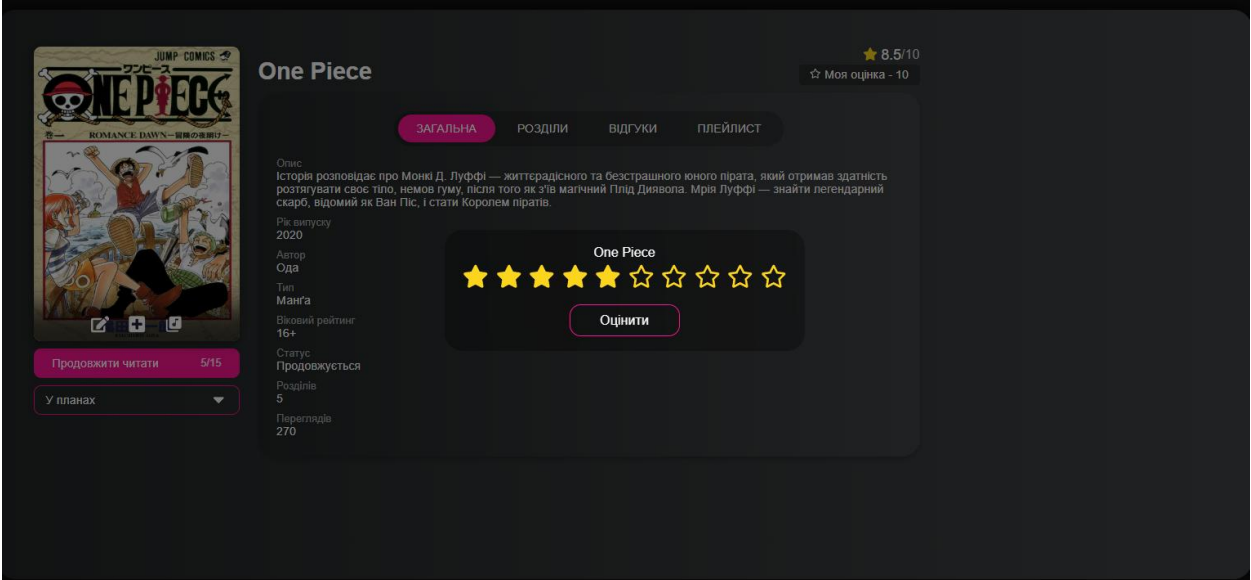


Рис. 3.11. Сторінка коміксу. Оцінювання

Для того щоб читати комікс, потрібно або натиснути на кнопку “Продовжити читати” або “Почати читати”. Також на вкладці “розділи” можна обрати потрібний розділ. Після цього ми попадемо на сторінку читання коміксу. В шапці можемо бачити такий функціонал: повернутися до сторінки з коміксом, обрати розділ, музичний програвач, інформація про кімнату для спільного читання, вибір треку в музичній колекції, додавання коментарів до розділу та налаштування.

Далі на самій сторінці, можемо бачити сторінка самого розділу коміксу, зліва знизу бачимо кнопку для швидкого прокручування до потрібної сторінки, да справа по центру кнопку для автоматичного прокручування сторінки.

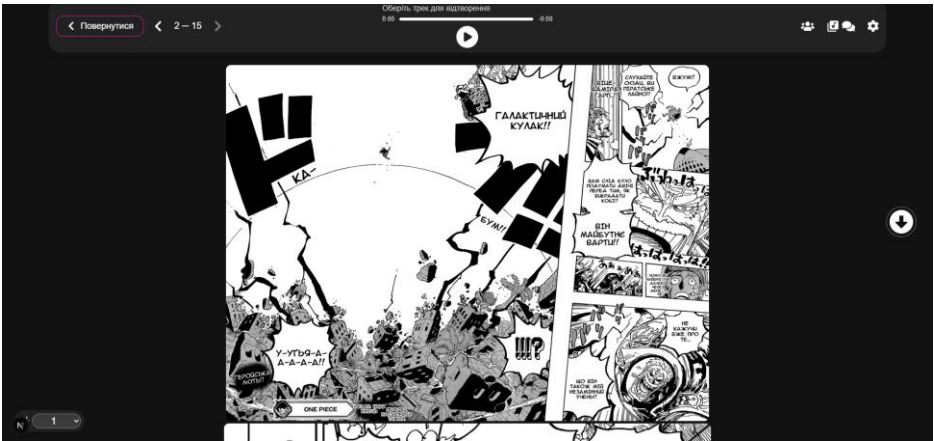


Рис. 3.12. Сторінка читання коміксу

В списку розділів можемо переключитися відразу на потрібний розділ та поставити вподобання розділу, який сподобався.

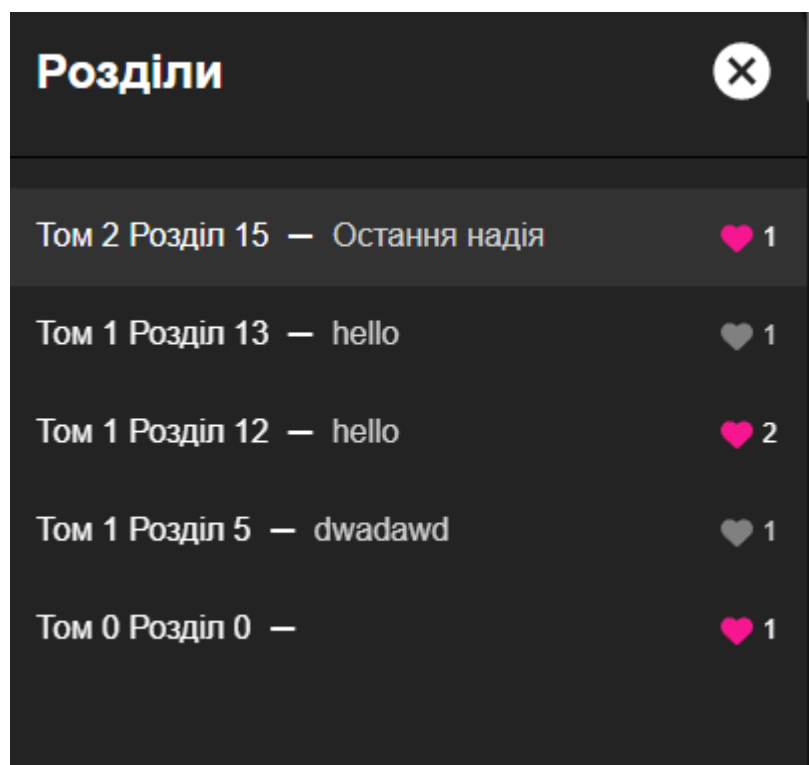


Рис. 3.13. Список розділів на сторінці читання

В налаштуваннях можна обрати тип читання: вертикальний, горизонтальний та тему: темну, світлу.

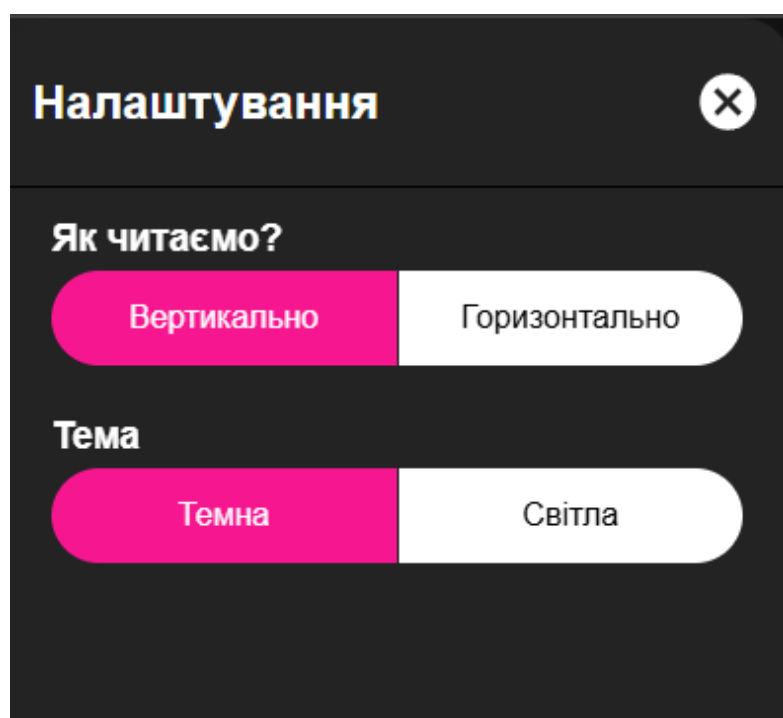


Рис. 3.14. Налаштування на сторінці читання

		Шарбоков І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

У коментарях до розділу, бачимо саму можливість залишити коментар та усі залишені коментарі.

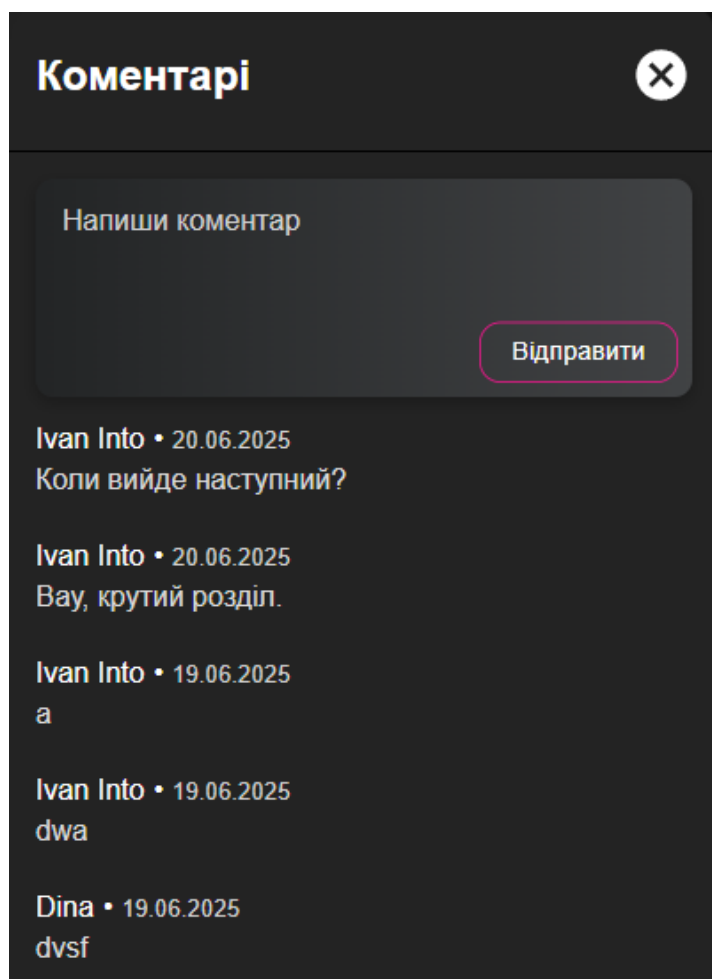


Рис. 3.15. Коментарі до розділу

У вкладці з кімнатою для спільного перегляду спочатку бачимо кнопку створити кімната, яка є доступною тільки для авторизованих користувачів.

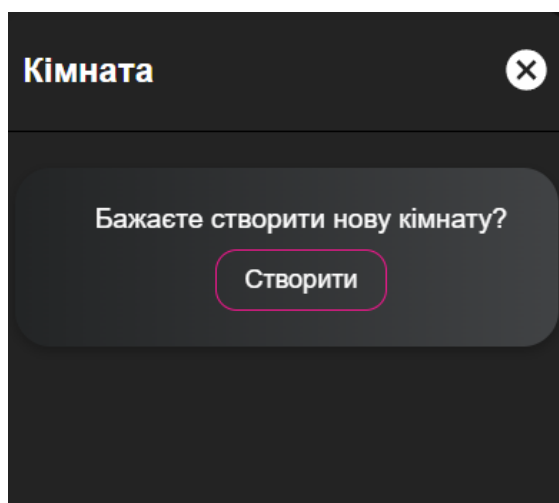


Рис. 3.16. Можливість створення кімнати

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

Після того як ми створити кімнату або приєдналися до вже існуючої, можемо побачити посилання для приєднання, список учасників, чат, та можливість вийти з кімнати.

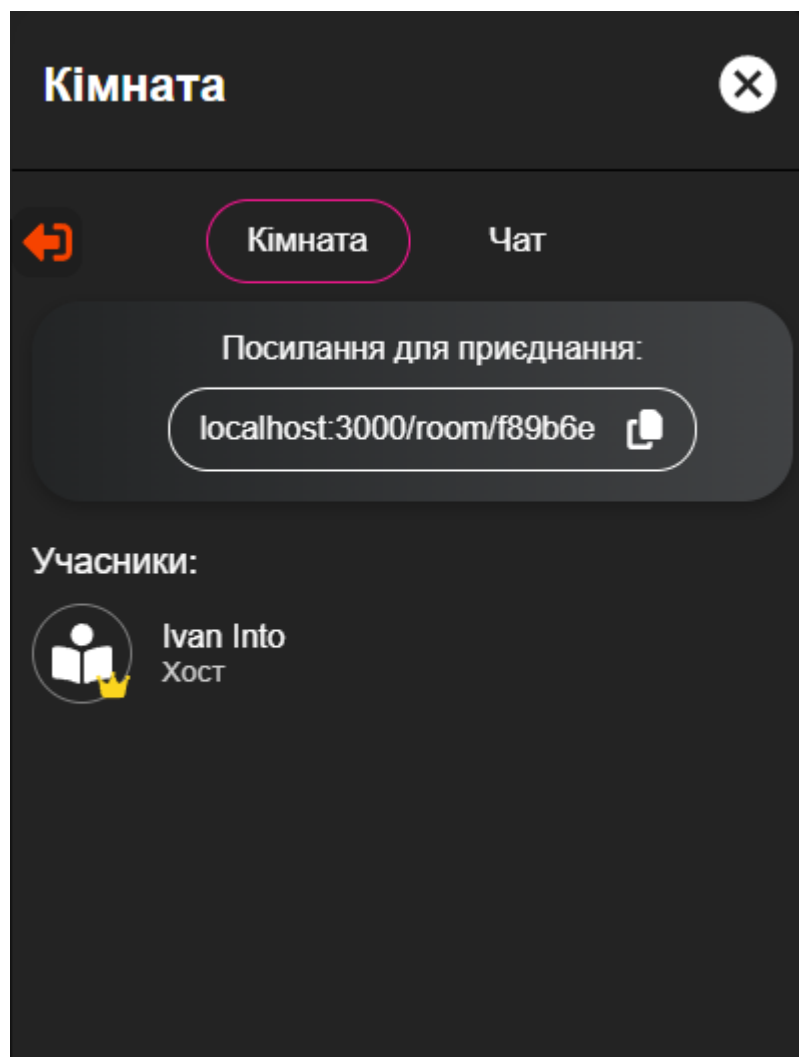


Рис. 3.17. Вікно з інформацією про кімнату

Також в кімнаті є чат, в якому учасники можуть спілкуватися між собою під час читання коміксу:

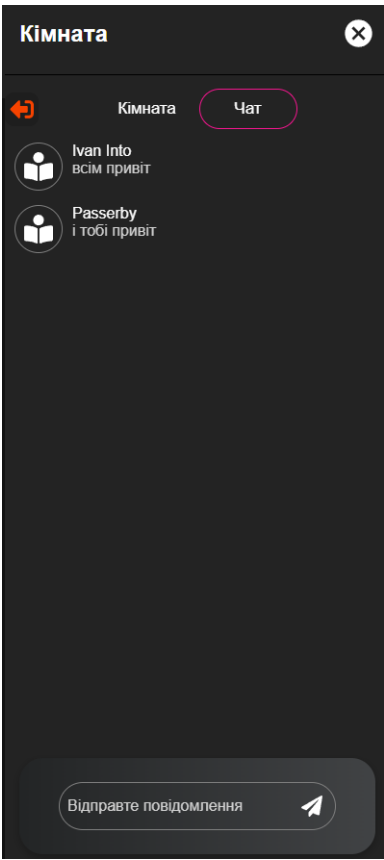


Рис. 3.18. Чат в кімнаті

Далі перейдемо на сторінку каталогу, де можемо бачити можливість фільтрації, сортування та пошуку коміксів.

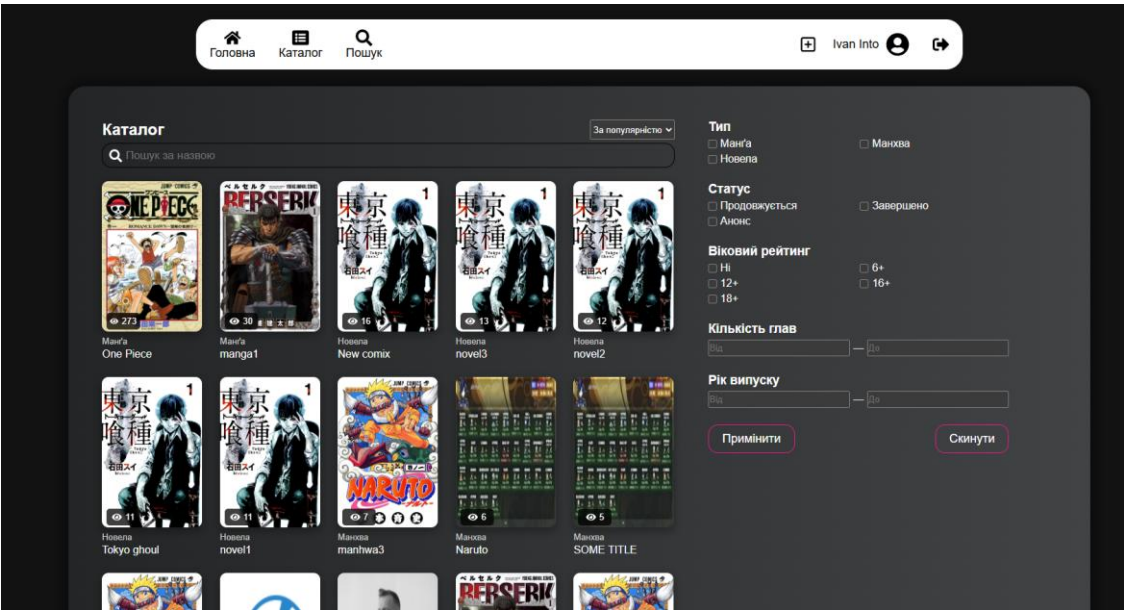


Рис. 3.19. Каталог в додатку

В звичайному пошуку, ми можемо знайти потрібний комікс відразу просто вписав частину його назви.

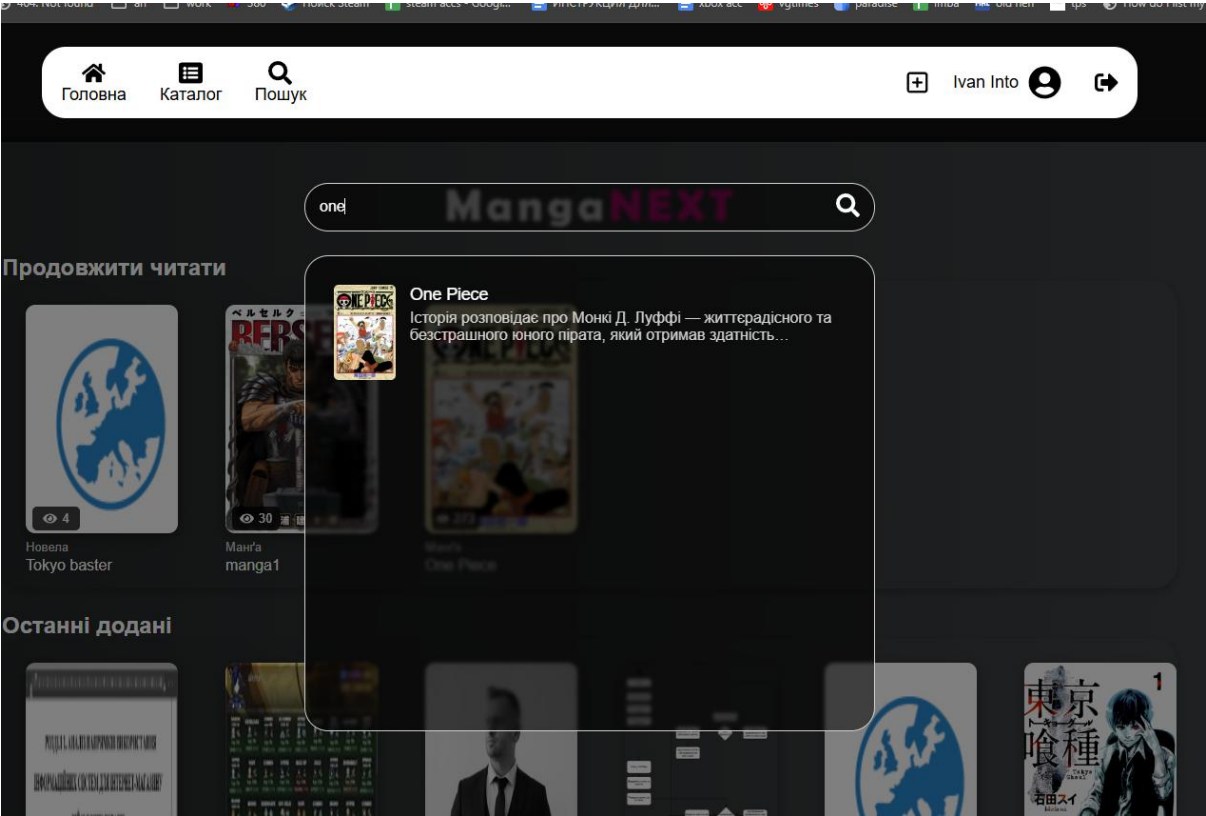


Рис. 3.20. Пошук в додатку

Кожен користувач може додати свій власний комікс. На сторінці коміксу йому потрібно заповнити усі поля що вказані та завантажити обкладинку.

The screenshot shows the 'Додати новий комікс' (Add new manga) form. It contains several input fields: 'Назва' (Name), 'Опис' (Description), 'Автор' (Author), 'Рік випуску' (Release year), 'Тип' (Type) with a dropdown menu showing 'Манґа', 'Вікове обмеження' (Age restriction) with a dropdown menu showing 'Немає', 'Статус' (Status) with a dropdown menu showing 'Продовжується', and 'Обкладинка' (Cover) with an upload icon. A 'Додати' (Add) button is at the bottom.

Рис. 3.21. Сторінка додавання коміксу в додатку

Після того як користувач додав комікс у нього є можливість його редагувати та додавати розділи і треки до нього.

ГоловнаКаталогПошук

Ivan Into

Редагувати комікс - One Piece

Назва

One Piece

Опис

Історія розповідає про Монкі Д. Лuffy — життєрадісного та безстрашного юного пірата, який отримав здатність розтягувати своє тіло, немов гуму, після того як з'їв маг'яний Плід Диявола. Мрія Лuffy — знайти легендарний скарб, відомий як Ван Піс, і стати Королем піратів.

Автор

Ода

Рік випуску

2020

Тип

Манґа

Вікове обмеження

16+

Статус

Продовжується

Обкладинка



Змінити

Рис. 3.22. Сторінка редагування коміксу в додатку

ГоловнаКаталогПошук

Ivan Into

Додати розділ

Номер тому

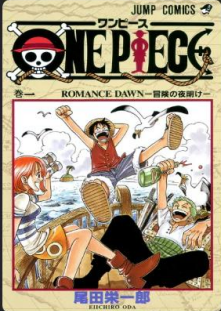
1

Номер розділу

2

Назва

Новий розділ



One Piece



1



2



3



4



5



6



7



8



9

Додати

Рис. 3.23. Сторінка додавання нового розділу до коміксу

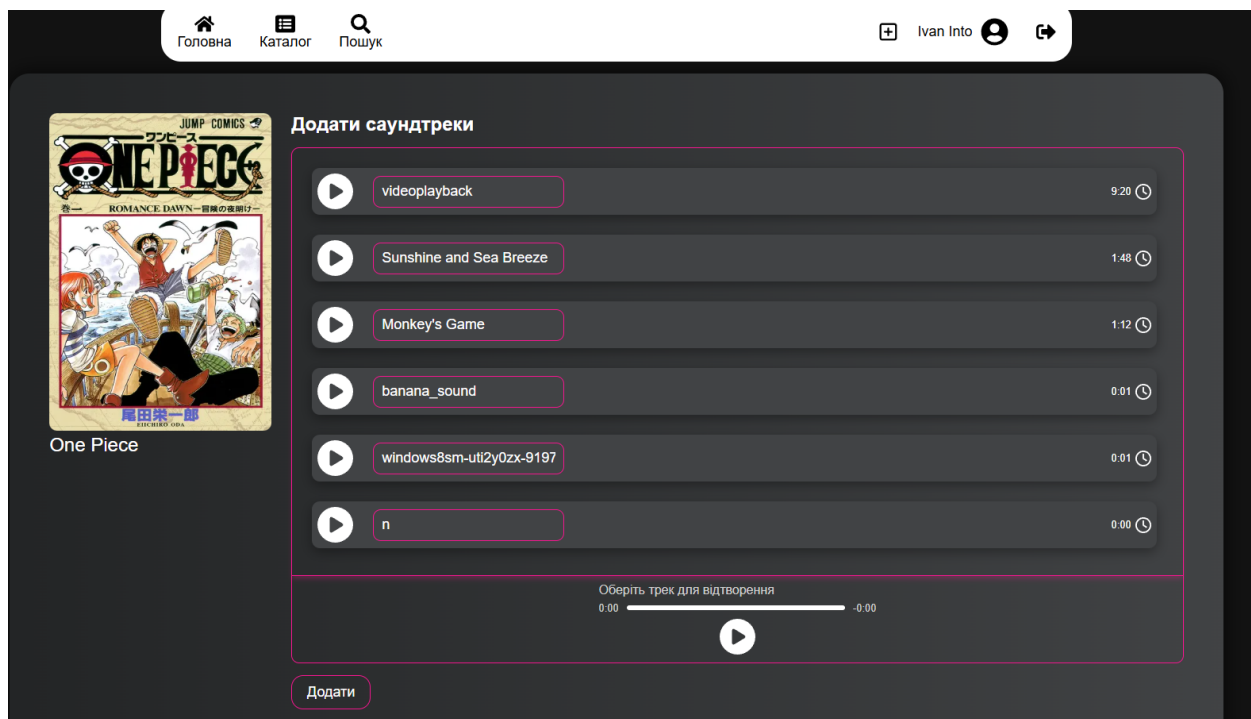


Рис. 3.24. Сторінка додавання саундтреків до коміксу

На сторінці власного профілю, користувач може змінити свою аватарку, бачити власні списки коміксів, та видаляти комікси зі списків.

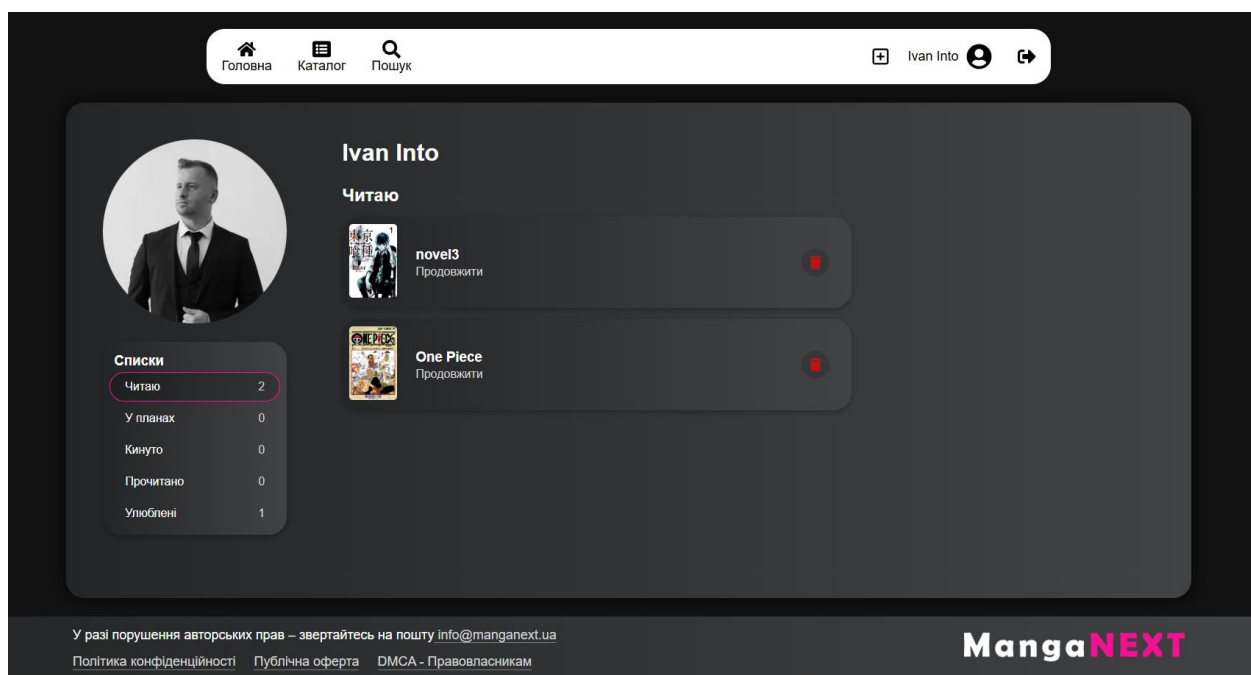


Рис. 3.25. Сторінка профілю в додатку

Під час авторизації користувач може обрати вхід за поштою, паролем або через Google. Для реєстрації користувачу ще також потрібно вказати його ім'я.

Рис. 3.26. Вікно входу в обліковий запис

Рис. 3.27. Вікно реєстрації облікового запису

		Шарбоков І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

3.3. Тестування роботи програмного комплексу

Під час тестування вебдодатку було використано онлайн-сервіси GTMetrix та PageSpeed.

GTMetrix – це інструмент, призначений для оцінювання продуктивності та швидкості завантаження вебсторінок. Він формує детальний звіт, у якому відображено ключові метрики, зокрема час завантаження сторінки, її розмір, кількість запитів до сервера та інші показники, що впливають на загальну ефективність сайту.

Після проведення аналізу було сформовано звіт, який містить розгорнуту інформацію про продуктивність вебресурсу (рис. 3.23).

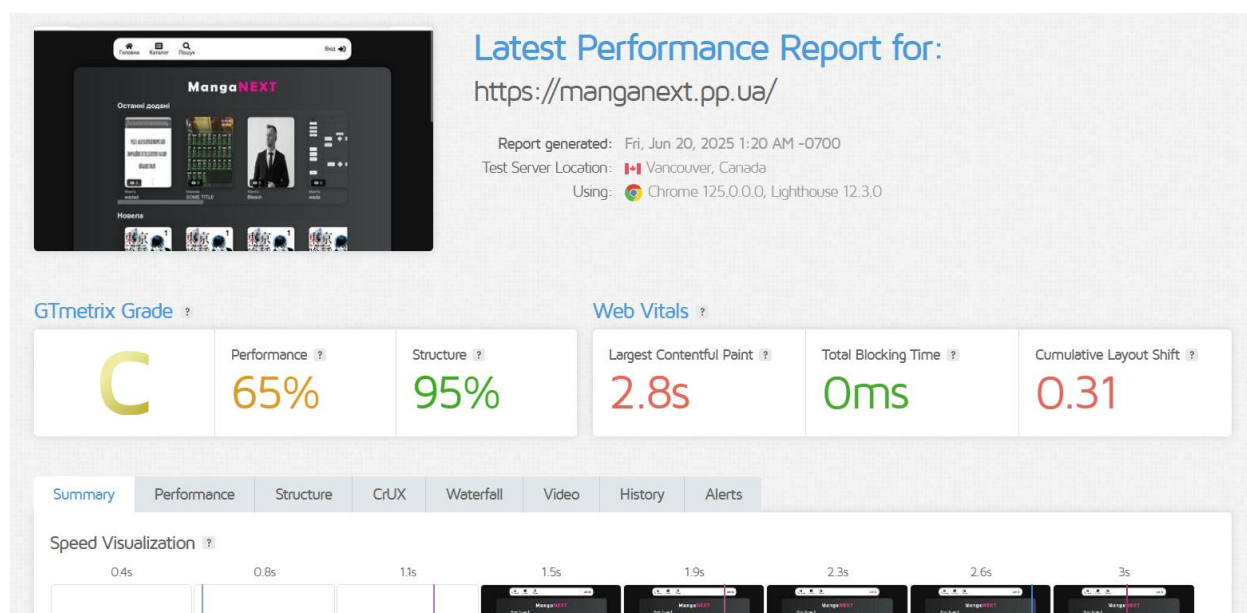


Рис. 3.28. Тестування додатку за допомогою GTMetrix

Звіти GTMetrix надають детальну інформацію про продуктивність вебсайту, зокрема загальний час завантаження сторінки, час відображення основного вмісту, кількість серверних запитів, об'єм сторінки, швидкість завантаження на різних типах інтернет-з'єднання тощо. Ці дані дозволяють оцінити ефективність роботи сайту та виявити шляхи для покращення його швидкодії (рис. 2.24).

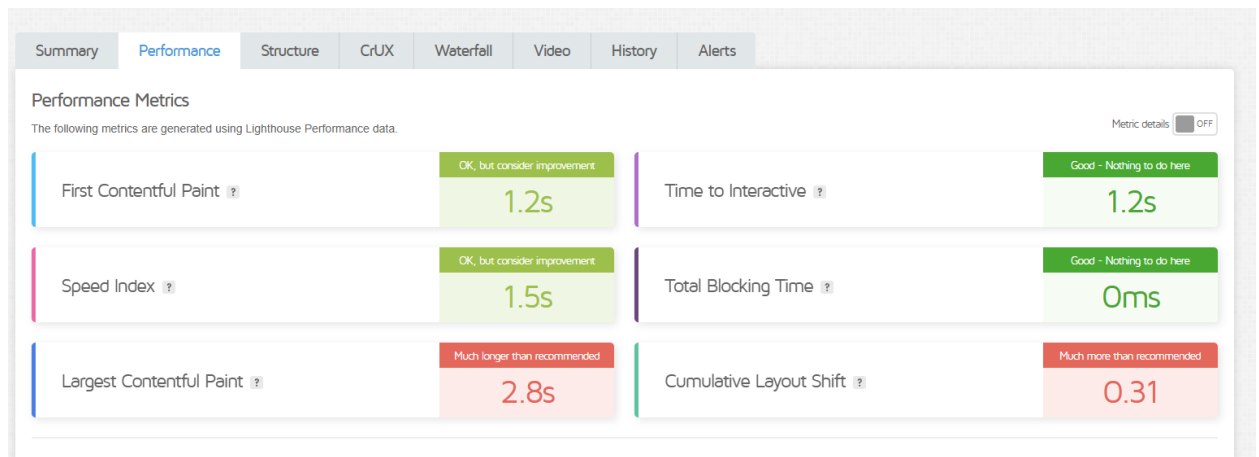


Рис. 3.29. Тестування швидкості вебдодатку за допомогою GTMetrix

GTMetrix також пропонує широкий набір інструментів і функцій для покращення продуктивності вебсайту. Серед них – збереження та порівняння звітів, відстеження змін у швидкодії сайту, автоматичний моніторинг продуктивності, сповіщення про виявлені проблеми та інші корисні можливості.

Pagespeed (рис. 3.30) – це вебресурс, розроблений командою Google, який пропонує інструменти та поради для підвищення швидкості завантаження вебсторінок. Його основна мета – допомогти веброзробникам покращити ефективність і продуктивність своїх сайтів.

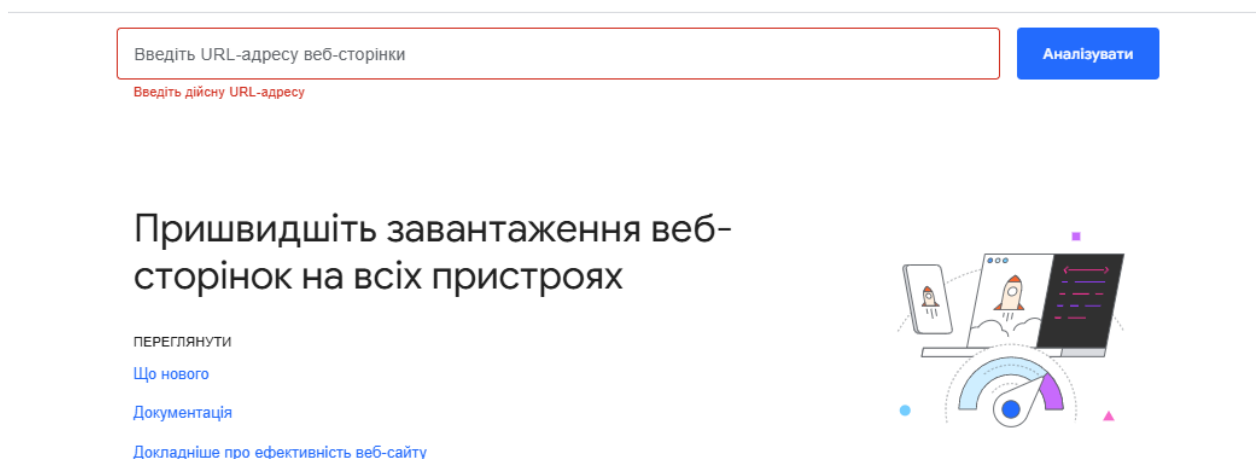


Рис. 3.30. Веб сайт Pagespeed

Інструмент для аналізу швидкості сторінок дозволяє користувачам ввести URL свого вебсайту та отримати детальний звіт щодо його швидкості завантаження. Сервіс оцінює такі параметри, як час завантаження сторінки, її розмір, кількість серверних запитів та інші важливі показники (рис. 3.26).

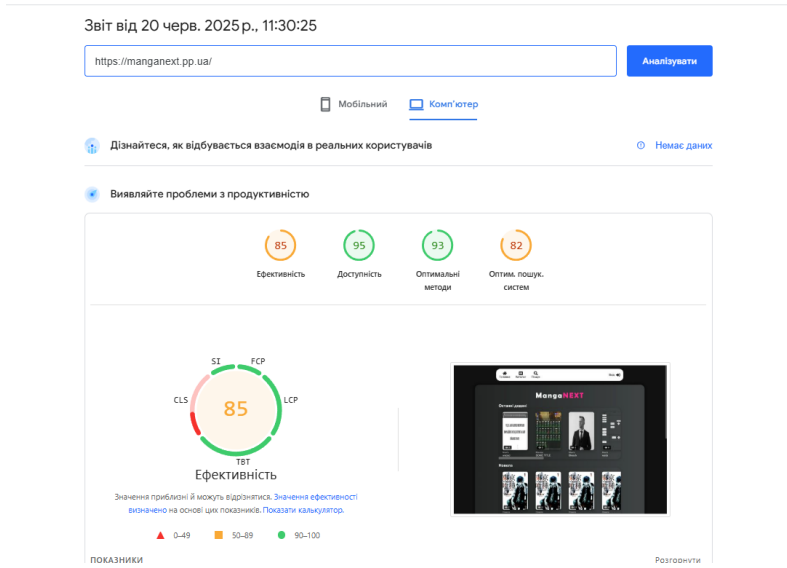


Рис. 3.31. Тестування веб-додатку за допомогою Pagespeed

Рекомендації з оптимізації. Вебсайт надає детальний перелік порад і рекомендацій щодо покращення швидкості завантаження сторінок. Ці поради ґрунтуються на перевірених практиках і рекомендаціях Google з оптимізації вебресурсів.

Перевірка мобільної адаптації. Крім аналізу швидкості, вебсайт також здійснює оцінку пристосованості ресурсу до мобільних пристроїв. Він пропонує рекомендації щодо адаптивного дизайну, розміщення елементів контенту та інших чинників, які впливають на зручність користування сайтом з мобільних пристроїв (рис. 3.32).

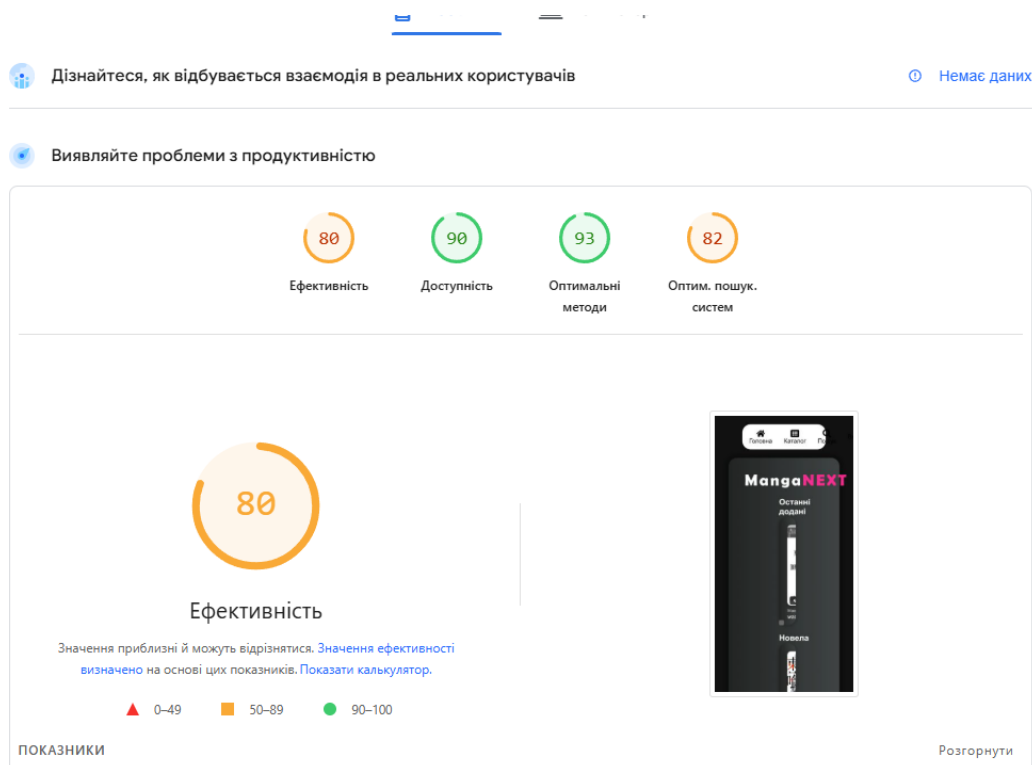


Рис. 3.32. Тестування веб-додатку за допомогою Pagespeed

Огляд ключових метрик. Вебсайт дозволяє переглядати різноманітні показники швидкодії, зокрема Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS), Total Blocking Time (TBT) та інші. Це дає змогу зосередитися на окремих аспектах продуктивності та виявити й усунути проблеми, що негативно впливають на загальну ефективність сайту (рис. 3.33).

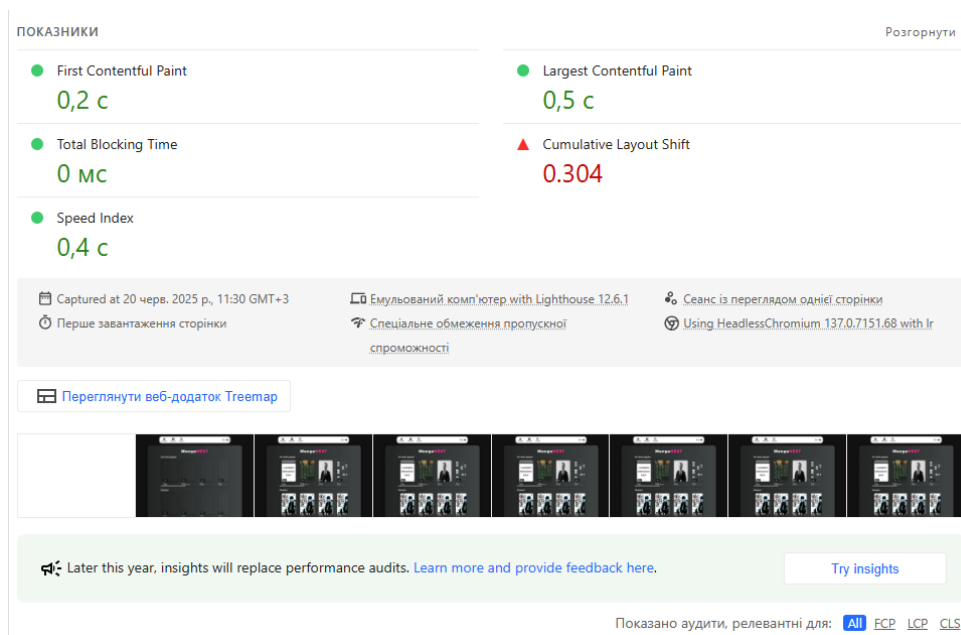


Рис. 3.33. Тестування вебдодатку за допомогою Pagespeed

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

Поради з покращення. Вебсайт пропонує конкретні рекомендації для підвищення швидкості завантаження сторінок. Серед них – поради щодо стиснення зображень, налаштування кешування, мінімізації CSS і JavaScript файлів, використання асинхронного завантаження ресурсів та інші ефективні методи оптимізації (рис. 3.34).

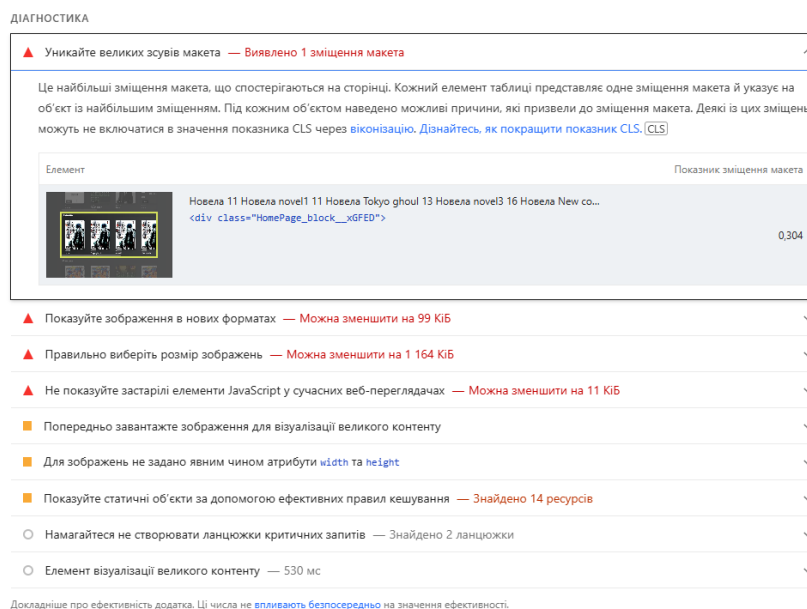


Рис. 3.34. Тестування вебдодатку за допомогою Pagespeed

Висновки до третього розділу

У даному розділі було детально розглянуто інтерфейс, функціональні можливості та порядок роботи з веб-платформою для читання коміксів. Описано процес розгортання системи, її архітектуру та взаємодію основних компонентів, зокрема фронтенду, бекенду, серверів баз даних і WebSocket-сервера. Завдяки використанню діаграм розгортання та компонентів стало можливим наочно представити структуру системи та її логічну організацію.

Інтерфейс платформи реалізований у зручному для користувача форматі: з головною сторінкою, каталогом, профілем користувача, функціоналом додавання коміксів та можливістю спільного читання з чатом і музичним супроводом. Особлива увага приділена зручності навігації, адаптивності інтерфейсу та персоналізації взаємодії.

Для оцінки продуктивності вебдодатку було проведено тестування за допомогою інструментів GTMetrix та PageSpeed. Обидва сервіси надали розгорнуті звіти щодо швидкості завантаження сторінок, їх розміру, кількості запитів та інших технічних параметрів. Отримані результати дозволили виявити потенційні вузькі місця у продуктивності та сформулювати рекомендації для подальшої оптимізації. Таким чином, вебплатформа не лише відповідає сучасним вимогам функціональності, а й демонструє хороші показники продуктивності та зручності користування.

		Шаробок І. В.			ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		74

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було спроектовано та реалізовано веб-платформу для читання коміксів, яка орієнтована на сучасного користувача та враховує актуальні тенденції цифрового споживання контенту. Метою роботи було створення функціонального, зручного та інноваційного інструменту, що дозволяє користувачам переглядати комікси у зручному форматі, отримуючи при цьому розширений мультимедійний досвід.

На початковому етапі роботи було проведено аналіз предметної області, вивчено існуючі рішення, які надають подібні послуги – від базових платформ для читання до повноцінних комікс-сервісів. Було визначено, що переважна більшість платформ надає лише базовий функціонал, обмежуючи взаємодію користувача з контентом лише пасивним переглядом. Відсутність інноваційних можливостей, таких як супровідний звук чи спільне читання, вказувала на вільну нішу на ринку, що й зумовило вибір напряму розробки.

На основі аналізу сформульовано основні вимоги до платформи: підтримка великої кількості графічного контенту, адаптивність інтерфейсу для різних пристроїв, зручна система навігації, можливість персоналізації, а також впровадження інноваційних функцій для покращення користувацького досвіду.

Було розроблено архітектуру вебдодатку, яка включає клієнтську та серверну частини, а також базу даних для зберігання інформації про користувачів, комікси, жанри, закладки, історію перегляду та активність. Було реалізовано систему реєстрації, авторизації, ролей користувачів (звичайні користувачі, адміністратори), механізми безпеки та захисту персональних даних.

Особливу увагу в платформі приділено двом унікальним функціям, що значно підвищують інтерактивність та емоційне занурення користувачів у процес читання. Перша – музичне супроводження, яке дозволяє читати комікси під фонову музику, що відповідає настрою та жанру конкретного твору. Це створює глибший емоційний зв'язок із сюжетом і підсилює враження від візуального контенту. Для реалізації цієї функції було впроваджено систему динамічного підкл-

		Шаробок І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		75

ючення аудіотреків до окремих коміксів із можливістю керування відтворенням. Друга – кімнати для спільного читання, які дають змогу кільком користувачам одночасно переглядати один і той самий комікс у синхронізованому режимі. У середині таких кімнат також доступний чат для спілкування між читачами. Це створює ефект спільного перегляду, подібного до онлайн-кінотеатрів, але адаптованого до формату графічного оповідання. Було також спроектовано та реалізовано інтерфейс користувача з урахуванням принципів UX/UI дизайну: мінімалізм, інтуїтивність, доступність функцій.

Для адміністрації реалізовано окремий модуль управління контентом: завантаження та редагування коміксів, категоризація, перегляд статистики активності користувачів, керування правами доступу. Це дозволяє ефективно модерувати платформу та реагувати на потреби аудиторії.

В результаті роботи створено повноцінну веб-платформу, яка поєднує в собі інструменти для читання, соціальної взаємодії та розширеного мультимедійного супроводу. Такий підхід не лише розширює функціональність традиційних комікс-сервісів, а й створює нові можливості для залучення користувачів та формування активної спільноти навколо платформи.

Таким чином, поставлені у роботі цілі були повністю досягнуті, а отриманий результат має як практичну цінність для кінцевих користувачів, так і потенціал для подальшого розвитку – зокрема, у напрямку мобільної версії, підтримки інтерактивних коміксів, інтеграції з соціальними мережами або впровадження підписної моделі монетизації.

		Шаробоків І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WebToon [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.webtoons.com/>
2. Tapas [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.tapas.com/>
3. MangaPlus [Електронний ресурс] – Режим доступу до ресурсу:
<https://mangaplus.shueisha.co.jp/>
4. ComiXology [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.amazon.com/kindle-dbs/comics-store/>
5. WebComics [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.webcomicsapp.com/>
6. JavaScript | MDN Web Docs [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
7. Node.js [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodejs.org/>
8. React [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/>
9. React useState [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react/useState>
10. React useContext [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react/useContext>
11. React useEffect [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react/useEffect>
12. React useRef [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react/useRef>
13. React useState [Електронний ресурс] – Режим доступу до ресурсу:
<https://react.dev/reference/react/useState>
14. Next.js [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/>

		Шаробок І. В.			ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		77

15. Next.js Deploying [Електронний ресурс] – Режим доступу до ресурсу:
<https://nextjs.org/docs/app/getting-started/deploying>
16. Next.js Route Handlers and Middleware [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/app/getting-started/route-handlers-and-middleware>
17. Next.js Image Optimization [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/app/getting-started/images>
18. Next.js Caching and Revalidating [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/app/getting-started/caching-and-revalidating>
19. Vercel – Deployment for Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://vercel.com/>
20. W3Schools [Електронний ресурс] – Режим доступу до ресурсу: <https://www.w3schools.com/>
21. npm – Node Package Manager [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/>
22. PostgreSQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
23. Redis – In-memory data structure store [Електронний ресурс] – Режим доступу до ресурсу: <https://redis.io/>
24. Redis Pub/Sub [Електронний ресурс] – Режим доступу до ресурсу: <https://redis.io/docs/latest/develop/interact/pubsub/>
25. Redis Insight [Електронний ресурс] – Режим доступу до ресурсу: <https://redis.io/insight>
26. Socket.IO [Електронний ресурс] – Режим доступу до ресурсу: <https://socket.io/>
27. MDN Web Docs – WebSockets [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
28. GitHub [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/>

29. GitLab [Електронний ресурс] – Режим доступу до ресурсу:
<https://gitlab.com/>

30. NextAuth.js [Електронний ресурс] – Режим доступу до ресурсу:
<https://next-auth.js.org/>

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		79

ДОДАТКИ

ДОДАТОК А

Схема бази даних в prisma

```
generator client {
  provider      = "prisma-client-js"
  previewFeatures = ["fullTextSearchPostgres"]
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id          Int          @id @default(autoincrement())
  email       String       @unique
  name        String
  password    String?
  avatar      String?
  Review      Review[]
  comics      Comic[]
  ComicList   ComicList[]
  LastChapter LastChapter[]
  Rating      Rating[]
  ChapterComment ChapterComment[]
  ChapterLike ChapterLike[]
}

model Comic {
  id          Int          @id @default(autoincrement())
  title       String
  description  String
  author      String
  cover       String
  releaseYear Int
  type        ComicType   @default(manga)
  views       Int          @default(0)
  age         Int          @default(0)
  status      ComicStatus @default(continues)
  ownerId     Int          @default(3)
  chapters    Chapter[]
  reviews     Review[]
  tracks       Track[]
  User        User         @relation(fields: [ownerId], references: [id])
  ComicList   ComicList[]
  LastChapter LastChapter[]
  Rating      Rating[]
}

model Chapter {
  id          Int          @id @default(autoincrement())
  title       String
  volumeNumber Int
  chapterNumber Int
  comicId     Int
  Comic       Comic        @relation(fields: [comicId], references: [id])
  pages       Page[]
  LastChapter LastChapter[]
}
```

```

ChapterComment ChapterComment[]
ChapterLike ChapterLike[]

@@unique([volumeNumber, chapterNumber, comicId])
@@unique([chapterNumber, comicId])
}

model Page {
  id Int @id @default(autoincrement())
  image String
  number Int
  chapterId Int
  Chapter Chapter @relation(fields: [chapterId], references: [id], onDelete: Cascade)
}

model Review {
  id Int @id @default(autoincrement())
  timestamp DateTime
  content String
  comicId Int
  userId Int
  Comic Comic @relation(fields: [comicId], references: [id])
  User User @relation(fields: [userId], references: [id])
}

model LastChapter {
  id Int @id @default(autoincrement())
  userId Int
  comicId Int
  chapterId Int
  Chapter Chapter @relation(fields: [chapterId], references: [id], onDelete: Cascade)
  Comic Comic @relation(fields: [comicId], references: [id])
  User User @relation(fields: [userId], references: [id])

  @@unique([userId, comicId])
}

model Track {
  id Int @id @default(autoincrement())
  title String
  src String
  likes Int @default(0)
  comicId Int
  Comic Comic @relation(fields: [comicId], references: [id])
}

model ComicList {
  id Int @id @default(autoincrement())
  userId Int
  comicId Int
  type ListType
  User User @relation(fields: [userId], references: [id])
  Comic Comic @relation(fields: [comicId], references: [id])

  @@unique([userId, comicId])
}

model Rating {
  id Int @id @default(autoincrement())
  userId Int
  comicId Int

```

```

rating Int
User User @relation(fields: [userId], references: [id])
Comic Comic @relation(fields: [comicId], references: [id])

@@unique([userId, comicId])
}

model ChapterComment {
  id Int @id @default(autoincrement())
  userId Int
  chapterId Int
  content String
  timestamp DateTime @default(now())
  User User @relation(fields: [userId], references: [id])
  Chapter Chapter @relation(fields: [chapterId], references: [id], onDelete: Cascade)
}

model ChapterLike {
  id Int @id @default(autoincrement())
  userId Int
  chapterId Int
  User User @relation(fields: [userId], references: [id])
  Chapter Chapter @relation(fields: [chapterId], references: [id], onDelete: Cascade)

  @@unique([userId, chapterId])
}

enum ComicType {
  novel
  manhwa
  manga
}

enum ComicStatus {
  continues
  completed
  ann
}

enum ListType {
  reading
  plan
  drop
  read
  favourite
}

```

ДОДАТОК Б

Отримання інформації про комікс та пов'язаних даних

```

import prisma from '@lib/prisma'
import ComicPage from '@components/ComicPage/ComicPage';
import {auth} from '@auth';

export default async function Page({params}) {
  const session = await auth()
  const id = Number((await params).id)
  const comic = await prisma.comic.findUnique({
    where: {
      id: id
    },
  },

```

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

include: {
  chapters: {
    orderBy: {
      chapterNumber: 'asc'
    },
    include: {
      ChapterLike: {
        select: {
          userId: true
        }
      }
    }
  },
  reviews: {
    include: {
      User: {
        select: {
          name: true,
        }
      }
    }
  },
  ComicList: {
    where: {
      userId: session?.user.id,
    },
    select: {
      type: true
    }
  },
  ...(session && {
    LastChapter: {
      where: {
        userId: session?.user.id || null,
      },
      include: {
        Chapter: {
          select: {
            chapterNumber: true
          }
        }
      }
    },
    Rating: {
      where: {
        userId: session?.user.id,
      }
    }
  })
}
})

const ratings = await prisma.rating.findMany({
  where: {
    comicId: id
  }
})

if (ratings.length > 0) {
  comic.averageRating = ratings.reduce((sum, rating) => sum + rating.rating, 0) / ratings.length
  comic.averageRating = comic.averageRating.toFixed(1)
}

```

```

    }

    await prisma.comic.update({
      where: {
        id: id
      },
      data: {
        views: {
          increment: 1
        }
      }
    })
    console.log(comic)

    return <ComicPage comic={comic} session={session}/>
  }
}

```

Отримання коміксів за пошуком, фільтрацією, сортуванням

```

import React from 'react';
import CatalogPage from "@components/CatalogPage/CatalogPage";
import prisma from "@lib/prisma";

const Page = async ({searchParams}) => {
  const params = await searchParams;
  console.log(params);

  let comics = await prisma.comic.findMany({
    where: {
      title: {
        contains: params.q,
        mode: 'insensitive'
      },
      type: Array.isArray(params.type) ? {
        in: params.type
      } : params.type,
      status: Array.isArray(params.status) ? {
        in: params.status
      } : params.status,
      releaseYear: {
        ...(params.from_year && {gte: Number(params.from_year)}),
        ...(params.to_year && {lte: Number(params.to_year)}),
      },
      age: {
        ...(params.age ? Array.isArray(params.age) ? {
          in: params.age.map(Number)
        } : {equals: Number(params.age)} : null)
      }
    },
    orderBy: {
      [params.sort ?? 'views']: params.sort === 'chapters' ? {
        count: 'desc'
      } : 'desc'
    },
    include: {
      chapters: {
        select: {
          _count: {
            select: {
              ChapterLike: true
            }
          }
        }
      }
    }
  });
}

```

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
  },
  _count: {
    select: {
      ComicList: true
    }
  }
}
});

if (params.from_ch) {
  comics = comics.filter(comic => {
    const count = comic.chapters.length;
    return count >= params.from_ch
  })
}
if (params.to_ch) {
  comics = comics.filter(comic => {
    const count = comic.chapters.length;
    return count <= params.to_ch
  })
}

comics = comics.map(comic => {
  const likes = comic.chapters.reduce(
    (sum, chapter) => sum + chapter._count.ChapterLike,
    0
  );
  return {...comic, likes};
});

return (
  <CatalogPage comics={comics}/>
);
};

export default Page;

```

ДОДАТОК В

Арі для димічного доступу до локального файлового середовища

```

import fs from 'fs';
import { NextResponse } from 'next/server';
import path from "path";

export async function GET(req, { params }) {
  const dir = (await params).dir.join("");
  if (!dir) {
    return new NextResponse(null, { status: 500 });
  }
  // Prevent path traversal attacks
  if (dir.indexOf('.') >= 0) {
    return new NextResponse(null, { status: 400 });
  }

  let isAudio = false;
  const ext = path.extname(dir).toLowerCase();
  const mimeTypes = {
    '.mp3': 'audio/mpeg',

```

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    '.wav': 'audio/wav',
    '.ogg': 'audio/ogg',
  };
  if(mimeTypes[ext]) {
    isAudio = true;
  }
  if(isAudio) {
    try {
      const data = fs.readFileSync('public/' + dir);
      const contentType = mimeTypes[ext]

      return new NextResponse(data, {
        status: 206,
        headers: {
          'Content-Type': contentType,
          'Content-Length': data.length.toString(),
          'Cache-Control': 'public, max-age=31536000',
          'Content-Range': `bytes ${0}-${data.length-1}/${data.length}`
        },
      });
    } catch (error) {
      return new NextResponse(null, { status: 500 });
    }
  }

  try {
    // Read and serve the file
    const data = fs.readFileSync('public/' + dir);

    return new NextResponse(data);
  } catch (error) {
    return new NextResponse(null, { status: 500 });
  }
}

```

ДОДАТОК Г

Компонент ProgressBar

```

import React, {useEffect, useRef} from 'react';
import styles from './ProgressBar.module.css'

const ProgressBar = ({value, inSeek, outSeek, onSeek, canSeeking}) => {
  const [currentValue, setCurrentValue] = React.useState(value);
  const isSeeking = useRef(false);
  const barRef = useRef(null)

  useEffect(() => {
    setCurrentValue(value);
  }, [value]);

  useEffect(() => {
    const handleMouseDown = (e) => {
      if(!canSeeking) return;
      isSeeking.current = true;
      const value = setValueBySeek(e);
      inSeek && inSeek(value);
    };

    const handleMouseUp = (e) => {

```

		Шаробоків І. В.			ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (!isSeeking.current) return;
    isSeeking.current = false;
    outSeek && outSeek();
  };

  const handleMouseMove = (e) => {
    if (!isSeeking.current) return;
    const value = setValueBySeek(e);
    onSeek && onSeek(value);
  };

  barRef.current.addEventListener('mousedown', handleMouseDown);
  document.addEventListener('mouseup', handleMouseUp);
  document.addEventListener('mousemove', handleMouseMove);

  return () => {
    barRef.current?.removeEventListener('mousedown', handleMouseDown);
    document.removeEventListener('mouseup', handleMouseUp);
    document.removeEventListener('mousemove', handleMouseMove);
  }
}, [canSeeking])

function setValueBySeek(e) {
  const rect = barRef.current.getBoundingClientRect();
  const value = (e.clientX - rect.left) / rect.width;
  const clampedValue = Math.max(0, Math.min(1, value));
  setCurrentValue(clampedValue);
  return clampedValue;
}

return (
  <div className={styles.bar} ref={barRef}>
    <div className={styles.track}>
      <div className={styles.fill} style={{transform: `scaleX(${currentValue})`}}/>
    </div>
    <div className={styles.thumb}
      style={{transform: `translateX(${currentValue * barRef.current?.clientWidth}px)`}}/>
    </div>
  </div>
);
};

export default ProgressBar;

```

Компонент Player

```

import React, {useEffect, useRef, useState} from 'react';
import styles from './Player.module.css';
import {FaCirclePause, FaCirclePlay} from "react-icons/fa6";
import ProgressBar from "@components/Form/ProgressBar";
import {useAudio} from "@context/AudioContext";
import '@app/globals.css'

export function timeToString(time) {
  if (!time) return '0:00';
  return Math.trunc(time / 60) + ':' + String(Math.trunc(time % 60)).padStart(2, '0')
}

const Player = () => {
  const {audioRef, isPlay, togglePlay, progress, setTrack, title, src} = useAudio();

  useEffect(() => {

```

		Шаробок І. В.			ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return () => {
      setTrack()
    }
  }, [])

function setCurrentTimeByProgress(value) {
  let currentTime = value * audioRef.current.duration
  if(!currentTime) currentTime = 0;
  audioRef.current.currentTime = currentTime;
}

const timerCurrent = timeToString(Math.round(audioRef.current.currentTime))
const timerRemain = timeToString(Math.round(audioRef.current.duration) -
Math.round(audioRef.current.currentTime))

return (
  <div className={styles.player}>
    <div style={{
      display: "flex",
      gap: 5,
      alignItems: "center",
      justifyContent: "flex-start",
      width: '100%',
      minHeight: '1em',
      lineHeight: '1em'
    }}>
      {src && <>
        <div className={styles.name}>{title}</div>
        <div>—</div>
        <div className={styles.author}>One Piece OST</div>
      </>
      {
        !src && <div style={{color: 'lightgray'}}>
          Оберіть трек для відтворення
        </div>
      }
    </div>
    <div className={styles.barWrapper}>
      <div className={styles.timer}>
        {timerCurrent}
      </div>
      <ProgressBar value={src ? progress : 0}
        onSeek={(value) => {
          setCurrentTimeByProgress(value)
        }}
        inSeek={(value) => {
          setCurrentTimeByProgress(value)
          audioRef.current.pause();
        }}
        outSeek={() => {
          audioRef.current.play();
        }}
        canSeeking={!src}
      />
      <div className={styles.timer}>
        -{timerRemain}
      </div>
    </div>
    <div onClick={togglePlay} className={'icon'}>
      {isPlay ? <FaCirclePause size={45}/> : <FaCirclePlay size={45}/>}
    </div>
  )

```

```

    </div>
  );
};

export default Player;

```

Реалізація автоматичного прокручування сторінки

```

import React, {useEffect, useState} from 'react';
import styles from "@components/ReadPage/ReadPage.module.css";
import {FaArrowAltCircleDown} from "react-icons/fa";
import {useAudio} from "@context/AudioContext";
import {useTheme} from "@context/ThemeContext";

const AutoScroll = () => {
  const {isPlay, togglePlay} = useAudio();
  const [lastPlay, setLastPlay] = useState(isPlay);

  const [autoScroll, setAutoScroll] = useState(false);
  const {theme, setTheme} = useTheme();

  useEffect(() => {
    if (isPlay && !autoScroll) {
      togglePlay();
      setLastPlay(false);
    } else if (lastPlay === isPlay) {
      togglePlay()
      setLastPlay(true);
    }
    if(!autoScroll) return;
    const intervalId = setInterval(() => {
      window.scrollTo({
        top: window.innerHeight / 3,
        behavior: "smooth"
      })
      if(window.innerHeight + window.scrollY >= document.body.scrollHeight)
        setAutoScroll(false);
    }, 2000)
    return () => {
      clearInterval(intervalId);
    }
  }, [autoScroll])
  return (
    <div className={styles.autoScroll} style={{outline: autoScroll ? '1px solid deeppink' : '1px solid grey'}}>
      <div className={styles.icon}>
        <FaArrowAltCircleDown size={50} onClick={() => setAutoScroll(val => !val)}
          color={theme === 'dark' ? 'white' : 'black'}/>
      </div>
    </div>
  );
};

export default AutoScroll;

```

Форма для завантаження треків

```

'use client'
import UploadBox from "@components/UploadBox/UploadBox";
import React, {useState} from "react";
import Player from "@components/Player/Player";
import styles from "../UploadTrackBox.module.css";
import path from 'path'
import {TrackItem} from "@components/TrackItem/TrackItem";

```

		Шаробоків І. В.			ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const UploadTrackBox = () => {
  const [tracksData, setTracksData] = useState([]);

  const onSetFiles = (files) => {
    Array.from(files).forEach((file, i) => {
      const audio = new Audio();
      audio.preload = 'metadata';
      audio.src = URL.createObjectURL(files[i]);
      audio.onloadedmetadata = () => setTracksData(prev => {
        const updated = [...prev]
        updated[i] = {
          src: audio.src,
          title: path.basename(`${file.name}`, path.extname(file.name))
        }
        return updated
      })
    })
  }

  const onChangeTitle = (currentSrc, title) => {
    setTracksData(prev => {
      const updated = [...prev]
      const target = updated.find(item => item.src === currentSrc)
      target.title = title
      return updated
    })
  }

  return (
    <UploadBox onSetFiles={onSetFiles}>
      <div className={styles.main}>
        <div className={styles.tracks}>
          {!tracksData.some((track) => {return !track}) && tracksData.map((track, i) => {
            console.log(tracksData)
            return <TrackItem key={i} trackSrc={track.src} trackTitle={track.title}
onChangeTitle={onChangeTitle}/>
          )}}
        </div>
        <div className={styles.playerWrapper}>
          <Player/>
        </div>
      </UploadBox>
    )
  );
}

export default UploadTrackBox;

```

ДОДАТОК Г

Аудіо контекст

```

'use client';

import {createContext, useContext, useRef, useState, useEffect} from 'react';

const AudioContext = createContext(null);

export const useAudio = () => {
  const ctx = useContext(AudioContext);
  if (!ctx) throw new Error("useAudio must be used within AudioProvider");
}

```

		Шаробоків І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return ctx;
  }
};

export const AudioProvider = ({children}) => {
  const audioRef = useRef(null);
  const [isPlay, setIsPlay] = useState(false);
  const [src, setSrc] = useState("");
  const [progress, setProgress] = useState(0);
  const [title, setTitle] = useState("");
  const playlistRef = useRef({});
  const [playlist, setPlaylist] = useState({});
  const currentIndex = useRef(0);

  const togglePlay = () => {
    const audio = audioRef.current;
    if (!audio) return;
    if (audio.paused) {
      audio.play().catch(err => console.warn("Play error:", err));
    } else {
      audio.pause();
    }
  };

  const nextTrack = () => {
    currentIndex.current = (currentIndex.current + 1) % playlistRef.current.tracks.length;
    const track = playlistRef.current.tracks[currentIndex.current];
    setTrack(track.src, track.title, false)
  }
  const previousTrack = () => {
    currentIndex.current = (currentIndex.current - 1) % playlistRef.current.tracks.length;
    const track = playlistRef.current.tracks[currentIndex.current];
    setTrack(track.src, track.title, false)
  }

  const setTrack = (src="", title="", clearPlaylist=true) => {
    console.log('track set', src, title, clearPlaylist)
    if(clearPlaylist) setPlaylist({})
    setSrc(src)
    setTitle(title)
  }

  useEffect(() => {
    const audio = audioRef.current;
    if (!audio) return;

    audio.volume = 0.1

    const handlePlay = () => setIsPlay(true);
    const handlePause = () => setIsPlay(false);
    const handleEnded = () => {
      if(playlistRef.current?.tracks)
        nextTrack();
    }

    audio.addEventListener("play", handlePlay);
    audio.addEventListener("pause", handlePause);
    audio.addEventListener("ended", handleEnded);

    return () => {
      audio.removeEventListener("play", handlePlay);
      audio.removeEventListener("pause", handlePause);
    }
  }, [audioRef, playlistRef]);
};

```

```

    };
  }, []);

useEffect(() => {
  const audio = audioRef.current;
  if (!audio) return;
  audio.src = src;
  src && audio.play();
}, [src]);

useEffect((e) => {
  console.log(e)
  playlistRef.current = playlist;
  if (!playlist?.tracks) return;
  currentIndex.current = 0;
  const track = playlist.tracks[0]
  setTrack(track.src, track.title, false)
}, [playlist])

useEffect(() => {
  let animId;
  if (isPlay) animId = requestAnimationFrame(frame);

  function frame() {
    console.log('frame');
    animId = requestAnimationFrame(frame);
    let progress_ = audioRef.current.currentTime / audioRef.current.duration;
    if (isNaN(progress_)) {
      progress_ = 0;
    }
    setProgress(progress_);
  }

  return () => {
    cancelAnimationFrame(animId)
  }
}, [isPlay])

return (
  <AudioContext.Provider
    value={{audioRef, isPlay, togglePlay, src, progress, title, setPlaylist, setTrack, nextTrack, previousTrack,
playlist}}>
    {children}
    <audio ref={audioRef}/>
  </AudioContext.Provider>
);
};

```

Контекст модального вікна

```

'use client';

import {createContext, useContext, useEffect, useState} from "react";
import styles from './Modal.module.css'

const ModalContext = createContext(null);

export const useModal = () => {
  const ctx = useContext(ModalContext);
  if (!ctx) throw new Error("useModal must be used within ModalProvider");
  return ctx;
};

```

		Шаробоків І. В.			ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

export const ModalProvider = ({children}) => {
  const [content, setContent] = useState(null);
  const showModal=(modalContent)=>{
    document.body.style.overflowY="hidden";
    setContent(modalContent)
  }
  const hideModal = ()=> {
    document.body.style.overflowY="scroll";
    setContent(null);
  }

  return (
    <ModalContext.Provider value={{showModal,hideModal}}>
      {content !== null && <div className={styles.modal} onClick={()=>hideModal()}>
        <div onClick={(e)=>e.stopPropagation()}>
          {content}
        </div>
      </div>}
      {children}
    </ModalContext.Provider>
  );
};

```

Контекст кімнати для спільного перегляду

```

'use client';
import React, {createContext, useContext, useEffect, useState} from 'react';
import {usePathname, useRouter, useSearchParams} from "next/navigation";
import {getRoom} from "@app/actions";
import {io} from "socket.io-client";

const RoomContext = createContext(null);

export const useRoom = () => {
  const ctx = useContext(RoomContext);
  if (!ctx) throw new Error("useRoom must be used within RoomProvider");
  return ctx;
};

export const RoomProvider = ({children, session}) => {
  const router = useRouter();
  const searchParams = useSearchParams();
  const pathName = usePathname();

  const [room, setRoom] = useState(null);
  const [socket, setSocket] = useState(io(`https://socket.manganext.pp.ua`, {
    autoConnect: false,
    withCredentials: true,
    auth: session?.user
  })));

  const joinRoom = async (roomId) => {
    const room = await getRoom(roomId);
    if (room) {
      socket.connect()
      socket.emit('join', roomId, () => {
        socket.emit('get_members', (members) => {
          room.members = new Map(members.map(item => [item.id, {name: item.name}]));
          console.log(room.members);
          setRoom(room);
        })
      })
    }
  })
};

```

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    } else setRoom(null)
    return !!room;
  }

  const leaveRoom = () => {
    socket.emit('leave');
    router.push(pathName)
  }

  const emitChapter = (chapter) => {
    console.log('emit')
    socket.emit('set_chapter', chapter);
  }

  const emitScroll = (scroll) => {
    console.log('emit-scroll')
    socket.emit('set_scroll', scroll);
  }

  const emitMessage = (message) => {
    console.log('emit-message')
    socket.emit('add_message', message);
  }

  const emitKick = (userId) => {
    socket.emit('kick', userId);
  }

  useEffect(() => {
    console.log('init it')
    setSocket(socket);
    socket.on('join', (member) => {
      setRoom(room => {
        if (!room || room.members.has(member.id)) return room;
        const updated = new Map(room.members);
        updated.set(member.id, {name: member.name});
        console.log(updated.members);
        return {...room, members: updated};
      })
    })
    socket.on('leave', (id) => {
      setRoom(room => {
        const updated = new Map(room.members);
        updated.delete(id);
        console.log(updated.members);
        return {...room, members: updated};
      })
    })
    socket.on('set_chapter', (chapter) => {
      setRoom(room => {
        return {...room, chapterNumber: chapter}
      })
    })
    socket.on('set_scroll', (scroll) => {
      console.log('get scroll', scroll)
      setRoom(room => {
        return {...room, scroll: scroll}
      })
    })
    socket.on('add_message', (message) => {
      console.log(message)
    })
  })

```

```

    setRoom(room => {
      return {...room, messages: [...room.messages, message]};
    })
  })
  socket.on('kick', () => {
    leaveRoom()
  })

  return () => {
    socket.off('join');
    socket.off('leave');
    socket.off('set_chapter');
    socket.off('set_scroll');
    socket.off('add_message');
    socket.off('kick');
    socket.disconnect()
  }
}, []);

useEffect(() => {
  console.log(searchParams)
  async function handleRoom(roomId) {
    await joinRoom(roomId)
  }

  const roomId = searchParams.get('room')
  roomId ? handleRoom(roomId) : setRoom(null)
}, [searchParams.get('room')])

useEffect(() => {
  console.log()
  room?.chapterNumber !== null && room?.comicId !== null &&
  router.push(`/comics/${room.comicId}/chapters/${room.chapterNumber}?${searchParams}`)
}, [room?.chapterNumber, room?.comicId])

useEffect(() => {
  room?.scroll !== null
  && window.scrollTo({
    top: room.scroll * (document.documentElement.scrollHeight - window.innerHeight),
    left: 0,
    behavior: 'smooth'
  })
}, [room?.scroll])

useEffect(() => {
  console.log(room)
  if (!room) {
    console.log('no room')
    //router.push(pathName)
  }
}, [room])

return (
  <RoomContext.Provider value={{room, emitChapter, emitScroll, emitMessage, emitKick, leaveRoom}}>
    {children}
  </RoomContext.Provider>
);
};

```

Контекст для зміни теми

'use client';

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						95
Змн.	Арк.	№ докум.	Підпис	Дата		

```
import {createContext, useContext, useState} from "react";

const ThemeContext = createContext(null);

export const useTheme = () => {
  const ctx = useContext(ThemeContext);
  if (!ctx) throw new Error("useTheme must be used within ThemeProvider");
  return ctx;
};

export const ThemeProvider = ({children}) => {
  const [theme, setTheme] = useState('dark');

  return (
    <ThemeContext.Provider value={{theme, setTheme}}>
      {children}
    </ThemeContext.Provider>
  );
};
```

ДОДАТОК Д

Socket.io сервер

```
import {Server} from 'socket.io'
import {createClient} from "redis";
import cookie from "cookie"
import {decode} from "@auth/core/jwt"
import crypto from "crypto"

const redis = createClient({
  password: process.env.REDIS_PW,
  socket: {
    host: process.env.REDIS_HOST,
    port: process.env.REDIS_PORT
  }
})
redis.on('error', err => console.log('Redis Client Error', err));
await redis.connect()

const io = new Server({
  cors: {
    origin: ['http://localhost:3000', 'http://192.168.0.101:3000', 'https://manganext.pp.ua'],
    credentials: true
  }
});

// FOR COOKIE AUTH

// io.use(async (socket, next) => {
//   console.log('socket auth')
//   try {
//     const token = (cookie.parse(socket.handshake.headers.cookie))['authjs.session-token'];
//     console.log(token)
//     const decoded = await decode({
//       token: token,
//       secret: process.env.AUTH_SECRET,
//       salt: 'authjs.session-token'
//     })
//   }
//   console.log('socket connecting..', socket.id)
//   socket.data = {id: decoded.id, name: decoded.name}
```

		Шаробоків І. В.			ПІЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

//   } catch (e) {
//     const id = crypto.randomBytes(4).toString('hex')
//     socket.data = {id: id, name: 'Passerby'}
//   }
//   next()
// })

// FOR AUTH FIELD AUTH

io.use(async (socket, next) => {
  if(socket.handshake.auth.id) {
    socket.data = socket.handshake.auth
  } else {
    const id = crypto.randomBytes(4).toString('hex')
    socket.data = {id: id, name: 'Passerby'}
  }
  next()
})

io.on('connection', async (socket) => {
  socket.on('join', async (roomId, cb) => {
    const exists = await redis.exists(`rooms:${roomId}`);
    if (!exists) return;
    const rooms = Array.from(socket.rooms)
    for (const room of rooms.slice(1)) {
      socket.leave(room)
      io.to(room).emit('leave', socket.data.id);
    }

    socket.join(roomId);
    socket.to(roomId).emit('join', socket.data);
    cb()
  })
  socket.on('leave', () => {
    const rooms = Array.from(socket.rooms)
    for (const room of rooms.slice(1)) {
      socket.leave(room)
      io.to(room).emit('leave', socket.data.id);
    }
  })
  socket.on('get_members', async (cb) => {
    const room = Array.from(socket.rooms)[1]
    const sockets = await io.in(room).fetchSockets()
    const members = sockets.map(socket => socket.data)
    const uniqueMembers = Array.from(
      new Map(members.map(item => [item.id, item])).values()
    );
    console.log(uniqueMembers);
    cb(uniqueMembers)
  })

  socket.on('set_chapter', async (chapter) => {
    const room = Array.from(socket.rooms)[1]
    const host = await redis.hGet(`rooms:${room}`, 'host')
    if (Number(host) !== socket.data.id) return;
    console.log('set chapter')
    socket.to(room).emit('set_chapter', chapter);
    void redis.hSet(`rooms:${room}`, 'chapterNumber', chapter)
  })
  socket.on('set_scroll', async (scroll) => {

```

		Шаробок І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const room = Array.from(socket.rooms)[1]
const host = await redis.hGet(`rooms:${room}`, 'host')
if (Number(host) !== socket.data.id) return;
socket.to(room).emit('set_scroll', scroll);
void redis.hSet(`rooms:${room}`, 'scroll', scroll)
})

socket.on('kick', async(userId)=> {
const room = Array.from(socket.rooms)[1]
const host = await redis.hGet(`rooms:${room}`, 'host')
if (Number(host) !== socket.data.id) return;

const sockets = await io.in(room).fetchSockets()
const kickSocket = sockets.find((socket) => socket.data.id === userId);
console.log(kickSocket.data.id)
kickSocket.emit('kick');
})

socket.on('add_message', async (message) => {
const room = Array.from(socket.rooms)[1]
io.in(room).emit('add_message', {name: socket.data.name, message});
await redis.xAdd(`rooms:${room}:messages`, '*', {name: socket.data.name, message: message})
})

socket.on('disconnecting', () => {
console.log(socket.data)
const room = Array.from(socket.rooms)[1]
io.to(room).emit('leave', socket.data.id);
})
})
io.listen(3001)

```

ДОДАТОК Е

Таблиця 1

Структура бази даних

Модель	Поля	Зв'язки	Опис
User	name, email, avatar	Review	Дані про користувачів
Comic	title, description, author, cover, releaseYear, type, views, age, status, chapters, reviews	Chapter, Review	Дані про комікси

		Шаробоків І. В.			ПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		98

Chapter	Title, volumeNumber, chapterNumber, comicId	Comic, LastRead	Окремий розділ у коміксі
Page	image, number, chapterId	Chapter	Сторінка розділу
Review	timestamp, content, comicId, userId	Comic, User	Відгук користу- вача про комікс
LastChapter	userId, chapterId, timestamp	User, Chapter	Останній прочи- таний розділ
Track	title, src, likes, comicId	Comic	Аудіотрек до ко- міксу
ComicList	userId, comicId, type	User, Comic	Збереження комі- ксу в списку
Rating	userId, comicId, rating	User, Comic	Рейтинг коміксу
ChapterComment	userId, chapterId, content, typestamp	User, Chapter	Коментарій до розділу
ChapterLike	userId, chapterId	User, Chapter	Вподобання до розділу