**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ ВІДЕО ЗАСОБАМИ VUE.JS ТА TMDB API»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
А.О. Хліпітько
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Хліпітьку Артему Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку для перегляду відео засобами Vue.js TMDb API

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування, реалізація та тестування вебзастосунку для перегляду відео.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 07.03.2024 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 29.03.2024 | |
| | | | |
| 2. | Збір вихідних даних. | 26.04.2024 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 31.05.2024 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 06.09.2024 | |
| | | | |
| 5. | Розробка третього розділу. | 13.11.2024 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 20.11.2024 | |
| | кваліфікаційної роботи магістра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 13.12.2024 | |

Студент _____ А.О. Хліпітько
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

## Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку для перегляду відео засобами Vue.js та TMDb API»: 62 с., 28 рис., 22 джерела.

ВЕБЗАСТОСУНОК, РЕКОМЕНДАЦІЙНА СИСТЕМА, CYPRESS, FIREBASE, TMDB API, VUE.JS.

Об’єкт дослідження: процес розробки вебзастосунку для перегляду відео.

Мета роботи: створення вебзастосунку для перегляду відео засобами Vue.js та TMDb API.

Методи дослідження: для досягнення поставленої мети був використаний підхід, що поєднує розробку фронтенду з використанням Vue.js та взаємодію з сервером за допомогою TMDb API. У процесі розробки було використано компонентний підхід Vue.js для покращення організації коду та забезпечення простоти у підтримці.

Результатом роботи є вебзастосунок для пошуку і перегляду відео, створений на основі Vue.js з інтеграцією даних через TMDb API. Основна увага приділялася аналізу вимог до функціональності, зокрема до зручності інтерфейсу та швидкого пошуку відеоматеріалів. У рамках роботи було досліджено можливості рекомендаційних систем для персоналізації контенту. Також реалізовано функції авторизації через Firebase, що забезпечує зберігання особистих налаштувань користувачів. Надійність застосунку перевірено засобами автоматизованого тестування за допомогою Cypress, що дозволило виявити і усунути помилки у відображенні та взаємодії з даними, підвищуючи стабільність системи. Розроблений застосунок може слугувати основою для подальших проєктів у сфері потокового відео та персоналізованого пошуку, демонструючи ефективність інтеграції Vue.js з TMDb API, а також застосування сучасних інструментів для забезпечення надійності й масштабованості системи.

# SUMMARY

Master’s qualifying paper “Development of a Web Application for Playing Video Using Vue.js and TMDb API”: 62 pages, 28 figures, 22 references.

WEB APPLICATION, RECOMMENDATION SYSTEM, CYPRESS, FIREBASE, TMDB API, VUE.JS.

The object of the study is the process of developing a web application for viewing videos.

The aim of the study is the creation of a web application for video viewing using Vue.js and the TMDb API.

The methods of research are a combined approach integrating frontend development with Vue.js and server interaction through the TMDb API. The development process employed Vue.js’s component-based approach to improve code organization and ensure ease of maintenance.

The result of the thesis is a web application for searching and viewing videos, built using Vue.js with integration of data via the TMDb API. The main focus was on analyzing functionality requirements, particularly user interface convenience and fast video material search. In the course of the work, the possibilities of recommendation systems for content personalization were explored. Additionally, authentication features were implemented via Firebase, ensuring the storage of user settings. The reliability of the application was tested using automated testing tools such as Cypress, which helped identify and fix issues with data display and interaction, improving the system’s stability. The developed application can serve as a basis for further projects in the field of video streaming and personalized search, demonstrating the effectiveness of the integration of Vue.js with the TMDb API, as well as the use of modern tools to ensure the reliability and scalability of the system.

**ЗМІСТ**

Завдання на кваліфікаційну роботу……..…….…..... 2
Реферат ..... 4
Summary ..... 5
Вступ..... 8
1 Аналіз предметної області..... 10
1.1 Огляд сучасних тенденцій у розробці вебзастосунків для перегляду відео ..... 10
1.2 Аналіз програмного забезпечення для розробки..... 13
1.3 Огляд рекомендаційних систем..... 17
1.3.1 Surprise ..... 20
1.3.2 RecBole..... 21
1.3.3 LightFM ..... 22
1.3.4 TensorRec ..... 22
1.4 Висновки до розділу 1 ..... 23
2 Проєктування вебзастосунку для перегляду відео ..... 24
2.1 Проєктний аналіз вебзастосунку ..... 24
2.2 Опис функціональних вимог..... 25
2.3 Опис нефункціональних вимог..... 28
2.4 UML-діаграми..... 29
2.4.1 Діаграма прецедентів (Use Case Diagram)..... 29
2.4.2 Діаграма послідовності (Sequence Diagram) ..... 31
2.4.3 Діаграма розгортання (Deployment Diagram)..... 33
2.5 ER-діаграма бази даних користувачів..... 34
2.6 Висновки до розділу 2 ..... 35
3 Програмна реалізація вебзастосунку ..... 37
3.1 Структура проєкту ..... 37
3.2 Основні компоненти, інтерфейс користувача та інтеграція з Firebase 38

3.2.1 Компонент головної сторінки (MovieList) ..... 38
3.2.2 Header ..... 40
3.2.3 Компонент деталей фільму (MovieDetail) ..... 41
3.2.4 Сторінка актора (ActorDetail) ..... 43
3.2.5 Роутинг (Vue Router) ..... 45
3.2.6 Аутентифікація користувачів через Firebase ..... 45
3.2.7 Збереження вподобаних фільмів у Firebase ..... 47
3.2.8 Профіль користувача ..... 49
3.3 Тестування додатку засобами Cypress ..... 53
3.4 Висновки до розділу 3 ..... 57
Висновки ..... 59
Перелік посилань ..... 60

# ВСТУП

У сучасному цифровому світі інтернет перетворився не лише на невичерпне джерел знань, але й на важливу платформу для розваг та навчання. Швидкі темпи розвитку технологій сприяють зростанню популярності вебзастосунків для перегляду відео, які стають улюбленим способом проведення вільного часу для мільйонів користувачів. У цьому контексті виникає актуальність створення вебзастосунків, які не лише надають доступ до відео контенту, а й надають користувачам зручний інтерфейс та нові можливості для організації та взаємодії з вмістом.

Мета даної роботи полягає в розробці вебзастосунку для перегляду відео за допомогою фреймворка Vue.js та використанням TMDb API, який забезпечує швидкий та зручний доступ до широкого спектру відео матеріалів.

Об’єктом дослідження є процес розробки вебзастосунку, який забезпечуватиме користувачам зручний та ефективний інтерфейс для перегляду відео. Предметом дослідження є технології Vue.js та TMDb API для розробки вебзастосунку для перегляду відео, включаючи вибір інструментів, розробку функціональності та інтеграцію з зовнішнім API.

Однією з ключових особливостей роботи є використання Vue.js для ефективної організації коду та покращення користувацького досвіду. Інтеграція TMDb API дозволяє отримувати актуальну інформацію про фільми, а Firebase забезпечує авторизацію та збереження персональних налаштувань. Використання Cypress гарантує надійність і стабільність вебзастосунку.

Практична значимість роботи полягає у створенні функціонального та інтуїтивно зрозумілого інструменту для перегляду відео онлайн.

Структурно робота складається з трьох розділів. У першому розділі проводиться аналіз предметної області, що охоплює огляд сучасних

тенденцій у веброзробці для відеоконтенту, дослідження рекомендаційних систем і аналіз вибраних технічних рішень. У другому розділі представлено проєктування застосунку, детально описані функціональні та нефункціональні вимоги, а також надано UML-діаграми для візуалізації архітектури та основних взаємодій. Третій розділ фокусується на програмній реалізації, включаючи інтеграцію з Firebase для зберігання вподобань користувачів, аутентифікацію, використання TMDb API для отримання інформації про відеоконтент і автоматизоване тестування засобами Cypress для забезпечення стабільності роботи застосунку.

Розроблений вебзастосунок може бути використаний як основа для подальших розробок або впроваджений для надання користувачам зручного доступу до відеоконтенту.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Огляд сучасних тенденцій у розробці вебзастосунків для перегляду відео

В сучасному цифровому світі відео-стрімінг став одним з найпопулярніших способів розваги та інформаційного споживання. Різноманітні сервіси, що представляють широкий спектр контенту, від розважальних відео до освітніх ресурсів, зробили цей формат надзвичайно привабливим для мільйонів користувачів у всьому світі. Це вплинуло на розвиток вебплатформ, що дозволяють користувачам переглядати відео безпосередньо в браузері або через спеціальні додатки на мобільних пристроях.

Зростання популярності відео-стрімінгу значно впливає на розвиток вебзастосунків, оскільки вони повинні пристосовуватися до змінних потреб та вимог користувачів. З одного боку, це заохочує розвиток нових технологій та функціоналу, таких як висока якість відео, інтерактивні функції, вдосконалені алгоритми рекомендацій та індивідуальні налаштування. З іншого боку, це ставить перед розробниками завдання забезпечити стабільну та ефективну роботу своїх платформ у вимогливих умовах сучасного медіа-споживання.

Зручний інтерфейс є однією з ключових вимог користувачів до вебзастосунків для перегляду відео. Користувачі хочуть, щоб навігація по платформі була інтуїтивно зрозумілою і пропонувала швидкий доступ до необхідного контенту. Наявність контенту також важлива – користувачі хочуть мати можливість переглядати відео в будь-який момент та в будь-якому місці, без зайвих обмежень. Можливість персоналізації є ще одним важливим критерієм, оскільки користувачі очікують індивідуального підходу до своїх інтересів та вподобань, щоб отримувати рекомендації та контент згідно своїх смаків.

Для задоволення потреб користувачів та ефективної розробки вебзастосунків, розробники мають великий вибір інструментів та програмного забезпечення. Ці програмні утиліти охоплюють усі аспекти від створення та редагування відео-контенту до розробки та підтримки вебзастосунків для його програвання.

Одним із основних інструментів потокового онлайн-відео та відтворення вебдодатками є YouTube API [1]. Це набір інтерфейсів програмування додатків, які компанія Google пропонує для взаємодії з YouTube. Використовуючи YouTube API, розробники можуть інтегрувати у своє вебплатформи функції YouTube, зокрема пошук, відтворення відео, доступ до метаданих та багато іншого. Це дає змогу створювати зручні та потужні платформи для потокового перегляду відео із використанням різноманітних можливостей YouTube.

Крім того, існують спеціалізовані бібліотеки та фреймворки для потокового відео, такі як Video.js та Plyr. Вони пропонують гнучкість у налаштуванні потокового відео на вебплатформах, підтримуючи різні формати відео, адаптивний дизайн та додаткові функції, такі як плавне відтворення і підтримка субтитрів.

Video.js – це популярний HTML5 відеоплеєр з відкритим вихідним кодом, який легко інтегрується у вебдодатки та підтримує безліч відеоформатів [2]. Його основні функції включають підтримку HTML5 відео та Flash fallback для старіших браузерів, адаптивний дизайн, який забезпечує зручний перегляд на різних пристроях, та підтримку плагінів для додаткової функціональності. Завдяки модульній архітектурі, Video.js дозволяє легко налаштовувати плеєр під потреби конкретного проєкту, забезпечуючи високу продуктивність та сумісність з сучасними веббраузерами. Активна спільнота розробників забезпечує регулярні оновлення та виправлення помилок, що робить цей плеєр надійним інструментом для відео-стрімінгу.

Іншим HTML5 медіаплеєром з відкритим вихідним кодом є Plyr, який підтримує різні типи медіа, включаючи відео та аудіо формати [3]. Plyr

пропонує простий та інтуїтивний інтерфейс, який можна легко налаштувати відповідно до вимог проєкту. Основні функції цього плеєра включають підтримку YouTube та Vimeo, адаптивний дизайн та легку інтеграцію у вебзастосунки. Plyr також підтримує субтитри та інші текстові накладки, що робить його привабливим для користувачів з особливими потребами. Plyr пропонує простий у використанні інтерфейс з мінімалістичним дизайном, який спрощує інтеграцію та може використовуватися для різноманітних додатків, починаючи від простих відео-програвачів і закінчуючи складними мультимедійними платформами.

Ще однією важливою особливістю сучасних вебзастосунків для перегляду відео є можливість використання бібліотек для адаптації плеєрів під різні типи контенту. Наприклад, HLS.js дозволяє відтворювати HLS (HTTP Live Streaming) відео у браузерах, що не підтримують цей формат безпосередньо, розширюючи можливості відтворення на різних пристроях [4]. Завдяки HLS.js, розробники можуть додавати до своїх застосунків підтримку потокового відео високої якості та забезпечити стабільний перегляд на всіх сучасних платформах.

Отже, вибір інструментів та готових рішень для розробки вебзастосунків для відео-стрімінгу залежить від конкретних вимог проєкту. Наявність різноманітних бібліотек, API та фреймворків дозволяє розробникам створювати як прості відеоплеєри, так і комплексні мультимедійні платформи з розширеними можливостями.

Серед найбільш популярних відео-платформ, які визначають сучасні тенденції у розробці вебзастосунків, можна виділити YouTube, Netflix, Hulu та Amazon Prime Video. Кожен з цих сервісів пропонує унікальні функції, що впливають на інтерфейси та функції вебзастосунків. Наприклад, YouTube дозволяє безкоштовно переглядати контент, але з рекламою, і накладає обмеження на перегляд оригінальних серіалів та фільмів. Netflix спеціалізується на високоякісному контенті та оригінальних продукціях, але вимагає платної підписки. Hulu, з іншого боку, пропонує можливість

перегляду нових епізодів телевізійних шоу в день їхнього виходу, хоча має обмеження щодо міжнародної доступності. Ці платформи допомагають визначати напрямки розвитку вебзастосунків для перегляду відео, створюючи прості у використанні інтерфейси, орієнтовані на швидкий доступ до контенту, зручну навігацію та адаптивний дизайн для різних пристроїв.

## 1.2 Аналіз програмного забезпечення для розробки

У сучасному цифровому середовищі розвиток вебзастосунків для перегляду відео базується на використанні різних програмних засобів та технічних рішень. Для забезпечення успішного функціонування та високої якості таких сервісів розробники використовують широкий спектр інструментів та технологій.

YouTube, як один з найбільших вебсервісів для відео-контенту, використовує численні технології для розробки та підтримки свого сервісу. Основними компонентами програмного забезпечення YouTube є вебсайт та мобільні додатки, розроблені з використанням HTML, CSS та JavaScript. Основні фреймворки включають AngularJS та Polymer для забезпечення динамічності та інтерактивності користувацького інтерфейсу. Основною мовою програмування, що використовується для бекенд-процесів та управління великими наборами даних на YouTube, є Python. Також використовуються інші мови програмування, адаптовані для специфічних завдань різних компонентів, включаючи C, C++, Go та Java. Вибір цих середовищ програмування відрізняється від компонента до компонента та від команди до команди, але вирішальним фактором є призначення для кінцевого користувача та серверна функціональність.

Для забезпечення потокової передачі відео YouTube використовує високоефективні сервери та CDN (Content Delivery Network), такі як Google

Global Cache, що дозволяє забезпечити стабільну та швидку передачу великого обсягу відео-контенту користувачам по всьому світу. Також YouTube надає свій API для взаємодії з контентом, що дозволяє розробникам інтегрувати функціонал YouTube у свої вебзастосунки. На фронтенді YouTube використовує JavaScript для створення динамічних та інтерактивних інтерфейсів, тоді як мобільні додатки розробляються з використанням Java для Android та Objective-C (або Swift) для iOS. Мови C та C++ використовуються для критично важливих компонентів, що забезпечують плавне відтворення відео та ефективну обробку [5].

Netflix, як провідний стрімінговий сервіс, використовує мікросервісну архітектуру, яка дозволяє розділити великий проєкт на невеликі, самодостатні компоненти. Основні технології включають Java, Scala та Node.js. Netflix активно використовує Spring Boot для створення мікросервісів та Apache Kafka для обробки потокових даних. Для обробки та передачі відео-контенту Netflix використовує власну платформу Open Connect, яка включає високоефективні сервери та CDN для забезпечення стабільної та швидкої доставки контенту. Важливою частиною інфраструктури є також використання алгоритмів машинного навчання та штучного інтелекту для персоналізації рекомендацій та покращення користувацького досвіду. Для побудови користувацького інтерфейсу Netflix використовує React.js, що забезпечує динамічність та інтерактивність [6].

Hulu також використовує сучасні технології та програмне забезпечення для розробки та підтримки своїх вебзастосунків. Як і Netflix, Hulu базується на мікросервісній архітектурі з використанням технологій, таких як Java, Node.js та Python для серверної частини. Hulu також використовує AWS (Amazon Web Services) для хостингу та обробки даних. Для фронтенду Hulu використовує React.js, що дозволяє створювати швидкий та інтуїтивно зрозумілий користувацький інтерфейс. Hulu також використовує CDN для забезпечення високоякісної та стабільної передачі відео-контенту користувачам. Крім того, важливою частиною програмного забезпечення

Hulu є алгоритми рекомендацій, які допомагають користувачам знаходити цікавий контент на основі їхніх переглядів та вподобань [7].

Усі ці вебсервіси для перегляду відео використовують ряд спеціалізованих технологій та програмного забезпечення для забезпечення якісного та стабільного функціонування своїх платформ. Основними технологіями є JavaScript-фреймворки (AngularJS, React.js), серверні мови програмування (Python, Java, Scala, Node.js), а також високопродуктивні сервери та CDN для забезпечення швидкої та стабільної передачі відео. Розробники цих сервісів постійно вдосконалюють свої технології та шукають нові способи покращення користувацького досвіду та продуктивності платформ.

В даній роботі було обрано стек технологій, до якого входять: Vue.js для фронтенду, Node.js для бекенду та FireBase для зберігання даних, а також для авторизації через Google Account. Крім того, використовується інтеграція з TMDb API для отримання даних про відеоконтент.

Vue.js – це прогресивний фреймворк для створення користувацьких інтерфейсів [8]. Його головні переваги включають:

– простота вивчення: завдяки простій та інтуїтивно зрозумілій структурі, Vue.js легко вивчати навіть новачкам;
– компонентний підхід: дозволяє розбити інтерфейс на незалежні, легко керовані компоненти, що спрощує розробку та тестування;
– висока продуктивність: завдяки використанню віртуального DOM Vue.js забезпечує високу продуктивність та швидкість рендерингу;
– велика спільнота та підтримка: широкий спектр плагінів, бібліотек та інструментів, розроблених спільнотою.

Node.js – це середовище виконання JavaScript, яке дозволяє виконувати код на сервері [9]. Основні переваги включають:

– асинхронна обробка: Node.js використовує неблокуючий, асинхронний I/O, що дозволяє обробляти велику кількість запитів з високою продуктивністю;

- єдиний стек мов: використання JavaScript для обох фронтенду та бекенду спрощує процес розробки та підтримки;
- велика кількість модулів: Node.js має велику кількість модулів та бібліотек, доступних через npm, що дозволяє швидко інтегрувати різноманітні функціональності.

Firebase – це хмарна платформа від Google, яка надає різні сервіси для розробників, включаючи базу даних, аутентифікацію та хостинг [10]. Основні переваги Firebase у цьому проєкті:

- Firebase Realtime Database: використовується для зберігання та синхронізації даних у реальному часі, це дозволяє швидко отримувати доступ до даних користувачів та оновлювати їх, що є важливим для функціональності вебзастосунку;
- Firebase Authentication: надає можливість реалізувати авторизацію користувачів через Google Account, цей функціонал забезпечує простий та безпечний вхід для користувачів вебзастосунку, що полегшує їх взаємодію з системою та доступ до персоналізованого контенту;
- легка інтеграція: Firebase легко інтегрується з іншими технологіями, такими як Vue.js та Node.js, що прискорює розробку та забезпечує зручність у роботі з даними.

TMDb API (The Movie Database API) надає доступ до великої бази даних фільмів, серіалів та іншого відео-контенту [11]. Використання цього API забезпечує наступні переваги:

- широкий вибір контенту: дозволяє отримувати дані про фільми, серіали, акторів та інші пов’язані дані;
- легкість інтеграції: простий і зрозумілий інтерфейс API дозволяє легко інтегрувати його у вебзастосунок;
- актуальність даних: постійне оновлення даних забезпечує актуальність інформації про відео-контент.

Cypress – це потужний інструмент для автоматизованого тестування вебзастосунків, що дозволяє писати тести для фронтенду, які виконуються безпосередньо в браузері. У цьому проєкті Cypress використовується для тестування функціональності користувацького інтерфейсу та взаємодії з вебзастосунком [12].

До основних переваг використання Cypress належить:

– легкість у використанні: Cypress має простий і інтуїтивно зрозумілий API, що дозволяє швидко почати писати тести навіть без глибоких знань у тестуванні;
– швидкість виконання: тести виконуються швидко завдяки тому, що Cypress запускається безпосередньо в браузері, що дозволяє швидко виявляти помилки;
– реалістичне тестування: Cypress дає змогу тестувати поведінку користувача в реальному браузері, що дозволяє перевіряти інтерфейси так, як це робить реальний користувач.

Таким чином, використання цих технологій у поєднанні з TMDb API створює можливості для розробки конкурентоспроможного та сучасного вебзастосунку, здатного задовольнити вимоги користувачів щодо перегляду та взаємодії з відео-контентом. Інтеграція з Cypress для автоматизованого тестування забезпечує високий рівень якості та стабільності вебзастосунку, дозволяючи ефективно перевіряти функціональність та взаємодію користувачів з інтерфейсом. Це дозволяє швидко виявляти та виправляти помилки, що підвищує надійність системи та зручність для користувачів.

## 1.3 Огляд рекомендаційних систем

Рекомендаційні системи стали невід’ємною складовою сучасних вебзастосунків для перегляду відео, оскільки саме вони сприяють ефективному пошуку нового контенту, адаптованого до вподобань та історії

переглядів кожного користувача. Їхнє використання дозволяє створити персоналізований досвід, підвищуючи задоволеність користувачів, залученість до платформи та час перебування на ній. Це, у свою чергу, позитивно впливає на популярність сервісу і його конкурентоспроможність. Основу сучасних рекомендаційних систем складають різноманітні алгоритми машинного навчання, зокрема collaborative filtering, content-based filtering, а також гібридні методи, що забезпечують найбільш точні рекомендації.

Collaborative Filtering (CF) є одним із найбільш поширених підходів у створенні рекомендацій. Цей метод заснований на аналізі вподобань великої кількості користувачів для виявлення схожостей між ними або контентом. У CF виділяють два основних підходи: User-Based CF та Item-Based CF. Підхід User-Based CF орієнтований на пошук користувачів зі схожими інтересами. Наприклад, якщо двоє користувачів дивляться багато однакових відео, система може запропонувати контент, який один з них вже бачив, а інший ще ні. Item-Based CF, натомість, зосереджується на аналізі схожості між елементами контенту на основі оцінок. Якщо користувач подивився певне відео, система може порекомендувати інші, які високо оцінювали ті самі користувачі. Обидва методи мають свої сильні і слабкі сторони: User-Based CF чудово працює в умовах активної спільноти користувачів, але стикається з проблемою «холодного старту» для нових користувачів. Item-Based CF краще пристосований для швидкого оцінювання нових елементів контенту, що робить його менш вразливим до проблеми «холодного старту» [13].

Content-Based Filtering (CBF) фокусується на аналізі характеристик самих елементів контенту. Цей метод аналізує метадані, як-от жанр, режисер, актори, ключові слова, і рекомендує відео, схожі на ті, що користувач дивився раніше. Наприклад, якщо користувач зазвичай обирає фільми одного жанру або з участю певного актора, система підбирає подібний контент. Content-Based Filtering має ключову перевагу, оскільки не залежить від історії переглядів інших користувачів і не стикається з проблемою «холодного старту» у випадку нових користувачів. Однак, цей підхід має свої обмеження:

він часто надає лише контент, схожий на вже відомий користувачу, і може пропускати нові або несподівані рекомендації [14].

Гібридні методи дозволяють поєднувати переваги Collaborative та Content-Based Filtering для отримання більш релевантних і точних рекомендацій [15]. Наприклад, система може спочатку використовувати collaborative filtering для визначення подібних користувачів, а потім content-based filtering для аналізу характеристик контенту, який вони оцінювали, і створення релевантних рекомендацій. Існує кілька підходів до комбінування цих методів: каскадний (використання одного методу для генерації початкового набору рекомендацій і уточнення його іншим), паралельний (генерація рекомендацій різними методами одночасно і подальше комбінування результатів) та метод зважування (надання різної ваги рекомендаціям, залежно від точності і надійності кожного з методів). Netflix є прикладом успішного застосування гібридних методів, де платформа аналізує історію переглядів, рейтинги та інші параметри для забезпечення точних рекомендацій.

Глибоке навчання стало важливою складовою сучасних рекомендаційних систем, надаючи змогу обробляти значні обсяги даних з метою складного аналізу і створення точних рекомендацій [16]. Використання нейронних мереж дозволяє рекомендаційним системам обробляти послідовні дані, такі як історія переглядів, та враховувати контекст і порядок відео. Наприклад, рекурентні нейронні мережі (RNN) та нейронні мережі з довготривалою пам’яттю (LSTM) дають змогу аналізувати послідовність переглядів, враховуючи їх контекст. Convolutional Neural Networks (CNN) використовуються для виявлення візуальних та аудіо характеристик контенту, що дозволяє рекомендаціям ґрунтуватися на візуальній подібності. Крім того, автоенкодери та варіаційні автоенкодери (VAE) сприяють зменшенню розмірності даних та виявленню латентних характеристик контенту, що допомагає у створенні рекомендацій на основі

прихованих патернів [17]. Усе це дозволяє рекомендаційним системам враховувати складні зв’язки і значно покращує релевантність та точність рекомендацій, що відповідають інтересам користувача.

Оскільки глибоке навчання надає нові можливості для створення ефективних рекомендаційних систем, важливо також звернути увагу на готові рішення, які допомагають розробникам реалізувати ці технології у своїх проєктах. Існує кілька потужних бібліотек, що спрощують побудову та тестування рекомендаційних систем, зокрема Surprise та RecBole. Ці інструменти не лише знижують час розробки, але й забезпечують різноманітність алгоритмів та моделей, які можуть бути використані для адаптації до конкретних потреб проєкту.

### 1.3.1 Surprise

Surprise – це бібліотека на Python, яка створена для розробки рекомендаційних систем [18]. Вона дозволяє використовувати різні алгоритми колаборативної фільтрації, завдяки чому легко створювати прототипи й тестувати їх ефективність.

Головні особливості Surprise – це підтримка таких алгоритмів, як SVD, KNN Basic та KNN With Means. У бібліотеці є доступ до популярних наборів даних, наприклад, MovieLens [19], а також можливість працювати з власними наборами. Для оцінки точності моделей передбачені інструменти для крос-валідації, що допомагає аналізувати якість рекомендацій.

З технічної точки зору, Surprise є зручним рішенням для швидкого створення прототипів рекомендаційних систем. Бібліотека підтримує різноманітні алгоритми, що дозволяє знайти оптимальний підхід під конкретні завдання. Завдяки інтеграції з іншими популярними Python-бібліотеками, процес роботи з даними стає простішим, а продуктивність системи – вищою.

### 1.3.2 RecBole

RecBole – це потужна бібліотека для побудови рекомендаційних систем на основі глибокого навчання. Вона підтримує понад 70 різних моделей рекомендаційних систем, включаючи Factorization Machines, Graph Neural Networks та Autoencoders, що дає змогу створювати складні та ефективні рекомендаційні системи. Це дозволяє вибирати найвідповідніші моделі для конкретних завдань, враховуючи потреби та характеристики даних [20].

Бібліотека підтримує різні типи даних, такі як рейтинги, послідовні дані та графові структури. Це робить її гнучким інструментом для роботи з широким спектром даних, що дозволяє адаптувати систему до специфічних вимог проєкту. Також, RecBole надає інструменти для крос-валідації, оцінки моделей та гіперпараметричної оптимізації, що допомагає підвищити точність і стабільність моделі на етапі тестування.

Ще однією важливою особливістю є інтеграція з популярними бібліотеками глибокого навчання, такими як PyTorch та TensorFlow. Це дозволяє розробникам без проблем інтегрувати RecBole в свої проєкти та використовувати всю потужність сучасних фреймворків для глибокого навчання, що спрощує розробку та масштабування моделей.

У технічному аналізі RecBole виглядає як потужний інструмент для створення рекомендаційних систем, здатний обробляти складні та великі набори даних.

Підтримка різних моделей та алгоритмів забезпечує гнучкість у виборі оптимального підходу, а інструменти для оцінки та оптимізації допомагають покращити точність та ефективність системи. Взаємодія з популярними бібліотеками глибокого навчання спрощує розробку, що є важливим для масштабування моделей у реальних умовах.

### 1.3.3 LightFM

LightFM є ще одним популярним рішенням для розробки гібридних рекомендаційних систем [21]. Вона підтримує як collaborative, так і content-based підходи, що робить її дуже адаптивною для створення складних рекомендаційних систем, які задовольняють різні вимоги. Це дозволяє розробникам модифікувати платформу для різноманітного набору функцій, поєднуючи переваги обох методів.

Одна з ключових переваг LightFM полягає в здатності використовувати розріджені дані, що є важливим для точності рекомендацій у випадку як невеликих, так і великих наборів даних. Ця характеристика сприяє зменшенню проблем неактивних елементів у рекомендаційних системах, що є часто виникаючою проблемою в реальних умовах, де набори даних можуть бути неповними або нерівномірно розподілені.

LightFM включає різноманітні алгоритми, такі як WARP, k-OS WARP, BPR та власний лог-лосс, що дозволяє вибрати найбільш ефективний метод для конкретної задачі. Ці алгоритми точно налаштовані для сортування вмісту, значно підвищуючи точність рекомендацій і гарантуючи їх відповідність користувачам.

З технічної точки зору, LightFM є дуже зручним для проєктів, що потребують обробки розріджених даних. Алгоритми, що використовуються в цій бібліотеці, допомагають покращити якість ранжування контенту, а гнучкість налаштування та оптимізації параметрів дозволяє досягти найефективніші результати для конкретної умови та завдання.

### 1.3.4 TensorRec

TensorRec – це бібліотека для створення рекомендаційних систем на основі TensorFlow, що дозволяє повністю налаштовувати структуру моделі відповідно до конкретних вимог. Вона підтримує як collaborative, так і

content-based рекомендації, що дає змогу створювати гнучкі системи для різних сценаріїв використання [22].

Однією з ключових переваг TensorRec є її модульна структура, яка дозволяє створювати індивідуальні елементи моделі. Це дає розробникам можливість змінювати модель відповідно до особливих вимог проєкту, забезпечуючи високу гнучкість та дозволяючи налаштовувати кожен етап рекомендаційного процесу.

TensorRec також підтримує тренування на GPU, що значно прискорює процес навчання моделей. Це дозволяє значно зменшити час, необхідний для розробки і тестування великих моделей, що є важливим фактором для проєктів з великими обсягами даних.

З технічної точки зору, TensorRec підходить для складних проєктів, що вимагають значної кастомізації моделей. Модульна структура дає змогу налаштовувати архітектуру моделей відповідно до конкретних потреб, а підтримка GPU дозволяє швидко навчати складні моделі, що робить TensorRec потужним інструментом для розробки систем рекомендацій.

## 1.4 Висновки до розділу 1

В даному розділі проведено аналіз предметної області вебзастосунків для перегляду відео. Розглянуто сучасні тенденції у цій сфері, включаючи вимоги користувачів до інтерфейсу, доступності контенту та персоналізації. Описано існуючі рішення та технічні аспекти розробки таких платформ на прикладі популярних сервісів, таких як YouTube, Netflix та Hulu. Порівняно їх функціональні можливості та технології, що використовуються для забезпечення високої якості та продуктивності. Представлено вибір технологій для розробки вебзастосунку, зокрема Vue.js, TMDb API та Firebase, та їх переваги для створення конкурентоспроможного продукту.

# 2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ ВІДЕО

## 2.1 Проєктний аналіз вебзастосунку

У цьому розділі проводиться детальний проєктний аналіз розробки вебзастосунку для перегляду відео з використанням технології Vue.js та інтеграцією з TMDb API. Метою аналізу є визначення архітектури застосунку, проєктування його основних компонентів, а також розробка UML-діаграм і ER-діаграми для бази даних.

Проєктний аналіз є важливим етапом розробки будь-якого програмного забезпечення, оскільки він дозволяє детально планувати всі аспекти проєкту, виявляти можливі проблеми на ранніх стадіях та забезпечувати більш ефективну реалізацію запланованих функцій. Важливість проєктного аналізу полягає в наступному:

- визначення вимог і обмежень: аналіз вимог користувачів допомагає зрозуміти, які функції повинні бути реалізовані у вебзастосунку, а також які обмеження існують для їхньої реалізації;
- створення архітектури системи: проєктний аналіз дозволяє створити чітку архітектуру системи, визначити основні компоненти, їхні взаємозв’язки та взаємодію (це включає як фронтенд, так і бекенд частини застосунку, базу даних та інтеграцію з зовнішніми сервісами, такими як TMDb API);
- планування розробки: добре структурований проєктний аналіз дозволяє ефективно планувати процес розробки, а також визначати пріоритети для різних етапів роботи;
- покращення якості програмного забезпечення: використання UML-діаграм, ER-діаграм та інших засобів моделювання допомагає забезпечити високу якість програмного забезпечення за рахунок кращого розуміння структури системи та її компонентів (це сприяє

зменшенню помилок та покращенню підтримки і розширюваності коду);

– ефективне використання ресурсів: проєктний аналіз дозволяє оптимізувати використання часових та технічних ресурсів, що є критичним для успішного завершення проєкту у встановлені терміни;

Таким чином, проєктний аналіз є ключовим етапом у розробці вебзастосунку, який забезпечує основу для успішної реалізації проєкту, підвищення якості програмного забезпечення та ефективного управління ресурсами. У цьому розділі буде детально розглянуто всі аспекти проєктного аналізу, необхідні для створення функціонального та зручного вебзастосунку для перегляду відео з використанням Vue.js та TMDb API.

## 2.2 Опис функціональних вимог

Для успішної реалізації проєкту були визначені наступні функціональні вимоги. Кожна з них спрямована на забезпечення користувачів зручними інструментами для пошуку, перегляду та взаємодії з фільмами, а також персоналізованого досвіду через систему рекомендацій, а саме:

а) пошук фільмів:

1) користувач повинен мати можливість здійснювати пошук фільмів за ключовими словами (назва фільму, ім’я актора тощо). Пошук здійснюється за допомогою інтеграції з базою даних TMDb API, що дозволяє отримати список релевантних результатів;
2) фільми, які не мають зображень або інших необхідних метаданих, не відображаються у результатах пошуку. Функція пошуку має включати підтримку пагінації, щоб користувач міг переглядати велику кількість результатів;

б) вибірка за жанром:

1) користувач може здійснювати фільтрацію фільмів за певними жанрами (наприклад, комедія, драма, фантастика тощо);

2) система повинна пропонувати список доступних жанрів на основі даних TMDb API та відображати лише ті фільми, які відповідають вибраному жанру;

в) перегляд детальної інформації про фільм:

1) для кожного фільму користувач може переглядати детальну інформацію, яка включає: оригінальну назву фільму, опис, рейтинг, дата виходу, жанри, інформація про акторів та режисерів;

2) вся інформація завантажується через TMDb API та відображається у структурованому вигляді, щоб користувач міг легко знайти потрібну інформацію;

г) перегляд трейлера фільму:

1) на сторінці з детальною інформацією про фільм користувач може переглянути трейлер до фільму;

2) інтеграція з відеосервісами (наприклад, YouTube) дозволяє відображати трейлер безпосередньо на сторінці фільму, не покидаючи вебзастосунок;

д) відображення рекомендацій:

1) після перегляду детальної інформації про фільм користувачеві надаються рекомендації схожих фільмів, ці рекомендації базуються на жанрі, режисерові, акторах або інших характеристиках поточного фільму та отримуються через TMDb API;

е) створення профілю користувача:

1) користувач повинен мати можливість створити власний профіль, який міститиме особисту інформацію та дані про вподобані фільми;

2) профіль користувача зберігається в Firebase Realtime Database (у профілі можна змінювати особисті налаштування, додавати або видаляти фільми зі списку вподобаних, а також переглядати персональні рекомендації);

3) користувач може редагувати свій профіль, а саме завантажувати нове зображення профілю (зображення завантажується та зберігається в Firebase Storage, після чого автоматично оновлюється на профільній сторінці користувача, під час зміни зображення використовується асинхронна обробка файлів для забезпечення швидкого завантаження та відображення нової аватарки без необхідності оновлення сторінки);

є) авторизація через Google Account:

1) авторизація користувачів здійснюється через Google Account за допомогою Firebase Authentication (це забезпечує простий і безпечний вхід, зручний для користувачів, оскільки не потрібно створювати новий обліковий запис для користування вебзастосунком);

ж) власні рекомендації на основі вподобаних фільмів:

1) на основі фільмів, які користувач додав до списку «вподобаних», система генерує персоналізовані рекомендації (використовуючи Firebase Realtime Database та TMDb API, система підбирає схожі фільми та відображає їх у профілі користувача у форматі горизонтального слайдера);

2) фільми, які вже є у списку «вподобаних», виключаються з рекомендацій;

з) відстеження активності користувачів:

1) у вебзастосунку зберігається інформація про фільми, які користувач вподобав (це дозволяє створювати історію активності, яка може бути використана для надання користувачеві більш релевантних рекомендацій);

и) синхронізація даних у реальному часі:

1) завдяки використанню Firebase Realtime Database усі дії користувачів (створення профілю, додавання вподобаних фільмів, тощо) автоматично зберігаються і синхронізуються з сервером у реальному часі – це дозволяє користувачам бачити зміни на будь-якому пристрої миттєво, без необхідності оновлювати сторінку.

## 2.3 Опис нефункціональних вимог

Нефункціональні вимоги описують критерії, які визначають, як система повинна працювати, а не що вона повинна робити. Вони охоплюють різноманітні аспекти, такі як продуктивність, безпека, надійність та зручність використання. Для даного вебзастосунку визначено наступні нефункціональні вимоги.

*Продуктивність*. Вебзастосунок повинен забезпечувати швидке завантаження сторінок та їх компонентів. Час завантаження основної сторінки не повинен перевищувати 2 секунд на стандартних з’єднаннях. Пошук фільмів і отримання рекомендацій повинні відбуватися за рахунок асинхронних запитів, що забезпечує миттєвий відгук користувача. Використання фреймворку Vue.js дозволяє оптимізувати продуктивність завдяки реактивному управлінню станом додатку та компонентному підходу, що зменшує час на рендеринг.

*Безпека*. Система повинна гарантувати безпеку даних користувачів, зокрема під час авторизації через Google-аккаунт за допомогою Firebase Authentication. Всі облікові дані користувачів не зберігаються на сервері, а обробляються через безпечні протоколи Firebase, такі як HTTPS та OAuth 2.0.

*Надійність*. Вебзастосунок повинен бути надійним, з низьким рівнем помилок. У разі виникнення проблем користувач повинен отримувати

зрозумілі повідомлення про помилки, а система повинна відновлюватися без втрати даних. Системні збої не повинні впливати на доступність основних функцій.

*Зручність використання*. Інтерфейс користувача повинен бути інтуїтивно зрозумілим і зручним для використання. Вебзастосунок повинен бути доступним на різних пристроях, включаючи мобільні телефони та планшети, з адаптивним дизайном, що забезпечує коректне відображення на різних розмірах екранів. Використання CSS Grid та медіа-запитів у стилях забезпечує адаптивність інтерфейсу, дозволяючи комфортно переглядати контент на різних пристроях.

*Документованість*. Весь код повинен бути документованим, з детальними коментарями, які пояснюють ключові рішення та функції. Це спростить подальше обслуговування та розвиток проєкту.

Таким чином, нефункціональні вимоги є важливими для забезпечення якості вебзастосунку, його продуктивності та безпеки, що у свою чергу вплине на загальний досвід користувача.

## 2.4 UML-діаграми

У рамках проєктування вебзастосунку було створено ряд UML-діаграм. Ці діаграми допомагають у розумінні зв’язків між компонентами системи та візуалізують, як дані зберігаються, обробляються та використовуються.

### 2.4.1 Діаграма прецедентів (Use Case Diagram)

Ця діаграма ілюструє різні сценарії використання системи, що дозволяє зрозуміти взаємодію користувачів (актора) з вебзастосунком (див. рис. 2.1). Основним актором є користувач, який має можливість авторизуватися через

Google, що забезпечує безпечний доступ до персоналізованого контенту. Після успішної авторизації користувач може переглядати свій профіль, де відображаються його особисті дані, включаючи список вподобаних фільмів, а також редагувати ці дані, змінюючи, наприклад, профільне зображення або інші налаштування.




Рисунок 2.1 – Діаграма прецедентів

На основі вподобань та історії переглядів система пропонує користувачу персоналізовані рекомендації фільмів. Користувач може переглядати ці рекомендації та обирати фільми для подальшого перегляду. Крім того, він має можливість фільтрувати фільми за різними жанрами, такими як драма, комедія, трилер тощо. Система забезпечує зручний інтерфейс для вибору жанру, що дозволяє швидко знаходити потрібні фільми.

Користувач також може ввести назву фільму в поле пошуку. Система виконує асинхронний запит до бази даних і надає результати, що відповідають критеріям пошуку. Коли користувач натискає на конкретний фільм, він має змогу переглянути деталі, включаючи опис, трейлер, рейтинг та акторський склад, що забезпечує повну інформацію про обраний фільм.

Крім того, користувач може зберігати фільми у своєму списку вподобаних для подальшого перегляду, що дозволяє йому зберігати цікаві

фільми і швидко знаходити їх у майбутньому. Кожен із цих прецедентів описує важливі функції системи, які дозволяють користувачеві ефективно взаємодіяти з вебзастосунком, задовольняючи його потреби у перегляді та управлінні відеоконтентом.

### 2.4.2 Діаграма послідовності (Sequence Diagram)

Діаграма послідовності для сценарію перегляду фільму, що включає всі дії, які виконуються після виклику методу *ViewMovie()*. Діаграма показує, як система отримує деталі фільму, кредитні дані, трейлер і рекомендації, а також перевіряє, чи сподобався фільм користувачу (див. рис. 2.2).

Спочатку перевіряється статус авторизації через AuthService. Після отримання даних про користувача, запитуються дані про фільм, акторів, трейлер та рекомендації з API. Далі, перевіряється, чи сподобався фільм користувачу, звертаючись до Database.

Коли дані про фільм завантажені, інтерфейс користувача (UI) оновлюється, щоб відобразити деталі фільму, акторів, трейлер та рекомендації. Користувач переглядає всю інформацію про фільм, а потім може натиснути на актора, щоб дізнатися більше про нього. У цей момент сайт викликає метод, який перенаправляє на детальну інформацію про актора, запитуючи відповідні дані з API та оновлюючи UI для відображення деталей актора.

Коли користувач натискає кнопку “Like”, MovieDetail знову перевіряє статус авторизації. Після цього компонент переключає статус “Like” у Database, а UI оновлюється відповідно до нового статусу.

Останнім кроком є вибір користувачем рекомендованого фільму. MovieDetail переходить до деталей цього фільму, запитує нові дані з API та оновлює UI, щоб відобразити нову інформацію про фільм.

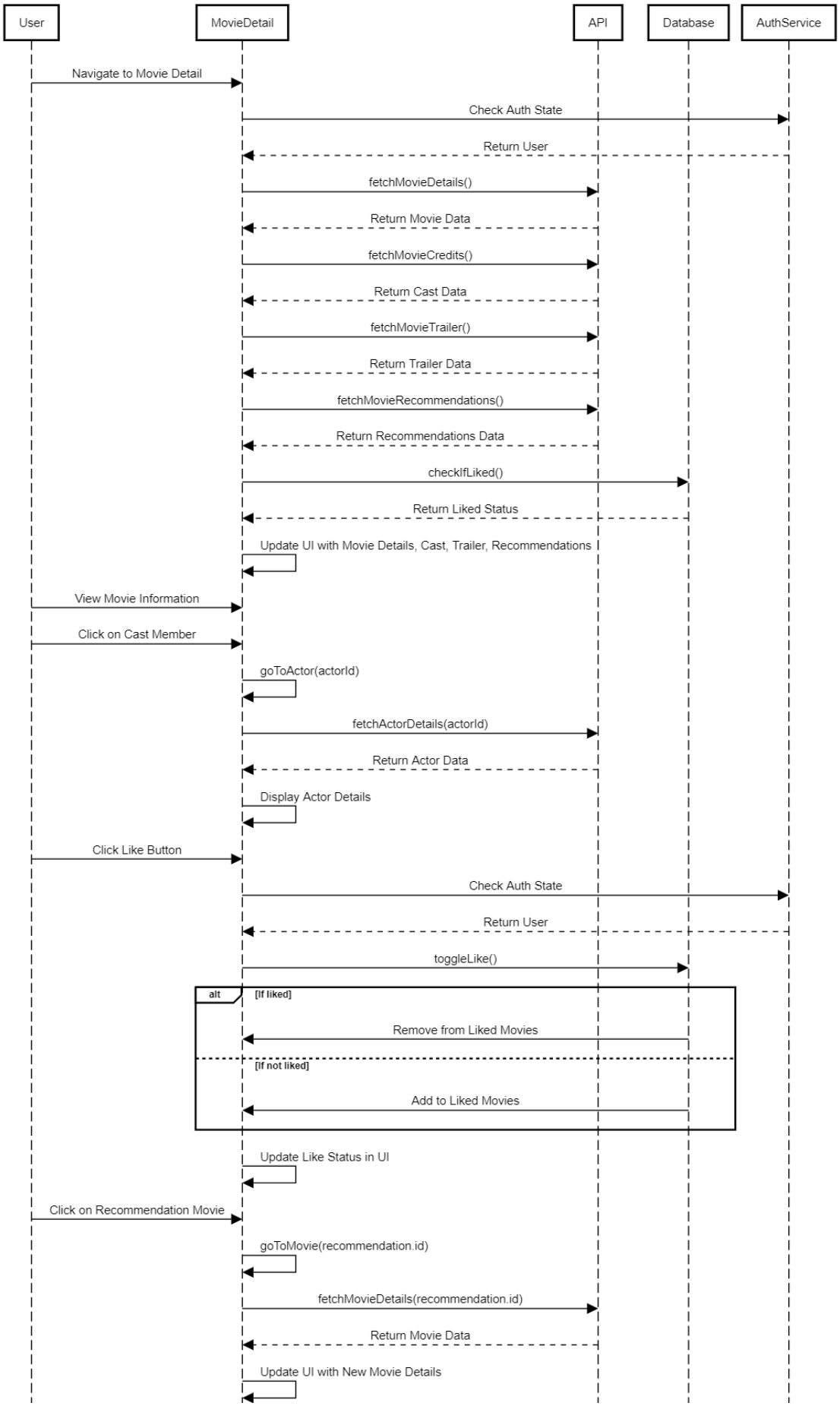



Рисунок 2.2 – Діаграма послідовності

### 2.4.3 Діаграма розгортання (Deployment Diagram)

У даній діаграмі розгортання представлено архітектуру системи, що складається з трьох основних компонентів: пристрою користувача, вебсервера та хмари Firebase (див. рис. 2.3).

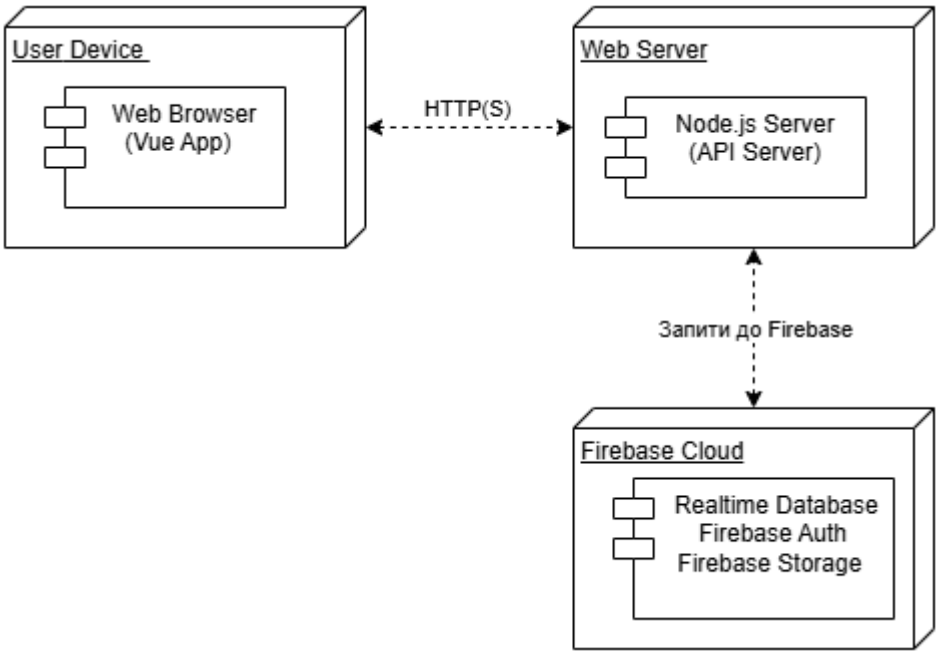



Рисунок 2.3 – Діаграма розгортання

User Device – це пристрій, який використовує користувач (ноутбук, комп’ютер або телефон), на якому працює веббраузер. У цьому браузері реалізований вебдодаток на основі Vue, що дозволяє користувачу взаємодіяти з системою, виконуючи різноманітні дії, такі як перегляд фільмів, авторизація та управління профілем.

Web Server – це вебсервер, на якому розгортається сервер API, реалізований за допомогою Node.js. Цей сервер обробляє запити користувачів, взаємодіє з базою даних і повертає результати запитів до вебдодатка.

Firebase Cloud – у цій частині системи використовується Firebase для управління даними та аутентифікації. Firebase включає Realtime Database для

зберігання даних, Firebase Authentication для управління аутентифікацією користувачів, а також Firebase Storage для зберігання зображень профілю. Ці компоненти працюють разом, щоб забезпечити безпечне та ефективне зберігання даних та підтримку функцій, таких як управління користувачами та зберігання контенту.

## 2.5 ER-діаграма бази даних користувачів

Під час реалізації проєкту було спроєктовано ER-діаграму для бази даних Firebase Realtime Database, яка використовується для зберігання інформації про користувачів та їхні улюблені фільми (див. рис. 2.4). Основною метою проєктування цієї діаграми було забезпечення ефективної організації даних для зручного доступу та масштабованості.

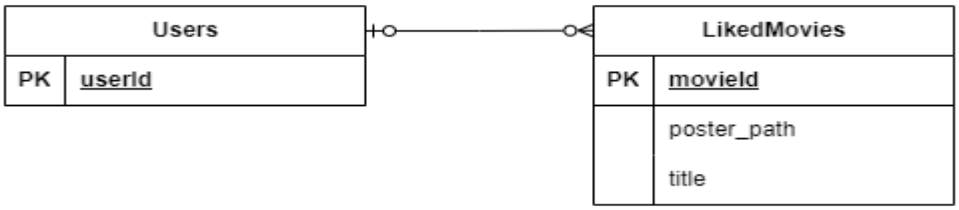



Рисунок 2.4 – ER-діаграма

*Сутність Users.* Кожен користувач в базі даних ідентифікується за унікальним ключем, згенерованим за допомогою Firebase Authentication, що дозволяє створити надійний і унікальний ідентифікатор для кожного користувача (наприклад, *wc7ukovsWUMzAW2vdETfTlvpPas1*). Цей ідентифікатор створюється автоматично під час реєстрації користувача в системі, що гарантує його унікальність і безпеку. В рамках кожного користувача зберігається вкладений об’єкт likedMovies, який містить інформацію про фільми, що були додані до улюблених.

*Сутність LikedMovies.* Ця сутність містить інформацію про фільми, які користувач позначив як улюблені. Кожен фільм також ідентифікується за

унікальним movieId, який виступає ключем. Структура кожного фільму включає наступні атрибути:

– movieId – унікальний ідентифікатор фільму;
– poster_path – шлях до постеру фільму;
– title – назва фільму.

Взаємозв’язки:

– Users пов’язана з LikedMovies у відношенні «один до багатьох», оскільки один користувач може мати багато улюблених фільмів;
– фільми можуть повторюватись у різних користувачів, що відображає реальну ситуацію, де один і той самий фільм може бути улюбленим у кількох людей.

Приклад:

а) користувач з ідентифікатором `*wc7ukovsWUMzAW2vdETfTlvpPas1*` додав до улюблених фільми, такі як:

1) The Lord of the Rings: The Fellowship of the Ring (ID: 120);
2) The Godfather (ID: 238);
3) Avatar (ID: 19995);

б) інший користувач, з ідентифікатором `*zGSiADR8dXQ2qkat4bAcCDbxsmT2*`, має улюблені фільми, серед яких:

1) Deadpool & Wolverine (ID: 533535);
2) Inside Out 2 (ID: 1022789).

Ця структура дозволяє легко масштабувати базу даних та додавати нові фільми для користувачів, зберігаючи дані про їхні уподобання.

## 2.6 Висновки до розділу 2

У цьому розділі проведено комплексний аналіз предметної області вебзастосунків для перегляду відео, визначено основні технології та методи розробки. Розглянуто важливість проєктного аналізу, який включає

визначення вимог, побудову архітектури системи та планування розробки.

У рамках розробки вебзастосунку були побудовані UML-діаграми, які деталізують структуру та взаємодію основних компонентів системи, включаючи користувачів, фільми та їх взаємозв’язки. Ці діаграми допомогли формалізувати процеси обробки даних та взаємодії з зовнішніми API, що сприяло кращому розумінню архітектури системи. Також створено ER-діаграму, яка моделює структуру бази даних у Firebase Realtime Database, відображаючи логічні зв’язки між сутностями, такими як користувачі та їх улюблені фільми. Завдяки цьому вдалось забезпечити ефективне зберігання та обробку даних, що сприяє підвищенню продуктивності та масштабованості системи.

Значну увагу приділено визначенню функціональних вимог до вебзастосунку, що дозволяють забезпечити ефективну взаємодію з користувачем. Реалізація цих вимог сприятиме створенню системи, яка не тільки відповідатиме актуальним потребам користувачів, але й забезпечить високу якість обслуговування, що, в свою чергу, сприятиме підвищенню задоволеності та лояльності користувачів.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

## 3.1 Структура проєкту

Цей розділ присвячений опису структури проєкту вебзастосунку, який розроблений з використанням фреймворку Vue.js для фронтенду та Firebase для бекенду. Структура проєкту організована таким чином, щоб забезпечити максимальну зручність у навігації, управлінні кодом та підтримці проєкту в майбутньому.

Основні папки і файли проєкту розподілені на дві основні частини: фронтенд і бекенд. Папка *frontend* містить усі файли, необхідні для відображення користувацького інтерфейсу та логіки, що обробляється на стороні клієнта. Бекенд-частина реалізована за допомогою Firebase, який включає налаштування для зберігання даних у Firebase Realtime Database, автентифікацію та інші серверні функції.

Фронтенд-частина організована в декілька основних папок:

– src – коренева папка застосунку, де зберігається весь вихідний код. Вона включає компоненти, маршрути, сховище стану та інші функціональні частини;
– components – містить окремі Vue-компоненти, що є основними будівельними блоками інтерфейсу (сюди входять модульні частини інтерфейсу, такі як картки фільмів, навігаційне меню, панелі профілю тощо);
– router – містить налаштування маршрутизації, що відповідає за навігацію між сторінками застосунку, включаючи головну сторінку, пошук, сторінки з рекомендаціями, деталі фільмів і профіль користувача.

Для налаштування оточення та керування залежностями використовується Node Package Manager (npm). Залежності Vue.js були

встановлені через Vue CLI – інструмент командного рядка, який дозволяє створювати й керувати проєктами Vue.js. Крім того, Firebase інтегрований у проєкт для обробки автентифікації та зберігання даних користувачів. Це включає пакет *firebase*, який дозволяє взаємодіяти з Firebase Realtime Database, обробляти автентифікацію через Google та інші функції, необхідні для бекенд-частини.

Чітка структура проєкту забезпечує підтримку чистої архітектури, зручність в розширенні функціоналу та обслуговуванні застосунку, а використання Firebase спрощує роботу з даними та забезпечує масштабованість.

## 3.2 Основні компоненти, інтерфейс користувача та інтеграція з Firebase

Застосунок побудовано з використанням декількох основних компонентів, які забезпечують зручну та інтуїтивну взаємодію користувача з платформою. Кожен компонент відіграє свою роль у створенні цілісного та функціонального інтерфейсу, який адаптований під різні потреби користувача. Це дозволяє створити цілісну архітектуру, де фронтенд та бекенд ефективно доповнюють один одного, забезпечуючи безперебійний досвід для користувача.

### 3.2.1 Компонент головної сторінки (MovieList)

Головна сторінка застосунку (*MovieList.vue*) відображає список популярних або нових фільмів, використовуючи дані, отримані з TMDb API. Компонент отримує дані за допомогою асинхронних запитів до API і динамічно оновлює вміст, забезпечуючи швидке відображення списку

фільмів для користувача. Дані фільмів зберігаються в стані компонента, а потім виводяться за допомогою *v-for*, що створює окремі картки фільмів для кожного об'єкта (див. рис. 3.1). На картках відображаються зображення постера, назва, рейтинг та інша базова інформація про фільм (див. рис. 3.2).

```
<template>
  <div class="movie-list">
    <h1>{{ title }}</h1>
    <div v-if="loading" class="loading">Loading...</div>
    <div v-else class="movies-container">
      <div v-for="movie in movies" :key="movie.id" class="movie">
        <router-link :to="{ name: 'MovieDetail', params: { id: movie.id } }">
          <img :src="getImageUrl(movie.poster_path)" :alt="movie.title" />
          <div class="movie-info">
            <h2>{{ movie.title }}</h2>
            <p>Release Year: {{ getReleaseYear(movie.release_date) }}</p>
            <p>Genres: {{ getGenres(movie.genre_ids) }}</p>
            <div :class="['rating', ratingColor(movie.vote_average)]">{{ movie.vote_average.toFixed(1) }}</div>
          </div>
        </router-link>
      </div>
    </div>
    <Navigation :currentPage="currentPage" :totalPages="totalPages" @goBack="goBack" @goForward="goForward" />
  </div>
</template>
```

Рисунок 3.1 – Шаблон для виведення карток фільмів

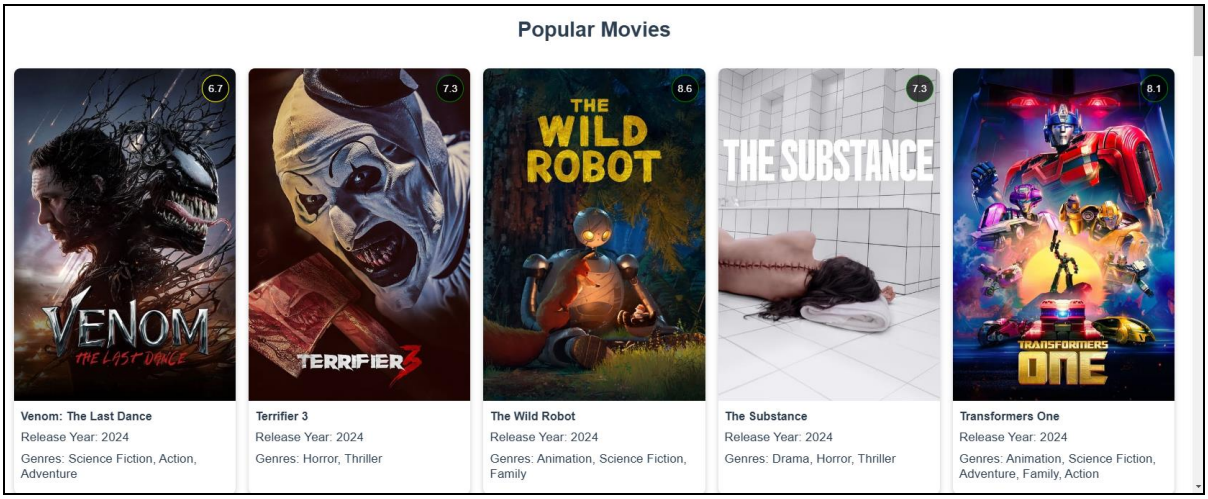



Рисунок 3.2 – Зовнішній вигляд головної сторінки

Щоб полегшити навігацію, компонент *MovieList* реалізує пагінацію, що дозволяє користувачам переходити між сторінками результатів. Пагінація реалізована з урахуванням динамічного завантаження даних при зміні сторінки. При переході на іншу сторінку компонент виконує новий запит до TMDb API, завантажуючи наступний набір фільмів. Пагінація зберігає

поточний стан сторінки, щоб користувач міг повернутися до попередньої сторінки після оновлення (див. рис. 3.3).




Рисунок 3.3 – Зовнішній вигляд пагінації в моєму додатку

### 3.2.2 Header

У компоненті *Header.vue* реалізується навігаційна панель, яка дозволяє користувачам легко переходити між різними категоріями фільмів, такими як популярні, нові, найкращі або майбутні. Кожна категорія представлена у вигляді посилання, яке веде до відповідного маршруту, а також є випадаюче меню для вибору жанрів фільмів, що динамічно генерується на основі масиву жанрів.

Крім того, компонент містить панель автентифікації: якщо користувач увійшов у систему, відображається його аватар, а якщо ні – кнопка для входу. Функції переходу до профілю або на сторінку входу реалізовані через методи, які активуються при кліку на відповідні елементи.

У верхній частині компонента є пошукова панель, яка дозволяє користувачам вводити запити на пошук фільмів. Пошук активується за допомогою кнопки або натискання клавіші Enter, після чого користувач перенаправляється на сторінку з результатами.

Стан автентифікації користувача перевіряється за допомогою Firebase, що дозволяє динамічно відображати інформацію про користувача в інтерфейсі. Компонент *Header.vue* відображається на всіх сторінках додатку, що забезпечує зручний та швидкий доступ до навігаційних елементів для користувачів. Вся ця функціональність стилізована з використанням CSS, що забезпечує привабливий та зручний для користувачів інтерфейс (див. рис. 3.4).

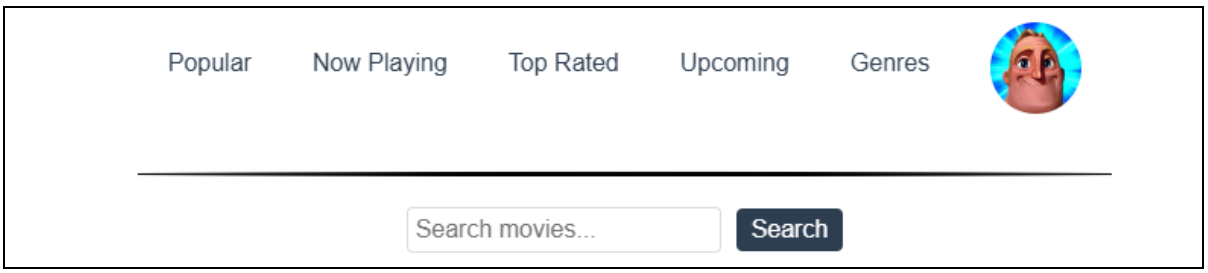



Рисунок 3.4 – Зовнішній вигляд компоненту Header

### 3.2.3 Компонент деталей фільму (MovieDetail)

Компонент MovieDetail відповідає за детальне відображення інформації про окремий фільм. Користувач може переглянути опис фільму, рейтинг, жанри, трейлер, список акторів, а також додати фільм до «вподобаних» (див. рис. 3.5).




Рисунок 3.5 – Зовнішній вигляд сторінки з фільмом

Компонент отримує дані про фільм через динамічний маршрут (наприклад, */movie/:id*). Запит до TMDb API надає інформацію про фільм, таку як назва, опис, жанри, рейтинг, а також посилання на трейлер. Вся інформація виводиться у зрозумілому форматі, включаючи постер фільму, слоган, жанри, огляд, та користувацький рейтинг. Також передбачено

відображення трейлера у форматі iframe, що забезпечує можливість перегляду відео безпосередньо на сторінці.

Компонент також відображає список акторів, які брали участь у фільмі. Кожен актор представлений у вигляді картки, на яку можна натиснути для отримання більш детальної інформації про них. Це дозволяє користувачам дізнатися більше про улюблених акторів (див. рис. 3.6).

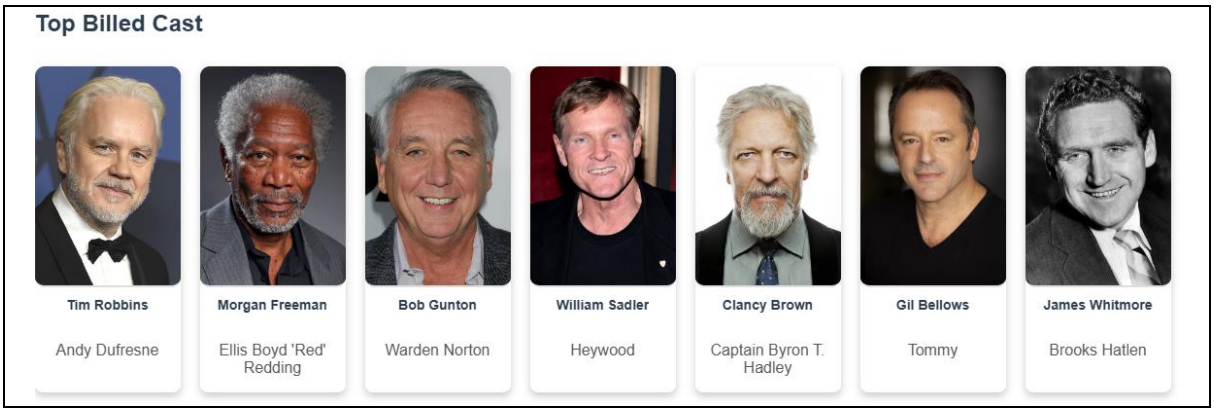



Рисунок 3.6 – Картки з акторами

На основі поточного фільму, компонент *MovieDetail* також показує рекомендовані фільми, які можуть зацікавити користувача. Це досягається шляхом виконання додаткового запиту до TMDb API, який повертає список фільмів, що мають подібні жанри чи тематику. Рекомендовані фільми виводяться у вигляді горизонтального списку, що дозволяє користувачу легко їх переглядати (див. рис. 3.7).

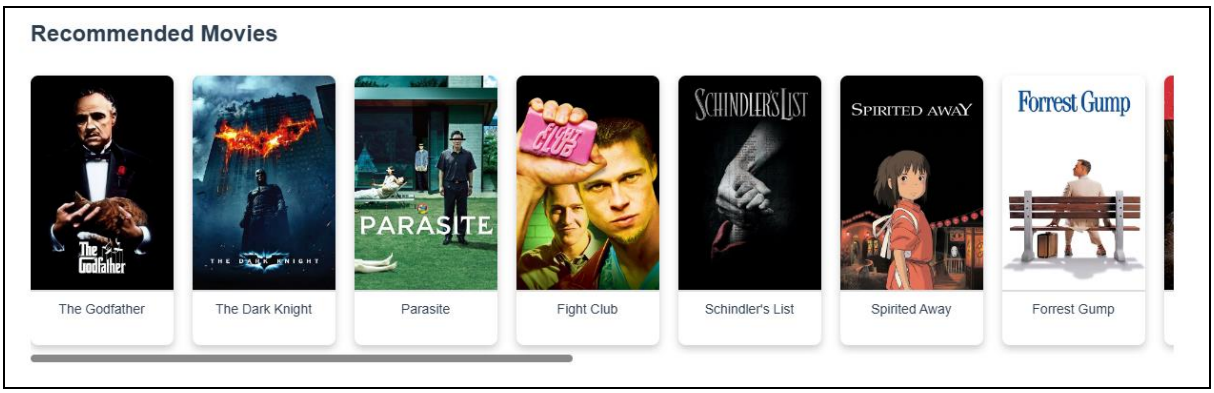



Рисунок 3.7 – Картки з рекомендованими фільмами

Користувач може додати фільм до списку «вподобаних» натисканням на спеціальну кнопку. Стан кнопки змінюється в реальному часі, і користувач одразу бачить зміни без перезавантаження сторінки. При натисканні кнопки інформація про фільм додається в Firebase Realtime Database (див. рис. 3.8).

```
async checkIfLiked() {
  if (this.user) {
    const db = getDatabase(app);
    const likedMoviesRef = ref(db, `users/${this.user.uid}/likedMovies/${this.movie.id}`);
    const snapshot = await get(likedMoviesRef);
    this.isLiked = snapshot.exists();
  }
},
async toggleLike() {
  if (this.user) {
    const db = getDatabase(app);
    const likedMoviesRef = ref(db, `users/${this.user.uid}/likedMovies/${this.movie.id}`);

    if (this.isLiked) {
      // Remove from liked movies
      await set(likedMoviesRef, null);
      this.isLiked = false;
    } else {
      // Add to liked movies
      await set(likedMoviesRef, {
        movieId: this.movie.id,
        title: this.movie.title,
        poster_path: this.movie.poster_path,
      });
      this.isLiked = true;
    }
  }
},
```

Рисунок 3.8 – Код для обробки кнопки «вподобати»

### 3.2.4 Сторінка актора (ActorDetail)

Сторінка актора реалізується через компонент *ActorDetail*, який надає користувачам детальну інформацію про вибраного актора. Основним завданням є відображення біографії актора, а також списку фільмів, у яких він брав участь. Біографічні дані включають основні факти про актора, такі як дата народження, місце народження, та основні етапи кар'єри. Крім того, сторінка містить секцію зі списком фільмів, що надає можливість користувачам переглядати роботи актора та переходити до детальної інформації про кожен фільм (див. рис. 3.9).

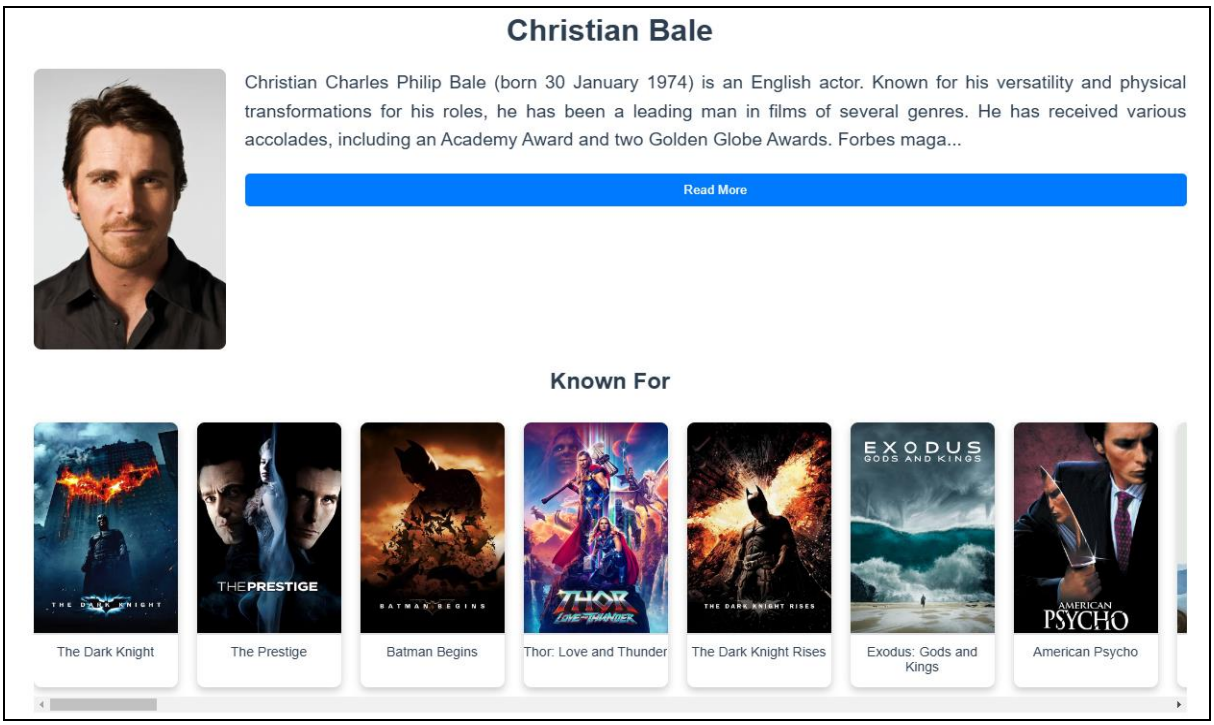



Рисунок 3.9 – Сторінка з актором

Для покращення візуальної привабливості та зручності використання інформація на сторінці структурована в зрозумілій формі. Зліва від біографії розташоване фото актора, яке підвищує впізнаваність та привертає увагу користувачів. Біографія представлена у вигляді тексту з можливістю розгортання; для цього використовуються кнопки «Читайте більше», які відкривають додаткову інформацію про актора, що дозволяє користувачам глибше ознайомитися з його життям та кар'єрою без перевантаження інтерфейсу.

Для отримання даних про актора реалізована логіка, яка взаємодіє з API TMDb. При завантаженні компонента *ActorDetail* відправляється запит до API з параметром ідентифікатора актора. У відповідь отримуються дані, які включають не лише біографічну інформацію, але й список фільмів, у яких актор брав участь. Це дозволяє динамічно оновлювати контент на сторінці, надаючи користувачам актуальну та релевантну інформацію. Логіка обробки помилок також впроваджена, щоб у разі проблеми з отриманням даних надати користувачам зрозуміле повідомлення про помилку та можливість повторити запит.

### 3.2.5 Роутинг (Vue Router)

У проєкті використовуємо Vue Router для налаштування динамічних маршрутів, які дозволяють користувачам переглядати деталі фільмів і акторів. Кожен маршрут визначає унікальний шлях до компонентів, які відображають відповідні дані. Наприклад, маршрут для деталей фільму виглядає як `/movie/:id`, де `:id` – це параметр, що передає ідентифікатор фільму. Аналогічно, маршрут для акторів реалізується через `/actor/:id`. Це забезпечує гнучкість в отриманні даних через API, дозволяючи завантажувати та відображати конкретну інформацію відповідно до обраного фільму або актора.

Щоб покращити користувацький досвід, реалізовано механізм збереження стану сторінки при навігації та оновленні. Використовуючи Vue Router, ми можемо зберігати параметри маршруту та стан компонентів, що дозволяє користувачам повернутися на попередню сторінку без втрати інформації. Наприклад, при переході між фільмами або акторами, всі дані, такі як список рекомендованих фільмів або інформація про вподобані, залишаються доступними. Це реалізується шляхом використання параметрів маршруту та Vuex для зберігання стану, що забезпечує плавну навігацію без повторного запиту до API, зберігаючи час завантаження і покращуючи загальний користувацький досвід.

### 3.2.6 Аутентифікація користувачів через Firebase

Firebase забезпечує зручні та безпечні інструменти для аутентифікації користувачів. У цьому проєкті використовується Google OAuth для реєстрації та входу, що дозволяє користувачам увійти в систему за допомогою їхніх Google-акаунтів (див. рис. 3.10).

Для реалізації аутентифікації через Google OAuth використовується Firebase Authentication, яка дозволяє інтегрувати авторизацію Google з

мінімальними налаштуваннями. При натисканні кнопки «Увійти через Google» користувач перенаправляється на сторінку авторизації Google, після чого отримані токени автентифікації зберігаються у Firebase. Цей процес є простим та безпечним для користувача, оскільки він уникає необхідності створювати нові паролі.

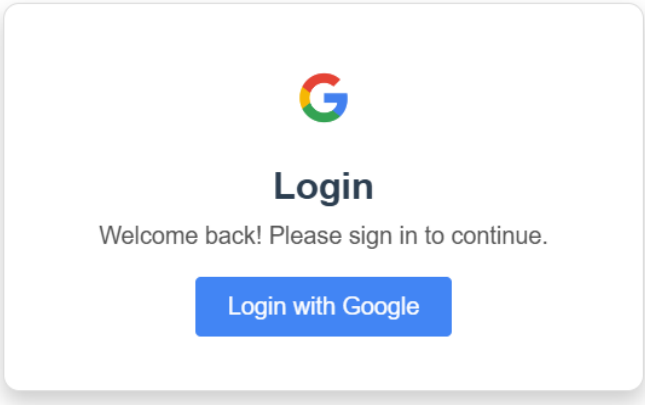



Рисунок 3.10 – Сторінка авторизації

Після успішного входу Firebase зберігає стан авторизації користувача, щоб він залишався увійшовшим навіть після перезавантаження сторінки. Крім того, передбачена логіка перенаправлення користувача на головну сторінку, що робить процес більш інтуїтивним та зручним. Firebase також дозволяє обробляти події виходу з системи, очищаючи сесію і перенаправляючи користувача на головну сторінку з можливістю авторизації.

Нижче представлено фрагмент коду, який реалізує вхід через Google OAuth за допомогою Firebase Authentication. Він демонструє, як саме відбувається процес автентифікації (див. рис. 3.11).

Метод loginWithGoogle виконує аутентифікацію через Google OAuth. При натисканні кнопки “Login with Google” цей метод відкриває спливаюче вікно, де користувач може обрати свій обліковий запис Google для входу. Функція signInWithPopup(auth, provider) зберігає токен у Firebase при успішному вході, забезпечуючи безпечну автентифікацію користувачів.

Відстеження стану аутентифікації реалізовано за допомогою onAuthStateChanged, що активується кожного разу, коли змінюється стан

авторизації користувача. Якщо користувач вже автентифікований, його автоматично перенаправляють на головну сторінку.

```
<script>
import { getAuth, GoogleAuthProvider, signInWithPopup, onAuthStateChanged } from "firebase/auth";
import { app } from "../firebase"; // Шлях до Firebase-конфігурації

export default {
  name: 'Login',
  methods: {
    async loginWithGoogle() {
      const auth = getAuth(app); // Ініціалізація аутентифікації
      const provider = new GoogleAuthProvider(); // Провайдер для Google OAuth
      try {
        await signInWithPopup(auth, provider); // Вхід через спливаюче вікно
      } catch (error) {
        console.error("Error during login:", error); // Обробка помилок
      }
    }
  },
  created() {
    const auth = getAuth(app);
    onAuthStateChanged(auth, (user) => {
      if (user) {
        this.$router.push({ name: 'home' }); // Перенаправлення на головну сторінку
      }
    });
  }
};
</script>
```

Рисунок 3.11 – Реалізація аутентифікації в коді

Цей підхід забезпечує просту та безпечну інтеграцію Google OAuth через Firebase, а також покращує зручність користування додатком завдяки автоматичному збереженню стану авторизації та логічному перенаправленню після входу в систему.

### 3.2.7 Збереження вподобаних фільмів у Firebase

Для зберігання вподобаних фільмів користувача застосовується Firebase Realtime Database. Вибір цієї бази даних обґрунтований її можливістю оновлювати дані в реальному часі та забезпечувати швидкий доступ до них.

Структура даних у Realtime Database організована таким чином, щоб спростити збереження та отримання вподобаних фільмів для кожного користувача. Для цього створюється колекція likedMovies в розділі кожного користувача, де кожен запис представляє ідентифікатор фільму, який користувач додав до вподобаних (див. рис. 3.12).




Рисунок 3.12 – Приклад збережених даних

Ця структура дозволяє зберігати основну інформацію про фільми, щоб уникнути надлишкових запитів до API при відображенні списку вподобаних фільмів.

Реалізація функціоналу складається з двох основних кроків. Перший – перевірка вподобань. Після завантаження сторінки виконується перевірка, чи був обраний фільм доданий у вподобання користувача. Ця перевірка здійснюється функцією *checkIfLiked*, яка звертається до Firebase Realtime Database і визначає, чи є запис з поточним ідентифікатором фільму (див. рис. 3.13).

```
async checkIfLiked() {
  if (this.user) {
    const db = getDatabase(app);
    const likedMoviesRef = ref(db, `users/${this.user.uid}/likedMovies/${this.movie.id}`);
    const snapshot = await get(likedMoviesRef);
    this.isLiked = snapshot.exists();
  }
},
```

Рисунок 3.13 – Код функції перевірки вподобань

Другим кроком є збереження та видалення вподобаних фільмів. Функція *toggleLike* перевіряє поточний стан вподобання фільму. Якщо фільм

вже є у вподобаннях, функція видаляє його з бази даних, а якщо ні – додає новий запис з інформацією про фільм (ідентифікатор, назва та шлях до постеру) (див. рис. 3.14). Обидві операції виконуються асинхронно та інтегруються з інтерфейсом, що дозволяє користувачеві миттєво бачити результати взаємодії з базою даних без необхідності перезавантаження сторінки.

```
async toggleLike() {
  if (this.user) {
    const db = getDatabase(app);
    const likedMoviesRef = ref(db, `users/${this.user.uid}/likedMovies/${this.movie.id}`);

    if (this.isLiked) {
      // Видалення з вподобаних фільмів
      await set(likedMoviesRef, null);
      this.isLiked = false;
    } else {
      // Додавання до вподобаних фільмів
      await set(likedMoviesRef, {
        movieId: this.movie.id,
        title: this.movie.title,
        poster_path: this.movie.poster_path,
      });
      this.isLiked = true;
    }
  }
},
```

Рисунок 3.14 – Код функції збереження та видалення вподобаних фільмів

### 3.2.8 Профіль користувача

Сторінка профілю користувача надає зручний доступ до всієї інформації, пов’язаної з уподобаннями користувача в системі, зокрема до вподобаних фільмів та рекомендацій на основі цих уподобань. Основними компонентами сторінки є таблиця з вподобаними фільмами, горизонтальний скролер для рекомендованих фільмів і функцію завантаження аватарки.

На сторінці профілю передбачена можливість змінювати аватарку профілю. Користувач може завантажити нове зображення, яке автоматично відображатиметься на сторінці. Для зберігання аватарок використовується

Firebase Storage, що дозволяє безпечно завантажувати та зберігати зображення в хмарному сховищі. Після завантаження зображення отримується його URL, який зберігається разом із даними профілю користувача, що дозволяє швидко відображати аватарку в інтерфейсі (див. рис. 3.15).

```
async uploadImage(event) {
  // Отримуємо файл, який користувач вибрав через input
  const file = event.target.files[0];
  // Перевіряємо, чи файл існує і чи є користувач
  if (file && this.user) {
    // Ініціалізуємо Firebase Storage
    const storage = getStorage(app);
    // Створюємо посилання на зображення в хмарному сховищі, яке буде зберігатися під унікальним id користувача
    const storageRef = ref(storage, `profile_images/${this.user.uid}`);
    try {
      // Завантажуємо файл у Firebase Storage
      await uploadBytes(storageRef, file);
      // Отримуємо URL завантаженого зображення
      const photoURL = await getDownloadURL(storageRef);
      // Оновлюємо профіль користувача, додаючи нове зображення
      await updateProfile(getAuth(app).currentUser, { photoURL });
      // Оновлюємо локальну змінну photoURL, щоб відобразити нове зображення
      this.photoURL = photoURL;
      // Перезавантажуємо сторінку, щоб відобразити оновлення
      location.reload();
    } catch (error) {
      // Виводимо помилку, якщо щось пішло не так
      console.error("Error uploading image:", error);
    }
  }
```

Рисунок 3.15 – Функція для завантаження зображення користувачем

Функціонал збереження та відображення вподобаних фільмів дозволяє користувачам переглядати, керувати та швидко отримувати доступ до улюблених фільмів. Всі вподобані фільми зберігаються у Firebase Realtime Database, що забезпечує актуальність даних у режимі реального часу (див. рис. 3.16).

Кожного разу, коли користувач додає або видаляє фільм із вподобаних, Firebase Realtime Database автоматично оновлює дані на стороні клієнта, що забезпечує безперервну синхронізацію даних без потреби в ручному оновленні сторінки. Логіка завантаження даних побудована так, щоб при кожному завантаженні профілю система виконувала запит до Firebase для відображення актуального списку вподобаних фільмів у таблиці (див. рис. 3.17).

**Liked Movies**

| # | Title |
|---|---|
| 1 | The Dark Knight |
| 2 | The Godfather |
| 3 | The Shawshank Redemption |
| 4 | 12 Angry Men |
| 5 | Avatar |
| 6 | The Avengers |
| 7 | Interstellar |
| 8 | Despicable Me 4 |
| 9 | The Gentlemen |
| 10 | Deadpool & Wolverine |
| 11 | Fly Me to the Moon |
| 12 | Inside Out 2 |
| 13 | Kill |

Рисунок 3.16 – Відображення уподобаних фільмів

```
fetchLikedMovies(userId) {
  // Отримуємо доступ до Firebase Realtime Database
  const db = getDatabase(app);
  // Створюємо посилання на список улюблених фільмів користувача в базі даних
  const likedMoviesRef = dbRef(db, `users/${userId}/likedMovies`);
  // Встановлюємо слухача на зміни в базі даних
  onValue(likedMoviesRef, (snapshot) => {
    // Отримуємо дані з бази
    const data = snapshot.val();
    // Якщо дані є
    if (data) {
      // Перетворюємо об'єкт даних на масив, додаючи id до кожного фільму
      this.likedMovies = Object.keys(data).map(key => ({
        id: key,
        ...data[key]
      }));
      // Викликаємо функцію для отримання рекомендацій
      this.fetchRecommendations();
    } else {
      // Якщо дані відсутні, ініціалізуємо масив порожнім
      this.likedMovies = [];
    }
  });
},
```

Рисунок 3.17 – Функція для відображення уподобаних фільмів

Назва кожного фільму є клікабельною. При натисканні на назву користувач автоматично перенаправляється на сторінку з детальною інформацією про фільм, де може переглянути трейлер, акторський склад та інші деталі. Якщо користувач ще не додав жодного фільму до вподобаних, на

сторінці профілю замість таблиці відображається повідомлення про можливість додавання фільмів, що робить інтерфейс інформативним та зручним для нових користувачів.

Рекомендаційна система розроблена для того, щоб допомогти користувачам знаходити нові фільми, які можуть їх зацікавити на основі попередніх вподобань. Основний функціонал побудований на API сервісу The Movie Database (TMDb), за допомогою якого отримуються релевантні рекомендації (див. рис. 3.18).

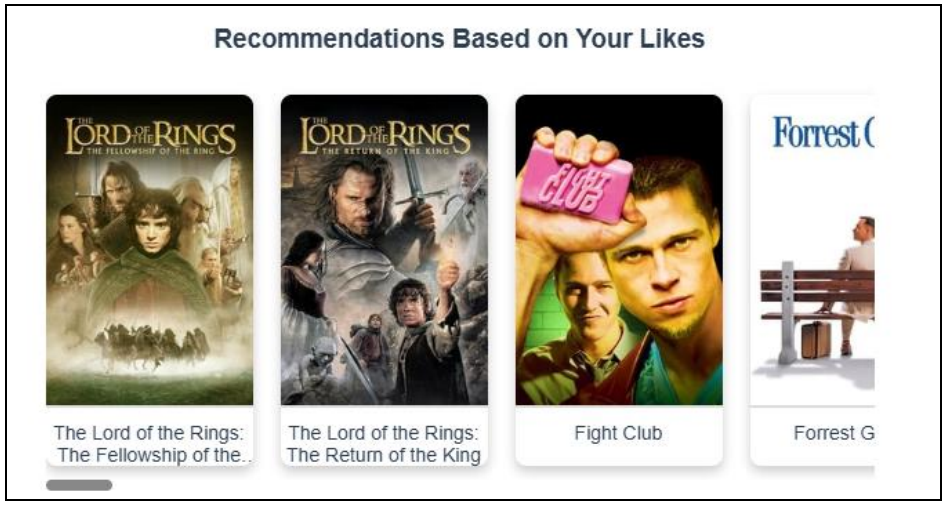



Рисунок 3.18 – Відображення рекомендацій на основі вподобаних

Алгоритм рекомендацій працює на основі фільмів, які користувач позначив як «вподобані». Використовуючи *movieId* кожного фільму, система надсилає запити до TMDb API для отримання пов’язаних фільмів, що дозволяє надавати користувачеві рекомендації, які відповідають його смаковим уподобанням. Наприклад, для кожного вподобаного фільму виконується запит *https://api.themoviedb.org/3/movie/{movieId}/recommendations*, який повертає схожі фільми. Рекомендовані фільми фільтруються для виключення дублювання тих, що вже є у списку вподобаних. Оскільки рекомендованих фільмів може бути багато, зроблено щоб відображало 5 найкращих за рейтингом на основі кожного фільму, що сподобався (див. рис. 3.19).

```
// Створюємо набір для зберігання унікальних рекомендацій
const uniqueRecommendations = new Set();
// Створюємо набір з ID улюблених фільмів, щоб легко виключати їх з рекомендацій
const likedMovieIds = new Set(this.likedMovies.map(movie => Number(movie.id))); // Переконуємося, що id є числами

// Створюємо масив запитів до API для кожного улюбленого фільму
const requests = this.likedMovies.map(movie =>
  fetch(`https://api.themoviedb.org/3/movie/${movie.id}/recommendations`, options)
    .then(response => response.json())
    .then(response => {
      // Обробка результатів рекомендацій
      const topRecommendations = response.results
        .filter(rec => {
          const recId = Number(rec.id); // Переконуємося, що rec.id є числом
          return recId !== Number(movie.id) && !likedMovieIds.has(recId); // Виключаємо фільми, які вже є в улюблених, і дублікати
        })
        .sort((a, b) => b.popularity - a.popularity) // Сортуємо за популярністю
        .slice(0, 5); // Беремо топ 5 рекомендацій

      // Додаємо рекомендації до набору, щоб уникнути дублювання
      topRecommendations.forEach(rec => uniqueRecommendations.add(rec));
    })
);

// Чекаємо, поки всі запити будуть виконані
await Promise.all(requests);

// Додаткове фільтрування для уникнення повторів і виключення улюблених фільмів
```

Рисунок 3.19 – Функція для відображення рекомендацій

На сторінці профілю рекомендації представлені у вигляді горизонтального скролера, що забезпечує зручну навігацію по списку фільмів і додає естетичної привабливості інтерфейсу. Кожен елемент скролера відображає постер фільму та основну інформацію, а додані ефекти при наведенні забезпечують інтерактивність перегляду. Кожен рекомендований фільм є клікабельним, і при натисканні на постер користувач може перейти на сторінку деталей фільму.

Таким чином, сторінка профілю з таблицею вподобаних фільмів та блоком рекомендацій дозволяє користувачам ефективно переглядати улюблені фільми, отримувати швидкий доступ до нових релевантних рекомендацій і зручний доступ до деталей про зацікавлені фільми.

## 3.3 Тестування додатку засобами Cypress

У цьому підрозділі розглядається процес тестування функціональності додатку з використанням засобу автоматизації тестів Cypress. Cypress дозволяє не тільки перевіряти роботу інтерфейсу користувача (UI), а й

взаємодію з бекенд-сервісами, такими як Firebase. Тестування допомагає забезпечити стабільність і надійність додатку перед його запуском на продуктивному середовищі.

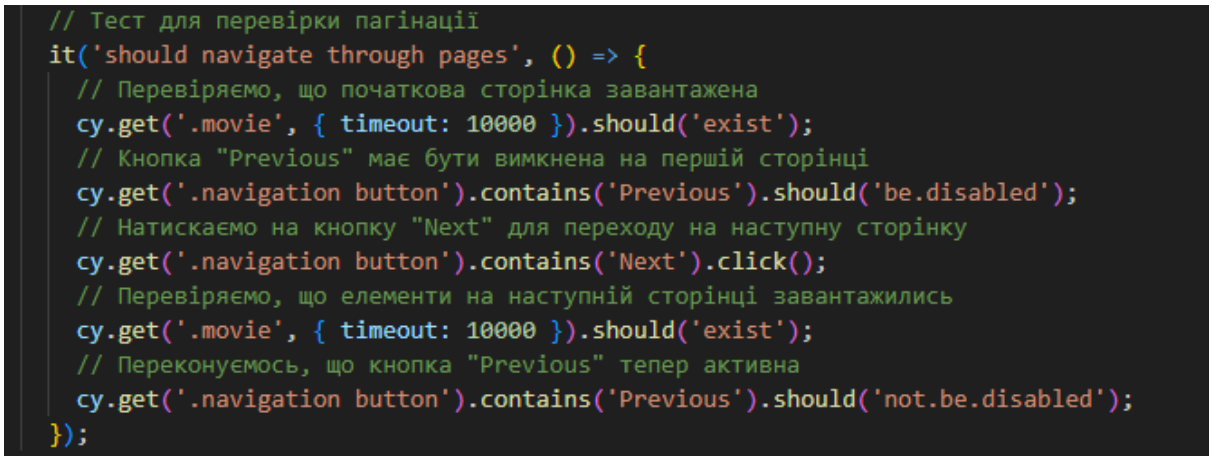
Перевірка пошуку фільмів починається з того, що перевіряється наявність поля для вводу тексту. Потім вводиться запит, наприклад, “Inception”, і після натискання кнопки пошуку перевіряється, чи з’являються результати, що містять введену назву в заголовках фільмів (див. рис. 3.20).



```
// Тест для перевірки пошуку
it('should search for movies based on input', () => {
  const searchTerm = 'Inception';
  // Знаходимо поле пошуку за placeholder і вводимо запит
  cy.get('input[placeholder="Search movies..."]', { timeout: 10000 }).should('exist').type(searchTerm);
  // Знаходимо кнопку пошуку за текстом і натискаємо на неї
  cy.contains('button', 'Search').click();
  // Перевіряємо, що відображаються результати, які відповідають запиту
  cy.get('.movie', { timeout: 10000 }).should('exist');
  cy.get('.movie-info h2').each(($title) => {
    cy.wrap($title).should('contain.text', searchTerm);
  });
});
```

Рисунок 3.20 – Код для тестування пошуку фільмів

Тестування пагінації фокусується на перевірці кнопок “Next” та “Previous”. Спочатку перевіряється, що кнопка “Previous” неактивна на першій сторінці. Потім натискається кнопка “Next”, і перевіряється, чи змінилися фільми на новій сторінці, а також чи стала кнопка “Previous” активною (див. рис. 3.21).



```
// Тест для перевірки пагінації
it('should navigate through pages', () => {
  // Перевіряємо, що початкова сторінка завантажена
  cy.get('.movie', { timeout: 10000 }).should('exist');
  // Кнопка "Previous" має бути вимкнена на першій сторінці
  cy.get('.navigation button').contains('Previous').should('be.disabled');
  // Натискаємо на кнопку "Next" для переходу на наступну сторінку
  cy.get('.navigation button').contains('Next').click();
  // Перевіряємо, що елементи на наступній сторінці завантажились
  cy.get('.movie', { timeout: 10000 }).should('exist');
  // Переконуємось, що кнопка "Previous" тепер активна
  cy.get('.navigation button').contains('Previous').should('not.be.disabled');
});
```

Рисунок 3.21 – Код для тестування пагінації

Перевірка відображення деталей фільмів передбачає виведення важливих елементів фільму після завантаження сторінки, таких як постер, назва, опис, жанри, рейтинг тощо (див. рис. 3.22).

```
describe('Movie Detail Page Tests', () => {
  beforeEach(() => {
    const movieId = 550; // ID тестового фільму
    cy.visit(`/movie/${movieId}`);
  });
  it('should display movie details after loading', () => {
    // Чекаємо завершення завантаження
    cy.get('.loading').should('not.exist');
    // Перевіряємо наявність елементів з даними про фільм
    cy.get('.movie-header').should('exist');
    cy.get('.movie-poster').should('exist');
    cy.get('.movie-info h1').should('exist'); // Назва фільму
    cy.get('.tagline').should('exist'); // Теглайн, якщо він є
    cy.get('.genres').should('exist'); // Жанри
    cy.get('.user-score').should('exist'); // Рейтинг
    cy.get('.overview').should('exist'); // Опис
  });
  it('should display trailer if available', () => {
    cy.get('.loading').should('not.exist');
    // Перевіряємо наявність трейлера
    cy.get('.trailer').should('exist');
    cy.get('.trailer iframe').should('have.attr', 'src').and('include', 'youtube.com');
  });
  it('should display recommendations', () => {
    cy.get('.loading').should('not.exist');
    // Перевіряємо наявність рекомендованих фільмів
    cy.get('.recommendations-list .recommendation-item').should('have.length.greaterThan', 0);
    cy.get('.recommendations-list .recommendation-item').first().click(); // Клікаємо на перший рекомендований фільм
    cy.url().should('include', '/movie/'); // Перевіряємо, що ми перейшли на сторінку рекомендованого фільму
  });
```

Рисунок 3.22 – Код для тестування сторінки з фільмами

Тестування відображення трейлера перевіряє, чи правильно вбудовуються трейлери для фільмів, якщо вони доступні. Після завантаження сторінки перевіряється, чи є елемент, що містить трейлер, а також чи вбудоване відео з YouTube.

Рекомендації фільмів тестуються на наявність списку рекомендованих фільмів, що відображаються на сторінці. Перевіряється, чи можна перейти на детальну сторінку фільму, натискаючи на картку одного з фільмів.

Тестування деталей актора перевіряє, чи коректно завантажуються і відображаються дані про актора на сторінці (див. рис. 3.23). Спочатку перевіряється, чи список акторів відображається без затримок після завантаження сторінки. Далі, перехоплюється запит до API для отримання інформації про актора. Після цього натискається на першого актора зі списку,

і перевіряється, що сторінка була оновлена, а URL змінився, включаючи ідентифікатор актора. Далі перевіряється, чи коректно відображаються елементи: ім’я актора, зображення та список відомих фільмів, в яких він брав участь.

```
it('should display actor details', () => {
  // Чекаємо завершення завантаження списку акторів
  cy.get('.loading').should('not.exist');
  // Перевіряємо наявність списку акторів
  cy.get('.cast-list .cast-member').should('have.length.greaterThan', 0);
  // Перехоплюємо запит до API для отримання даних актора
  cy.intercept('GET', 'https://api.themoviedb.org/3/person/**').as('getActorData');
  // Клікаємо на першого актора
  cy.get('.cast-list .cast-member').first().click();
  // Чекаємо завершення запиту до API
  cy.wait('@getActorData');
  // Перевіряємо, що ми перейшли на сторінку актора
  cy.url().should('include', '/actor/');
  // Перевіряємо наявність імені актора
  cy.get('.actor-detail h1').should('be.visible');
  // Перевіряємо, що зображення актора відображається
  cy.get('.actor-image').should('be.visible');
  // Перевіряємо, що список відомих фільмів відображається
  cy.get('.known-for-list .known-for-item').should('have.length.greaterThan', 0);
});
```

Рисунок 3.23 – Тестування сторінки актора

Тестування логіну через Firebase перевіряє, чи правильно працює процес авторизації (див. рис. 3.24). Спочатку перевіряється, чи користувач вже авторизований. Якщо ні, виконуються введення даних і натискання кнопки “Login”. Після цього перевіряється, що користувач потрапляє на домашню сторінку.

Завдяки проведеному тестуванню функціональності додатку було перевірено основні процеси взаємодії користувача з інтерфейсом. Під час тестування було виявлено одну помилку, пов’язану з тим, що на сторінці з фільмами починали обробляти дані, не дочекавшись їх повного завантаження. І хоча це не призводило до некоректного відображення інформації, воно все ж таки могло спричинити неочікувані помилки при подальших взаємодіях з інтерфейсом, таких як неправильне відображення

елементів або некоректна робота деяких функцій. Завдяки тестуванню ця помилка була своєчасно виявлена і виправлена. В результаті було досягнуто більш стабільної роботи додатку, що забезпечило коректне завантаження та обробку даних перед виконанням подальших дій.

```
beforeEach(() => {
  cy.visit('/login'); // Перехід на сторінку логіну
});

it('should login using email and password if not already logged in', () => {
  // Перевіряємо, чи користувач уже авторизований
  cy.window().then((window) => {
    const isUserLoggedIn = window.localStorage.getItem('firebase_auth') !== null;

    // Якщо користувач не авторизований, то виконуємо логін
    if (!isUserLoggedIn) {
      cy.get('.hidden-email-login').invoke('show');
      const email = Cypress.env('FIREBASE_TEST_EMAIL');
      const password = Cypress.env('FIREBASE_TEST_PASSWORD');
      cy.get('input[type="email"]').type(email);
      cy.get('input[type="password"]').type(password);
      cy.get('button').contains('Login with Email').click();

      // Перевіряємо, чи користувач на домашній сторінці після успішного логіну
      cy.url().should('include', '/');
    } else {
      // Якщо користувач вже авторизований, просто перевіряємо домашню сторінку
      cy.url().should('include', '/');
    }
```

Рисунок 3.24 – Тестування авторизації

Таким чином, тестування не тільки підтвердило стабільність роботи додатку, але й допомогло виявити та усунути помилки, що можуть негативно вплинути на користувацький досвід.

### 3.4 Висновки до розділу 3

Реалізація вебзастосунку з використанням фреймворку Vue.js для фронтенд-частини та Firebase для бекенду дозволила створити масштабовану та зручну структуру проєкту, яка повністю відповідає поставленим вимогам.

Ключові функціональні компоненти, такі як головна сторінка, панель навігації, сторінки з детальною інформацією про фільми та акторів, модуль аутентифікації через Firebase та можливість зберігати вподобані фільми, були успішно реалізовані, надаючи користувачам інтерактивну і безперервну роботу. Cypress тестування підтвердило стабільність роботи всіх модулів додатка, а також виявило й допомогло усунути незначні помилки, пов’язані з асинхронним завантаженням даних, що поліпшило загальну функціональність. Таким чином, створений вебзастосунок повністю відповідає технічним вимогам проєкту, забезпечуючи інтуїтивно зрозумілий інтерфейс і надійну роботу для користувачів.

# ВИСНОВКИ

В кваліфікаційній роботі розроблено вебзастосунок для перегляду відео з використанням Vue.js та інтеграцією рекомендаційної системи, заснованої на даних TMDb API, яка пропонує фільми на основі жанрів, вподобань користувачів та популярності.

В першому розділі роботи було детально описані сучасні тенденції у сфері веброзробки та створення рекомендаційних систем, які стали основою для проєктування та реалізації продукту. Обґрунтовано вибір Vue.js для фронтенду, Firebase для зберігання даних і аутентифікації. Інтеграція з TMDb API дозволила надати доступ до актуальних даних про фільми, а також реалізувати механізм рекомендацій, на основі вподобань користувачів.

В другому розділі проведено проєктний аналіз, на основі якого визначено основні функціональні та нефункціональні вимоги до системи. Для опису структури застосунку та процесу взаємодії користувачів із системою наведено діаграму прецедентів, діаграму послідовності, діаграму розгортання та ER-діаграму бази даних.

У третьому розділі продемонстровано програмну реалізацію вебдодатку. Детально описано структуру проєкту та ключові компоненти, як-от Vue Router для організації маршрутизації, що забезпечує зручну навігацію між сторінками. Інтеграція Firebase дозволила реалізувати функції аутентифікації користувачів. Окрему увагу приділено технології Cypress, яка була використана для автоматизації тестування, зокрема для перевірки коректності відображення фільмів, роботи системи рекомендацій та аутентифікації.

Результати роботи можуть бути використані для подальшого розвитку платформи, зокрема для додавання нових функцій, інтеграції рекомендаційних алгоритмів або монетизації через рекламні платформи.

# ПЕРЕЛІК ПОСИЛАНЬ

1. API Reference, YouTube Data API. *Google for Developers*. URL: https://developers.google.com/youtube/v3/docs (дата звернення: 06.06.2024).

2. GitHub – sampotts/plyr: A simple HTML5, YouTube and Vimeo player. *GitHub*. URL: https://github.com/sampotts/plyr (дата звернення: 08.06.2024).

3. Video.js API docs. URL: https://docs.videojs.com/ (дата звернення: 08.06.2024).

4. GitHub – video-dev/hls.js: HLS.js is a JavaScript library that plays HLS in browsers with support for MSE. *GitHub*. URL: https://github.com/video-dev/hls.js/ (дата звернення: 15.06.2024).

5. Kumari R. Top technologies and programming language behind YouTube. *TechGig*. URL: https://content.techgig.com/technology/top-technologies-and-programming-language-behind-youtube/articleshow/104493727.cms (дата звернення: 13.07.2024).

6. Babu S. S. Technologies and programming languages used by Netflix. URL: https://faun.pub/technologies-and-programming-languages-used-by-netflix-63d8810e5880 (дата звернення: 15.07.2024).

7. Tech H. Introducing Hulu's Modern Web Stack. *Medium*. URL: https://medium.com/hulu-tech-blog/introducing-hulus-modern-web-stack-d0be4cf4f439 (дата звернення: 15.07.2024).

8. Garaguso P. D., Zander O. Vue. js 3 Design Patterns and Best Practices: Develop Scalable and Robust Applications with Vite, Pinia, and Vue Router. Packt Publishing, Limited, 2023. P. 1–49.

9. Mardan A. Practical Node.js. Berkeley, CA : Apress, 2018. P. 51–305. URL: https://doi.org/10.1007/978-1-4842-3039-8 (дата звернення: 18.07.2024).

10. Google's Mobile and Web App Development Platform. *Firebase*. URL: https://firebase.google.com/ (дата звернення: 18.07.2024).

11. Cypress testing solutions. *Cypress Documentation*. URL: https://docs.cypress.io/app/get-started/why-cypress (дата звернення: 20.07.2024).

12. Getting Started. *The Movie Database (TMDB)*. URL: https://developer.themoviedb.org/docs/getting-started (дата звернення: 21.07.2024).

13. Papadakis H., Papagrigoriou A., Panagiotakis C. Collaborative filtering recommender systems taxonomy. *Knowledge and Information Systems*. 2022. Vol. 64, no. 1. P. 35–74. URL: https://doi.org/10.1007/s10115-021-01628-7 (дата звернення: 21.07.2024).

14. Lops P., de Gemmis M., Semeraro G. Content-based and collaborative techniques for tag recommendation: an empirical evaluation. *Journal of Intelligent Information Systems*. 2012. Vol. 40, no. 1. P. 41–61. URL: https://doi.org/10.1007/s10844-012-0215-6 (дата звернення: 01.08.2024).

15. Afoudi Y., Lazaar M., Al Achhab M. Hybrid recommendation system combined content-based filtering and collaborative prediction using artificial neural network. *Simulation Modelling Practice and Theory*. 2021. Vol. 113. P. 102375. URL: https://doi.org/10.1016/j.simpat.2021.102375 (дата звернення: 03.08.2024).

16. Zhang S., Tay Y., Yao L. Deep Learning for Recommender Systems. *SpringerLink*. 2022. URL: https://doi.org/10.1007/978-1-0716-2197-4_5 (дата звернення: 07.08.2024).

17. He X., Liao L., Zhang H. Neural Collaborative Filtering. *WWW '17: 26th International World Wide Web Conference*, Perth Australia. Republic and Canton of Geneva, Switzerland, 2017. URL: https://doi.org/10.1145/3038912.3052569 (дата звернення: 11.08.2024).

18. Hug N. *Surprise*. URL: https://surpriselib.com/ (дата звернення: 11.08.2024).

19. MovieLens 20M Dataset. *Kaggle: Your Machine Learning and Data Science Community*. URL: https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset (дата звернення: 11.08.2024).

20. RecBole. URL: https://recbole.io/ (дата звернення: 15.08.2024).

21. GitHub – lyst/lightfm: A Python implementation of LightFM, a hybrid recommendation algorithm. URL: https://github.com/lyst/lightfm (дата звернення: 15.08.2024).

22. GitHub – jfkirk/tensorrec: A TensorFlow recommendation algorithm and framework in Python. URL: https://github.com/jfkirk/tensorrec (дата звернення: 16.08.2024).