МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проект

здобувача освіти денної форми навчання
РП.08.16.000.ДП

ПАНКА
КИРИЛА ДМИТРОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: **«Розробка програмного забезпечення»**

Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме-контенту

Проектний матеріал складається з пояснювальної записки на 76 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Панка К.Д.)
Керівник _____ (Жадан А.С.)

Консультанти:
з економічного розділу _____ (Канський М.Ю.)
з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)
з нормоконтролю _____ (Петрашова В.І.)
старший консультант _____ (Кривченко Ю.В.)

До захисту допущений
Голова циклової комісії _____ (Кривченко Ю.В.)
Завідувач відділення _____ (Краснокутська К.Г.)

Захист «21» 06 2025 р. Протокол ЕК № 1
Оцінка ЕК 5/90

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Панка Кирилу Дмитровичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме- контенту.*

затверджена наказом по коледжу від "*14*" *листопада* 2024р. № *246*

2. Термін здачі закінченого проекту

3. Вихідні данні до проекту

1. Передбачити GUI у стилі Flat;

2. Застосовувати прогресивний фреймворк Next.js;

3. Впровадити технології WebSocket;

4. Реалізувати пошук за допомогою штучного інтелекту.

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

1. Огляд предметної області; 2. Вибір та обґрунтування технологій; 3. Проєктування дизайну веб-застосунку; 4. Проєктування архітектури веб-застосунку; 5. Розробка серверного веб-застосунку; 6. Розробка клієнтського веб-застосунку; 7. Економічний розрахунок; 8. Аспекти охорони праці та техніки безпеки.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Титул; Переваги проєкту; Вибір технологій; Клієнтська частина; Реалізація хука usePlayer; Алгоритм пошуку; Алгоритм сумісного перегляду; Алгоритм реєстрації та авторизації користувача через Zustand; UI/ UX дизайн (дизайн мапа); UI/ UX дизайн (мобільний вигляд); UI/ UX дизайн (ПК вигляд); Висновки.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Жадан А.С.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 12.05.2025

Керівник *Жадан А.С.* (підпис)

Завдання прийняв до виконання *Панка К.Д.* (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Формування вступу*	15.05.2025	Виконано
2	*Дослідження предметної області*	16.05.2025	Виконано
3	*Огляд аналогів*	19.05.2025	Виконано
4	*Вибір технічної літератури*	20.05.2025	Виконано
5	*Аналіз технологій розробки*	21.05.2025	Виконано
6	*Проєктування веб-дизайну веб-застосунку*	22.05.2025	Виконано
7	*Проєктування архітектури веб-застосунку*	27.05.2025	Виконано
8	*Розробка клієнтського веб-застосунку*	02.06 2025	Виконано
9	*Тестування створеного веб-застосунку*	06.06.2025	Виконано
10	*Оформлення пояснювальної записки*	11.06.2025	Виконано
11	*Підготовка графічних матеріалів*	13.06.2025	Виконано
12	*Економічний розрахунок*	13.06.2025	Виконано
13	*Опис аспектів охорони праці та техніки безпеки*	13.06.2025	Виконано
14	*Підведення висновків*	13.06.2025	Виконано
15	*Підготовка доповіді для захисту*	16.06.2025	Виконано

Дипломник (підпис)

Керівник (підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 16 000. 00 ДП	Дипломний проект		
А4			РП. 08. 16 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 01 000. 00 ДП		
Розробив		Панка К.Д.		6.06.25	Розробка веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме-контенту	Літ.	Аркуш 4 / Аркушів 76
Перевірив		Жадан А.С.		6.06.25		Н Д П	
Н. контр.		Петрашова В.І.		6.06.25		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.					

ЗМІСТ

ВСТУП

Аніме-контент стає дедалі популярнішим серед користувачів різного віку, що підвищує попит на зручні інструменти для його пошуку, перегляду та спільного дозвілля. Більшість платформ не забезпечують гнучкого підбору аніме чи функцій синхронного перегляду. Тому розробка веб-застосунку, що поєднує інтелектуальний пошук і спільний перегляд, є актуальною та відповідає сучасним потребам.

Метою є створення веб-застосунку, який забезпечує користувачам зручний інтерфейс для пошуку аніме-контенту за допомогою інтелектуальних фільтрів та можливість переглядати відео разом із друзями в режимі реального часу.

У розробці застосовано системний підхід, який передбачає поетапний аналіз, проєктування та реалізацію програмного забезпечення. Також використовуються принципи компонентної розробки, клієнт-серверної архітектури та модульного програмування.

Для реалізації веб-застосунку було обрано сучасний стек технологій, який забезпечує ефективну, надійну та масштабовану розробку. Основу клієнтської частини становить фреймворк Next.js, що дозволяє реалізувати зручний інтерфейс та гнучку маршрутизацію. Серверна логіка побудована з використанням Node.js, що забезпечує високу продуктивність і швидкодію. Мова TypeScript використовується для підвищення надійності коду та зручності супроводу проєкту. Для реалізації функціоналу спільного перегляду в реальному часі застосовується бібліотека Socket.IO, яка забезпечує двостороннє з’єднання між клієнтами. Зберігання даних про користувачів та аніме-контент організоване за допомогою MongoDB – гнучкої документно-орієнтованої бази даних.

Розроблений веб-застосунок призначений для широкого кола користувачів, які цікавляться аніме. Він дозволяє ефективно шукати контент відповідно до власних уподобань і ділитися переглядом з іншими в режимі реального часу, що сприяє покращенню користувацького досвіду та соціальній взаємодії в онлайн-середовищі.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд предметної області

1.1.1 Сучасні тенденції та актуальність інтелектуального пошуку в веб-застосунках

Інтернет налічує велику кількість контенту різного формату, серед якого дедалі важче орієнтуватися та знаходити потрібну інформацію. Це, безспосередньо, зумовлює потребу на вдосконалення алгоритмів пошуку які можуть допомогти зорієнтувати користвачів на потрібний їм матераіл не тільки з чітким і конкретно сформованим запитом, а також і з нечітким, розпливчастим запитом. З цим завданням на поточний момент часу справляються механізми інтелектуального пошуку.

Інтелектуальний пошук являє собою технологічний підхід, який поєднує класичні методи фільтрації та аналіз пошукового запиту. Він дозволяє користувачу зручно та швидко знайти те, що йому цікаво. Основною метою є надання користувачу конкретної або близької відповіді на точний або розпливчатий запит.

У межах цього дипломного проєкту інтелектуальний пошук використовується для того, щоб користувач міг знаходити аніме не лише за назвою чи жанром, а й за коротким описом чи змістом. Це особливо корисно, коли людина не пам’ятає точну назву, але пригадує, про що йшлося. Такий підхід робить пошук більш гнучким і природним – користувач просто вводить те, що пам’ятає, а застосунок допомагає знайти найбільш схожі серіали. Це значно покращує досвід користування і дозволяє відкривати новий контент навіть за мінімальними підказками.

1.1.2 Огляд існуючих рішень для спільного онлайн-перегляду

На поточний час вже існють сервіси, які дозволяють користувачам перегладати відео у режимі реального часу. Існує два способи подивитись відео разом: стріминг екрану одного з користувачів або додаток, який в реальному часі може керувати прогрувачем. Обидва варіанти мають як свої плюси та і свої мінуси.

Стрімінг–найпростіший спосіб передачі зображення багатьом користувачам.

Потрібний результат, а саме сумісний перегляд, досягається даним способом дуже просто, достатньо знайти сервіс, в якому є можливість підключення до голосового каналу та можливість трансляції зображення. Як приклад такого сервісу Discord, застосунок для гравців який дозволяє створювати кімнати для спілкування, або Teamspeak, сутність якого така сама.

Плюсом даного методу являється його простота.

Із мінусів стрімінгу можна виділити низьку якість потоку, достатньо часто бітрейт падає і зображення виглядає неякісно, або має велику затримку, а також, для сумісного перегляду не вистачає зручного керування медіа, а саме поставити відео на паузу, налаштувати програвач під себе.

Другим способом являються додатки або застосунки які розраховані на сумісний перегляд. Прикладом таких є Watch2Gether та Teleparty від Netflix. Їх суть полягає у тому, що створюється веб пространство, як кімната, в якому користувачам надається програвач і в цьому програвачі всі бачать відео.

Плюсами даного методу являються: можливість налаштувати програвач, керувати медіа може кожен, хто дивиться.

Мінусами даного методу є те, що ці портали зазвичай розраховані на певні джерела самого медія, наприклад Teleparty працює не на всіх сервісах, а тільки на тих, які офіційно підтримуються, такі як Netfilx, HBO, Prime Video.

1.1.3 Вимоги до функціональності та зручності використання

Для того що веб-застосунок був приємним і зручним для використання він має відповідати певним вимогам – як технічним, так і естетичним. На сьогоднішній день не достатньо просто гарного застосунку або просто швидкого та зручного застосунку, треба щоб він був приємним як у використанні, так і на вигляд. Чи не малу увагу також обов’язково треба приділити доступності застосунку, щоб кожен користувач міг легко пересуватися у застосунку та не заплутатися.

Для цього важливо враховувати сучасні стандарти веб-дизайну та рекомендації з юзабіліті. Зручний інтерфейс має бути інтуїтивно зрозумілим і передбачати логічну структуру навігації. Також слід забезпечити адаптивність застосунку для комфортного використання на різних пристроях і розмірах екранів.

Рисунок 1.1. Вимоги до функціональності та зручності використання

1.2 Вибір та обґрунтування технологій

1.2.1 Організація архітектури застосунку на базі Next.js

Для розробки клієнтської частини застосунку було обрано фреймворк Next.js, який поєднує постоту роботи React.js [1] із гнучкою маршрутизацією, можливостями серверного рендерингу сторінки, а також підтримкою серверних дій. Однією з ключових переваг Next.js є його модульність і постійна еволюція, саме тому в проєкті було використано структуру App Router, яка є рекомендованим підходом починаючи з тринадцятої версії фреймворку.

App Router являє собою нову систему маршрутизації, яка прийшла на заміну попередньому Page Router, та привнесла велику кількість змін у DX та UX. Основою App Router є директорія app/, яка описує маршрутизацію через вкладені теки. На відміну від Page Router у App Router кожна тека складає певний маршрут, в той час як у Page Router маршрутом був виконавчий файл, це дало змогу зробити більш глибоку конструкцію для будування вкладених шляхів у застосунку. В середині теки маршруту має знаходитись файл page.tsx який являє собою сторінку, та являється точкою входу, що символізує те, що тека є сторінкою. Окрім page.ts у теці також можуть знаходитись файли layout.tsx – для указання розмітки сторінки, loading.tsx – для того щоб вказати що буде відображено на етапі завантаження сторінки, error.tsx для випадку якщо на сторінці станеться помилка та not-found.tsx щоб вказати на базову помилку 404. Це дозволяє гнучко керувати рендерингом

інтерфейсу на різних рівнях маршруту. Крім того, App Router дозволяє будувати динамічні та паралельні маршрути.

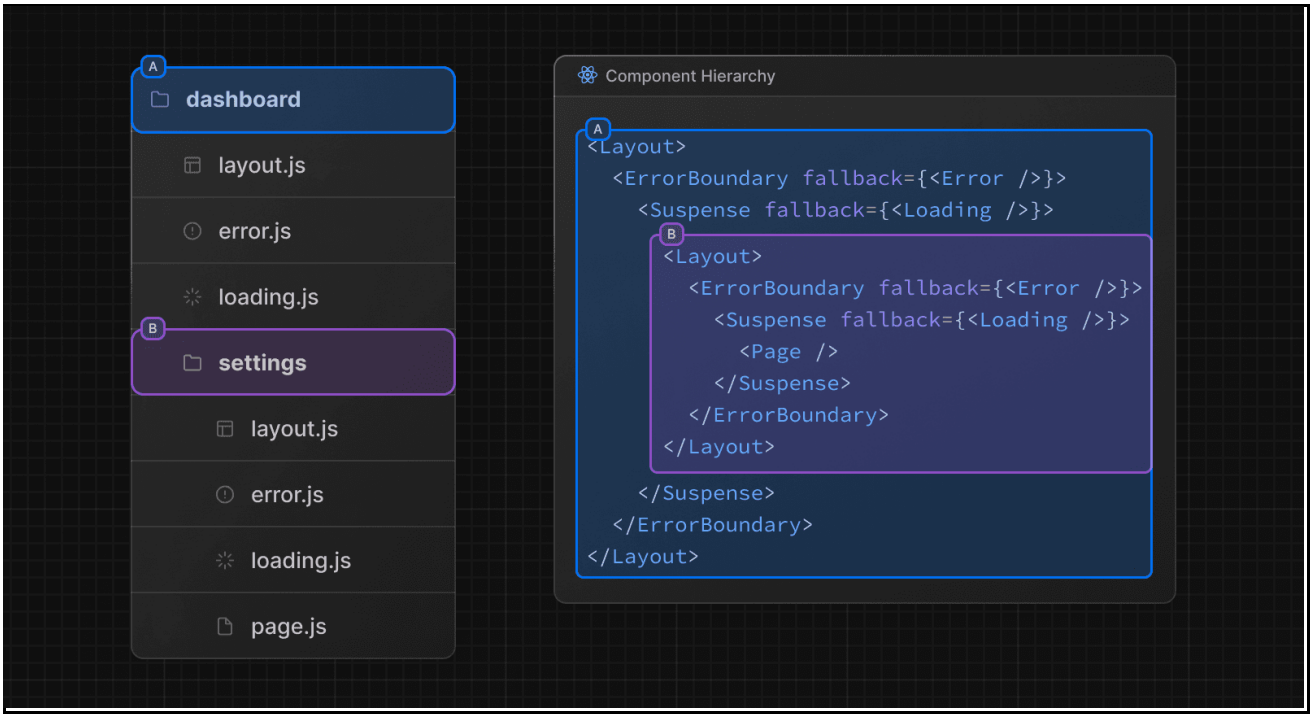


Рис. 1.2. Приклад базової App Router структури

Динамічні маршрути, це ті маршрути, які можуть змінюватися динамічно. Так, для того щоб показати, наприклад, сторінку з аніме не потрібно писати параметри запиту як /anime?anime=назва_аніме, достатньо створити динамічний маршрут (Рис. 1.2.) та для того щоб викликати сторінку написати /anime/назва_аніме. У даний спосіб можна уникнути надлишкової логіки з вилученням назви та зайвими властивостями у запиті.

app/blog/[slug]/page.tsx TypeScript

```tsx
export default async function Page({
  params,
}: {
  params: Promise<{ slug: string }>
}) {
  const { slug } = await params
  return <div>My Post: {slug}</div>
}
```

Рисунок 1.3. Приклад вмісту динамічного руту

Паралельним маршрутом [3] являється можливість одночасно показувати декілька незалежних частин документа у рамках однієї сторінки. Такий підхід дозволяє зробити гнучку навігацію, в якій сторінка може бути як частиною іншої сторінки, так і повноцінною сторінкою. За допомогою паралельних маршрутів можна зробити сторінку, яка буде мати власний маршрут та можливість відкриватися додатково на викликаючій сторінці, не змінюючи поточну навігацію. Так, за допомогою цього методу можна зробити вікно авторизації як модальним, так і окремою сторінкою. Для цього у директорії app/ треба створити теку, яка буде починатися з символу «@» (для прикладу назвемо теку @modal). В середині теки визначаємо шлях, за яким буде визначатися модальне вікно, зазвичай його називають так само, як і шлях до повноцінної сторінки, але до назви теки додають префікс «(.)», для прикладу зробимо модальне вікно для авторизації тому назвемо теку «(.)login». Після виконання цих при переході на /login в залежності від типу переходу ми будемо бачити модальне вікно або повну сторінку створеного шляху.

Клієнтська частина застосунку побудована за принципом компонетної архітектури, що є звичною практикою у сучасних фронтенд-розробках, особливо при роботі з React. Це означає, що інтерфейс розбитий на окремі частини – компоненти, кожен із яких відповідає за свій шматок функціональності: відображення кнопки, поля введення, цілого чату чи відеоплеєра.

Такий підхід зручний тим, що дозволяє працювати над невеликими частинами інтерфейсу окремо, не ламаючи інші. Якщо потрібно змінити вигляд однієї кнопки – немає потреби переглядати весь код програми. До того ж, один і той самий компонент можна використовувати в різних місцях проєкту або навіть у зовсім інших застосунках, що добре економить час.

Ще одна перевага – простота масштабування. Коли проєкт росте, додавання нового функціоналу зводиться до створення нових компонентів або розширення існуючих. Це дозволяє розвивати програму поступово, не переписуючи її з нуля.

Компоненти також добре піддаються тестуванню. Можна окремо перевірити, як працює конкретний блок без необхідності запускати всю програму – це значно спрощує пошук і виправлення помилок.

Крім того, така архітектура добре підходить для командної роботи. Кілька людей можуть одночасно працювати над різними компонентами без конфліктів, що пришвидшує розробку й робить її ефективнішою.

У випадку з React це не просто рекомендований підхід – це сам принцип, на якому побудовано фреймворк. Компонентна архітектура дає змогу створювати динамічні, адаптивні інтерфейси, які легко підтримувати й оновлювати у майбутньому. Власне, саме завдяки цьому React так активно використовується в сучасній веб-розробці.

Підтримка типів реалізована за допомогою TypeScript [2], а всі типи винесені в окрему теку @types, що допомагає уникати дублювання та зменшує ризик помилок при взаємодії між клієнтом і сервером.

Було реалізовано окремий сервер, який підключає Socket.io до того ж порту, що дозволяє використовувати сервер Next.js одночасно з Websocket, також реалізовано деякі хуки, що реалізують бізнес-логіку взаємодії з плеєром, фільтрами, пошуком тощо.

1.2.2 Огляд способів керування станом у React.

У React стан являє собою набір даних, які впливають на вигляд і поведінку компонентів. Коли додаток складається з кількох незалежних частин, які мають спільну інформацію (наприклад, дані користувача, налаштування, статус плеєра), з’являється потреба у централізованому способі зберігання і обміну цими даними. І саме для цього використовуються стейт-менеджери – інструменти, які дозволяють створити єдине джерело правди, з якого компоненти можуть отримувати актуальний стан і змінювати його.

У цьому застосунку немає великої кількості глобального стану, тому складні рішення тут були б зайвими. Основне, для чого використовується стейт-менеджер – це збереження профілю користувача після авторизації. Замість того щоб кожного разу запитувати сервер про поточного користувача, його дані один раз зберігаються у стані, після чого доступні з будь-якого місця в застосунку.

Наприклад, сторінки, які потребують авторизації, перевіряють, чи є в стані користувач. Компоненти на кшталт аватара, меню або рекомендацій також можуть

отримувати цю інформацію прямо зі стору, без зайвих запитів. Це дозволяє пришвидшити взаємодію та зменшити навантаження на сервер.

Чому не Redux?

Redux являється добре знайомим багатьом розробникам, він має велику екосистему, документацію та інструменти для налагодження. Але водночас Redux потребує надчіткої структури, що не підходить для простої задачі, як збереження профілю користувача. Redux вимагає гнучкого налаштування, навіть якщо стану в застосунку – всього кілька полів.

Zustand виявився простішим, швидшим у впровадженні та більш сучасним. Його API базується на хуках, і він не потребує зайвої структури. Все, що потрібно – створити стор за допомогою функції, і далі отримувати або оновлювати дані в компонентах через прості хуки. Це робить код легшим для розуміння та підтримки.

Таблиця 1.1. Порівняння Redux з Zustand

Характеристика	**Redux**	**Zustand**
Підхід до керування станом	Через actions, reducers, типи	Через функцію з хуком create()
Наявність шаблонного коду	Багато: дії, редюсери, типи, провайдер	Мінімум шаблонного коду
Налаштування	Потрібен Redux Provider, конфігурація store	Жодних провайдерів, store створюється одразу
API	Імперативне, часто непряме	Декларативне, гнучке, просте
Складність навчання	Потребує часу, особливо з middleware	Простий, легко інтегрується навіть новачками
Можливість інтеграції з хуками	Потрібна додаткова бібліотека (react-redux)	Працює одразу з хуками
Підходить для великих проєктів	Так	Так, але краще для середніх і малих
Підходить для простих задач	Надмірний	Оптимальний

1.2.3 Забезпечення реального часу: огляд WebSocket і використання Socket.IO

У застосунку реалізована функція спільного перегляду аніме. Це означає, що кілька користувачів можуть одночасно знаходитися в «кімнаті» і дивитися одне й

те саме відео. Для цього всі учасники мають бачити однаковий момент у плеєрі, бачити, коли хтось натискає «пауза» або «відтворити», а також – мати змогу спілкуватися в чаті. Таку взаємодію не вийде організувати через звичайні HTTP-запити – потрібна миттєва передача даних між клієнтом і сервером. Для реалізації функцій, які потребують миттєвого обміну даними між клієнтом і сервером, таких як чат або синхронізація перегляду, у нашому застосунку було використано WebSocket у зв’язці з бібліотекою Socket.IO [5].

Веб сокетом [6] називають технологію, яка дозволяє встановити постійне з’єднання між клієнтом і сервером. На відміну від класичного HTTP-запиту, де кожне звернення вимагає нового з’єднання, WebSocket дозволяє одразу обом сторонам (клієнту і серверу) надсилати дані один одному в будь-який момент часу без затримки на «рукостискання». Це суттєво зменшує час очікування і дає змогу створювати "живі" інтерфейси – чат, повідомлення, індикатори онлайн-статусу тощо.

Зв'язок через WebSocket починається з HTTP-рукостискання. Після встановлення з'єднання воно залишається відкритим, і клієнт і сервер можуть обмінюватися даними незалежно один від одного в будь-який час.

Проте сам WebSocket – це низькорівневий інструмент. Для більш зручної розробки в реальних проєктах, де потрібно обробляти події, працювати з кімнатами або повторно під’єднувати клієнта у випадку втрати з’єднання, ми використали Socket.IO бібліотеку, яка спрощує роботу з WebSocket і додає набагато більше можливостей.

Перевагою Socket.io є те, що ця бібліотека дозволяє легко організовувати спілкування в межах певний кімнат, обробляти події join, leave, message, що зручно при побудові логіки, забезпечити автоматичне повторне з’єднання та надає можливість зробити http-fallback, тобто якщо з'єднання не вдасться встановити через WebSocket, з'єднання повернеться до HTTP-long-polling.

Крім того, використання Socket.IO забезпечує високу стабільність роботи застосунку навіть за нестабільного інтернет-з’єднання. Це дозволяє користувачам отримувати безперервний досвід взаємодії без необхідності вручну оновлювати

сторінку або відновлювати з’єднання.

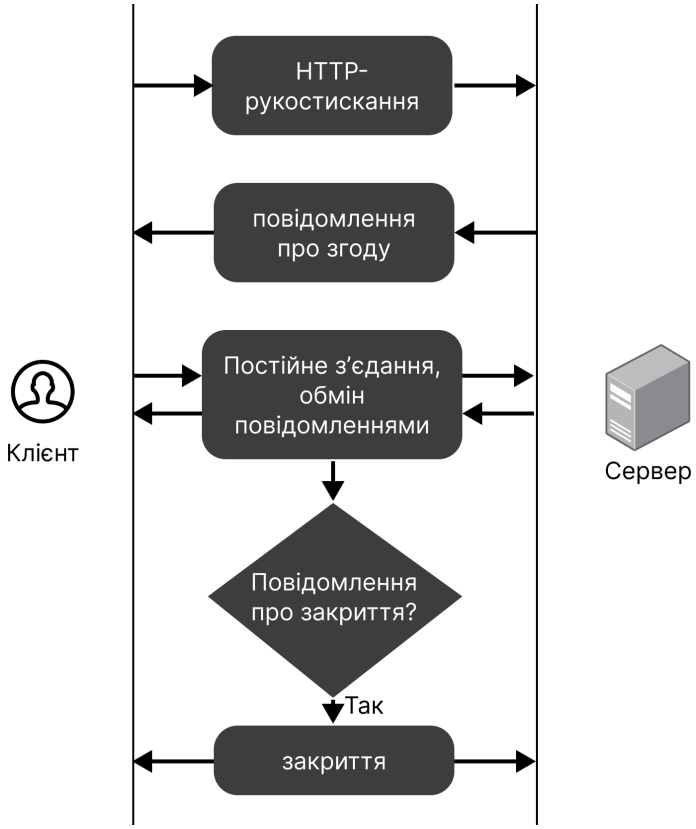


Рисунок 1.4. Схема роботи технології WebSockets

1.2.4 Серверна логіка з Node.js та Express.js

Для обробки запитів на сервері, взаємодії з базою даних та керування кімнатами для спільного перегляду в застосунку використовується зв’язка Node.js та Express.js. Такий підхід дозволяє швидко будувати надійний сервер з мінімальними накладними витратами.

Node.js – це середовище виконання JavaScript на сервері. Завдяки асинхронній моделі воно добре підходить для обробки великої кількості одночасних запитів, що особливо важливо у веб-застосунку з функціональністю реального часу.

Express.js – це мінімалістичний фреймворк, який спрощує створення маршрутів, обробку запитів і підключення проміжного програмного забезпечення (middleware). Він забезпечує достатньо гнучкості, щоб будувати як прості API, так і повноцінну серверну логіку для масштабного застосунку.

У цьому проєкті Express відповідає за:

- маршрути для авторизації, пошуку та взаємодії з даними;
- інтеграцію з базою даних MongoDB;
- керування сесіями користувачів (разом з express-session);
- валідацію та обробку даних, що надходять від клієнта.

Для зберігання даних використовується MongoDB, що ідеально підходить для структурування гнучких, неформалізованих даних, як-от профілі користувачів чи дані про кімнати. Бібліотека mongoose застосовується для моделювання цих даних і взаємодії з ними на рівні об'єктів.

Такий стек дозволяє легко масштабувати застосунок, а також забезпечує гарну інтеграцію з клієнтською частиною на Next.js. Оскільки і сервер, і клієнт написані JavaScript/TypeScript, взаємодія між ними є простою та зрозумілою.

1.2.5 Використання MongoDB як бази даних: структура, моделі, запити

MongoDB стала основою для зберігання всіх даних у межа застосунку. Це нереляційна (NoSQL) база даних, яка зберігає інформацію у вигляді документів формату JSON (фактично BSON). Основною перевагою для проєкту стало те, що вона дозволяє швидко розпочати розробку без потреби створювати складні структури таблиць і зв’язків між ними, як у випадку з SQL-базами. Крім того, її хмарний варіант MongoDB Atlas значно спрощує налаштування, адміністрування та резервне зберігання даних.

Документно-орієнтована структура MongoDB дозволяє працювати з даними, що мають складну вкладену структуру, напряму. Це дуже зручно для застосунків, які зберігають об'єкти з багатьма рівнями вкладеності, наприклад, профілі користувачів, списки вподобаного контенту, налаштування персоналізації тощо. Завдяки цьому я мав змогу створити гнучку модель користувача, яка містить не лише базові поля, а й вкладені структури – наприклад, списки з аніме, історію перегляду, статистику по жанрах тощо.

Для взаємодії між Node.js сервером та MongoDB було використано бібліотеку Mongoose. Вона надає зручний API для створення схем та моделей, які відображають структуру документів у колекціях. Замість того, щоб вручну писати запити до бази, Mongoose дозволяє описати модель (наприклад, користувача чи

аніме) у вигляді окремого класу або функції, після чого працювати з даними як з об’єктами JavaScript. Такий підхід полегшує налагодження, додає типізацію, а також автоматично перевіряє правильність структури перед збереженням документа.

У застосунку всі запити до бази винесені в окремі сервіси. Це реалізовано з урахуванням принципів серверної архітектури: кожен сервіс відповідає за окрему сутність (наприклад, користувача або аніме) та включає методи пошуку, створення, оновлення і видалення. Такий підхід дозволяє:

- централізувати бізнес-логіку;
- обробку помилок;
- забезпечити повторне використання коду;
- краще масштабувати застосунок у майбутньому.

Також реалізовано механізми валідації. Вони працюють на рівні схем Mongoose, що дозволяє перевіряти правильність даних ще до того, як вони потраплять у базу. Наприклад, можна вказати, що поле email має бути унікальним та відповідати шаблону електронної адреси, а поле username не повинно бути коротшим за 3 символи. Це допомагає уникати типових помилок ще на етапі розробки.

1.2.6 Зберігання сесій з express-session і автентифікація

У будь-якому сучасному застосунку, який передбачає роботу з профілями користувачів, дуже важливо організувати автентифікацію – процес підтвердження особи користувача. Цей механізм не тільки забезпечує персоналізований доступ до функціоналу, але й гарантує безпеку користувацьких даних. У рамках мого проєкту було реалізовано збереження сесій за допомогою middleware-пакету express-session.

Принцип роботи дуже простий: коли користувач успішно проходить автентифікацію, сервер «запам’ятовує» його, створюючи сесію. У подальших запитах сервер перевіряє, чи є у користувача активна сесія, і дозволяє доступ до потрібних частин застосунку. Це дозволяє уникати повторної автентифікації на кожному кроці.

Основна перевага express-session – це серверне зберігання інформації про користувача, тобто всі критично важливі дані зберігаються не в браузері, а на стороні сервера. Завдяки цьому забезпечується вищий рівень безпеки, оскільки сторонні користувачі не мають прямого доступу до вмісту сесій. Кожній сесії присвоюється унікальний ідентифікатор, який передається клієнту у вигляді cookie.

Процес входу/виходу реалізовано так, щоб користувачі мали змогу комфортно працювати з застосунком. Після входу сервер створює сесію, де зберігається sessionID користувача. Вихід із системи очищає сесію, таким чином припиняючи доступ до персоналізованих частин сайту. Доступ до захищених сторінок перевіряється на кожному запиті, що дозволяє уникнути несанкціонованого доступу.

У межах автентифікації також використовується хешування паролів. Це означає, що навіть якщо зловмисник отримає доступ до бази даних, він не побачить реальні паролі – лише хеші, які практично неможливо розшифрувати без спеціальних засобів. Для хешування використовується популярна бібліотека bcrypt.

Такий підхід до збереження сесій дозволяє досягти:

- простоти реалізації;
- надійного контролю автентифікації;
- можливості масштабування в майбутньому;
- централізованого управління сесіями (через MongoDB або Redis, якщо буде потреба).

1.2.7 Реалізація інтелектуального пошуку за допомогою ШІ: підхід, логіка, обґрунтування

Пошук – це одна з найважливіших функцій у будь-якому застосунку, де користувач взаємодіє з великим обсягом контенту. У рамках даного проєкту потрібно було реалізувати такий механізм пошуку, який не просто шукає за точним співпадінням, а й мав змогу шукати більш розпливчасті запити, а саме механізм, який не обмежується лише точним співпадінням введеного слова з назвою, а й

розуміє приблизні запити, орієнтується в синонімах, підтримує декілька мов і навіть намагається «здогадатися», що мав на увазі користувач. Саме тому було прийнято рішення впровадити інтелектуальний пошук, який використовує прості ШІ-алгоритми.

Основна мета полягала в тому, щоб пошук працював гнучко, тобто не вимагав від користувача точного формулювання. Наприклад, навіть якщо користувач помилиться у назві, або напише її іншою мовою, або просто введе опис сюжету – система все одно повинна допомогти йому знайти відповідне аніме. Щоб це реалізувати, пошук був розбитий на кілька рівнів логіки.

Рівні реалізації пошуку:

1. Пошук у базі даних – система шукає аніме серед уже збережених у внутрішній базі даних. Тут перевіряється назва англійською, українською, японською (ромаджі), а також можливі варіації написання. Такий базовий пошук дозволяє швидко знайти аніме, яке вже було переглянуто чи збережено.
2. Розширений пошук – Аніме, яке шукає користувач, шукається в базі даних. До цієї відповіді також додаються аніме, які були знайдені за межами бази даних, в цьому випадку на сторонньому сервісі «My Anime List» (MAL), який являється самою великою базою даних про аніме. Нові знайдені тайтли можуть одразу зберігатися у внутрішню базу, щоб у майбутньому уникати повторних звернень і прискорити пошук.
3. Пошук з ШІ – Аніме шукається не тільки за приблизною назвою, а й за приблизним описом. ШІ, який аналізує введений запит, формує припущення щодо того, яке саме аніме може матися на увазі, та видає відповідь. Відповідь, яку надав штучний інтелект, обробляється та результат шукається у внутрішній та зовнішній (MAL) базі даних, та відправляється назад користувачу. У даний спосіб, навіть не зовсім конкретне чи неточне формулювання може привести до правильного результату.

Для реалізації цього рівня пошуку були розглянуті сучасні API, які надають доступ до потужних мовних моделей.

Найзручніші з них:

1. ChatGPT API (від OpenAI) – дозволяє надсилати текстові запити та отримувати розгорнуті відповіді. Модель розуміє контекст, опис, логічні зв’язки.
2. Gemini API (від Google) – підходить для завдань з аналізу природної мови, може працювати з багатомовними запитами, має вбудовані функції резюмування чи переформулювання.
3. OpenRouter API – це платформа, яка надає доступ до кількох різних мовних моделей (у тому числі Claude, Mistral, GPT тощо) через єдиний інтерфейс, що дозволяє швидко тестувати різні підходи без потреби змінювати логіку додатку.

На практиці, будь-який із цих варіантів може бути інтегрований для обробки описових запитів користувача. Система надсилає введений запит до API, отримує припущення про назву або ключові слова, і вже потім шукає відповідність у базах.

Для цього веб-застосунку було обрано саме OpenRouter API, так як він налічує безкоштовні моделі.

1.3 Проєктування застосунку

1.3.1 Архітектура клієнт-серверної взаємодії

Клієнт-серверна модель передбачає собою розподілену архітектура додатків, яка розподіляє завдання або робоче навантаження між серверами і клієнтами. У цій моделі клієнт надсилає серверу запит на отримання даних, які зазвичай обробляються на стороні сервера. Потім сервер повертає запитані дані клієнту.

Клієнти, як правило, не діляться ресурсами один з одним, а покладаються на сервер, який надає запитувані ресурси або послуги. Поширеними прикладами клієнт-серверної моделі є системи електронної пошти та Всесвітня павутина (WWW), де поштові клієнти взаємодіють з поштовими серверами, а веб-браузери запитують ресурси з веб-серверів.

Коли ми говоримо про «клієнт», ми маємо на увазі пристрій (зазвичай комп'ютер, смартфон або додаток), який запитує та отримує послуги від сервера.

Тобто, клієнт – деякий об'єкт, який ініціює зв'язок, запитуючи дані або ресурси від сервера. Наприклад, веб-браузери, такі як Google Chrome, Mozilla Firefox або Safari, є поширеними клієнтськими програмами, які запитують дані з сервера для відображення веб-сторінок.

З іншого боку, сервер який є віддаленим комп'ютером або системою, яка надає дані, ресурси або послуги клієнтам. Він прослуховує вхідні клієнтські запити, обробляє їх і надсилає необхідну інформацію назад. Сервер може обробляти кілька клієнтських запитів одночасно.

Наприклад, веб-сервери розміщують веб-сайти, а сервери баз даних зберігають і обслуговують бази даних для додатків. Простіше кажучи, клієнт надсилає запит на сервер, а сервер обслуговує його доти, доки дані або послуга доступні в його системі.

Клієнт-серверна модель має кілька переваг, які роблять її популярною в мережевих і розподілених системах:

1. Централізоване управління даними: Всі дані зберігаються на централізованому сервері, що полегшує управління, оновлення та резервне копіювання.
2. Економічна ефективність: Оскільки сервер виконує більшу частину обробки, клієнти вимагають менше ресурсів і можуть бути простішими пристроями, що знижує витрати.
3. Масштабованість: Як клієнти, так і сервери можна масштабувати окремо. Сервери можна модернізувати для обслуговування більшої кількості клієнтів, а нові клієнти можуть бути додані без значних змін в інфраструктурі сервера.
4. Відновлення даних: Централізоване зберігання даних на сервері дозволяє краще відновлювати дані та спрощує стратегії резервного копіювання.
5. Безпека: Заходи безпеки, такі як брандмауери, шифрування та автентифікація, можуть бути централізовані на сервері, забезпечуючи захист конфіденційних даних.

Типи архітектури клієнт-сервер:

1. Дворівнева архітектура: Найпростіша форма, де клієнти та сервери взаємодіють безпосередньо.
2. Трирівнева архітектура (Рис. 1.5): Між клієнтом і сервером додається проміжний рівень, який часто називають middleware.
3. рівнева архітектура: Включає кілька проміжних рівнів, таких як безпека і бізнес-логіка, для управління складною обробкою даних і забезпечення безпеки.

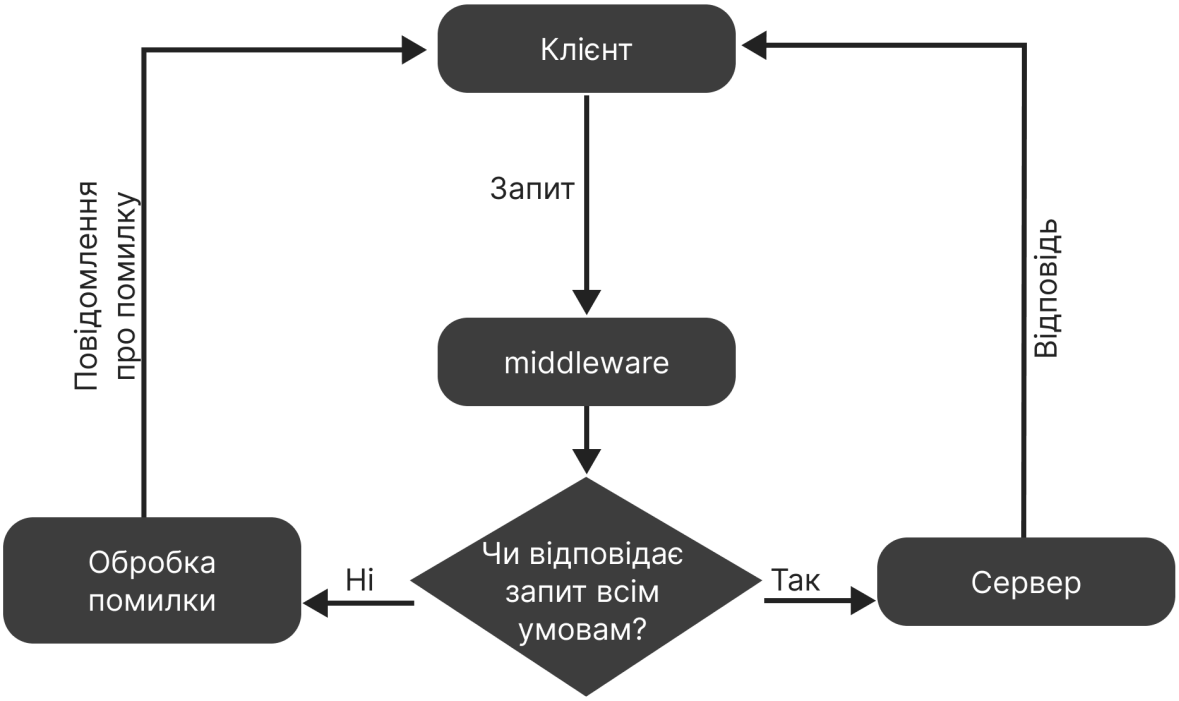


Рисунок 1.5. Трирівнева архітектура

Які є альтернативи клієнт-серверної взаємодії?

Хоча клієнт-серверна модель є найбільш традиційною для веб-застосунків, існують інші архітектурні підходи, кожен з яких має свої особливості, переваги та сфери застосування. До таких альтернатив належать архітектура мікросервісів, архітектурний шаблон MVC, а також peer-to-peer (P2P) моделі.

Попри свою популярність, клієнт-серверна модель не є єдиним способом побудови мережевих застосунків. Залежно від завдань, вимог до масштабування, швидкості реакції чи відмовостійкості, можуть застосовуватись такі альтернативні підходи:

1. Мікросервісна архітектура. Це сучасний підхід до побудови програм, де вся система складається з незалежних компонентів – сервісів, кожен з яких реалізує одну конкретну функцію (наприклад, аутентифікація, аналітика, пошук). Кожен сервіс може бути розроблений, оновлений і масштабований окремо. Сервіси зазвичай взаємодіють між собою через HTTP або повідомлення (черги подій). Переваги: масштабованість, легкість у підтримці, можливість незалежної розробки модулів. Недоліки: складність у налаштуванні зв'язків і моніторингу, необхідність у додаткових інструментах (реєстр сервісів, логування, трейсинг).
2. Peer-to-Peer (P2P). У цій архітектурі відсутній централізований сервер. Усі пристрої (вузли) рівноправні: вони можуть і відправляти, і приймати дані. P2P моделі часто використовуються в месенджерах, торент-мережах, блокчейн-технологіях і застосунках із прямим обміном файлами. Переваги: висока стійкість до збоїв, відсутність центрального вузла, зменшення навантаження на сервер. Недоліки: важче забезпечити безпеку, контроль доступу, а також підтримку однакової версії застосунку на всіх вузлах.
3. Serverless-архітектура (FaaS – Function as a Service). У цьому підході розробник пише окремі функції, які запускаються лише тоді, коли надходить запит. Уся інфраструктура керується постачальником (наприклад, AWS Lambda, Google Cloud Functions), а серверна частина «прихована» від розробника. Переваги: не потрібно налаштовувати або обслуговувати сервери, автоматичне масштабування. Недоліки: холодний старт (затримка першого виклику), обмеження по тривалості виконання, складніше організувати стан.
4. Event-Driven Architecture (EDA). Ця архітектура базується на ідеї, що компоненти системи взаємодіють через події. Наприклад, одна частина застосунку генерує подію (користувач зробив замовлення), інші реагують на неї (оновлення складу, надсилання листа). Переваги: висока асинхронність, гнучкість, легше масштабування подій. Недоліки: важче відслідковувати повний ланцюг подій, складніша відладка.

1.3.2 Схема маршрутизації в Next.js

Вебзастосунок побудований з використанням фреймворку Next.js, у якому реалізовано систему маршрутизації на основі файлової структури (File System Routing). В основі архітектури застосовано нову модель App Router, представлений починаючи з версії 13. Усі маршрути, компонування, завантаження та обробка даних структуровано в директорії src/app.

Кожна папка в src/app може містити такі файли:

- page.tsx – основний вміст сторінки.
- loading.tsx – компонент, що показується під час завантаження сторінки.
- not-found.tsx – індивідуальна обробка 404 для певного маршруту.

Усі шляхи генеруються автоматично на основі структури директорій.

Папки в дужках, як-от (anime), використовуються для групування логічно пов'язаних сторінок, без створення власного URL-шляху. Тобто шлях /anime/[slug] відповідає src/app/(anime)/anime/[slug]/page.tsx, але частина (anime) у URL не відображається. Це дозволяє організовувати код за функціональністю, не порушуючи структуру URL.

Папка @modal є паралельною маршрутизацією (parallel routes) – новою функцією App Router. Вона дозволяє рендерити модальні компоненти поряд з основною сторінкою, не перезавантажуючи весь вміст. Зокрема, маршрути на кшталт /(.)login або /(.)search відкривають модальні вікна поверх поточної сторінки.

Підтримуються динамічні сегменти в URL за допомогою квадратних дужок:
– /anime/[slug] – перегляд аніме за ідентифікатором (наприклад, /anime/attack-on-titan).
– /anime/[slug]/room/[roomCode] – кімната спільного перегляду з кодом.

Next.js автоматично витягує значення параметрів (наприклад, slug, roomCode) через params у функціях generateMetadata, getServerSideProps, useParams() тощо.

Основними маршрутами є маршрути зазначені у таблиці 1.2

У директорії src/app/api створені серверні маршрути (route.ts), що працюють

подібно до REST API. Приклад структури:

- /api/anime/[slug] – завантаження інформації про аніме;
- /api/anime/episode – завантаження епізоду (через POST-запит);
- /api/list – генерація списку аніме згідно фільтрів;
- /api/login / /api/registration – обробка авторизації;
- /api/profile – дані користувача;
- /api/search – результати пошуку;
- /api/chart – статистика вподобань.

API-файли також можуть використовувати GET, POST та інші HTTP-методи прямо в route.ts (експорт функцій GET(), POST() і т. д.).
Також додані not-found.tsx, сторінка для відображення помилки 404, та layout.tsx, головна обгортка, в яку вбудовуються всі сторінки.

Таблиця 1.2. Шляхи роутів

Шлях	**Призначення**
/	Головна сторінка
/anime/[slug]	Опис, плеєр і дані конкретного аніме
/anime/[slug]/room/[roomCode]	Спільний перегляд у кімнаті
/anime/search	Сторінка пошуку
/login	Вхід до системи
/registration	Реєстрація нового користувача
/profile	Профіль користувача
/list	Каталог аніме зі зручним фільтруванням

1.3.3 Проєктування інтерфейсу користувача (UI)

У процесі розробки вебзастосунку велика увага приділялась зручності користувача (User Experience, UX) та сучасному, чистому інтерфейсу (User Interface, UI). Дизайн інтерфейсу був попередньо спроєктований у сервісі Figma, що дозволило структурувати компоненти, протестувати візуальні сценарії та забезпечити узгодженість елементів на всіх сторінках.

Було прийнято рішення дотримуватися мінімалістичного, але сучасного дизайну, який поєднує в собі інтуїтивність та візуальну ієрархію. Всі компоненти інтерфейсу побудовані відповідно до принципів дизайн-патерну ShadCN,

адаптованого до власних потреб, із впровадженням адаптивної кольорової гами, натхненної підходами Material UI.

Інтерфейс використовує багаторівневу структуру за допомогою концепції "глибини" (depth), яка задається через змінні, засновані на color-mix(in oklab, ...). Чим вищий візуальний шар елемента (наприклад, модальне вікно або ховерна підказка), тим світлішим є його фон. Це дозволяє зберігати контрастність і візуальну послідовність як у світлій, так і темній темі.

Під час проєктування було враховано низку сучасних принципів UI/UX:

1. Мінімалізм і акценти – дизайн містить мінімальну кількість відволікаючих елементів, акценти зроблені на важливих зонах: назвах, обкладинках, кнопках.
2. Контрастність і читабельність – текстові блоки мають високу контрастність щодо фону.
3. Адаптивність – інтерфейс створений із врахуванням різних розмірів екранів (від мобільного до широкоформатного монітору).
4. Компонентний підхід – усі UI-елементи мають уніфікований стиль (кнопки, карти, поля вводу тощо), що спрощує підтримку і масштабування.

Основними кольорами стали білий для акцентного кольору, та сірий з ефектами глибини, як фоновий колір. Для перемикання між світлою та темною темою було застосовано css функцію light-dark() [4].

Функція light-dark() дозволяє задати два кольори для властивості – повертає один з двох варіантів кольору, визначаючи, якщо розробник встановив світлу або темну колірну схему або користувач запросив світлу або темну тему – без необхідності інкапсуляції кольорів теми у запиті до медіа-функції preferences-color-scheme.

Кнопки мають однакову висоту, округлення (border-radius: 12px), а при наведенні – м’який ховер-ефект.

Головна сторінка.

Головна сторінка веб-застосунку розміщена за адресою /. Вона є основною точкою входу в систему для більшості користувачів. Тут реалізовано відображення

популярних аніме, аніме за рекомендаціями, підбірок аніме, а також історії перегляду (якщо користувач авторизований).

Це забезпечує швидкий доступ до релевантного контенту та сприяє персоналізованому досвіду.

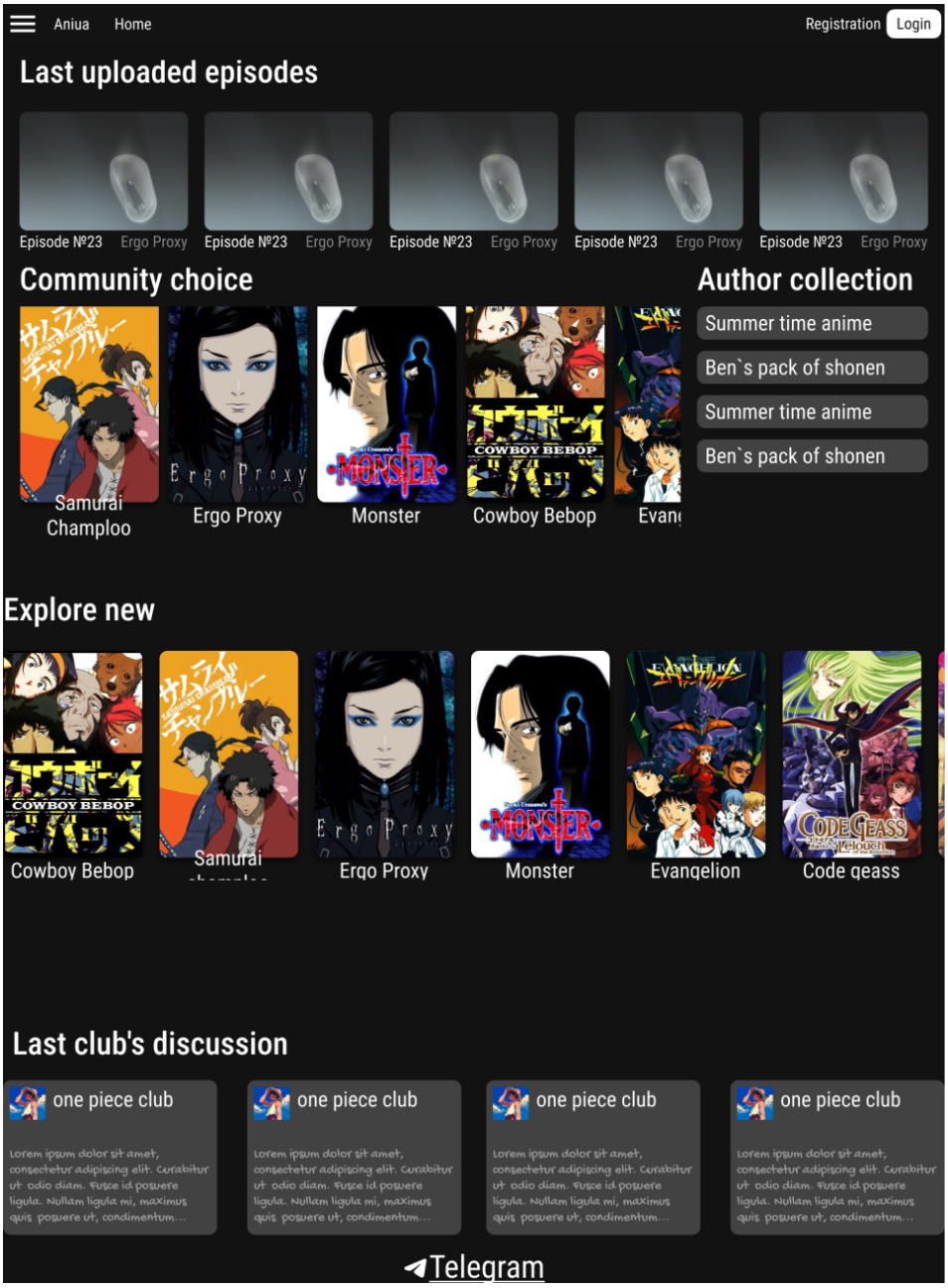


Рисунок 1.6. Головна сторінка

Сторінка каталогу аніме.

На сторінці /list реалізовано повноцінний каталог аніме. Користувач може застосовувати фільтри за жанром, роком, типом, кількістю серій та іншими параметрами. Також доступна можливість сортування за популярністю, рейтингом або алфавітом.

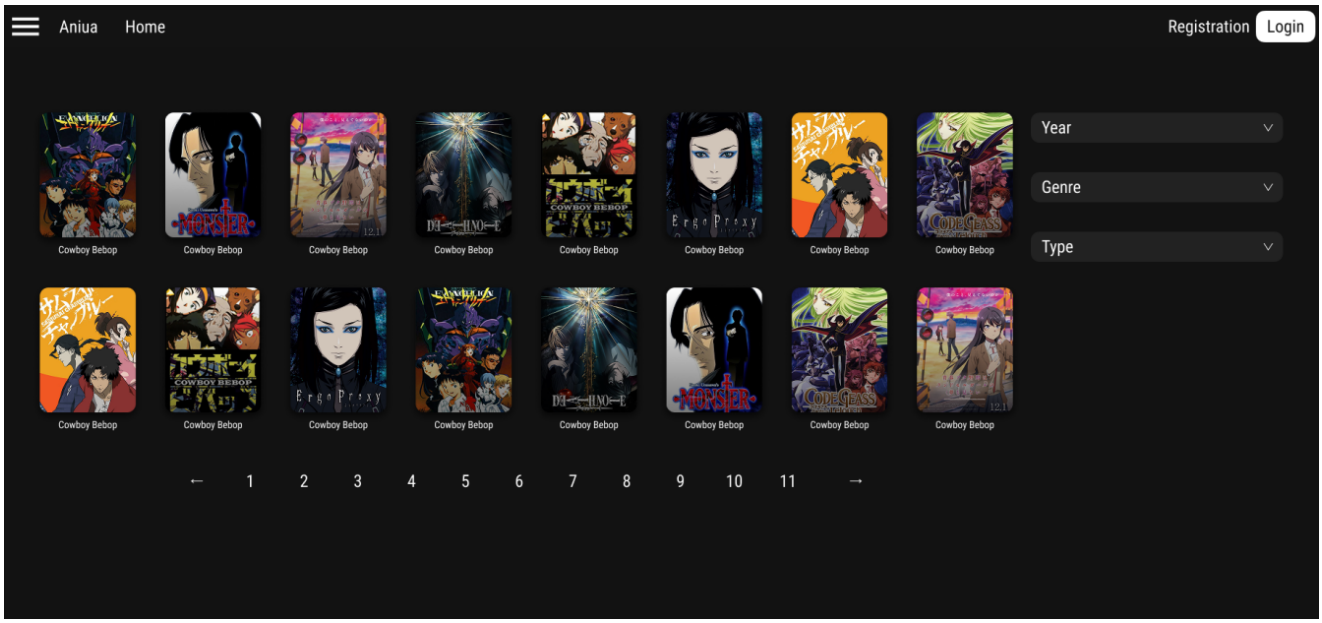


Рисунок 1.7. Сторінка зі списком аніме та фільтром

Сторінка аніме.

Ця сторінка призначена для перегляду конкретного аніме. Вона містить:

- відеоплеєр (у форматі iframe);
- загальну інформацію про аніме (опис, жанри, рік, студія);
- список серій (якщо релевантно);
- можливість додавання до власного списку перегляду.

Компоненти авторизації та реєстрації

Форма логіну та реєстрації реалізована як окремий компонент, який може відображатися у двох форматах:

- у вигляді модального вікна (при взаємодії з кнопкою входу без повного переходу);
- як повноцінна сторінка (/login, /register) – для зручності у випадках прямого переходу або з мобільних пристроїв.

Відеоплеєр підтримує адаптивний дизайн, що дозволяє комфортно переглядати аніме на різних пристроях, включно з мобільними телефонами та планшетами. Загальна інформація представлена у структурованому вигляді, що забезпечує легке і швидке ознайомлення з основними характеристиками аніме. Список серій містить функціонал позначення переглянутих епізодів, що допомагає користувачам відслідковувати прогрес перегляду. Компоненти авторизації

забезпечують надійну перевірку введених даних та підтримують функції відновлення паролю для зручності користувачів.

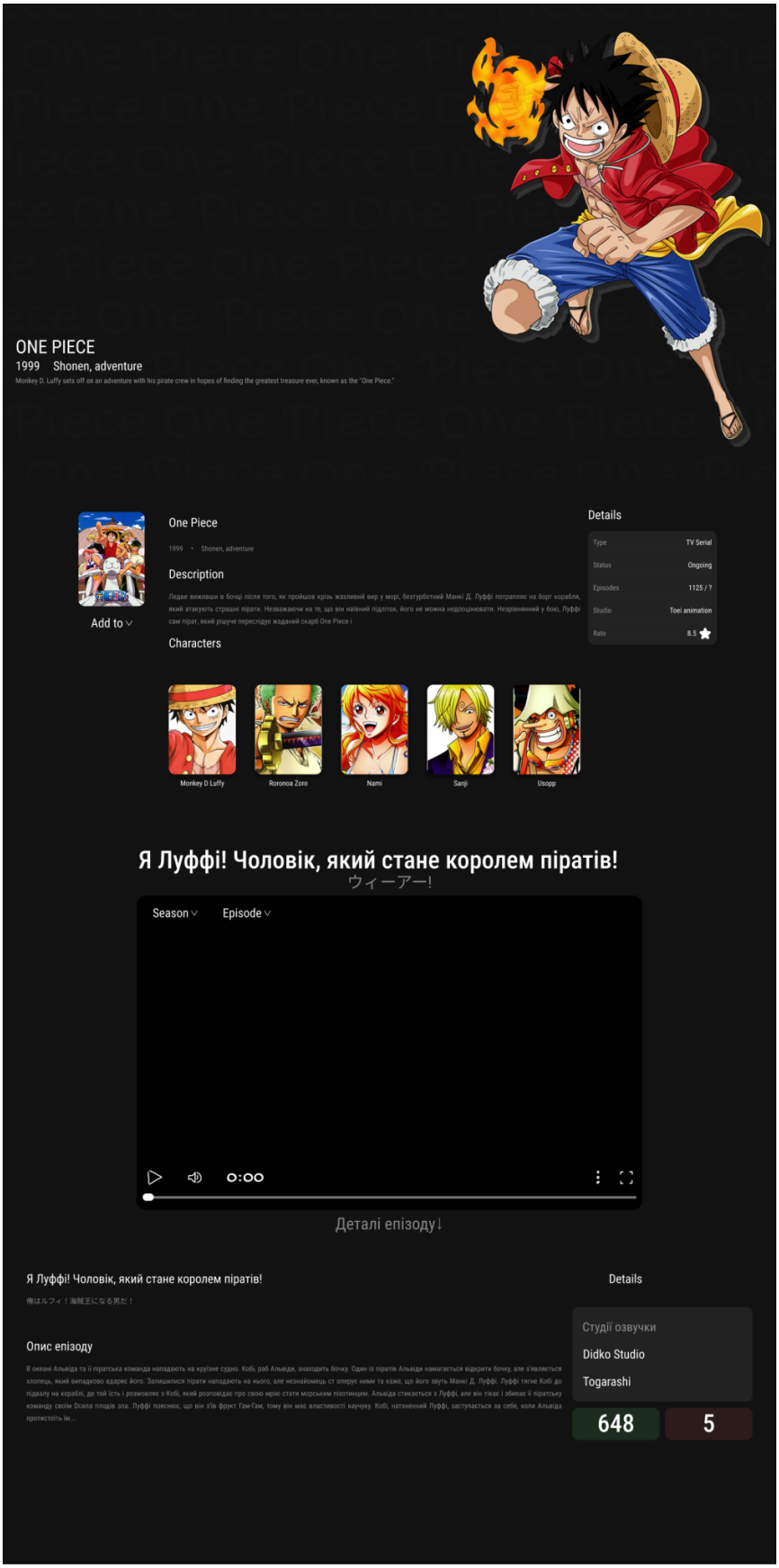


Рисунок 1.8. Сторінка перегляду аніме

Компонент пошуку.

Функціонал пошуку реалізований аналогічно: існує компонент пошуку, який:

- може бути викликаний як модальне вікно поверх будь-якої сторінки (наприклад, через клавішу швидкого доступу або клік по іконці) (Рис. 1.9);
- або як окрема сторінка з повноцінною видачею результатів (/search?q=...).

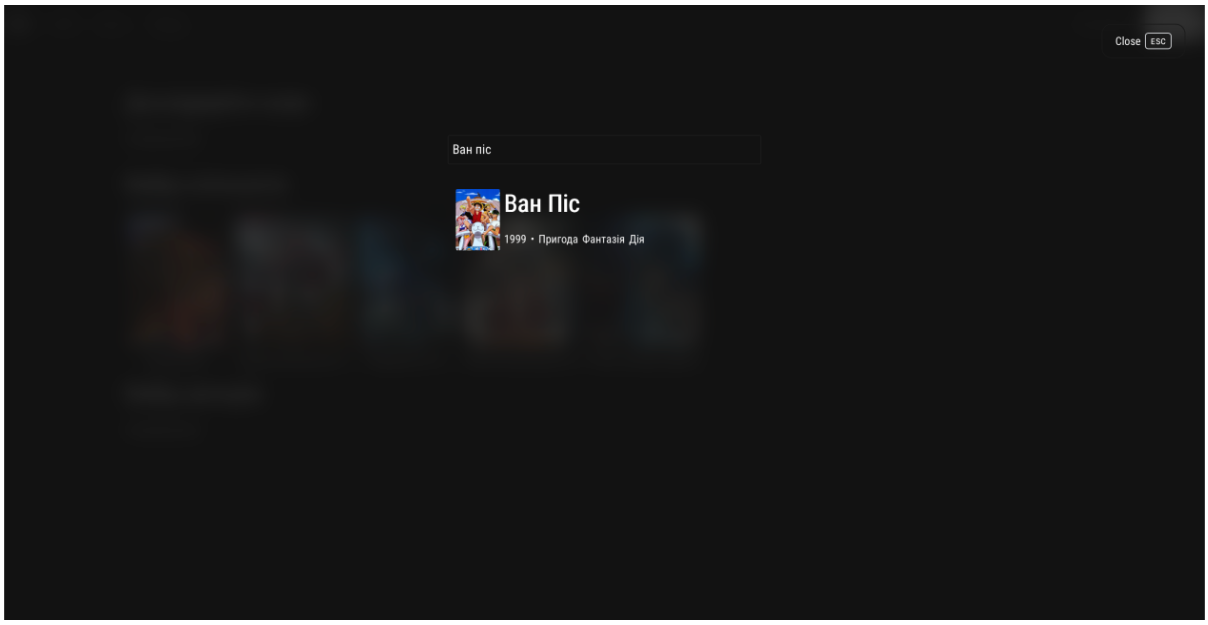


Рис. 1. 9 Компонент пошуку у вигляді модального вікна

1.3.4 Розробка користувацького досвіду (UX) для сумісного перегляду

Однією з ключових особливостей веб-застосунку є можливість сумісного перегляду аніме. Це дозволяє декільком користувачам одночасно переглядати одне й те саме відео в синхронізованому режимі в реальному часі, спілкуючись між собою та створюючи ефект присутності. Такий функціонал потребує ретельно продуманого UX-дизайну, оскільки він включає не лише перегляд контенту, а й соціальну взаємодію.

Основні принципи UX-дизайну для сумісного перегляду:

Простота підключення до перегляду.

Користувач може створити або приєднатись до кімнати перегляду за посиланням, отриманим від іншого учасника. Для створення нової кімнати достатньо натиснути кнопку “W2G”, яка переадресує користувача до створенної кімнати.

Синхронізація відеоплеєра.

Усі учасники кімнати бачать один і той самий момент відео. Перемотка або пауза ініціатора кімнати синхронізується для інших. UX-поведінка враховує відставання та перепідключення: користувач, який приєднався пізніше, автоматично переміщується до поточного моменту.

Мінімалізм і фокус на контенті.

Інтерфейс сумісного перегляду не перевантажений елементами (Рис. 1.10). Основний фокус – відео. Елементи керування відео згорнуті або винесені в окремі панелі. Це дозволяє зберігати чистий вигляд сторінки.

Рис. 1. 10 Вигляд плеєру під час сумісного перегляду

Можливість взаємодії в реальному часі.

Для зручності користувачів реалізований чат, в якому учасники можуть обмінюватися повідомленнями під час перегляду. Візуальне оформлення чату мінімалістичне, щоб не відволікати від відео.

Адаптивність.

UX реалізовано з урахуванням як десктопних, так і мобільних користувачів (Рис. 1.11). У мобільному вигляді інтерфейс автоматично приховує панелі, але залишає доступ до основних дій (відтворення, чат, вихід).

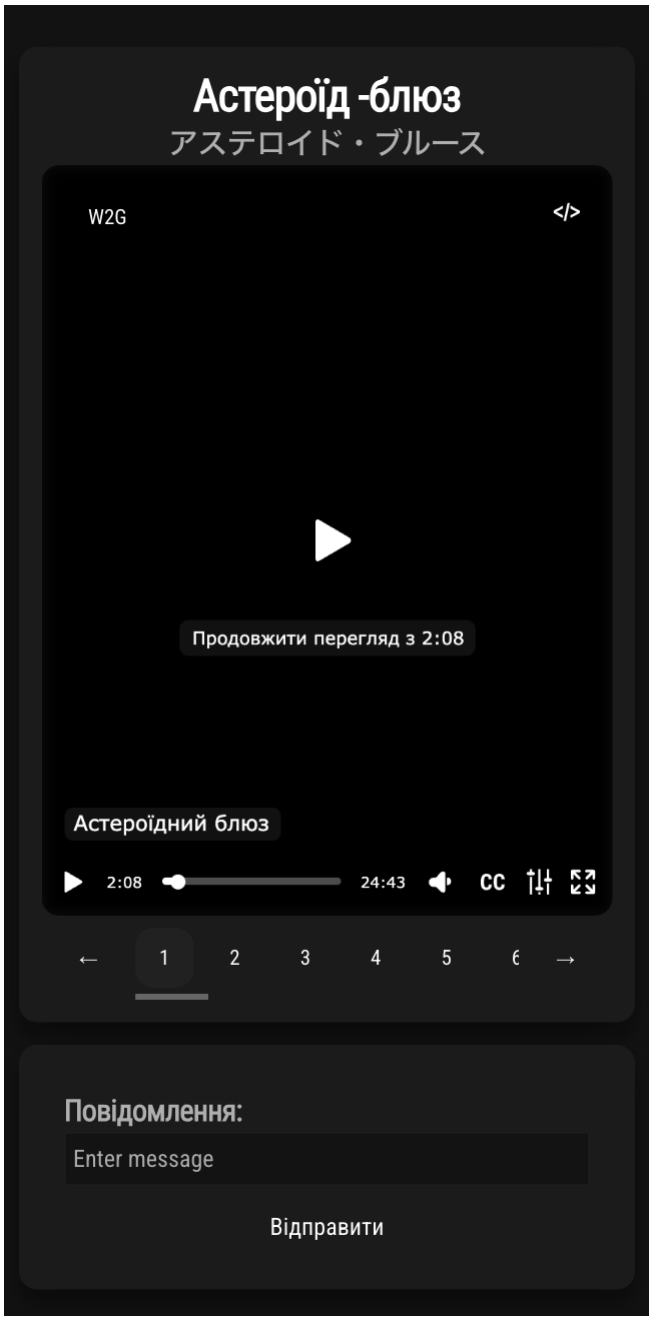


Рис. 1. 11 Мобільний вигляд сумісного перегляду

1.4 Реалізація функціоналу застосунку

1.4.1 Налаштування середовища розробки

Для розробки веб-застосунку було налаштоване сучасне середовище з використанням фреймворку Next.js, менеджера пакетів pnpm, редактора коду Visual Studio Code, а також таких інструментів як Tailwind CSS, ESLint, Prettier та ін. Це середовище дозволяє ефективно реалізувати SPA з серверною рендерингом, сучасною адаптивною верстkою та модульною архітектурою компонентів.

Visual Studio Code.

Visual Studio Code (VS Code) є основним інструментом для написання коду. Цей редактор має широкі можливості розширення, підтримує TypeScript, React, ESLint, Git і має потужну систему дебагінгу.

Набір рекомендованих розширень:

- ESLint – для lint-аналізу коду;
- Prettier – для автоматичного форматування;
- Tailwind CSS IntelliSense – автопідказки класів;
- GitLens – розширення для роботи з Git;
- Error Lens – зручна робота з помилками.

Ініціалізація проєкту

Для створення Next.js-застосунку було використано pnpm create next-app@latest, з наступними параметрами:

- App Router (app/);
- TypeScript;
- TailwindCSS;
- ESLint;
- Prettier;

Структура проєкту:

```
aniua/
├─ app/
│  ├─ layout.tsx
│  ├─ page.tsx
├─ components/
├─ styles/
├─ public/
├─ utils/
├─ types/
├─ .eslintrc.json
├─ tailwind.config.ts
├─ tsconfig.json
├─ .env.local
└─ pnpm-lock.yaml
```

Конфігурація Tailwind CSS:

```
import type { Config } from 'tailwindcss';
export default {
  content: [
    './src/pages/**/*.{js,ts,jsx,tsx,mdx}',
    './src/components/**/*.{js,ts,jsx,tsx,mdx}',
    './src/app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {
```

```
        colors: {
          textCol: 'var(--textColor)',
          textSemiCol: 'var(--textSemiColor)',
          background: 'var(--background)',
          c01dp: 'var(--01dp)',
          c02dp: 'var(--02dp)',
          transparent01dp: 'var(--t01dp)',
          transparent00dp: 'var(--t00dp)',
        },
        height: {
          'calc-screen-minus-4rem': 'calc(100svh – 4rem)',
        },
      },
    },
    plugins: [],
  } satisfies Config;
```

Залежності проєкту.

Проєкт використовує як основні залежності для клієнтської частини, так і dev-залежності для лінтингу, форматування та стилізації.

Основні залежності:

```
"dependencies": {
  "next": "^15.3.0",
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "socket.io": "^4.8.1",
  "zustand": "^5.0.2",
  "i18next": "^24.2.0",
  "react-i18next": "^15.4.0",
  "lodash.debounce": "^4.0.8",
  "react-hot-toast": "^2.5.2",
  "react-hook-form": "^7.54.0",
  "clsx": "^2.1.1",
  "@emotion/styled": "^11.13.5",
  "@mui/icons-material": "^6.1.8",
  "@mui/x-charts": "^7.23.2"
}
Dev-залежності:
"devDependencies": {
  "tailwindcss": "^3.4.17",
  "typescript": "^5.7.2",
  "eslint": "^8.57.1",
  "husky": "^9.1.7",
  "prettier": "^3.4.2",
  "lint-staged": "^15.4.0",
  "@eslint/js": "^9.17.0",
  "@types/react": "^18",
  "@types/node": "^20",
  "postcss": "^8.4.49",
  "autoprefixer": "^10.4.20"
}
```

Налаштування package.json:

```
"scripts": {
  "dev": "nodemon server.js",
  "dev:next": "next dev --turbopack --hostname 0.0.0.0",
  "build": "next build",
  "start": "NODE_ENV=production nodemon server.js",
```

```
    "lint": "next lint && prettier --write .",
    "prepare": "husky"
  },
  "lint-staged": {
    "*.{js,ts,tsx,css}": [
      "npm run lint",
      "git add"
    ]
  }
```

Налаштування змінних середовища. У .env.local зберігаються чутливі параметри:

```
NEXT_PUBLIC_BASE_URL=http://localhost:3000/api
NEXT_PUBLIC_API_URL=mongodb://localhost:27017/aniua
```

Завдяки такому підходу до конфігурації, середовище розробки є гнучким, розширюваним та зручним для колективної роботи над масштабним застосунком з підтримкою SSR, модальної навігації та адаптивного інтерфейсу.

1.4.2 Розробка клієнстьої частини

Next.js був обраний як фреймворк, що поєднує можливості клієнтської та серверної частини, дозволяючи реалізувати гібридні сторінки з підтримкою SSR (Server-side rendering), ISR (Incremental Static Regeneration) та SPA-навігації. Основними можливостями, які використовувались у проєкті, стали App Router, генерація метаданих для SEO та формування заголовків сторінок та middleware для попередньої обробки запитів.

Клієнтська частина програми побудована на основі компонентного підходу з використанням React, де функціональні компоненти відповідають за відображення, а логіка обробки даних винесена в окремі кастомні хуки. Такий підхід дозволив досягти високої гнучкості та повторного використання коду. Як приклад компоненту розглянемо компонент Player (src/app/(anime)/anime/Components/PlayerSection/Player.tsx).

Одним із ключових клієнтських компонентів є Player, що відповідає за перегляд аніме-серій. Він включає декілька підкомпонентів:

- EpisodeInfo – відображає назви епізоду на різних мовах;
- StudioDropdown – випадаючий список для вибору студії озвучення;
- EpisodeList – список доступних епізодів;
- iframe – основний елемент, у якому відтворюється відео.

Компонент підтримує синхронізацію перегляду через WebSocket (реалізовано в хук usePlayerSocket), що дає змогу одночасного перегляду кільком користувачам у рамках однієї кімнати (аналог Watch2Gether). Також забезпечено динамічне завантаження відео у відповідь на вибір користувача – зокрема зміну студії озвучення або конкретного епізоду.

Передача стану до Player здійснюється за допомогою спеціального хука usePlayer, що розглянуто нижче.

Кастомний хук usePlayer

Хук usePlayer інкапсулює усю бізнес-логіку, пов’язану з завантаженням та керуванням плеєром. Він виконує такі функції:

Завантаження списку епізодів.

При отриманні slug (унікального ідентифікатора аніме) викликається функція fetchEpisodeList, яка через API отримує список доступних епізодів.

Початкова ініціалізація стану.

Якщо знайдено епізоди, автоматично викликається функція handleEpisode, яка налаштовує URL відео, заголовки, список студій та інші параметри.

Зміна студії.

За допомогою функції handleStudio користувач може змінити студію озвучення, що викликає повторне оновлення плеєра із новим джерелом.

Обробка завантаження.

Стан isPlayerLoading дозволяє відображати індикатор завантаження під час очікування відео.

Такий розподіл логіки між компонентом і хуком дозволяє ізолювати побічні ефекти (useEffect), забезпечити чистоту компоненту та зробити хук повторно використовуваним для інших компонентів, які також можуть потребувати схожого функціоналу.

Хук також обробляє зміну поточного епізоду, що дозволяє користувачу швидко перемикатися між серіями без перезавантаження сторінки. Усі виклики API в хуку обгорнуті в try-catch блоки для обробки можливих помилок та покращення стабільності застосунку. Крім того, usePlayer інтегрується з

контекстом WebSocket, що дозволяє синхронізувати стан плеєра між кількома користувачами в реальному часі.

Рисунок 1.12. Схема роботи хука usePlayer

Глобальний стан через Zustand.

Для управління глобальним станом застосунку використовується легка та продуктивна бібліотека Zustand, яка дозволяє створювати центральні сховища (store) з прозорим API та зручною інтеграцією в React-компоненти без потреби в

контекстах чи громіздкому Redux.

Store користувача.

Один із ключових сто̀рів – це userStore, який відповідає за збереження профілю користувача, стан авторизації та контроль за гідратацією (тобто моментом, коли стан з локального сховища вже відновлено на клієнті).

```
interface UserStateStore {
  user: UserProfileInterface;
  hydrated: boolean;
  isLoggedIn: boolean;
  setLoginState: (loggedIn: boolean) => void;
  setHydrated: (value: boolean) => void;
  setUser: (user: UserProfileInterface) => void;
  removeUser: () => void;
}
```

У Zustand store реалізовано такі функції:

- setUser – зберігає інформацію про користувача у стані, а також виставляє прапорець авторизації;
- removeUser – скидає користувача до початкових значень та відключає прапорець входу;
- setLoginState – дозволяє явно виставити статус авторизації, який додатково кешується в localStorage;
- setHydrated – вказує, що гідратація стану завершена;
- persist – middleware, який автоматично зберігає стан користувача в локальному сховищі, з подальшим відновленням при перезавантаженні сторінки;
- devtools – дозволяє вбудовану інтеграцію зі сторонніми інструментами розробника.

```
export const useUserStore = create<UserStateStore>()(
  devtools(
    persist(
      (set) => ({
        user: defaultUser,
        hydrated: false,
        isLoggedIn: false,
        setHydrated: (value: boolean) => set({ hydrated: value }),
        setLoginState: (loggedIn: boolean) => {
          localStorage.setItem('isLoggedIn', loggedIn ? 'true' : 'false');
          set({ isLoggedIn: loggedIn });
        },
        setUser: (user: UserProfileInterface) => {
          set({ user, isLoggedIn: true });
          localStorage.setItem('isLoggedIn', 'true');
        },
```

```
            removeUser: () => {
              set({ user: defaultUser, isLoggedIn: false });
              localStorage.setItem('isLoggedIn', 'false');
            },
          }),
          {
            name: 'userStore',
            onRehydrateStorage: () => (state) => {
              if (state && !state?.user) {
                state.user = defaultUser;
              }
              state?.setHydrated(true);
            },
          },
        ),
        {
          name: 'userStore',
        },
      ),
    );
```

У даний спосіб, гідратація гарантує, що після першого рендеру на клієнті, додаток отримає актуальні дані користувача, синхронізовані з localStorage.

Хук useUserProfile.

Щоб забезпечити логіку роботи з профілем користувача (отримання, оновлення або видалення), створено кастомний хук useUserProfile, який працює з userStore. Він:

- витягує необхідні методи зі сто̀ру (включаючи user, setUser, removeUser);
- запускає запит на отримання профілю з API після гідратації стану, якщо користувач авторизований;
- в разі помилки – виводить повідомлення через toast та очищує стан;
- підтримує повторне використання в будь-якому компоненті, де потрібен доступ до даних користувача.

```
useEffect(() => {
  if (!hydrated) return;
  if (isLoggedIn) fetchUserProfile();
}, [hydrated]);
```

У даний спосіб, функція fetchUserProfile виконується лише після повного завантаження сто̀ру, що унеможливлює ранні виклики API до готовності клієнтського середовища.

Переваги підходу.

Використання Zustand у поєднанні з кастомними хуками дозволило:

- централізувати зберігання користувацького стану;

- уникнути дублювання логіки у компонентах;
- забезпечити прозоре управління авторизацією;
- легко інтегрувати з локальним сховищем та React-екосистемою без зайвої складності.

1.4.3. Побудова механізму синхронного перегляду (Socket.IO)

Для реалізації функціональності спільного перегляду відео було використано бібліотеку Socket.IO, яка дозволяє встановити постійне двостороннє з'єднання між клієнтом та сервером. У цьому випадку сервер Socket.IO розміщений безпосередньо на тому ж Node.js-сервері, що й Next.js, що дозволяє об'єднати маршрутизацію сторінок та WebSocket-логіку в межах одного процесу.

Ініціалізація Socket.IO на сервері.

У кореневому файлі запуску Next.js створено HTTP-сервер через createServer, який передається як аргумент при ініціалізації екземпляру Server з socket.io.

При підключенні клієнта оброблюються кілька основних подій:

1. join – клієнт приєднується до кімнати (roomCode), яка є сесією спільного перегляду.
2. send_state – клієнт надсилає свій поточний стан відео (URL і таймкод), який сервер пересилає іншому учаснику.
3. message / chat_message – звичайні або чат-повідомлення для кімнати.
4. disconnect – користувач покидає кімнату, і якщо вона порожня, її видаляють із пам’яті.

Код розділено на окремі обробники joinRoom, leaveRoom та onLobbyMessage для зручності підтримки й тестування.

Синхронізація стану відтворення.

Приєднання нових учасників до кімнати ініціює наступну послідовність:

1. Новий клієнт надсилає подію join з параметром roomCode.
2. Сервер додає клієнта до кімнати і, якщо в кімнаті вже є інші учасники, надсилає першому з них подію request_state з ідентифікатором нового клієнта.

3. Перший клієнт відповідає подією send_state, передаючи URL відео та позицію відтворення.
4. Сервер пересилає цю інформацію новому клієнту через подію state_update.

У даний спосіб, усі учасники отримують актуальний стан відео незалежно від того, коли вони приєдналися.

Обробка повідомлень.

Усі текстові повідомлення (включно з командами або чатами) оброблюються централізовано:

Подія message – використовується для керуючих команд (наприклад, "play", "pause").

Подія chat_message – для текстових повідомлень між учасниками перегляду, з передачею інформації про користувача (нікнейм, аватар).

Ці повідомлення транслюються всім користувачам у кімнаті, окрім відправника.

4. Управління кімнатами

Стан кожної кімнати зберігається у Map, де ключ – це roomCode, а значення – об'єкт з користувачами, поточним videoUrl і timecode. Це дозволяє легко відслідковувати стан кожної кімнати та очищати пам’ять, якщо кімната порожня.

```
rooms.set(roomCode, {
  users: new Set(),
  videoUrl: null,
  timecode: 0,
});
```

При виході користувача з кімнати (disconnect) – його видаляють із списку учасників, а якщо кімната стала порожньою – її повністю прибирають.

1.4.4 Реалізація серверної логіки

Серверна частина застосунку реалізована з використанням платформи Node.js у поєднанні з фреймворком Express.js, що забезпечує гнучкий і продуктивний обробник HTTP-запитів. Цей компонент виконує роль повноцінного бекенду, який обслуговує клієнтські запити, керує доступом до бази даних, здійснює автентифікацію та авторизацію користувачів, а також надає програмний інтерфейс (API) до всіх функціональних можливостей платформи – зокрема

пошуку аніме, керування профілем, роботу зі списками чи налаштуваннями.

З погляду архітектури, сервер структуровано за принципами розділення відповідальностей. Усі функції поділено на окремі логічні модулі – маршрути, контролери, моделі та сервіси. При надходженні запиту він спочатку обробляється маршрутизатором, який визначає шлях і передає запит відповідному контролеру. Контролер виконує перевірку, координує обробку запиту та звертається до сервісного шару – де, в залежності від ситуації, може бути реалізована логіка взаємодії з базою даних, зовнішнім API або іншими модулями системи.

Для роботи з базою даних використовується нереляційне сховище MongoDB у зв'язці з бібліотекою Mongoose, яка надає можливість чітко визначати структуру даних через схеми та моделі. Такий підхід забезпечує одночасно гнучкість NoSQL-рішень і контроль над структурованістю збереженої інформації.

Модель користувача зберігає не лише стандартні облікові дані, як-от ім'я користувача, пароль чи електронну пошту, а й містить додаткову інформацію: аватар, біографію, зовнішні посилання, CSS-налаштування профілю, рівень досвіду (XP), активність, а також список створених або збережених аніме-підбірок. Одна така підбірка (список) описується окремою схемою й містить власну назву, кількість тайтлів, рейтинг (вподобання/невподобання), а також список ідентифікаторів аніме.

Модель аніме є значно складнішою та деталізованішою. Вона зберігає повну інформацію про тайтл: оригінальні назви різними мовами (англійською, японською, синоніми), жанри, тип (наприклад, серіал чи фільм), рік і сезон виходу, опис, посилання на плеєр, тривалість, статус (триває чи завершено), рейтинг, оцінки з MyAnimeList, вік для перегляду, а також службові поля, як-от дата створення, останнього оновлення та унікальний slug для зручної адресації.

Основний серверний файл конфігурується як звичайний Express-додаток. Для збереження сесій використовується express-session, що дозволяє уникнути використання JWT. Сесії зберігаються у cookie з HTTP-only прапором, що забезпечує безпечніший варіант автентифікації.

```
import session from 'express-session';
app.use(
  session({
```

```
      secret: process.env.SECRET_KEY || 'supermegasecretkey',
      name: 'sessionid',
      resave: false,
      saveUninitialized: false,
      cookie: {
        httpOnly: true,
        maxAge: 30 * 24 * 60 * 60 * 1000, // 30 днів
      },
    })
);
```

Для імпорту файлів через псевдоніми використовується esm-module-alias. Це дозволяє не писати довгі шляхи типу ../../../utils/logger.js, а використовувати умовні позначення на кшталт # або !. Для визначення у головному файлі додано потрібні строки коду за документацією.

```
import addAliases from 'esm-module-alias';
addAliases({
  '#': './dist',
  '!': './',
});
У package.json це також враховано:
"aliases": {
  "#": "./dist/",
  "!": "./"
},
"imports": {
  "#/*": "./dist/**/*.js"
}
```

Збірка та запуск налаштовані так, щоб відокремити розробку (tsx watch) і продакшен (tsc + nodemon з ESM loader):

```
"scripts": {
  "dev": "tsx watch src/server.ts",
  "build": "tsc",
  "start": "nodemon --loader esm-module-alias/loader --no-warnings dist/server.js"
}
```

Серверна логіка пошуку реалізована у вигляді окремого сервісного класу SearchService, який виконує обробку запитів користувача на основі комбінації трьох джерел даних: локальної бази даних MongoDB, зовнішнього публічного API Jikan та спеціалізованого AI-сервісу. Вибір методу пошуку користувач здійснює самостійно, використовуючи певні ключові команди у запиті. Звичайний текстовий запит без префіксу опрацьовується лише локальною базою даних. Якщо ж запит розпочинається з команди /e, наприклад /e naruto, то система окрім локального пошуку додатково звертається до Jikan API для отримання більшої кількості результатів. У випадку, коли запит починається з команди /ai, наприклад /ai dark anime with psychological themes, активується пошук через мовну модель –

штучний інтелект формує перелік рекомендованих аніме, які далі перевіряються в локальній базі.

Якщо у базі не знаходиться достатньо релевантних результатів, викликається допоміжний метод, що надсилає запит до стороннього API Jikan. Пошуковий запит передається як параметр q, при цьому застосовуються додаткові обмеження, наприклад, фільтрація за типом. Отримані результати проходять попередню перевірку – з них вилучаються ті аніме, що вже наявні в локальній базі, аби уникнути дублювання. Для кожного унікального результату додатково завантажуються повні метадані через метод fetchAnimeByMal, після чого результат зберігається або повертається у відповідь.

У випадках, коли користувач здійснює запит з командою /ai, який є розмитим або описовим, система звертається до AI-сервісу рекомендацій. Спочатку формується промпт, який задає роль моделі як експерта з рекомендацій аніме. Далі на основі опису користувача модель генерує список рекомендованих назв у суворо формалізованому JSON-форматі, зокрема з вказаними MAL ID. Після отримання відповіді система виконує точний пошук кожної з рекомендованих назв у локальній базі, а за відсутності відповідника – звертається до API Jikan для отримання повних даних. У даний спосіб забезпечується комбінований пошук, що поєднує переваги локального кешування, публічних джерел та мовної обробки запитів через AI.

Окрім запитів до бази з аніме, реалізована повноцінна система взаємодії з обліковими записами користувачів. У публічному режимі доступна базова інформація про профіль, тоді як автентифіковані користувачі отримують доступ до повного функціоналу керування своїм обліковим записом. Це включає редагування особистих даних, зміну тем оформлення профілю, керування персональними списками аніме (наприклад, «Переглянуті», «У планах» або кастомні списки), а також отримання персоналізованих рекомендацій на основі аналізу жанрових вподобань. Уся інформація зберігається в базі даних у чітко визначеній структурі, що забезпечує швидкий доступ та надійне збереження стану користувацьких даних.

1.4.5 Інтеграція авторизації та збереження сесій

Для реалізації системи авторизації в проєкті використовується серверне збереження сесій через middleware express-session. Кожен користувач, що проходить успішну автентифікацію, отримує сесію, яка зберігається на сервері. Її параметри включають унікальне ім’я cookie, час життя та захист від доступу з JavaScript через httpOnly.

```
app.use(
  session({
    secret: process.env.SECRET_KEY || 'supermegasecretkey',
    name: 'sessionid',
    resave: false,
    saveUninitialized: false,
    cookie: {
      httpOnly: true,
      maxAge: 30 * 24 * 60 * 60 * 1000,
    },
  })
);
```

Коли клієнт надсилає запит на вхід, контролер loginController перевіряє наявність обов’язкових полів, а далі делегує логіку сервісному класу. Якщо користувача знайдено, а пароль успішно верифіковано за допомогою bcrypt, в об’єкт сесії додається поле user, що містить коротку інформацію про авторизованого клієнта:

```
export const loginController = async (req: Request, res: Response) => {
  const { username, password } = req.body;
  if (!username || !password) {
    return res.status(400).json({ message: "Username and password are required" });
  }
  try {
    const clientData = await userService.loginClient(username, password);
    req.session.user = {
      username: clientData.username,
    };
    res.status(200).json({
      success: true,
      message: "Login successful",
      client: clientData,
    });
  } catch (error) {
    const message = error instanceof Error ? error.message : "Unknown error";
    res.status(400).json({ message });
  }
};
```

У самому сервісі UserService здійснюється вся логіка роботи з MongoDB: пошук користувача за іменем, перевірка пароля, оновлення полів last_login і is_online, а також видалення чутливої інформації з результату перед поверненням.

```
async loginClient(username: string, password: string) {
  const client = await UserSchema.findOne({ username });
  if (!client) {
    throw new Error("User not found");
  }

  const isPasswordValid = await bcrypt.compare(password, client.password);
  if (!isPasswordValid) {
    throw new Error("Invalid password");
  }

  client.last_login = new Date();
  client.is_online = true;
  await client.save();

  const clientObject = client.toObject();
  delete (clientObject as any).password;
  return clientObject;
}
```

При реєстрації нового користувача в registerClient створюється обліковий запис із хешованим паролем, а також ініціалізується стартовий список аніме з полем is_default. У даний спосіб, одразу після створення облікового запису користувач отримує базову структуру для подальшої персоналізації профілю та колекцій.

Вся логіка реалізована за шаблоном маршрути → контролери → сервіси → моделі, що дозволяє централізовано керувати автентифікацією, зберігати стан користувача між запитами, а також масштабувати систему без порушення загальної архітектури.

1.4.6 Детальна реалізація механізму Socket.IO на сервері та клієнті

Для реалізації механізму синхронного перегляду в реальному часі було використано бібліотеку Socket.IO, яка забезпечує двосторонній зв’язок між сервером та клієнтом поверх протоколу WebSocket. Уся логіка взаємодії працює на тому ж сервері, де розміщується клієнтська частина, побудована за допомогою фреймворку Next.js.

Важливу роль у реалізації механізму синхронного перегляду відіграє можливість двосторонньої комунікації між iframe та основним додатком. Зокрема, сам iframe, у якому розміщується плеєр, підтримує надсилання повідомлень за допомогою `window.postMessage`, а React-компонент у відповідь слухає ці повідомлення через обробник `window.addEventListener('message', () => {})`. Завдяки

цьому стало можливим відслідковувати події з боку відеоплеєра, такі як старт, пауза, перемотування або оновлення часу. Ця функціональність і стала технічною основою для створення повноцінного синхронного перегляду між клієнтами – кожен з яких не просто отримує команду, а й реагує на неї так само, як це відбувалося б при власній взаємодії з плеєром.

Сервер створюється за допомогою модуля http, а також обгортки next, яка дозволяє обробляти як звичайні HTTP-запити, так і WebSocket-підключення. Після ініціалізації серверу, на ньому створюється екземпляр Socket.IO, до якого підключається кожен новий клієнт.

```
const dev = process.env.NODE_ENV !== 'production';
const hostname = 'localhost';
const port = 3000;
const app = next({ dev, hostname, port });
const handler = app.getRequestHandler();
app.prepare().then(() => {
  const httpServer = createServer(handler);
  const io = new Server(httpServer);
  io.on('connection', (socket) => {
    joinRoom(socket, rooms);
    onLobbyMessage(socket, rooms);

    socket.on('send_state', ({ requesterId, videoUrl, timecode }) => {
      console.log(`[SERVER]: Received state for ${requesterId}`);
      io.to(requesterId).emit('state_update', { videoUrl, timecode });
    });

    leaveRoom(socket, rooms);

    socket.on('error', (error) => {
      console.error('Socket error:', error);
    });
  });
  httpServer
    .once('error', (err) => {
      console.error(err);
      process.exit(1);
    })
    .listen(port, () => {
      console.log(`> Ready on http://${hostname}:${port}`);
    });
});
```

Основна логіка взаємодії з підключеннями поділена на кілька частин. По-перше, при підключенні кожного клієнта обробляються події входу в кімнату та виходу з неї. Під час приєднання клієнт передає унікальний roomCode, який дозволяє організувати окрему кімнату для спільного перегляду. Якщо кімната ще не існує, вона створюється динамічно, а всі нові користувачі додаються до неї через структуру типу Map, яка зберігає склад кімнати та поточний стан відео (посилання

на відтворюваний файл та час перегляду).

Якщо користувач приєднується до вже активної кімнати, сервер надсилає запит до першого учасника з проханням передати поточний стан перегляду, щоб новий клієнт міг синхронізуватися. Цей механізм дозволяє зберігати єдиний стан плеєра в рамках однієї кімнати.

Окремо реалізовано обробку подій типу message та chat_message. Вони дозволяють передавати між учасниками кімнати службові команди та текстові повідомлення. Усі такі події надсилаються не всім підключеним клієнтам, а лише тим, хто знаходиться в одній кімнаті, що знижує навантаження на сервер і підвищує ефективність взаємодії.

Також варто зазначити, що сервер веде базове логування всіх подій за допомогою утиліти logWithTime, що дозволяє бачити, хто, коли і яку дію виконав. Це спрощує налагодження та моніторинг активності в реальному часі.

У разі виникнення помилок, таких як втрати з'єднання або невірні дані, сервер генерує відповідні повідомлення про помилку. Наприклад, при спробі надсилання повідомлення поза межами кімнати, система виведе повідомлення про відсутність roomCode у клієнта.

Для роз’єднання користувача також передбачено обробку події disconnect. При її виклику користувач видаляється з відповідної кімнати. Якщо після цього кімната стає порожньою, вона автоматично видаляється зі списку активних.

Уся логіка спілкування між сервером і клієнтами реалізована таким чином, щоб забезпечити мінімальну затримку, актуальний стан плеєра та можливість розширення функціональності в майбутньому.

Клієнтська частина механізму спільного перегляду реалізована у вигляді кастомного React-хука usePlayerSocket. Цей хук відповідає за підключення до кімнати через Socket.IO, прийом повідомлень від інших учасників, надсилання стану відео, а також комунікацію з вбудованим відеоплеєром, який відображається через елемент <iframe>.

Хук приймає два параметри: ідентифікатор кімнати (roomCode) та посилання на сам iframe, в якому відтворюється відео. Після ініціалізації клієнт приєднується

до заданої кімнати за допомогою події join. Далі, при отриманні повідомлень через подію message, відбувається передача команд до iframe за допомогою методу postMessage. У даний спосіб, клієнт може дистанційно керувати станом відтворення: запускати відео, ставити на паузу або перемотувати до потрібного моменту.

Якщо новий користувач приєднується до кімнати, сервер надсилає іншим учасникам запит на поточний стан (request_state). Відповідно, клієнт, у якого є поточна інформація про відео, відправляє подію send_state, яка містить URL відео та його поточний час. Інший клієнт, що отримав цю інформацію, синхронізується: встановлює посилання у своєму iframe, а також перемотує відео до потрібного часу.

Щоб не відставати від поточного стану, клієнт слухає події з iframe через обробник window.addEventListener('message'). Коли користувач ставить відео на паузу, починає відтворення чи перемотує, ці дії транслюються на сервер, а звідти – до інших учасників кімнати. Передача часу (event: 'time') окремо записується у змінну time, яка надалі використовується для синхронізації стану.

У даний спосіб, плеєр всередині iframe завжди лишається пов’язаним з рештою учасників кімнати, навіть якщо його логіка ізольована. Стан між усіма учасниками підтримується актуальним, а події, які ініціює один клієнт, дублюються іншим – це ключовий принцип роботи функціональності спільного перегляду.

Хук також дозволяє оновлювати посилання на відео. Якщо користувач встановлює новий videoUrl, він автоматично передається через сокет усім іншим користувачам, що дозволяє перемикатися між епізодами або студіями без втрати синхронізації.

Уся комунікація відбувається на рівні об’єктів JSON зі структурою { command: string, additional?: string }, де command – це ключова дія, така як player_play, player_pause, player_seek або player_url, а additional використовується для передачі параметрів, таких як час чи URL.

Ця архітектура дозволяє легко масштабувати систему, додаючи нові типи

команд або повідомлень, зберігаючи при цьому незалежність плеєра від основного React-додатку. Кожен користувач отримує однакову взаємодію, незалежно від моменту приєднання до кімнати.

1.4.7 Реалізація інтелектуального пошуку за допомогою ШІ

Інтелектуальний пошук у веб-застосунку реалізовано у вигляді гнучкого API, який залежно від запиту користувача автоматично обирає джерело інформації: базу даних, зовнішнє API або мовну модель штучного інтелекту. Уся логіка пошуку зосереджена у сервісі SearchService, що забезпечує три основні сценарії (Рис. 1.13):

1. Стандартний пошук – виконується запит лише до локальної бази MongoDB.
2. Розширений пошук (/e) – виконується спочатку запит до бази, після чого ті самі або схожі дані шукаються в API Jikan.
3. AI-пошук (/ai) – запит надсилається до мовної моделі, яка повертає назви релевантних аніме, а вже потім система шукає ці аніме в базі або через Jikan.

Формат команди задається користувачем безпосередньо в рядку пошуку. У даний спосіб, реалізація підтримує адаптивний підхід – користувач сам вибирає спосіб пошуку відповідно до своїх потреб.

На рівні коду це реалізовано в роуті `/api/search`, де запит розділяється на команду (/ai, /e або порожню) і змістовну частину. Далі контролер визначає, який метод сервісу викликати:

- fetchSearch – для звичаних запитів, пошук у базі даних;
- searchOutApi – для розширених пошуків;
- searchOutAi – для пошуку за допомогою штучного інтелекту.

На практиці в searchOutAi використовується prompt-система, яка формує запит до мовної моделі у форматі системного і користувацького повідомлень. Відповідь очікується у вигляді масиву JSON-об'єктів із назвами аніме й MAL ID. Ці значення потім використовуються для запиту до локальної бази або API Jikan, щоб отримати повні дані.

Наприклад, якщо користувач ввів /ai коротке аніме про музику, то запит буде спочатку передано до мовної моделі, яка сформує список рекомендованих назв.

Далі ці назви перевіряються в базі. Якщо аніме не знайдене, воно запитується через Jikan API. У результаті клієнт отримає комбіновану вибірку з декількох джерел.

Уся ця логіка дозволяє забезпечити інтелектуальну адаптацію до складних запитів і створює гнучку, розширювану систему пошуку в реальному часі. Водночас реалізація залишається масштабованою завдяки розділенню запитів між базою, API та ШІ.

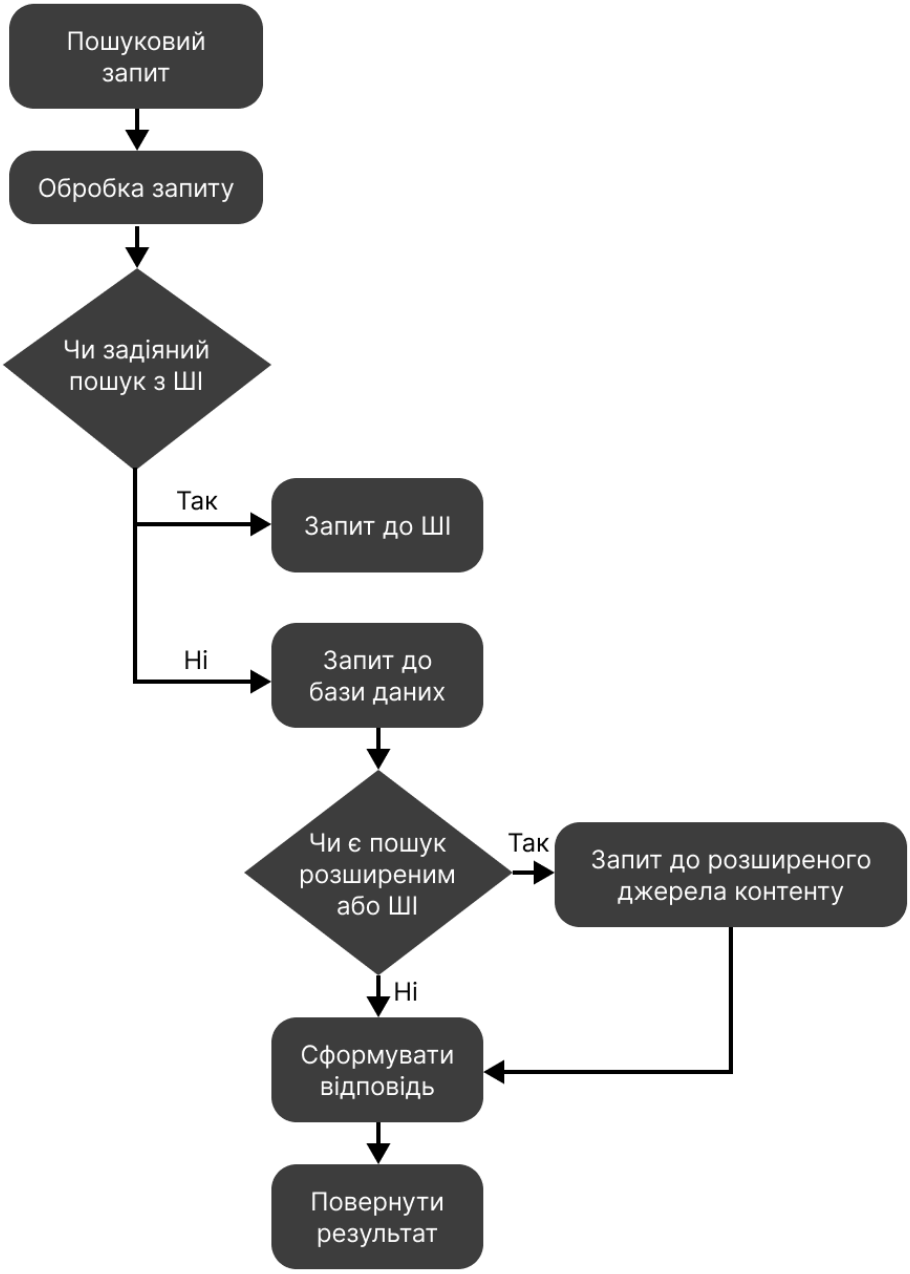


Рис. 1.13 Алгоритм роботи пошукового маршруту.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

З урахуванням специфіки теми – розробки веб-застосунку інтелектуального пошуку та спільного перегляду аніме-контенту – оцінювати ефективність такого проєкту доцільно не лише з економічної точки зору, а й враховуючи функціональні й соціальні аспекти. Залежно від того, який саме результат вважається основним – зручність пошуку, якість взаємодії між користувачами чи потенційна монетизація – можна говорити про функціональну, соціальну або економічну ефективність.

Ці показники тісно пов’язані з технічною реалізацією, інтерфейсними рішеннями, стабільністю роботи сервера, зручністю взаємодії з базою даних і можливостями масштабування. Також важливо враховувати, що потенційний результат застосунку – наприклад, приріст активної аудиторії або скорочення часу на пошук – може напряму залежати не лише від витрат на розробку, а й від зусиль, вкладених у оптимізацію, адаптацію під користувача та подальше просування.

2.2 Визначення витрат на розробку сайту

Для визначення витрат на розробку сайту (Вр) розрахуємо оплату праці виконавців, які могли б бути залучені до її виконання. Для реалізації проекту Web-системи використовуються наступні спеціалісти: фронтенд розробник, бекенд розробник, UI/UX дизайнер, CI/CD спеціаліст.

Складемо план-графік по розробці web-сайту і тривалості виконання робіт. Розподіл робіт по етапах і видах виконавців наведено в таблиці 3.1.

Таблиця 2.1. План-графік по розробці Web-сайту

№	Назва етапу	Час виконання (годин)	Посада виконавця
1	Аналіз вимог, створення структури бази даних	14	Fullstack
2	Розробка API та логіки взаємодії з клієнтом	34	Backend-розробник
3	Створення UI/UX макетів у Figma	24	UI/UX-дизайнер

Продовження таблиці 2.1

4	Створення компонентів інтерфейсу в Next.js	20	Frontend-розробник
5	Побудова сторінок, маршрутизація, адаптивність	26	Frontend-розробник
6	Реалізація глобального стану через Zustand	10	Frontend-розробник
7	Реалізація Socket.IO та логіки спільного перегляду	20	Frontend-розробник
8	Інтеграція інтелектуального пошуку з AI API	12	Фахівець з інтеграції ШІ.
ВСЬОГО:		160	

Під час розрахунку трудомісткості проєкту було сформовано перелік основних етапів і видів робіт, які необхідно виконати в межах розробки веб-застосунку. Цей перелік був узгоджений з керівником проєкту, в результаті чого частину робіт було уточнено, деякі етапи об’єднано або, навпаки, виключено з урахуванням їхньої доцільності. Такий підхід дозволив отримати оптимальну структуру робіт, адаптовану до обсягу завдань та конкретної мети дипломного проєкту.

Кожен вид робіт було пов’язано з відповідною посадою виконавця відповідного рівня кваліфікації. У випадках, коли певна задача передбачала участь фахівців різного профілю, роботу поділяли на декілька паралельних частин з окремим обліком трудових витрат. Це дозволило більш точно розрахувати витрати робочого часу та фінансові показники.

Для розрахунку заробітної плати використовувалася інформація про кількість залучених виконавців, загальний обсяг робіт, що виконувався кожною посадовою одиницею, а також середній рівень оплати праці за годину. Орієнтиром у фінансових розрахунках виступала мінімальна заробітна плата, встановлена відповідно до «Закону про Державний бюджет України» станом на 1 квітня 2025 року. Згідно з чинним законодавством, мінімальна місячна заробітна плата становить 8000 грн, що еквівалентно приблизно 48 грн на годину. Враховуючи характер робіт, їх складність та рівень відповідальності, були застосовані відповідні ставки заробітної плати, на основі яких і було сформовано підсумкову таблицю витрат на оплату праці.

Таблиця 2.2. Витрати на заробітну плату

№	Персонал	Етапи розробки	Кількість робочих годин (або днів)	Погодинна ставка (або денна ставка), грн.	Заробітна плата, грн.
1	Backend-розробник	Аналіз вимог, API, база даних, WebSocket, сесії, авторизація	64	50	3 200
2	Frontend-розробник	Клієнтська частина: маршрути, компоненти, Zustand, інтерфейс	48	50	2 400
3	UI/UX-дизайнер	Проєктування макетів, створення дизайну у Figma	16	50	800
4.	Фахівець з інтеграції ШІ.	Налаштування AI-запитів, обробка відповідей, генерація пошукових запитів	12	50	600
5	Fullstack (спільне тестування)	Інтеграція, перевірка, дебаг, тестування на помилки	29	50	1 450
ВСЬОГО:					$В_{зп}$ = 8 450 грн

До складу витрат на оплату праці також включаються податки, збори і інші обов'язкові платежі, встановлені системою оподаткування що діє. Розмір єдиного соціального внеску складає 22% від заробітної плати, розраховується за наступною формулою:

$$Вєсв = Взп \times 0,22 = 8\ 450 \times 0,22 = 1\ 859\ грн. \quad (2.1)$$

Загальні витрати (Вр) на розробку веб-сайту розраховуються як сума витрат на заробітну плату праці персоналу (Взп) та єдиного соціального внеску (Вєсв):

$$Вр = Взп + Вєсв = 8\ 450 + 1\ 859 = 10\ 309\ грн. \quad (2.2)$$

2.3 Визначення витрат на впровадження сайту

Витрати на впровадження сайту (Вв) складаються з двох складових:

- витрати на реєстрацію доменного імені на 1 рік (Вв1), що в залежності від доменого регіону та найменування варіюється від 75 до 750 грн. Візьмемо тариф за 147 гривень;
- витрати на реєстрацію в пошукових системах (Вв2), наприклад, Google, Rambler и т.п.).

$$B_в = B_{в1} + B_{в2} = 147 + 264 = 411 \text{ грн} \quad (2.3)$$

2.4 Визначення витрат на експлуатацію сайту

Витрати на експлуатацію сайту (Ве) включають вартість робіт з підтримки сайту в робочому стані і вартість послуг по продовженню доменного імені на 1 рік.

Роботи по підтримці сайту в робочому стані включають в себе:

- оновлення інформації;
- додавання та редагування контенту;
- видалення застарілих даних;
- створення резервних копій;
- моніторинг технічного стану.

У цьому випадку, обслуговування здійснюється адміністратором. У таблиці 3.3 визначаються постійні витрати як сума витрат на впровадження та експлуатацію сайту протягом року.

Таблиця 2.3. Постійні витрати

№	Стаття витрат	Вартість за рік, грн.
1	Подовження доменного імені	147
2	Витрати на хостинг	3 168 (264 грн на місяць)
3	Підтримка контенту	1 600
Всього:		$В_{пост}$ = 4 915

Загальні витрати (Вз) на розробку, впровадження та експлуатацію веб-сайту розрахуємо за наступною формулою:

$$Вз = Вр + Вв + Ве = 10\ 309 + 411 + 4\ 915 = 15\ 635 \text{ грн.} \quad (2.4)$$

2.5 Економічна ефективність застосунку

Економічна ефективность у застосунку відстутня, через відстутність комерційних інтеграцій або інтеграційних рекламний сервісів.

2.6 Соціальна ефективність застосунку

Оскільки розроблений веб-застосунок не є комерційним продуктом, а виконує функцію платформи для інтелектуального пошуку та спільного перегляду аніме-контенту, основною метою є досягнення соціального ефекту, а не прибутку.

До соціальної ефективності проєкту можна віднести такі аспекти:

1. Створення відкритої платформи для спільного перегляду аніме – користувачі можуть разом дивитись контент онлайн незалежно від місця перебування.
2. Доступ до релевантного контенту через інтелектуальний пошук – навіть неточні або неповні запити дозволяють знайти релевантне аніме завдяки ШІ.
3. Формування спільноти за інтересами – функції чату, рекомендацій, кабінету користувача сприяють соціальній взаємодії.
4. Популяризація аніме-культури серед молодіжної аудиторії, яка активно користується сучасними веб-ресурсами.
5. Інклюзивність – інтерфейс розроблено з урахуванням адаптивності та зручності для користувачів з різними пристроями та потребами.

Результатом впровадження веб-застосунку є не прибуток, а соціальний вплив, зокрема – забезпечення доступу до якісного аніме-контенту, розвиток онлайн-комунікацій та вдосконалення цифрової культури серед молоді.

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Основні положення

Під час виконання даного проєкту особливу увагу було приділено питанням охорони праці, оскільки робота здійснювалась переважно за комп'ютером. З метою забезпечення безпечних і комфортних умов праці було враховано чинні норми та рекомендації щодо організації робочого місця користувача ПК. У межах цього розділу розглянуто основні аспекти, що стосуються вимог до освітлення, мікроклімату, ергономічного розташування обладнання, дотримання норм електробезпеки та пожежної безпеки, а також рівня шуму в приміщенні. Усі ці чинники були враховані при створенні сприятливих умов для тривалої роботи за комп'ютером.

3.2 Негативні фактори під час роботи за комп'ютером

Тривала робота за комп'ютером може мати негативний вплив як на фізичний стан людини, так і на її психоемоційне здоров'я. Постійне перебування в одній позі спричиняє дискомфорт і біль у спині та шиї, порушує нормальний кровообіг, що, своєю чергою, погіршує живлення тканин, ослаблює судинні стінки і може призводити до їх деформації. Одноманітне навантаження на окремі групи м'язів часто викликає хронічні захворювання опорно-рухового апарату, а загальне зниження рухової активності – ризик розвитку ожиріння та порушень обміну речовин.

Тривалий зоровий контакт із монітором спричиняє перевтому очей: під час роботи відстань до зображення залишається сталою, тому м'язи ока змушені постійно напружуватись. Це може викликати погіршення зору. Окрім фізичних навантажень, значне розумове напруження викликає втому, зниження концентрації, а тривале перебування у цифровому середовищі негативно позначається на соціальній взаємодії. Такий стан може сприяти емоційному вигоранню, а робота у вечірній час – ще й порушенням сну через вплив синього світла.

3.3 Гігієнічні норми

3.3.1 Вимоги до приміщення

Для комфортного та безпечного розміщення обладнання було влаштоване робоче місце площею 9,0 м².

У приміщені, де організовано робоче місце з використанням комп'ютерної техніки, заборонено використовувати матеріали, здатні виділяти шкідливі хімічні речовини, зокрема деякі полімери. Тому стіни у такому середовищі були просто пофарбовані без застосування декоративних пластиків чи панелей. Колір стін відіграє важливу роль у формуванні сприятливої психологічної атмосфери та сприяє зниженню емоційного напруження. Яскраві або контрастні кольори, навпаки, можуть викликати швидку втому, зниження концентрації та подразнення, тому межах даного проєкту стіни робочого приміщення були обрані у пастельний тон, що відповідає санітарно-гігієнічним нормам.

Підлога приміщення має матове покриття з антиковзними та антистатичними властивостями, що знижує ризик травматизму і перешкоджає накопиченню електростатичного заряду.

3.3.2 Вимоги до робочого місця

Робочий стіл відповідає вимогам ергономіки і забезпечує оптимальне розміщення обладнання на робочій поверхні. Висота столу з комп'ютером становить 670 мм, ширина та глибина розраховані таким чином, щоб забезпечити можливість доступу до обладнання у межах досяжності. Під столом було влаштовано простір для ніг з висотою не менше ніж 600 мм, шириною не менше 500 мм і глибиною не менше 450 мм.

Робочий стілець був підібраний з регулюванням за висотою, кутом нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння. Також поверхня сидіння є плоскою, а передній край – заокругленим. Поверхня сидіння і спинки стільця напівм'яка з нековзним, повітронепроникним покриттям, що легко очищається і не електризується.

Монітор на робочій поверхні був розташовуваний на відстані 600-700мм від

користувача. Його положення забезпечує зручність зорового споглядання у вертикальній площині під кутом 30° до нормалі.

Клавіатура була розташовувана на відстані 190мм від краю. Висота середнього рядка клавіш не перевищує 30мм. Поверхня клавіатури матова з коефіцієнтом відбиття 0,4.

Мишка була розташовувана на відстані 210мм від краю. Форма миші ергономічна. Поверхня матова з коефіцієнтом відбиття 0,4.

Всі параметри можна побачити на рисунку 3.1.

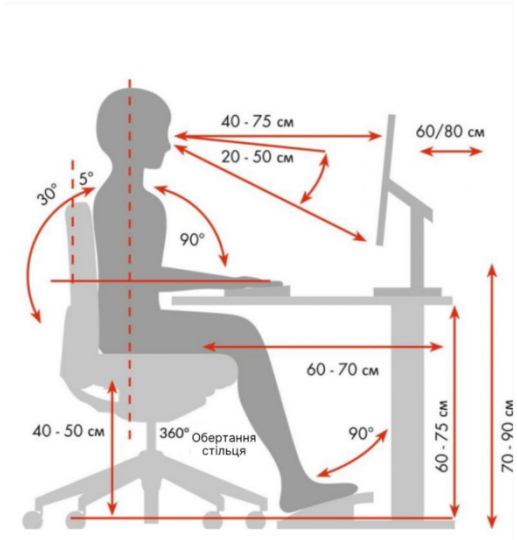


Рисунок 3.1. Параметри для робочого місця

3.3.3 Вимоги до освітлення

У приміщенні, де організовано робоче місце з комп'ютером, забезпечено достатній рівень природного освітлення, який відповідає нормативному значенню коефіцієнта природної освітленості – не менше ніж 1,5%. Для регулювання рівня освітлення природним світлом передбачено використання жалюзі. Робоче місце розташовуване так, щоб пряме сонячне світло не попадало в очі. Штучне освітлення приміщення було роблено у вигляді системи загального рівномірного освітлення. Рівень освітленості на робочому столі становить 300–500 лк.

3.3.4 Вимоги до шуму і вібрацій

Під час роботи рівень звуку не перевищував 45дБА. Такий рівень шуму забезпечує комфортні умови для тривалої розумової праці і не знижує

концентрацію уваги. Перевищення допустимих рівнів шуму може знижувати продуктивність праці, роздратування та підвищення стомлюваності.

3.3.5 Вимоги до мікроклімату

Приміщення з персональними комп'ютерами було обладнане системами опалення і кондиціонування повітря. Оптимальні значення параметрів мікроклімату у приміщенні становить 21–24°C температури повітря, 40–60% вологості повітря, і швидкості руху повітря – не більше 0,1 м/с. Також приміщення провітрюється кожні 1.5 години.

3.3.6 Електробезпека

Під час роботи з електричними пристроями необхідно суворо дотримуватися правил безпеки, щоб уникнути ураження струмом та інших небезпечних ситуацій. Перед початком роботи обов'язково проводиться візуальна перевірка технічного стану обладнання – особливу увагу приділяють цілісності кабелів, вилок, розеток та зовнішнього вигляду корпусу приладів. Усі виявлені пошкодження становлять потенційну небезпеку для здоров'я та життя робітника.

Після завершення роботи електроприлади обов'язково від'єднуються від мережі живлення, щоб запобігти короткому замиканню або перегріву. Не допускається використання електротехніки мокрими або вологими руками, оскільки це значно підвищує ризик ураження струмом. У випадку виявлення несправності пристрій слід негайно вимкнути з мережі, вилучити з експлуатації та передати на технічне обслуговування або ремонт.

3.4 Пожежна безпека

З метою попередження виникнення пожеж були вжиті необхідні заходи безпеки. Всі учасники розробки пройшли інструктаж з пожежної безпеки, а під час роботи суворо дотримувалися встановлених правил електробезпеки. Зокрема, уникалися перевантаження електромереж, не використовувалися пошкоджені або несправні прилади, обладнання не залишалося увімкненим без нагляду, а технічний стан розеток і подовжувачів регулярно перевірявся.

Приміщення було оснащене справними первинними засобами пожежогасіння – перевагу надано порошковим або вуглекислотним вогнегасникам, оскільки вони придатні для гасіння електрообладнання й не проводять електричний струм. Вогнегасник було розміщено у легкодоступному, зручному та безпечному місці, осторонь від джерел тепла. Поряд розміщувалася інструкція з його використання, а також контролювався термін придатності пристрою.

Окрім цього, для підвищення рівня безпеки було розроблено план евакуації, який було вивішено на видимих та доступних для всіх працівників місцях.

3.5 Основні заходи та засоби щодо збереження здоров’я

Щоб зберегти здоров’я виконувалися всі зазначені вище правила з охорони праці. Було дотримано режиму праці і відпочинку і робилися перерви по 5-10 хвилин. Виконувати вправи для очей і розминка для тіла, а у вільний час відвідувався спортивний зал.

Для зменшення впливу синього світла на очі на моніторі комп’ютера використовувався фільтр для зору (Night Shift), що прибирає синє світло і робить зображеня жовтим з більши теплішим відтінком.

Також кожен рік передбачено профілактичні медичні огляди у терапевта, офтальмолога та інших вузикх спецаілістів. Це дозволить вчасно виявити можливі порушення здоров’я та запобігти їх подальшому розвитку.

ВИСНОВКИ

У результаті виконання дипломної роботи було спроєктовано, реалізовано та протестовано веб-застосунок для інтелектуального пошуку та спільного перегляду аніме-контенту. Основна мета – розробка сучасного та інтерактивного інструменту, що поєднує функції персоналізованого пошуку з можливістю синхронного онлайн-перегляду – була досягнута.

У процесі роботи було проведено аналіз предметної області, розглянуто актуальні підходи до реалізації інтелектуального пошуку, опрацьовано наявні рішення для спільного перегляду мультимедійного контенту. Це дозволило сформувати чіткі вимоги до функціональності та обрати оптимальні інструменти для реалізації системи.

Технічно проєкт реалізовано за архітектурою клієнт-сервер, з використанням сучасних технологій: на фронтенді застосовано фреймворк Next.js у поєднанні з менеджером стану Zustand, а на сервері – Node.js з Express.js. Для зберігання даних використовується база MongoDB, доступ до якої здійснюється через бібліотеку Mongoose. Сесії авторизації реалізовано на основі express-session.

Особливу увагу приділено реалізації механізму спільного перегляду через Socket.IO, який дозволяє користувачам переглядати відео в режимі реального часу, синхронізуючи стан плеєра між учасниками кімнати. Також створено інтелектуальний механізм пошуку, який обробляє запити як до локальної бази, так і до стороннього API та до ШІ-моделі, забезпечуючи максимально точну й релевантну видачу результатів.

У процесі реалізації було дотримано принципів масштабованості, модульності та розширюваності коду. Система побудована таким чином, щоб легко підтримувати подальший розвиток, наприклад, додавання нових джерел даних, розширення функціоналу плеєра або впровадження нових мов для локалізації.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Порселло, Е., Бенкс, А. React: сучасні шаблони для розробки додатків : [пер. з англ.]. 2-е вид. – Київ : Видавництво «Діалектика», 2023. – 512 с.
2. Чорний, Б. Професійний TypeScript. Розроблення масштабованих JavaScript-додатків : навчальний посібник. – Харків : Видавництво «Фабула», 2022. – 448 с.
3. Routing: Parallel Routes | Next.js [Електронний ресурс] // Next.js Documentation – Офіційний сайт фреймворку Next.js. – Режим доступу: https://nextjs.org/docs/app/building-your-application/routing/parallel-routes – Назва з екрана. – Мова англійська. – Дата звернення: 15.05.2025.
4. light-dark() – CSS: Cascading Style Sheets | MDN [Електронний ресурс] // MDN Web Docs – Платформа Mozilla Developer Network. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/color_value/light-dark – Назва з екрана. – Мова англійська. – Дата звернення: 17.05.2025.
5. Socket.io Documentation [Електронний ресурс] // Socket.io – Офіційний сайт бібліотеки для роботи з WebSocket. – Режим доступу: https://socket.io/docs/v4/#what-socketio-is-not – Назва з екрана. – Мова англійська. – Дата звернення: 20.05.2025.
6. The WebSocket API (WebSockets) – Web APIs | MDN [Електронний ресурс] // MDN Web Docs – Офіційна документація Mozilla Developer Network. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API – Назва з екрана. – Мова англійська. – Дата звернення: 10.06.2025.

ДОДАТОК А. Фрагмент програмного коду клієнт-серверної логіки

```
// Server.ts

import { createServer } from 'node:http';
import next from 'next';
import { Server } from 'socket.io';

const dev = process.env.NODE_ENV !== 'production';
const hostname = 'localhost';
const port = 3000;
const app = next({ dev, hostname, port });
const handler = app.getRequestHandler();

import onLobbyMessage from './server/lobbyCommands.js';
import { joinRoom, leaveRoom } from './server/lobbyControls.js';

const rooms = new Map();

app.prepare().then(() => {
  const httpServer = createServer(handler);
  const io = new Server(httpServer);

  io.on('connection', (socket) => {
    joinRoom(socket, rooms);
    onLobbyMessage(socket, rooms);

    socket.on('send_state', ({ requesterId, videoUrl, timecode }) => {
      console.log(`[SERVER]: Received state for ${requesterId}`);
      io.to(requesterId).emit('state_update', { videoUrl, timecode });
    });

    leaveRoom(socket, rooms);

    socket.on('error', (error) => {
      console.error('Socket error:', error);
    });
  });

  httpServer
    .once('error', (err) => {
      console.error(err);
      process.exit(1);
    })
    .listen(port, () => {
      console.log(`> Ready on http://${hostname}:${port}`);
    });
});
// lobbyCommands.ts
import logWithTime from './log.js';

export default function onLobbyMessage(socket, rooms) {
  try {
    socket.on('message', (data) => {
      const { command, additional } = data;
      const roomCode = [...socket.rooms][1];
      if (roomCode) {
        logWithTime(`[Broadcasting in room ${roomCode}]: ${data}`);
        socket.to(roomCode).emit('message', { command, additional });
      } else {
```

```
        logWithTime(`No roomCode found for socket: ${socket.id}`);
      }
    });
    socket.on('chat_message', (data) => {
      const { command, additional, username, avatar } = data;
      const roomCode = [...socket.rooms][1];
      if (roomCode) {
        logWithTime(`[Broadcasting message in room ${roomCode}]: ${data}`);
        socket.to(roomCode).emit('chat_message', { command, additional, username, avatar
});
      } else {
        logWithTime(`No roomCode found for socket: ${socket.id}`, 'error');
      }
    });
  } catch (err) {
    logWithTime(`Failed to parse message: ${err.message}`, 'error');
  }
}

// lobbyControls.ts

import logWithTime from './log.js';

export function joinRoom(socket, rooms) {
  socket.on('join', ({ roomCode }) => {
    socket.join(roomCode);

    if (!rooms.has(roomCode)) {
      rooms.set(roomCode, { users: new Set(), videoUrl: null, timecode: 0 });
    }

    const room = rooms.get(roomCode);
    room.users.add(socket.id);

    logWithTime(`[ROOM ${roomCode}]: Socket ${socket.id} joined`);

    if (room.users.size > 1) {
      const firstUserId = [...room.users][0];
      socket.to(firstUserId).emit('request_state', { requesterId: socket.id, roomCode });
      logWithTime(`[ROOM ${roomCode}]: State requested from first user: ${firstUserId}`);
    }
  });
}

export function leaveRoom(socket, rooms) {
  socket.on('disconnect', () => {
    console.log(`[SERVER]: User ${socket.id} disconnected.`);

    for (const [roomName, users] of rooms.entries()) {
      if (users instanceof Set && users.has(socket.id)) {
        console.log(`[SERVER]: User ${socket.id} removed from room "${roomName}".`);

        if (users.size === 0) {
          rooms.delete(roomName);
          console.log(`[SERVER]: Room "${roomName}" deleted as it is now empty.`);
        }
        break;
      }
    }
  });
}
// usePlayer.ts
import { handleEpisode, i18n } from '@/utils/customUtils';
import FetchServiceInstance from '@/app/api';
```

```
import { useCallback, useEffect, useState } from 'react';
import { usePlayerStore } from '@/stores/playerHistory';

export const usePlayer = (slug: string | null) => {
  const [episodesList, setEpisodesList] = useState<EpisodeListInterface[] | null>(null);

  const [playerState, setPlayerState] = useState<playerStateInterface>({
    animeSlug: slug || '',
    episodeID: 0,
    chooseStudio: 0,
    studiosList: [],
    episodeUrl: null,
    episodeENTitle: '',
    episodeTitle: '',
    episodeJPTitle: '',
    isPlayerLoading: true,
  });

  useEffect(() => {
    const fetchEpisodes = async () => {
      if (!slug) return;
      const list = await FetchServiceInstance.fetchEpisodeList(slug);
      setEpisodesList(list);
    };
    fetchEpisodes();

    const { getEpisode } = usePlayerStore.getState();
    const lastEpisode = Number(getEpisode(slug || '')?.episodeID);
    const lastStudio = Number(getEpisode(slug || '')?.studio);
    if (lastEpisode) {
      setPlayerState((prev)  =>  ({  ...prev,  episodeID:  lastEpisode,  chooseStudio:
lastStudio }));
    }
    console.log(lastEpisode, lastStudio);
  }, [slug]);

  const handleStudio = useCallback(
    (index: number) => {
      setPlayerState((prevState) => ({ ...prevState, chooseStudio: index }));
    },
    [episodesList],
  );

  useEffect(() => {
    setPlayerState((prevState) => ({ ...prevState, isPlayerLoading: false }));
  }, [playerState.episodeUrl]);

  useEffect(() => {
    if (episodesList?.[0]?.id) {
      setPlayerState((prevState) => ({ ...prevState, studiosList: [] }));
      console.log(playerState);
      handleEpisode({
        playerState: setPlayerState,
        id: playerState.episodeID || episodesList[0].id,
        studio: playerState.chooseStudio,
        episodesList: episodesList[0],
        animeSlug: playerState.animeSlug,
      });
    } else {
      if (!playerState.isPlayerLoading) {
        setPlayerState((prevState) => ({
          ...prevState,
          episodeTitle: i18n.t('info.EpisodesNotFound'),
        }));
```

```
      }
    }
  }, [episodesList, playerState.chooseStudio]);

  return { playerState, setPlayerState, handleStudio, episodesList };
};

// usePlayerSocket.ts

import { useCallback, useEffect, useState } from 'react';
import { useSocket } from '../context/SocketContext';

const iframeCommands = {
  player_play: 'player_play',
  player_pause: 'player_pause',
  player_seek: 'player_seek',
};

const envCommands = {
  player_url: 'player_url',
  chat_message: 'chat_message',
} as const;

export const usePlayerSocket = (roomCode: string | null, iframe: HTMLIFrameElement | null)
=> {
  const socket = roomCode ? useSocket() : null;
  const [videoUrl, setVideoUrl] = useState<string | null>(null);
  let time = 0;

  if (!socket) {
    return {
      sendMessageToIframe: () => {},
      envCommands: { player_url: 'player_url', chat_message: 'chat_message' } as const,
      setVideoUrl: () => {}, // Default no-op function
    };
  }

  const sendMessageToIframe = useCallback(
    (messageToFrame: string, additional?: string | null) => {
      if (!iframe) {
        console.error('Iframe is not accessible');
        return;
      }

      if (!iframe?.contentWindow) {
        console.error('Iframe contentWindow is not accessible');
        return;
      }

      console.log('Sending message to iframe:', { command: messageToFrame, additional });
      const message = additional
        ? { command: messageToFrame, seek: Number(additional) }
        : { command: messageToFrame };
      iframe?.contentWindow?.postMessage(message, '*');
    },
    [iframe],
  );

  useEffect(() => {
    socket.emit('join', { roomCode });

    socket.on('message', (data: { command: string; additional?: string }) => {
      const { command, additional } = data;
      if (command in iframeCommands) {
```

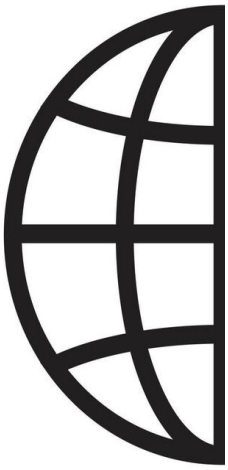
```
      sendMessageToIframe(command, additional || null);
    } else if (command === envCommands.player_url) {
      console.log(command, additional);
      iframe?.setAttribute('src', additional || '');
      document.querySelector('iframe')?.setAttribute('src', additional || '');
    }
  });

  socket.on('request_state', ({ requesterId }) => {
    console.log(`[CLIENT]: Received state request from ${requesterId}`);

    const videoUrl = iframe?.getAttribute('src') || null;
    const timecode = time;
    console.log(videoUrl, timecode);
    socket.emit('send_state', { requesterId, videoUrl, timecode });
  });

  socket.on('state_update', ({ videoUrl, timecode }) => {
    console.log(`@@@`, videoUrl, timecode);
    if (videoUrl) {
      document.querySelector('iframe')?.setAttribute('src', videoUrl);
    }
    if (timecode) {
      sendMessageToIframe(iframeCommands.player_seek, `${timecode}`);
    }
  });

  return () => {
    socket.off('message'); // Clean up listener
  };
}, [roomCode, sendMessageToIframe, iframe]);

useEffect(() => {
  const handleMessage = (event: MessageEvent) => {
    const message = event.data;
    if (typeof message === 'object' && socket.connected) {
      if (message.event === 'start' || message.event === 'play') {
        console.log('Emitting play command', message);
        socket.emit('message', { command: iframeCommands.player_play });
      }
      if (message.event === 'pause') {
        console.log('Emitting pause command', message);
        socket.emit('message', { command: iframeCommands.player_pause });
      }
      if (message.event === 'userseek') {
        console.log('Emitting seek command', message);
        socket.emit('message', {
          command: iframeCommands.player_seek,
          additional: `${message.data}`,
        });
      }
      if (message.event === 'time') {
        console.log(message.data);
        time = message.data;
      }
    } else {
      console.warn('Socket not connected, cannot emit');
    }
  };

  window.addEventListener('message', handleMessage);
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка веб-застосунку для інтелектуального пошуку та сумісного перегляду аніме-контенту

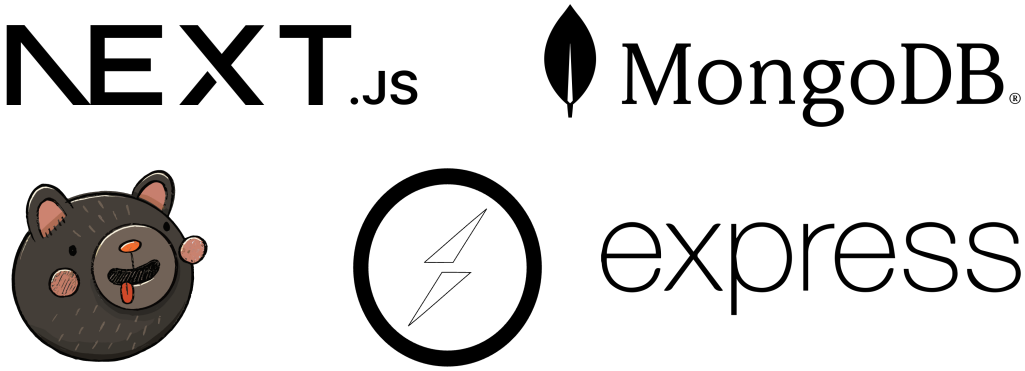
Дипломний проект студента групи 4РП-08:
Панка Кирила Дмитровича
Керівник: Жадан А. С.

Переваги проєкту



- Можливість зручно дивитись улюблений контент
- Синхронний перегляд та соціальна взаємодія
- Гнучний пошук

Вибір технологій



Клієнтська частина

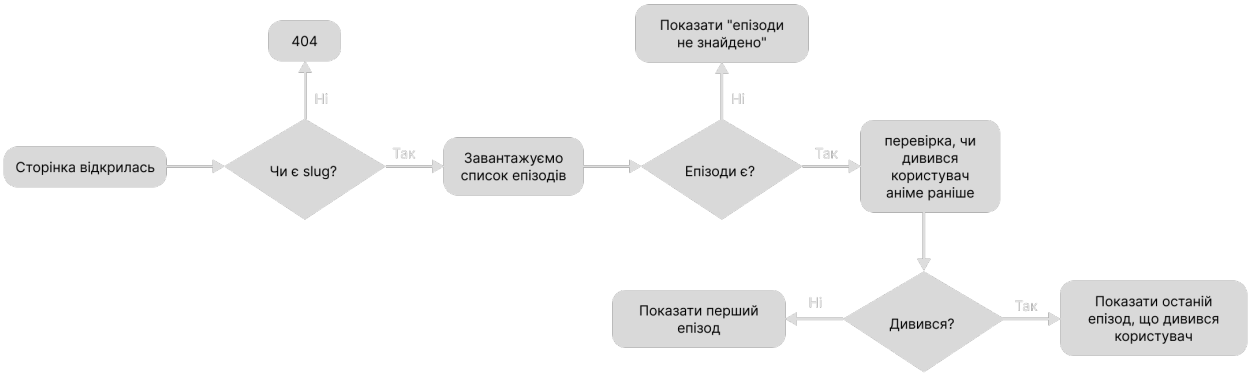
UI Компоненти
Відокремлені елементи інтерфейсу.

Кастомні Хуки
Перевикористовувана логіка.

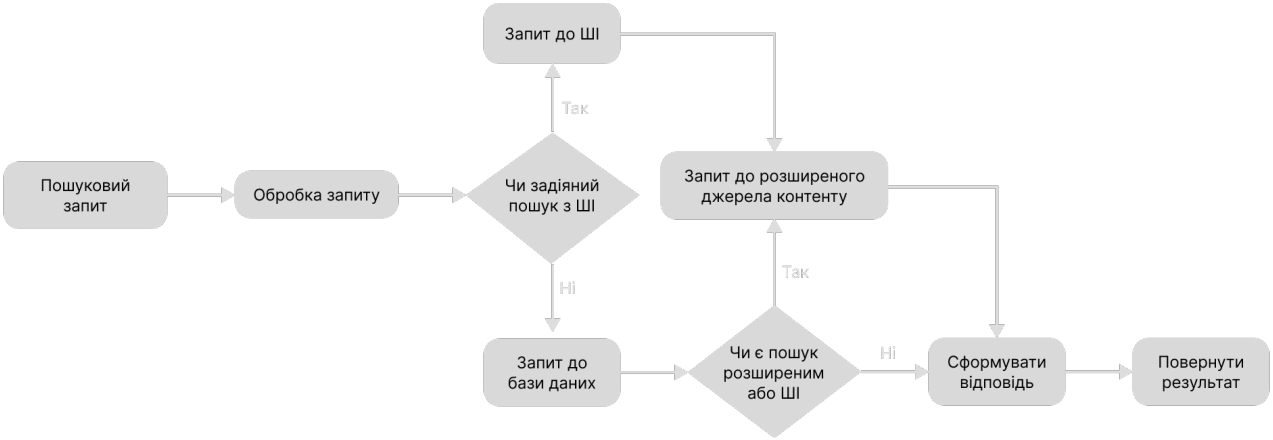
Логічні Компоненти
Інкапсулюють бізнес-логіку.

Глабільні стани
Відстутнісь пропс-дрілингу.

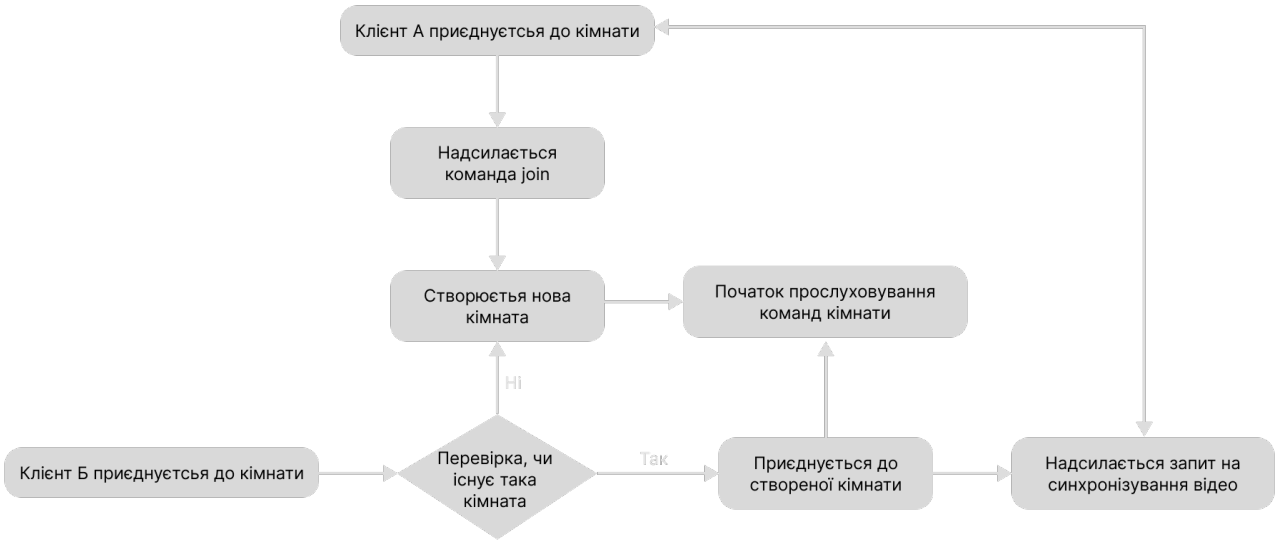
Реалізація хука usePlayer



Алгоритм пошуку



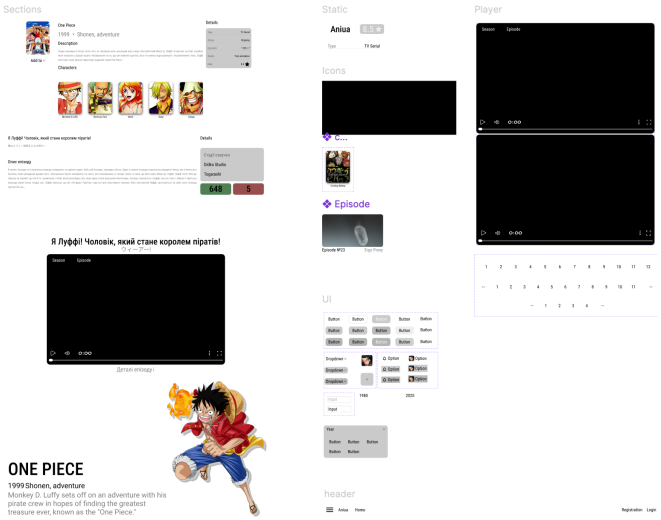
Алгоритм сумісного перегляду



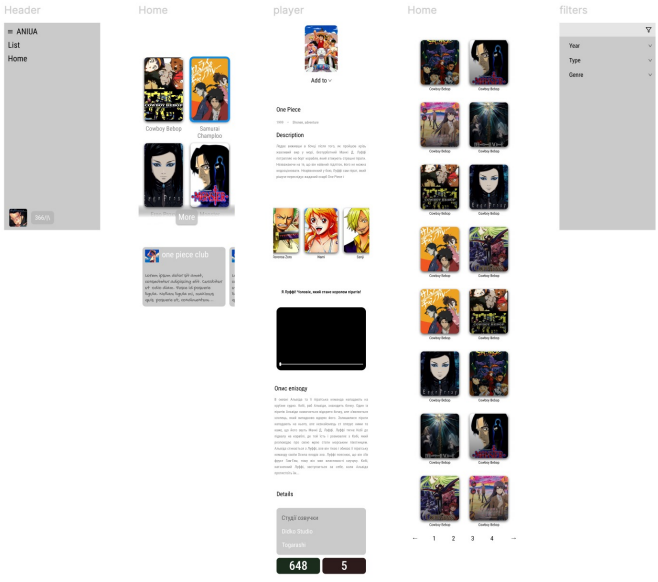
*За умови, що Клієнт А приєднується першим, а Клієнт Б - другим.

Алгоритм гідрації та авторизації користувача через Zustand

UI/UX дизайн | Дизайн мапа



UI/UX дизайн | Мобільний вигляд



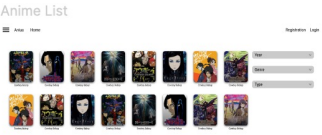
UI/UX дизайн | ПК вигляд основних сторінок

player

home

Profile

Anime List



Висновки

У ході розробки було створено сучасний веб-застосунок, що поєднує інтелектуальний пошук аніме та механізм сумісного перегляду в реальному часі. Реалізація клієнтської частини на базі Next.js і Zustand забезпечила гнучкість та високу продуктивність, а використання Socket.IO дозволило досягти ефективної синхронізації між користувачами. Інтеграція ШІ розширила функціональність пошуку, зробивши взаємодію з системою інтуїтивною та персоналізованою.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Панка Кирила Дмитровича

(прізвище, ім`я та по батькові)

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Жадан Артур Сергійович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме контенту*

Обсяг розрахунково-пояснювальної записки *76* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений проблемі інтелектуального пошуку та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 12 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Чітко описана архітектура клієнт-серверної взаємодії, з урахуванням паралельної маршрутизації та модульної структури.*

Впроваджено WebSocket-зв'язок через Socket.IO для реалізації реального часу.

д) основні недоліки дипломного проекту *Архітектура застосунку досить типова для Next.js-проєктів; відсутній власний підхід до масштабування, обробки помилок чи оптимізації. Синхронізація у реальному часі через WebSocket може створювати додаткове навантаження на сервер, особливо при великій кількості одночасних користувачів. Помилки у зображеннях блок-схем алгоритмів (наприклад, рис.1.12).*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис

«*20*» *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Панка Кирило Дмитрович

(прізвище, ім`я та по батькові)

Спеціальність: *121 " Інженерія програмного забезпечення"*

Освітньо-професійна програма: *«Розробка програмного забезпечення»*

Тема дипломного проєкту: *Розробка веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме-контенту*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЄКТУ

а) обсяг і якість виконання проєкту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проєкт виконано відповідно технічному завданню. Пояснювальна записка містить 76 сторінки. У пояснювальній записці виконано опис етапів розробки програмного забезпечення для веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме-контенту , а також його програмного забезпечення. Графічна частина складається з 12 слайдів мультимедійної презентації, які також містять схеми і скріншоти, передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.*

б) самостійність роботи над проєктом: *Протягом всього строку дипломного проєктування та переддипломної практики здобувач освіти Панка К.Д. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Панка К.Д. під час роботи над дипломним проєктом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою.*

Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проєкту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проєктування здобувач освіти мав змогу самостійно приймати рішення з реалізації алгоритмів і елемнетів веб-застосунку, що розробляв, та показав вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проєкту

Жадан Артур Сергійович

Місце роботи і посада керівника дипломного проєкту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії

Підпис

«16» грудня 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Панка Кирило Дмитрович,
здобувач освіти гр. 4РП-08, та

Жадан Артур Сергійович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка веб-застосунку інтелектуального-пошуку та сумісного перегляду аніме-контенту» (автор роботи – Панка К.Д., керівник роботи – Жадан А.С.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Панка К.Д. /

Керівник _____ / Жадан А.С. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Панки Кирила Дмитровича

на тему *Розробка веб-застосунку інтелектуального пошуку та сумісного перегляду аніме-контенту*

Висновок відповідальної особи за проведення нормоконтролю:

пояснювальна записка до дипломного проєкту виконана з некритичними порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 14.06.2025 р. значення коефіцієнту подібності в роботі становить 12,62%, коефіцієнт цитування – 1,49%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Панки К.Д.* (П.І.Б.)

проведена « *16* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/14/2025
Дата редагування 6/14/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка веб-застосунку інтелектуального пошуку та сумісного перегляду аніме-контенту

Автор
Панка Кирило Дмитрович
Науковий керівник / Експерт
Жадан Артур Сергійович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1

КЦ

25	14721	122004
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		3
Інтервали	A→	0
Мікропробіли		2
Білі знаки		0
Парафрази (SmartMarks)	a	65

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	101 0.69 %

2	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	72 0.49 %
3	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	65 0.44 %
4	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	57 0.39 %
5	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	52 0.35 %
6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	46 0.31 %
7	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	45 0.31 %
8	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	38 0.26 %
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/8da72e29-656f-4ee4-9b22-716dedf53ff5/content	37 0.25 %
10	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	36 0.24 %

з домашньої бази даних (3.54 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 **Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")**	**521 (15) 3.54 %**

з програми обміну базами даних (0.14 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Веб-застосунок для спільного програмування [redacted] 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	11 (1) 0.07 %
2	Kyrychenko_Oleynikov_Diplom_2025 [redacted] 6/8/2025 V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра кібербезпеки інформаційних систем, мереж і технологій)	10 (1) 0.07 %

з Інтернету (8.94 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	349 (35) 2.37 %
2	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	238 (22) 1.62 %
3	https://card-file.ontu.edu.ua/bitstreams/538ada8a-2c79-4b1e-b7d2-b0c97f68bc1c/download	127 (7) 0.86 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	100 (5) 0.68 %
5	https://card-file.ontu.edu.ua/bitstreams/9908b7a9-6b3e-46f5-a46e-84d83787cfd4/download	91 (7) 0.62 %
6	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	88 (3) 0.60 %
7	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	70 (7) 0.48 %
8	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	62 (2) 0.42 %
9	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	46 (3) 0.31 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/8da72e29-656f-4ee4-9b22-716dedf53ff5/content	37 (1) 0.25 %
11	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	32 (2) 0.22 %
12	https://github.com/tailwindlabs/tailwindcss/issues/13192	29 (1) 0.20 %
13	https://codereview.stackexchange.com/questions/286372/session-based-authentication-using-express-js	23 (1) 0.16 %
14	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	11 (1) 0.07 %
15	https://card-file.ontu.edu.ua/bitstreams/2129bb60-310e-4f6f-935d-31f4f403832e/download	7 (1) 0.05 %
16	https://card-file.ontu.edu.ua/bitstreams/f789da43-3034-4ad8-bf34-640a47414f93/download	6 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: **«Розробка програмного забезпечення»** **Група:** 4РП- 08

Дипломний **проект здобувача освіти денної форми навчання РП.** 08.16.000.ДП

ПАНКА
КИРИЛА ДМИТРОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: «Розробка програмного забезпечення»
Група: 4РП-08

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту на тему: