МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Веб-застосунок для підбору фільмів з урахуванням вподобань

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма **«Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-42
_____Крекотень М.В._____
(прізвище та ініціали)

Керівник _____Зосімов В.В._____
(прізвище та ініціали)
_____д.т.н., доцент_____
(науковий ступінь, звання)
Унікальність тексту 97% (https://my.plag.com.ua/)

Випускна кваліфікаційна робота бакалавра допущена до захисту
Рішенням кафедри *прикладних інформаційних систем*
Протокол № 14 від 23.05.2023 р.
зав. кафедри _____ Плескач В. Л.

Київ – 2023

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «Веб-застосунок для підбору фільмів з урахуванням вподобань»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Крекотень Микита Вікторович	

Назва роботи українською та англійською мовами:

Веб-застосунок для підбору фільмів з урахуванням вподобань
Web application for movie recommendations based on user preferences

МЕТА БАКАЛАВРСЬКОЇ РОБОТИ, ЗАВДАННЯ

Мета бакалаврської роботи: ефективна рекомендація фільмів з урахуванням вподобань користувача за допомогою веб-застосунку

План роботи:

1. Дослідження сучасних методів розроблення веб-застосунків
2. Аналіз архітектурних рішень, вибір технологій для реалізації веб-застосунку
3. Реалізація веб-застосунку для підбору фільмів з урахуванням вподобань

ПІБ, ступінь, звання наукового керівника роботи: д.т.н., доцент Зосімов В.В.

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2023	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2023	Виконано
9.	Подання роботи у першому варіанті	28.04.2023	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	22.05.2023	Виконано
12	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про	26.05.2023	Виконано

	допуск) на кафедру		
13	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	Виконано
14	Захист кваліфікаційної роботи бакалавра	28.06.2023	Виконано

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини кваліфікаційної роботи бакалавра	Обсяг, арк.
Титульний аркуш	1
Календарний план кваліфікаційної роботи бакалавра	2
Відомість кваліфікаційної роботи бакалавра	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	19
2	33
3	11
Висновки	1
Перелік використаних джерел	4
Додатки	4

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розроб н.</td><td>Крекотень М.В.</td><td></td><td></td><td rowspan="4">Відомість кваліфікаційної роботи бакалавра</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Зосімов В.В.</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Кравченко К.В.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Кваліфікаційна робота бакалавра: 83 с., 21 рис., 14 табл., 27 джерел, 3 дод.

Ця кваліфікаційна робота бакалавра присвячена проектуванню та розробленню веб-застосунку для підбору фільмів з урахуванням вподобань.

Метою кваліфікаційної роботи бакалавра є ефективна рекомендація фільмів з урахуванням вподобань користувача за допомогою веб-застосунку.

Для досягнення поставленої мети треба вирішити такі **завдання:**

- дослідити загально-теоретичні основи побудови веб-застосунків;
- дослідити сутність механізму підбору рекомендацій за вподобаннями;
- здійснити аналіз основних етапів підбору фільмів за вподобаннями;
- спроектувати, реалізувати, впровадити прототип веб-застосунку для підбору фільмів з урахуванням вподобань користувача.

Об’єкт дослідження

Процеси підбору контенту з урахуванням вподобань користувача.

Предмет дослідження

Програмно-технологічні засоби, загальні принципи, засади, підходи до створення веб-застосунку підбору фільмів з урахуванням вподобань користувача.

Методи дослідження

Метод аналізу даних, метод системного аналізу, метод моделювання.

Ключові слова: веб-застосунок, рекомендаційні системи, підбір фільмів.

ABSTRACT

Thesis: 83 pages, 21 figures, 14 tables, 27 sources, 3 appendices.

This thesis is dedicated to the design and development of a web application for movie recommendations based on user preferences.

The purpose of this thesis is an effective movie recommendation mechanism based on user preferences.

To achieve this goal you need to solve the following **tasks**:

- Research on the general theoretical principles of application development.
- Research of the essence of the mechanism for selecting recommendations based on preferences.
- Conduct an analysis of the main stages of movie selection based on preferences.
- Design, implement, and deploy a prototype web application for movie recommendations based on user preferences.

Object of study

The processes for selecting content based on user preferences.

Subject of study

Software and technological tools, general principles, foundations, and approaches for creating web applications for movie recommendations based on user preferences.

Research methods

Data analysis method, systems analysis method, modeling method.

Key words: web application, recommendation systems, movie selection.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API(Application Programming Interface) - програмний інтерфейс програми

HTTP(HyperText Transfer Protocol) - протокол передачі гіпертексту

ORM(Object-Relational Mapping) - об'єктно-реляційна проекція

CORS(Cross-Origin Resource Sharing) - спільне використання ресурсів з різних джерел

UI(User Interface) - користувацький інтерфейс

СУБД - Система управління базами даних

БД - База даних

ВСТУП

В сучасному світі, коли кількість фільмів та серіалів зростає з кожним роком, а вибір усіх смаків стає все більш різноманітним, часто важко знайти щось, що справді зацікавить. У такі моменти багато хто звертається до рекомендацій від друзів або використовує платформи для стрімінгу, що пропонують лише популярні фільми. Проте, індивідуальні вподобання та особливості кожної людини можуть вимагати більш персоналізованого підходу.

Актуальність даної теми зумовлена тим, що все більше людей надають перевагу перегляду фільмів та серіалів вдома. Однак, вибір фільму для перегляду може стати складною задачею, особливо коли на ринку присутні тисячі різних фільмів та серіалів різних жанрів та якості. Крім того, у кожної людини є свої вподобання та смаки, що ускладнює процес вибору. Розробка веб-застосунку для підбору фільмів з урахуванням вподобань може вирішити цю проблему та зробити процес вибору фільму простішим та зручнішим для користувачів. Крім того, у зв'язку зі зростанням популярності онлайн-кінотеатрів та стрімінгових сервісів, такий веб-застосунок може забезпечити конкурентну перевагу та привернути нових користувачів.

Метою кваліфікаційної роботи бакалавра є ефективна рекомендація фільмів з урахуванням вподобань користувача за допомогою веб-застосунку.

Завдання дослідження полягають у:

- Дослідженні загально-теоретичних основ побудови веб-застосунків.
- Дослідженні сутності механізму підбору рекомендацій за вподобаннями.

- Здійсненні аналізу основних етапів підбору фільмів за вподобаннями.
- Проектуванні, реалізації, впровадженні прототипу веб-застосунку для підбору фільмів з урахуванням вподобань.

Об’єктом дослідження кваліфікаційної роботи бакалавра є процеси підбору контенту з урахуванням вподобань користувача.

Предметом дослідження кваліфікаційної роботи бакалавра є програмно-технологічні засоби, загальні принципи, засади, підходи до створення веб-застосунку підбору фільмів з використанням фреймворку Vue та мови програмування JavaScript.

Методи дослідження: є моделі, методи та засоби аналізу теоретичних принципів побудови веб-застосунку для підбору фільмів з урахуванням вподобань.

Практичне значення одержаних результатів полягає в можливості створення зручного та корисного веб-застосунку для вибору фільмів та серіалів з урахуванням особистих вподобань користувачів. Розроблена система може допомогти залучити нових користувачів. Крім того, подібний інструмент може бути корисним для платформ онлайн-кінотеатрів та інших веб-сервісів, що пропонують послуги зі стрімінгу відео.

РОЗДІЛ 1
ЗАГАЛЬНОТЕОРЕТИЧНІ ЗАСАДИ ВЕБ-ЗАСТОСУНКІВ ПІДБОРУ ФІЛЬМІВ

1.1. Роль веб-застосунків для підбору фільмів у світі

У сучасному світі індустрія онлайн-кіно має значний вплив на наше життя, завдяки постійному розвитку технологій і широкій доступності Інтернету. Одним із найпопулярніших сервісів у цій сфері є веб-додатки для вибору фільмів, які допомагають користувачам знайти та переглянути фільм відповідно до своїх уподобань. [1]

1.1.1. Огляд сучасних трендів в онлайн-кіноіндустрії та їх вплив на популярність веб-застосунків для підбору фільмів

Сьогодні онлайн-кіноіндустрія та потокове відео переживають свій розквіт, що спричинено швидким розвитком технологій та зростанням доступності Інтернету. Багато людей вважають онлайн-платформи кращими варіантами для перегляду фільмів та серіалів через їх доступність та зручність. Онлайн-платформи, такі як Netflix, Amazon Prime Video, Hulu, Disney+ та HBO Max, стали дуже популярними серед людей різних вікових груп, що сприяє зростанню популярності онлайн-кіноіндустрії.[1]

Одним з найважливіших аспектів онлайн-кіноіндустрії є персоналізація. Багато потокових платформ використовують алгоритми машинного навчання та штучного інтелекту, щоб забезпечити користувачам персоналізовані рекомендації щодо фільмів та серіалів на основі їхніх вподобань. Це створює велику потребу в веб-додатках для підбору фільмів, які допомагають користувачам знайти ідеальний фільм за їхніми вподобаннями.

Одним з прикладів таких веб-додатків є IMDb, який пропонує користувачам різноманітні фільтри та рекомендації щодо фільмів, які можуть сподобатися. Рейтинги, огляди та коментарі від інших користувачів також

допомагають зрозуміти, чи варто дивитися конкретний фільм чи серіал. Іншими популярними веб-додатками для підбору фільмів є Rotten Tomatoes та Letterboxd.

Ще одним важливим аспектом популярності веб-додатків для підбору фільмів є їхня здатність допомагати користувачам з економією часу. Замість того, щоб витрачати години на пошук фільмів та серіалів в Інтернеті, користувачі можуть скористатися веб-додатками для отримання персоналізованих рекомендацій, що дозволяє їм знайти більше часу на перегляд улюблених фільмів та серіалів.

Крім того, зростання популярності онлайн-кіноіндустрії також стимулює розвиток нових технологій та інновацій в цій галузі. Наприклад, деякі потокові платформи вже випускають власні фільми та серіали, що є показником зростання конкуренції та залучення нових талантів до галузі.

В цілому, можна зробити висновок, що зростання популярності онлайн-кіноіндустрії сприяє зростанню популярності веб-додатків для підбору фільмів, які допомагають користувачам знайти ідеальний фільм за їхніми вподобаннями. Ці веб-додатки дозволяють користувачам економити час та зручно переглядати фільми та серіали, що збільшує їхню популярність та важливість у сучасному онлайн-світі.

1.1.2. Аналіз ринку веб-застосунків для підбору фільмів: конкуренція та ніші

Ринок веб-застосунків для підбору фільмів є досить конкурентним і розгалуженим. Серед найбільш відомих знаходяться такі, як Netflix, Amazon Prime, Imdb, та інші. Ці компанії використовують власні алгоритми рекомендацій, які враховують уподобання користувачів, їх історію перегляду, а також поведінку на платформі.

Крім цього, існують менш відомі, спеціалізовані веб-застосунки для підбору фільмів, які спрямовані на конкретну аудиторію, наприклад, фанатів

аніме чи фестивалів короткометражних фільмів. Такі веб-застосунки зазвичай забезпечують більш точні рекомендації, що враховують конкретні інтереси та потреби користувачів.

Оскільки більшість веб-застосунків для підбору фільмів є комерційними продуктами, то основним джерелом їх доходів є передплати та реклама. Це може вплинути на те, які фільми вони рекомендують користувачам, оскільки вони можуть спонсорувати деякі фільми або серіали.

Однак, на ринку є і безкоштовні веб-застосунки для підбору фільмів, які забезпечують якісні рекомендації без комерційних мотивів. Такі веб-застосунки зазвичай мають відкритий вихідний код, що дозволяє їм залучати розробників та підтримувати високий рівень розробки.

Отже, при проектуванні веб-застосунку для підбору фільмів необхідно враховувати конкурентну природу ринку та знайти свою нішу, щоб виділитися серед конкурентів. Для цього можна використовувати спеціалізовані алгоритми рекомендацій, які будуть враховувати конкретні інтереси та потреби користувачів. Крім цього, важливо мати високий рівень розробки та підтримувати вихідний код, щоб залучати співробітників та забезпечувати якість продукту.

Також важливо розуміти, що великі компанії вже мають велику базу користувачів та більший бюджет на маркетинг, тому важливо знайти свою нішу і звернути увагу на тих, хто шукає альтернативу. Наприклад, спеціалізуватися на маловідомих фільмах, аніме або фестивалях короткометражних фільмів, щоб залучити увагу любителів цих жанрів. Таким чином, можна збільшити свою аудиторію та підвищити конкурентоспроможність на ринку веб-застосунків для підбору фільмів.

1.1.3. Переваги та недоліки веб-застосунків для підбору фільмів у порівнянні з традиційними способами пошуку кінофільмів

Веб-застосунки для підбору фільмів мають деякі переваги порівняно з традиційними способами пошуку кінофільмів. Основні переваги таких веб-застосунків включають:

- Зручність. Користувачі можуть швидко та легко знайти фільми, які відповідають їхнім вподобанням та інтересам, без необхідності витрачати час на пошук самостійно. Веб-застосунки для підбору фільмів можуть зберігати вибір користувачів та рекомендувати нові фільми, які відповідають їхнім інтересам.
- Розширений вибір. Веб-застосунки для підбору фільмів можуть містити значно більший вибір фільмів, ніж традиційні види кінотеатрів або прокатних компаній. Користувачі можуть шукати фільми зі своїх уподобань, незалежно від того, чи були вони колись показані в кінотеатрі, або чи є у прокаті в даний момент.
- Індивідуальний підбір. Веб-застосунки для підбору фільмів можуть рекомендувати фільми, які відповідають конкретним інтересам користувачів, з урахуванням їхніх попередніх переглядів, рейтингів та відгуків.
- Доступність. Веб-застосунки для підбору фільмів можуть бути доступними з будь-якого пристрою з підключенням до Інтернету.

Незважаючи на переваги, веб-застосунки для підбору фільмів мають деякі недоліки, які можуть вплинути на їх ефективність та користування. Основні недоліки таких веб-застосунків включають:

- Недостатня точність рекомендацій. Хоча веб-застосунки для підбору фільмів можуть враховувати попередні вибори та інтереси користувачів, але іноді рекомендації можуть бути не точними або не відповідати реальним інтересам користувачів.

- Надмірна залежність від алгоритмів. Веб-застосунки для підбору фільмів можуть використовувати складні алгоритми для аналізу виборів користувачів та рекомендацій фільмів, але це може призводити до залежності від алгоритмів, що може бути недостатньо гнучким для задоволення різноманітних потреб користувачів.
- Обмеження доступу до нових релізів. Веб-застосунки для підбору фільмів можуть не мати доступу до нових релізів або бути обмеженими залежно від регіону проживання користувачів, що може призвести до обмеження вибору фільмів.
- Проблеми з приватністю даних. Веб-застосунки для підбору фільмів можуть збирати та аналізувати особисті дані користувачів, що може викликати проблеми з приватністю та безпекою даних.

Отже, веб-застосунки для підбору фільмів мають свої переваги, але також мають недоліки, які можуть впливати на їхню ефективність та користування. Для зменшення впливу цих недоліків можуть бути використані різні підходи, такі як поліпшення точності рекомендацій, збільшення гнучкості алгоритмів, поширення доступу до нових релізів та забезпечення безпеки та приватності даних користувачів. Такі заходи можуть покращити ефективність та користування веб-застосунків для підбору фільмів та зробити їх більш привабливими для користувачів.

1.1.4. Перспективи розвитку ринку підбору фільмів

Останні роки веб-застосунків для підбору фільмів показують стабільний ріст популярності.

Однією з головних тенденцій розвитку ринку є зростання впливу штучного інтелекту на функціональні можливості веб-застосунків для підбору фільмів. Інтелектуальні системи здатні аналізувати поведінку користувача, що

дозволяє більш точно рекомендувати фільми та зробити процес пошуку більш персоналізованим[2].

Крім того, очікується зростання конкуренції на ринку, яка буде спричинена появою нових веб-застосунків та зміною стратегій тих, що вже працюють на ринку. Нові рішення від розробників можуть містити у собі інноваційні функціональні можливості, які дозволять користувачам зробити більш точний та швидкий вибір фільмів.

Однак, необхідно враховувати також низку викликів та проблем, з якими стикаються веб-застосунки для підбору фільмів. Наприклад, питання захисту даних та приватності користувачів, недостатньо точних рекомендацій у випадку нестандартних запитів та інших факторів.

Усі ці фактори сприятимуть розвитку ринку підбору фільмів, але також вимагатимуть від розробників постійного удосконалення та модернізації веб-застосунків. На сьогоднішній день, більшість веб-застосунків для підбору фільмів використовують схожі алгоритми та методи рекомендації, що може призвести до втрати конкурентної переваги.

У цілому, ринок підбору фільмів має стабільні перспективи розвитку у зв'язку зі зростанням популярності веб-застосунків та розвитком нових технологій, але вимагає постійного удосконалення та модернізації, щоб задовольнити потреби користувачів та зберегти конкурентну перевагу на ринку[1].

1.2. Приклади веб-застосунків для підбору фільмів та їх порівняння

Ідея веб-застосунків для підбору фільмів не є новою. Сьогодні існує багато прикладів, які базуються на різних алгоритмах та методах рекомендацій.

1.2.1. Розгляд існуючих веб-застосунків для підбору фільмів та їх функціональності

Нижче наведені деякі приклади таких веб-застосунків та їх функціональні можливості.

IMDb - це один з найвідоміших та найпопулярніших веб-сайтів для пошуку фільмів та інформації про них. Сайт містить велику базу даних фільмів, серіалів та телепередач, а також відгуки глядачів, оцінки, трейлери та інші матеріали. IMDb пропонує рекомендації фільмів на основі інформації про попередні вибори глядачів, також доступна можливість пошуку фільмів за різними критеріями, наприклад, за жанром, режисером, актором, рейтингом тощо.

Netflix - це платформа для онлайн-перегляду фільмів та серіалів. При підборі фільмів для користувачів, Netflix використовує особисту систему рекомендацій, яка аналізує попередні вибори глядачів, та на основі цього пропонує подібні фільми. Крім того, Netflix дозволяє глядачам створювати свої списки фільмів та серіалів, що сподобалися, щоб потім знайти їх зручніше.

Letterboxd - це соціальна мережа для кінофілів, де можна зберігати списки фільмів, що вам сподобалися, відзначати переглянуті та бажані до перегляду фільми, а також ділитися враженнями про них з іншими користувачами. Letterboxd також пропонує персоналізовані рекомендації фільмів, які базуються на попередніх виборах та оцінках користувача, а також на тематичних списків, створених іншими користувачами. Крім того, на Letterboxd можна шукати фільми за жанрам, країною виробництва, режисером, акторами, а також за різними тегами, що дозволяє знайти фільми, які відповідають вашим смакам та інтересам.

Megogo - це платформа для онлайн-перегляду фільмів, серіалів, телепередач та іншого відеоконтенту. Крім можливості перегляду контенту, megogo також має функціонал для пошуку та підбору фільмів на основі інформації про попередні вибори користувачів, а також за критеріями, такими

як жанр, країна виробництва, рейтинг тощо. Окрім того, megogo має можливість створення списків фільмів та серіалів, які вам сподобалися, щоб потім знайти їх зручніше.

1.2.2. Порівняння існуючих веб-застосунків для підбору фільмів.

Веб-застосунки для підбору фільмів можна порівняти за наступними критеріями: функціональні можливості, тип рекомендацій, кількість фільмів у базі даних, цінова політика

Таблиця 1.1 — Порівняння існуючих веб-застосунків для підбору фільмів

Застосунок	Функціональні можливості	Тип рекомендацій	Кількість фільмів у базі даних	Цінова політика
IMDb	Пошук фільмів за жанром, режисером, актором, рейтингом тощо, відгуки глядачів, оцінки, трейлери та інші матеріали.	Колективні рекомендації на основі попередніх виборів глядачів.	Більше 7 мільйонів	Безкоштовний
Netflix	Особиста система рекомендацій, створення списків фільмів та серіалів.	Індивідуальні рекомендації на основі попередніх виборів глядачів.	Більше 15 тисяч	Підписка

Продовження таблиці 1.1

Megogo	Пошук та підбір фільмів за критеріями, такими як жанр, країна виробництва, рейтинг тощо, створення списків фільмів та серіалів.	Колективні рекомендації на основі попередніх виборів глядачів.	Більше 50 тисяч	Підписка
Letterboxd	Пошук та відбір фільмів за різними критеріями, персоналізовані рекомендації, створення списку перегляду	Колективні рекомендації на основі попередніх виборів глядачів.	Більше 2 мільйонів	Безкоштовно (з обмеженнями) або платна підписка

1.3. Принцип реалізації подібних програм

Для того, щоб зрозуміти, які алгоритми використовують веб-застосунки для підбору фільмів, необхідно розглянути основні підходи до рекомендаційних систем.

1.3.1. Аналіз алгоритмів для веб-застосунків підбору фільмів

Одним з найбільш популярних підходів є колаборативний фільтр. Цей підхід базується на тому, що якщо два користувачі подібні за своїми

уподобаннями, тобто мають схожий історії перегляду фільмів, то їм рекомендуються подібні фільми. Колаборативний фільтр може бути заснований на підсумовуванні оцінок, що ставлять користувачі фільмам, або на знаходженні схожих користувачів за допомогою аналізу їх історії перегляду.

Інший підхід - це контентний фільтр. Він базується на аналізі характеристик фільмів, таких як жанр, актори, режисер і т.д., і порівнює їх зі списком уподобань користувачів. Якщо користувач показав інтерес до фільмів з певними характеристиками, то йому рекомендуються фільми з подібними характеристиками.

Також можуть використовуватися гібридні підходи, що комбінують колаборативний та контентний фільтри.

Для покращення якості рекомендацій можуть використовуватися додаткові методи, такі як розпізнавання образів, аналіз текстів або використання машинного навчання.

1.3.2. Алгоритм колаборативного фільтру

Колаборативний фільтр - це один з основних підходів до рекомендаційних систем, який базується на аналізі взаємодії користувачів з певним контентом. Головна ідея полягає в тому, що якщо два користувачі відповідають за своїми вподобаннями, то вони можуть рекомендувати одне одному ті самі об'єкти, такі як фільми, музика, книги і т.д[3].

Існують два основних підходи до колаборативного фільтру: заснований на підсумовуванні оцінок та заснований на знаходженні схожих користувачів.

У першому підході для кожного об'єкта відстежується оцінка, яку ставлять йому користувачі. Для того, щоб зробити рекомендацію для конкретного користувача, знаходяться інші користувачі, які мають подібні смаки і оцінки, і на основі їхніх оцінок рекомендуються нові об'єкти.

У другому підході замість оцінок використовуються історії перегляду або купівлі об'єктів. За допомогою аналізу історії перегляду користувача

знаходяться інші користувачі, які мають схожу історію, і на основі їхніх переглядів рекомендуються нові об'єкти.

У реалізації колаборативного фільтру використовуються різні алгоритми машинного навчання, такі як кластеризація, нейронні мережі, байесівські мережі і т.д. Крім того, для покращення якості рекомендацій можуть використовуватися різні техніки обробки даних, такі як зменшення розмірності даних, факторизація матриць, розподільчі методи, а також методи зменшення шуму.

Одним з найбільш поширених алгоритмів колаборативного фільтру є Singular Value Decomposition (SVD), який базується на розкладі матриці оцінок користувачів і об'єктів на менші матриці, що репрезентують їхні характеристики. Цей алгоритм дозволяє зменшити розмірність даних і визначити складні залежності між користувачами та об'єктами, що дозволяє зробити більш точні рекомендації.

Ще одним алгоритмом, що використовується для колаборативного фільтру, є Alternating Least Squares (ALS), який базується на оптимізації розкладу матриці оцінок на основі альтернативної мінімізації квадратичної помилки. Цей алгоритм є більш ефективним за SVD, оскільки може працювати з більш великими наборами даних та більш гнучко пристосовуватися до зміни взаємодії користувачів з об'єктами.

Ще одним підходом до реалізації колаборативного фільтру є використання нейронних мереж. Наприклад, можна використовувати рекурентні нейронні мережі (RNN), які дозволяють моделювати послідовну взаємодію користувачів з об'єктами. Також можна використовувати глибокі нейронні мережі (DNN), які дозволяють моделювати складні залежності між користувачами та об'єктами.

Узагалі, реалізація колаборативного фільтру може бути складною задачею, оскільки вона вимагає великої кількості даних та обчислювальних ресурсів. Для покращення результатів можуть використовуватися різні методи і

підходи. Одним з них є використання глибокого навчання (deep learning) з метою створення моделей рекомендацій.

Нейронні мережі можуть забезпечити значно більш точні результати порівняно з традиційними методами колаборативного фільтрування. Зокрема, вони можуть автоматично виявляти складні залежності між різними факторами, які впливають на відповідність товару чи послуги конкретному користувачеві.

Крім того, можна використовувати гібридні підходи, що поєднують колаборативне та контентне фільтрування. Вони дозволяють враховувати не тільки поведінку користувачів, а й характеристики товарів чи послуг, що збільшує точність рекомендацій.

Нарешті, можна використовувати алгоритми підсиленого навчання (reinforcement learning), що дозволяють розробити систему, яка може вчитися та покращуватися з часом, навіть якщо отримані відгуки користувачів неповні або суперечливі.

Отже, використання різних методів та підходів може допомогти покращити результати колаборативного фільтрування та забезпечити більш точні та персоналізовані рекомендації користувачам.

1.3.3. Алгоритм контент фільтру

Контент фільтр - це алгоритм рекомендацій, який використовує відомості про властивості або характеристики об'єктів (контенту) для прогнозування того, що користувачі будуть цікавитися. Наприклад, якщо користувач шукає фільм з актором Томом Хенксом, то система може рекомендувати інші фільми з участю цього актора[3].

Реалізація контент-фільтра вимагає двох основних етапів: визначення характеристик контенту та розробки алгоритму для визначення подібності між контентом.

Визначення характеристик контенту: першим кроком у реалізації контент-фільтра є визначення характеристик контенту, які будуть

використовуватися для порівняння об'єктів. Наприклад, якщо ми використовуємо контент-фільтр для рекомендацій фільмів, то можливі характеристики можуть включати жанр, режисера, акторів, рейтинг, рік випуску тощо. Визначення характеристик контенту може бути автоматизовано за допомогою методів обробки природних мов або машинного навчання.

Розробка алгоритму для визначення подібності між контентом: після визначення характеристик контенту, наступним кроком є розробка алгоритму для визначення подібності між контентом. Цей алгоритм може використовувати методи статистичної аналітики або машинного навчання для порівняння характеристик контенту і визначення того, наскільки схожі вони.

Наприклад, одним з найпростіших алгоритмів для визначення подібності між контентом є косинусна схожість. Для цього кожен об'єкт (наприклад, фільм) може бути представлений як вектор, де кожна характеристика є координатою, а значення - значенням відповідної характеристики. Далі, вектори порівнюються за допомогою косинусної міри схожості, яка обчислює кут між векторами. Чим більше кут між векторами, тим менша схожість між ними.

Крім косинусної схожості, існує багато інших алгоритмів для визначення подібності між контентом, таких як рекомендації на основі байесових мереж, нейронних мереж або дерева рішень.

Після визначення подібності між контентом, система може зробити рекомендації користувачам на основі їхніх переглядів або рейтингів. Наприклад, якщо користувач переглянув деякий фільм з конкретним актором, система може рекомендувати інші фільми з цим актором або із схожими характеристиками (наприклад, того ж жанру чи режисера).

Отже, контент фільтр - це ефективний метод для рекомендаційних систем, особливо там, де недостатньо даних про користувачів або колаборативний фільтр не підходить. Реалізація контент-фільтру включає визначення характеристик контенту та розробку алгоритму для визначення подібності між ними. Цей підхід можна застосовувати для рекомендацій у

різних сферах, включаючи фільми, музику, книги, товари і послуги.

1.3.4. Підходи систем рекомендацій

Основні принципи систем рекомендацій, які були описані, базуються на різних підходах, що використовуються для побудови рекомендацій. Нижче представлено детальний огляд двох з них - системи на основі рейтингу та системи на основі лайків.

Системи на основі рейтингу будуються на основі оцінок користувачів певних об'єктів, наприклад фільмів. Ці оцінки можуть бути числовими (наприклад, від 1 до 5, або від 1 до 10). Для побудови рекомендаційної системи на основі рейтингу необхідно мати достатню кількість оцінок, які можуть бути використані для прогнозування подальших вподобань користувачів.

Переваги систем на основі рейтингу:

- Ефективні при наявності достатньої кількості даних.
- Забезпечують високу точність рекомендацій при залученні багатьох користувачів.

Недоліки систем на основі рейтингу:

- Потрібна достатня кількість оцінок для побудови рекомендацій.
- Складні у реалізації.
- Більший час виконання підбірки рекомендацій.

Системи на основі лайків, як правило, використовуються для побудови рекомендацій на основі поведінки користувачів в соціальних мережах. Наприклад, під час перегляду відео на TikTok користувач може натиснути кнопку "лайк" для вираження свого вподобання до контенту. Для побудови рекомендацій на основі лайків система використовує історію вподобань користувача, аналізуючи його попередні дії в соціальній мережі.

Переваги систем на основі лайків:

- Вони дуже прості для використання, оскільки вони не потребують спеціальних знань або взаємодії користувача з системою.

- Реалізація алгоритму підбірки рекомендацій значно легша порівнюючи з рейтинговою системою.
- Більша швидкість знаходження рекомендацій.

Недоліки систем на основі лайків:

Системи на основі лайків будуть менш точними, оскільки в рейтинговій системі більше варіантів оцінювання.

Отже, обидві системи рекомендацій мають свої переваги та недоліки і використовуються в залежності від контексту застосування. Системи на основі рейтингу є ефективними при наявності достатньої кількості даних, тоді як системи на основі лайків є ефективними при невеликій кількості даних та використовуються в основному в соціальних мережах. Варто також зазначити, що багато провідних компаній пішли шляхом лайкової системи, тому що більшість користувачів оцінювали речі рейтингом 4-5 або 1, що доволі легко перевести в лайк/дизлайк.

Для кращого розуміння порівняємо системи рекомендацій в таблиці.

Таблиця 1.2 — Схема моделі Variants

Система рекомендацій	Основні принципи	Переваги	Недоліки
Система на основі рейтингу	Оцінки користувачі в об'єктів	Ефективність при наявності достатньої кількості даних, висока точність	Потреба у достатній кількості оцінок, складність реалізації, більший час виконання алгоритму, складніша реалізація алгоритму

Продовження таблиці 1.2

Система на основі лайків	Вираження вподобань користувача до контенту	Простота використання, легкість реалізації алгоритму, швидкість алгоритму	Менша точність

1.4. Постановка задачі. Технічне завдання для розробки

Для того, щоб створити ефективний веб-застосунок для підбору фільмів, необхідно зрозуміти, які вимоги мають користувачі до такого сервісу.

1.4.1. Аналіз вимог користувачів до веб-застосунку для підбору фільмів

Основні вимоги користувачів до веб-застосунку для підбору фільмів можуть бути наступними:

- Зручність використання: веб-застосунок повинен мати інтуїтивно зрозумілий та легкий інтерфейс, що дозволяє користувачам легко та швидко знаходити необхідну інформацію. Крім того, важливо, щоб застосунок був доступним для користувачів з будь-якого пристрою, такого як комп'ютер, смартфон або планшет.
- Точність рекомендацій: веб-застосунок повинен бути здатний рекомендувати фільми, які відповідають інтересам та передвідомостям користувача. Для цього можуть використовуватись алгоритми машинного навчання та аналізу даних.
- Наявність різних фільтрів: веб-застосунок повинен мати можливість фільтрувати фільми за різними критеріями, наприклад жанром, рейтингом, роком випуску та іншими.

- Можливість зберігання списку обраних фільмів - веб-застосунок повинен мати можливість зберігання списку обраних фільмів користувача, щоб він міг зручно повернутися до цього списку пізніше.
- Наявність інформації про фільми: веб-застосунок повинен мати детальну інформацію про кожен фільм, таку як опис сюжету, трейлер, акторський склад, рейтинг та інші деталі.
- Врахування особистих налаштувань користувачів: веб-застосунок повинен мати можливість враховувати особисті налаштування та вподобання користувача, щоб рекомендації були максимально точними та відповідали індивідуальним потребам кожного користувача.

1.4.2. Опис архітектури веб-застосунку для підбору фільмів:

Архітектура веб-застосунку для підбору фільмів повинна базуватися на клієнт-серверній архітектурі. Клієнтська частина веб-застосунку повинна бути розроблена на базі веб-фреймворка, який забезпечує створення інтерфейсу користувача та взаємодію з сервером за допомогою протоколу HTTP. Серверна частина повинна бути розроблена з використанням мови програмування, яка забезпечує підключення до бази даних та реалізацію алгоритмів для рекомендацій фільмів.

Архітектура веб-застосунку повинна складатися з наступних компонентів:

- Клієнтська частина, що містить інтерфейс користувача та логіку взаємодії з сервером.
- Серверна частина, що містить алгоритми для рекомендацій фільмів та базу даних фільмів та користувачів.
- База даних фільмів, що містить інформацію про фільми та їх характеристики, користувачів, та їх вподобання фільмів.

Клієнтська частина повинна містити інтерфейс користувача, який буде відображати рекомендації фільмів та дозволяти користувачам оцінювати

фільми. Також, вона повинна забезпечувати взаємодію з серверною частиною за допомогою протоколу HTTP.

Серверна частина повинна містити алгоритми для рекомендацій фільмів, які будуть використовувати базу даних фільмів та користувачів для аналізу та підбору рекомендацій. Також, серверна частина повинна мати можливість зберігати та оновлювати дані в базі даних фільмів та користувачів.

База даних повинна містити інформацію про фільми, таку як назву, жанр, рік випуску, режисера та акторів, а також опис та посилання на трейлер. Крім того, повинна містити інформацію про користувачів, таку як їх ім'я, електронну пошту та вподобання фільмів.

Взаємодія між клієнтською та серверною частинами повинна здійснюватися за допомогою протоколу HTTP, з використанням RESTful API, який дозволяє передавати дані між клієнтом та сервером у вигляді структурованих запитів та відповідей.

Усі компоненти веб-застосунку повинні бути розгорнуті на веб-сервері, що забезпечить доступ користувачів до веб-застосунку за допомогою мережі Інтернет. Крім того, для забезпечення безпеки та захисту даних користувачів та фільмів, слід використовувати механізми аутентифікації та авторизації користувачів.

1.4.3. Постановка задачі. Технічне завдання для розробки веб-застосунку для підбору фільмів

Шляхом проведення аналізу вимог, опису необхідної архітектури веб-застосунку та принципів створення подібних веб-застосунків, було сформовано технічне завдання роботи.

Спроектувати веб-застосунок для підбору фільмів з урахуванням уподобань на основі системи лайків, що буде мати каталог популярних фільмів, а рекомендації будуть створені за допомогою гібридного алгоритму колаборативного та контент фільтрів

Для вирішення поставленої мети потрібно послідовно вирішити декілька задач:

- розробити зручний та легкий користувацький інтерфейс;
- розробити БД для зберігання даних про фільми та їх характеристики, користувачів їх даних та вподобань;
- створити сервер та з’єднати з базою даних ;
- з’єднати клієнтську та серверну частину;
- розробити алгоритм рекомендацій фільмів.

РОЗДІЛ 2

ПРОГРАМНО-ТЕХНОЛОГІЧНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ПІДБОРУ ФІЛЬМІВ З УРАХУВАННЯМ ВПОДОБАНЬ

2.1. Вибір технологій для розробки веб-застосунку

Для створення електронної системи я обрав такі технології створення web-застосунків:

1) Мова програмування - Javascript
2) Фреймворк для створення UI інтерфейсу - Vue
3) Сервер - Node.js та фреймворк Express.js
4) База Даних - PostgreSQL
5) Контейнеризація - Docker.

2.2. Клієнтська частина веб-застосунку

Фронтенд - це частина веб-розробки, яка відповідає за створення клієнтського інтерфейсу веб-сайту або веб-застосунку, з яким користувач взаємодіє безпосередньо через браузер. Фронтенд розробка займається створенням веб-сторінок та їх елементів, які користувач може бачити та з якими він може взаємодіяти. Фронтенд розробник використовує мови, такі як HTML, CSS та JavaScript, для створення сторінок та забезпечення їхньої функціональності.[6]

HTML (Hypertext Markup Language) - це мова розмітки, яка використовується для створення структури веб-сторінок, визначення рівня заголовку, тексту, посилань, таблиць, форм та інших елементів на сторінці.

CSS (Cascading Style Sheets) - це мова стилів, яка використовується для візуального оформлення веб-сторінок, включаючи зміну кольорів, розмірів та типів шрифтів, розташування елементів на сторінці та інше.

JavaScript - це скриптова мова програмування, яка використовується для створення динамічної функціональності на сторінці. JavaScript дозволяє розробникам створювати взаємодію користувача з веб-сторінкою, додавати анімацію, перевіряти правильність введення даних в форми та інше.

Із зростанням вимог щодо зручності використання та відповідності стандартам дизайну, фронтенд розробка стала ще більш важливою. Користувачі очікують, що веб-застосунок буде працювати швидко, безперебійно та мати привабливий та легко зрозумілий інтерфейс. Все це залежить від якості фронтенд розробки.

Успішний веб-застосунок повинен мати добре розроблений та оптимізований фронтенд, який дає можливість користувачам з легкістю взаємодіяти з системою та отримувати потрібну інформацію без затримок і помилок.

Усе це показує, що фронтенд є важливою складовою веб-розробки, оскільки він безпосередньо впливає на користувацький досвід та задоволеність від використання веб-застосунку.

2.2.1. Вибір фреймворка

В сучасному світі, коли веб-програмування є необхідною складовою багатьох проектів, використання Javascript фреймворків та бібліотек стає все популярнішим. Javascript має багато можливостей і дозволяє розробити будь-який веб-сайт, однак використання його у “голому” виді без бібліотек не є найкращим рішенням, бібліотеки дозволяють пришвидшити розробку та зменшити кількість коду. Раніше популярним рішенням була бібліотека JQuery, однак, сьогодні, на її місці посіли нові фреймворки та бібліотеки, що дають значно більше можливостей.

Фреймворк - це набір вже написаного коду, бібліотек та інструментів, що забезпечують загальну структуру та функціональність для розробки програмного забезпечення. Він надає розробникам зручний та ефективний

спосіб розробки додатків, зменшуючи час та зусилля, необхідні для написання коду з нуля. Фреймворки дозволяють розробникам сконцентруватися на розв'язанні конкретних завдань та деталей, використовуючи вже готові інструменти та компоненти, які забезпечують стандартизацію та підтримку певного стилю програмування.

Фронтенд фреймворки - це засоби для швидкої та ефективної розробки веб-додатків. Вони надають готовий набір інструментів та компонентів для створення інтерфейсу користувача, а також забезпечують підтримку різноманітних функцій, таких як маршрутизація, стан додатку, управління станом компонентів тощо.

Переваги використання фреймворків та бібліотек над "голим" JavaScript полягають у тому, що вони надають розробникам готові інструменти для швидкої та ефективної розробки веб-додатків. Ось декілька конкретних переваг:

1) Швидкість розробки: Фреймворки та бібліотеки надають зручний інтерфейс для розробки, що дозволяє зосередитися на бізнес-логіці додатку, а не на деталях роботи з DOM або AJAX запитами. Це дозволяє зменшити час розробки та зосередитися на вирішенні більш складних задач.

2) Підтримка різноманітних функцій: Фреймворки та бібліотеки надають набір готових рішень для роботи зі станом додатку, маршрутизації, валідації форм, анімації та інших функцій. Це дозволяє розробникам швидко реалізовувати складні функціональні вимоги, зменшуючи кількість коду, який потрібно написати.

3) Стандартизація: Фреймворки та бібліотеки надають стандартизовані підходи до розробки, що дозволяє зменшити кількість помилок, пов'язаних з використанням різних підходів. Крім того, стандартизація дозволяє з легкістю

використовувати різних розробників на проекті, які можуть бути знайомі з тими ж фреймворками та бібліотеками.

4) Підтримка спільноти: Фреймворки та бібліотеки мають широку спільноту розробників, яка надає підтримку та допомогу вирішенні проблем. Розробники можуть швидко знайти відповіді на форумах, знайти кодові приклади та документацію, що значно полегшує розробку.

5) Краща безпека: Фреймворки та бібліотеки вже містять безпекові заходи та виправлення багів, що дозволяє зменшити ризик вразливостей в додатку. Крім того, більшість фреймворків та бібліотек мають активну спільноту, яка швидко виявляє та виправляє можливі проблеми безпеки.

6) Розширюваність: Фреймворки та бібліотеки надають можливість розширення функціоналу додатку за допомогою плагінів та розширень. Це дозволяє розробникам легко внести нові функції та покращення в додаток.

Тестування: Фреймворки та бібліотеки надають зручний спосіб тестування додатку, що дозволяє виявляти та виправляти помилки раніше, зменшуючи час, необхідний для розробки.

Узагальнюючи, фреймворки та бібліотеки надають зручний та швидкий спосіб для розробки веб-додатків, дозволяючи зосередитися на бізнес-логіці додатку та реалізувати складні функціональні вимоги з меншим обсягом коду та з меншим ризиком вразливостей. Крім того, фреймворки та бібліотеки надають стандартизовані підходи та підтримку спільноти, що дозволяє ефективно працювати з командою розробників та швидко вирішувати проблеми.

Застосування фреймворків в фронтенді дозволяє зменшити час розробки, забезпечити більшу стабільність та надійність веб-додатків, а також забезпечити більшу масштабованість проекту. Фреймворки забезпечують

підтримку стандартів та найкращих практик у веб-розробці, а також зменшують кількість повторюваного коду. Вони також дозволяють розробникам працювати з великими командами та робити розробку більш організованою та систематичною.

Проте, використання фреймворків має свої недоліки. Один з них - залежність від сторонніх бібліотек, що може затримати процес розробки, якщо будь-яка з бібліотек буде змінена або зупинена. Також, використання фреймворків може зробити код менш зрозумілим для початківців, які ще не мають досвіду з певним фреймворком.

Отже, використання фреймворків у фронтенді є важливим інструментом для швидкої та ефективної розробки веб-додатків, але варто ретельно обирати фреймворк, який найкраще підходить для потреб проекту та команди розробників.

Однією з найпопулярніших бібліотек у фронтенді є React. React - це саме бібліотека, а не фреймворк, оскільки він не надає “жорстких” правил написання коду та архітектури. Його основна концепція полягає в розбитті інтерфейсу на незалежні компоненти, кожен з яких може мати свій внутрішній стан та логіку. React надає зручні інструменти для роботи з компонентами, такі як JSX - синтаксис, що дозволяє описувати компоненти з використанням HTML-подібних тегів та спеціальних властивостей.

Іншим популярним фреймворком є Angular. Він надає більш повний набір інструментів для розробки веб-додатків, зокрема механізми маршрутизації, перевірки валідності даних та управління станом додатку. Angular використовує TypeScript - сильно типізовану версію JavaScript, що дозволяє зменшити кількість помилок у коді.

Vue є ще одним популярним фреймворком у фронтенді. Він має більш легкий та простий до вивчення синтаксис, що робить його популярним серед

початківців у веб-розробці. Vue надає зручні інструменти для роботи з компонентами та реактивними даними. Також Vue являється найлегшим та найшвидшим фреймворком.

Таблиця 2.1 — Порівняння фронтенд фреймворків

Характеристика	Vue	React	Angular
Розмір	~20 КБ	~ 40 КБ	~500 КБ

Продовження таблиці 2.1

Тип	Фреймворк	Бібліотека	Фреймворк
Підтримка віртуального DOM	Так	Так	Так
Навчальна крива	Низька	Середня	Висока
Популярність	Середня	Висока	Низька
Спільнота	Середня	Висока	Низька
Швидкість	Висока	Середня	Низька

Для проекту найкращим варіантом буде фреймворк Vue, так як він взяв найкраще зі своїх конкурентів React та Angular, має більше можливостей ніж реакт, однак не переповнений ними на відміну від Angular. Веб-застосунок буде легко створити та підтримувати, а його швидкість буде найкраща.

2.2.2. Бібліотеки для веб-застосунку

Бібліотека - це колекція написаних кодом функцій та методів, які виконують певні завдання та можуть бути використані для спрощення розробки веб-додатків.

У веб-розробці бібліотеки використовуються для розв'язання різних задач, таких як маніпулювання DOM, робота з AJAX запитами, валідація форм, анімація та інше. Бібліотеки можуть бути написані на різних мовах програмування, таких як JavaScript, CSS, HTML та інші.

Одна з головних переваг використання бібліотек полягає у тому, що розробники можуть використовувати готові рішення замість того, щоб писати весь код з нуля. Це дозволяє зменшити час розробки та зосередитися на вирішенні більш складних завдань. Крім того, бібліотеки можуть мати широку спільноту розробників, що надає підтримку та допомогу вирішенні проблем.

Перелік основних бібліотек для фронтенду веб-застосунку:

1) Vuetify - це бібліотека компонентів для Vue.js, яка надає розробникам можливість швидко та ефективно створювати стильні та функціональні веб-додатки. Бібліотека включає в себе понад 80 різних компонентів, включаючи кнопки, форми, таблиці, каруселі та інші. Основною перевагою Vuetify є те, що вона дозволяє зосередитися на бізнес-логіці додатку, а не на деталях візуального оформлення. Компоненти в бібліотеці мають стильний та сучасний дизайн, що дозволяє розробникам швидко створювати вигляд своїх додатків. Окрім цього, Vuetify надає розробникам можливість налаштувати кольорову палітру, типографію та інші візуальні елементи своїх додатків з використанням Sass та стандартних CSS змінних.

2) Vue Router - це офіційний маршрутизатор для Vue.js. Це бібліотека, яка дозволяє створювати SPA (односторінкові додатки) з допомогою маршрутизації на

стороні клієнта. Vue Router дозволяє визначати маршрути та відображати відповідні компоненти на сторінці. Це дозволяє розбити додаток на набір компонентів та відображати їх на сторінках в залежності від URL-адреси. Таким чином, є можливість створити додаток зі сторінками, які не перезавантажуються під час переходів між ними, що забезпечує швидке та зручне користування.

Основні можливості, які надає Vue Router:

- Маршрутизація на стороні клієнта: Vue Router дозволяє використовувати маршрутизацію на стороні клієнта з допомогою HTML5 історії або хеш-роутингу, що дозволяє створювати додатки, які працюють без перезавантаження сторінки під час переходів між сторінками.
- Вкладена маршрутизація: Vue Router дозволяє створювати вкладені маршрути, що дозволяє створювати більш складні структури додатків.
- Перехоплення маршрутів: Vue Router дозволяє перехоплювати маршрути та перенаправляти користувачів на інші сторінки або дії, наприклад, для перевірки прав доступу або підтвердження запиту користувача.
- Параметризовані маршрути: Vue Router дозволяє створювати параметризовані маршрути, які дозволяють передавати параметри у вигляді URL-адреси.
- Динамічні маршрути: Vue Router дозволяє створювати динамічні маршрути, які дозволяють створювати маршрути на льоту з допомогою JavaScript.

Pinia - це бібліотека Vue.js для керування станом, яка дозволяє створювати та керувати стан додатку у декларативний спосіб. Основна ідея Pinia полягає в тому, що вона спрощує управління станом додатку, дозволяючи

розглядати його як збірну структуру даних, яка складається зі станів (state), геттерів (getters) та мутацій (mutations). Відповідно до цього, створюються зберігаючі об'єкти (store), які містять в собі ці елементи та забезпечують просту та зрозумілу інтерфейс для управління станом.

Axios - це бібліотека JavaScript для здійснення HTTP-запитів з браузера або з Node.js. Axios дозволяє легко взаємодіяти з зовнішніми API, отримувати та надсилати дані, обробляти помилки та інші HTTP-запити.

Основні функції, які надає Axios:

- Підтримка різних методів HTTP-запитів: Axios дозволяє використовувати різні методи запитів, такі як GET, POST, PUT, PATCH, DELETE, та інші.
- Підтримка інтерцепторів: Axios дозволяє додавати інтерсептори для перехоплення та обробки запитів та відповідей. Це дозволяє виконувати певні дії перед відправкою запиту або перед обробкою відповіді, наприклад, додавання заголовків або обробка помилок.
- Автоматична серіалізація та десеріалізація даних: Axios автоматично серіалізує дані у JSON для відправлення на сервер та десеріалізує отримані відповіді у JavaScript-об'єкти.
- Обробка помилок: Axios дозволяє виконувати обробку помилок при відправці запитів. Наприклад, ви можете перевірити код статусу відповіді та зробити відповідні дії в залежності від цього.
- Підтримка promises та async/await: Axios дозволяє використовувати promises та async/await для зручного роботи з запитами та відповідями.

- Підтримка заголовків та параметрів запиту: Axios дозволяє встановлювати заголовки запиту та параметри запиту для передачі додаткової інформації на сервер.
- Підтримка скачування файлів: Axios дозволяє скачувати файли на клієнт з сервера.

Загалом, Axios - це потужна та зручна бібліотека, яка дозволяє з легкістю здійснювати HTTP-запити з JavaScript-додатків та взаємодіяти з зовнішніми API.

2.2.3. Архітектура фронтенд частини

Архітектура - це планування та проектування структури та організації системи, що визначається його компонентами, їх взаємодією та відповідальностями. Це може включати проектування архітектури програмного забезпечення, архітектури баз даних, мережевої архітектури та інші.

Архітектура дозволяє розробникам проектувати складні системи, забезпечуючи зручну розширюваність та збільшення масштабу системи, зниження складності та забезпечення ефективності взаємодії між компонентами. Крім того, правильна архітектура може зменшити вартість розробки та підтримки системи.

Поняття low coupling та high cohesion є важливими для успішної архітектури системи.

Low coupling (слабка зв'язність) означає, що модулі програми повинні між собою максимально незалежні, тобто зміни в одному модулі не повинні впливати на роботу інших модулів. Це дозволяє знизити залежності між модулями та зробити систему більш гнучкою та легко змінюваною. Наприклад, якщо ми маємо два модулі, які взаємодіють між собою, то ми можемо зробити

так, щоб вони спілкувалися через інтерфейс, а не прямо між собою. Це зменшить залежність між модулями та зробить систему менш вразливою до змін.

High cohesion (високе зчеплення) означає, що код в межах одного модуля повинен бути пов'язаний між собою тісніше, тобто класи та функції, які пов'язані між собою, повинні бути в тому ж модулі. Це дозволяє зробити код більш зрозумілим та легко змінюваним, оскільки логічно пов'язаний код буде знаходитися в одному місці. Наприклад, якщо ми маємо модуль для роботи з базою даних, то всі класи та функції, які стосуються цього модуля, повинні бути в одному файлі або модулі.

Узагалі, принципи low coupling та high cohesion допомагають зменшити складність коду та зробити систему більш гнучкою та легко змінюваною. Вони часто використовуються в проектуванні архітектури програмного забезпечення.

Для веб-застосунку було обрано модульну архітектуру. Модульна архітектура означає, що програмне забезпечення розбито на невеликі, самодостатні модулі, які можна легко міняти та розширювати. Кожен модуль виконує певні функції та взаємодіє з іншими модулями тільки за допомогою визначеного API.

Модульна архітектура у фронтенді дозволяє забезпечити більшу гнучкість та швидкість розробки, оскільки окремі модулі можуть бути розроблені та тестовані незалежно. Крім того, це дозволяє більш легко зберігати та перевикористовувати код, оскільки модулі можна легко видалити або замінити без впливу на інші частини системи.

В проекті кожен модуль буде зберігатися під папкою views, наприклад: views/Movies. Модуль може мати свої сторінки, утиліти, компоненти та сервіси, що використовуються лише в цьому модулі, таким чином код буде мати високе зчеплення, а оскільки кожен модуль буде незалежним від інших модулей таким чином код буде мати слабку зв'язність.

2.2.4. Реалізація фронтенду

Фронтенд частина була розроблена за допомогою фреймворку Vue 3 та бандлеру Vite. Проект має наступну структуру:

Таблиця 2.2 — Структура файлів та папок кореневої папки клієнту

Назва папки	Опис
node_modules	Папка в проекті, яка містить усі залежності, встановлені за допомогою пакетного менеджера npm або Yarn.
public	Папка для публічних файлів, зазвичай тут зберігаються favicon іконки.
src	Папка де зберігається код проекту.
package.json	JSON файл, що має повну інформацію про проект: назву, відомості, залежності, які потрібні для роботи та залежності для розробки, що необхідні лише під час розробки.

Продовження таблиці 2.2

vite.config.ts	конфігураційний файл для Vite, який дозволяє налаштувати різні параметри збірки проекту.
tsconfig.json	це конфігураційний файл для TypeScript-компілятора, який визначає параметри проекту, такі як налаштування компіляції, шляхи до файлів, використання зовнішніх бібліотек та інші параметри.

index.html	Головний HTML файл до якого приєднується Vue.
Dockerfile	це текстовий файл, який містить інструкції для автоматизованого створення контейнера Docker. Він складається з інструкцій та команд, що описують, як система повинна бути налаштована та сконфігурована в контейнері Docker.

Головний код зберігається у папці src:

Таблиця 2.3 — Структура файлів та папок папки src клієнту

Назва папки / файлу	Опис
assets	Папка для зберігання всіх статичних файлів (картинки, іконки).
components	Папка для збереження компонентів, що можуть бути використані у кількох модулях.

Продовження таблиці 2.3

layouts	Папка зі всіми layout.
plugins	Папка для налаштування всіх плагінів(наприклад UI бібліотеки Vuetify).
constants	Папка для зберігання глобальних змінних, наприклад кольорів проекту.
http	Папка, де зберігаються запити до серверу.

router	Папка зі всіми маршрутами застосунку та налаштування vue-router.
store	Папка, де зберігається глобальний стан веб-застосунку та методи для його отримання та зміни.
types	Папка для зберігання глобальних типів веб-застосунку.
views	Папка для зберігання модулей проекту.
App.vue	Головний компонент.
main.ts	файл для підключення Vue, та настройки всіх плагінів для нього.

У Проекті було створено 2 модулі: Movies та Authorization

Таблиця 2.4 — Структура файлів та папок модулю Movies

components	Папка, де зберігається компоненти модулю.
HomePage.vue	Головна сторінка застосунку.

Продовження таблиці 2.4

MoviesPage.vue	Сторінка всіх фільмів з пошуком, фільтрацією та сортуванням.
MyLikesPage.vue	Сторінка де зберігаються всі вподобані фільми користувача.
SingleMoviePage.vue	Сторінка фільму з його детальним описом.

Також варто перелічити компоненти модулю Movies

Таблиця 2.5 — Компоненти модулю Movies

MasonryHero.vue	Секція фільмів, що відображаються у вигляді Masonry сітки.
MovieBanner.vue	Головний банер сайту.
MovieCard.vue	Картка фільму.
MovieHero.vue	Карусель блок фільмів.

Модуль Authorization:

Таблиця 2.6 — Структура файлів та папок модулю Authorization

Layouts	Папка, де зберігається layouts модулю.
SignInPage.vue	Сторінка авторизації
SignUpPage.vue	Сторінка реєстрації

2.3. Серверна частина веб-застосунку

Backend - це частина програмного забезпечення, яка відповідає за обробку даних та логіку бізнес-процесів на сервері, віддаленому від користувача. Бекенд зазвичай включає в себе серверні додатки, бази даних, інтерфейси програмування додатків (API), бібліотеки та інші інструменти для забезпечення взаємодії фронтенду (клієнтської частини) з сервером[6].

Основна функція бекенду - це обробка даних та відповідей на запити клієнтів. Він відповідає за збереження та обробку даних, роботу з базою даних, взаємодію з іншими службами та додатками, інтеграцію з зовнішніми API, а також за забезпечення безпеки даних.

2.3.1. Мова програмування

Для розробки серверів можна використовувати багато мов програмування, в кожної з яких є свої переваги, недоліки та особливості.

Перелік самих популярних мов програмування для створення серверної частини веб-застосунку:

Node.js: це серверна платформа, яка побудована на JavaScript. Вона дозволяє розробникам використовувати JavaScript як мову програмування для розробки бекенду. Node.js є дуже популярною мовою для бекенду, оскільки вона має велику спільноту розробників, вона легка у використанні та дозволяє розробникам швидко створювати масштабовані додатки.

Python: Python є однією з найпопулярніших мов програмування для бекенду. Вона є дуже простою у використанні та має велику кількість бібліотек, що дозволяє розробникам швидко створювати високоякісні додатки.

Ruby: Ruby є ще однією популярною мовою для бекенду. Вона має велику спільноту розробників, добре підходить для створення веб-додатків та має чудову підтримку фреймворків, таких як Ruby on Rails.

Java: Java є дуже потужною мовою програмування, яка підходить для великих та складних додатків. Вона має багато корисних бібліотек та фреймворків, які дозволяють розробникам швидко створювати бекенд додатків.

PHP: PHP є дуже популярною мовою програмування для веб-розробки та бекенду. Вона має велику кількість корисних бібліотек та фреймворків, що дозволяє розробникам швидко створювати високоякісні додатки.

Для проекту було обрано Node.js, це дозволить зробити розробку швидкою, а асинхронність, яка є однією з головних переваг Node.js, дозволить зробити сервер швидким та ефективним при високому навантаженні системи, а оскільки мова програмування Javascript, до розробки можуть бути задіяні і веб-розробники, оскільки мова програмування та сама.

2.3.2. Фреймворк серверної частини

Як і у клієнтській частині використання фреймворку у бекенді дозволяє розробникам ефективніше та швидше створювати додатки, спрощує процес розробки, забезпечує стандартизацію коду та допомагає підтримувати код в готовому стані. Фреймворки мають готові рішення для типових задач, що дозволяє уникнути рутинної роботи та скоротити час розробки. Крім того, використання фреймворку забезпечує більшу безпеку та стабільність додатку завдяки вбудованим механізмам перевірки та обробки помилок. Також фреймворки зазвичай мають активну спільноту, що дозволяє швидко отримувати відповіді на питання та вирішувати проблеми. Тому, використання фреймворку у бекенді є цілком доцільним для забезпечення швидкого та якісного розвитку додатків.

Express, Koa та Fastify - це три найпопулярніших фреймворки для Node.js, які дозволяють створювати швидкі та ефективні веб-сервери та веб-додатки. Розглянемо кожен з них детальніше:
Express - це дуже популярний та стабільний фреймворк для Node.js, який дозволяє створювати веб-додатки та RESTful API. Він має дуже велику кількість плагінів та модулів, які дозволяють розширювати його функціонал. Один з найбільших плюсів Express - це його простота в використанні та навчанні. Він пропонує мінімальний набір функцій, що дозволяє швидко створювати веб-додатки.

Koa - це новіший фреймворк порівняно з Express, він був створений командою розробників, які працювали над Express. Одна з головних переваг Koa - це те, що він дозволяє використовувати асинхронний JavaScript (async/await) для зручного управління запитами та відповідями на них. Також він має дуже просту структуру та дозволяє використовувати middleware, що дозволяє розширювати функціональність додатка.

Fastify - це досить новий фреймворк, що пропонує високу продуктивність та малу вагу. Він має дуже високу швидкість обробки запитів, оскільки використовує найсучасніші технології та асинхронний код. Fastify має дуже

добре розроблену документацію та ряд готових плагінів, що дозволяють додавати нові функції до додатку.

Для проекту було обрано фреймворк Express - він є найпопулярнішим серед наведених, має велике ком’юніті розробників та має всі необхідні інструменти для розробки. Він має просту структуру та легко розширюється за допомогою плагінів.

2.3.3. Бібліотеки серверної частини

Використання бібліотек у бекенді дозволяє розробникам ефективно виконувати рутинні завдання та оптимізувати процес розробки. Бібліотеки зазвичай містять готовий код, який можна використовувати безпосередньо у проекті. Це зменшує кількість часу, необхідного для написання коду, а також дозволяє уникнути помилок, пов'язаних з ручним написанням складних алгоритмів.

Перелік основних бібліотек у проекті:

Бібліотека Prisma - це сучасна ORM для баз даних, що дозволяє розробникам зручно працювати з базами даних з допомогою коду. Prisma підтримує декілька баз даних, включаючи PostgreSQL, MySQL та SQLite.

Основні переваги Prisma:

- Типізований код: Prisma дозволяє визначати моделі бази даних, поля та типи даних для кожної таблиці відповідно до схеми бази даних. Це допомагає уникнути помилок, пов'язаних з типами даних, та зробити код більш стійким до помилок.
- Простота використання: з Prisma можна працювати як з окремими запитами до бази даних, так і з використанням ORM-функцій, що дозволяє зручно та просто взаємодіяти з базою даних.

- Підтримка багатьох баз даних: Prisma підтримує декілька баз даних, що дозволяє розробникам використовувати її в різних проектах, не змінюючи коду.
- Швидкість: Prisma забезпечує швидку взаємодію з базою даних завдяки автоматичній генерації SQL-запитів та використанню оптимальних стратегій вибору даних.
- Підтримка схеми бази даних: Prisma дозволяє визначати схему бази даних, що дозволяє автоматично перевіряти правильність введення даних та робити міграції баз даних.

Бібліотека cors - це бібліотека, що дозволяє контролювати доступ до ресурсів на інших доменах з боку веб-браузера. Він додає необхідні HTTP заголовки, що дозволяють браузеру безпечно виконувати запити на ресурси на інших доменах.

Існує кілька методів вирішення проблеми CORS, однак cors - це один з найбільш популярних пакетів для роботи з CORS в Node.js. Він дозволяє налаштувати дозвіл для різних типів запитів, таких як GET, POST, DELETE і т.д., а також для різних типів даних, таких як рядки, об'єкти або файли.

Основні переваги використання cors:

- Забезпечення безпеки веб-додатків: cors дозволяє контролювати доступ до ресурсів на інших доменах, що забезпечує безпеку веб-додатків.
- Простота використання: cors є простим інструментом з простою настройкою, яка дозволяє швидко вирішити проблему CORS.

- Гнучкість: cors дозволяє налаштувати дозвіл для різних типів запитів і типів даних.
- Підтримка проксі-серверів: cors підтримує роботу з проксі-серверами, що дозволяє вирішувати проблему CORS в розподілених системах.

Отже, cors - це корисна бібліотека для контролю доступу до ресурсів на інших доменах, що забезпечує безпеку веб-додатків та простоту їх розробки.

Бібліотека jsonwebtoken - це найпопулярніших бібліотек для роботи з JWT (JSON Web Tokens) в середовищі Node.js. JWT є стандартом для створення токенів доступу, що використовуються для аутентифікації та авторизації користувачів.

jsonwebtoken надає зручний інтерфейс для генерації та перевірки JWT. Вона працює з даними у форматі JSON та дозволяє створювати токени з даними про користувача, що додатково можна зашифрувати з використанням ключа. Токен містить заголовок, корисні навантаження та підпис, які використовуються для перевірки та підтвердження ідентифікації користувача.

JWT - це стандарт відкритого формату для передачі безпечних даних між двома сторонами. JWT зазвичай використовують для аутентифікації та авторизації користувачів.

За допомогою jsonwebtoken можна реалізувати авторизацію користувачів на рівні API, що дозволяє захистити ресурси сервера від несанкціонованого доступу. Вона також дозволяє створювати різні рівні доступу до ресурсів, що дозволяє забезпечити безпеку даних та забезпечити захист від атак зламу системи.

2.3.4. Архітектура серверної частини

Архітектура бекенду - це організація компонентів, модулів та патернів проектування, які допомагають забезпечити якість, масштабованість та ефективність серверу.

Для серверу було обрано модульну архітектуру з кількох причин. Перш за все, модульна архітектура дозволяє розбити код на невеликі модулі, що робить його більш читабельним та легко змінюваним. Крім того, модульна архітектура дозволяє розподілити функціональність за функціональними групами, що полегшує розробку та тестування.

Кожний модуль повинен бути незалежним один від одного та складатися з певних компонентів:

1) Маршрути - це шаблони, які використовуються забезпечення спілкування між клієнтом та сервером. Коли клієнт надсилає запит на сервер, він включає шлях, який вказує, який ресурс він запитує. Шлях може містити параметри, такі як id чи назва продукту.

2) Контроллер - це компонент, який відповідає за обробку запитів від клієнтів. Контролер взаємодіє зі службовими компонентами бекенду, такими як база даних, сервіси, та інші, та повертає клієнту результати запиту у форматі відповіді HTTP.

3) Сервіс - це компонент, що відповідає за виконання конкретної функціональності додатку. Сервіс може бути використаний для виконання певної операції над даними, наприклад, обробки платежів, відправки електронної пошти, створення замовлень тощо. Він може також містити бізнес-логіку, яка визначає, як саме ці операції повинні бути виконані. Сервіси можуть бути відділені від контролерів, щоб забезпечити кращу організацію коду. Контролери відповідають за обробку запитів та взаємодію з клієнтом, тоді як сервіси відповідають за виконання дій над даними та інші операції, які повинні бути виконані в контексті запиту.

4) Репозиторій - це компонент, який відповідає за зберігання та доступ до даних. Зазвичай, це об'єктно-реляційна база даних, але може бути будь-який інший механізм зберігання даних, такий як NoSQL база даних, файлова система, тощо.

2.3.5. Реалізація серверної частини

Серверна частина була розроблена за допомогою Node.js та фреймворку Express. Проект має наступну структуру:

Таблиця 2.7 — Структура файлів та папок кореневої папки серверу

Назва папки	Опис
node_modules	Папка в проекті, яка містить усі залежності, встановлені за допомогою пакетного менеджера npm або Yarn.
package.json	JSON файл, що має повну інформацію про проект: назву, відомості, залежності, які потрібні для роботи та залежності для розробки, що необхідні лише під час розробки.
tsconfig.json	це конфігураційний файл для TypeScript-компілятора, який визначає параметри проекту, такі як налаштування компіляції, шляхи до файлів, використання зовнішніх бібліотек та інші параметри.
Dockerfile	це текстовий файл, який містить інструкції для автоматизованого створення контейнера Docker. Він складається з інструкцій та команд, що описують, як система повинна бути налаштована та сконфігурована в контейнері Docker.

Головний код зберігається у папці src:

Таблиця 2.8 — Структура файлів та папок папки src серверу

Назва папки	Опис
core	Папка, де зберігається алгоритми та реалізація головної функціональності системи, наприклад алгоритми для реалізації рекомендацій.
exceptions	Тут зберігаються модифіковані класи помилок для зручності дебаггінгу, наприклад ApiError, що має додаткове поле з кодом помилки.
middlewares	Папка, в якій зберігаються middleware проекту, наприклад для авторизації маршруту.
modules	Папка, в якій зберігаються модулі серверу.
prisma	Налаштування ORM Prisma, в тому числі міграції та схеми.
types	Глобальні типи проекту.
index.ts	файл налаштування серверу.
server.ts	файл для запуску сервера.
variables.ts	файл з глобальними змінними.

Проект має наступні модулі:

Таблиця 2.9 — Модулі серверу

Назва папки	Опис
Authorization	модуль авторизації
Movies	модуль фільмів
User	модуль юзерів
MovieRecommendationSystem	модуль рекомендаційної системи

Основні файли модулю, що використовуються в разі потреби:

1) module.routes.ts - файл всіх маршрутів модулю;
2) module.controller.ts - файл контроллеру модуля;
3) module.service.ts - файл бізнес-логіки модулю;
4) module.repository.ts - файл для роботи з БД модулю.

Також модуль може мати додаткові файли, наприклад для збереження утиліт, змінних, тощо.

2.4. База даних веб-застосунку

База даних - це сукупність даних, які організовані у певну структуру та зберігаються на комп'ютері або сервері для забезпечення їх доступності та зручного управління. База даних може включати в себе різноманітну інформацію, таку як дані про клієнтів, замовлення, статистику відвідувань сайту тощо.

Основна мета баз даних полягає у зберіганні великої кількості даних у структурованому та організованому форматі для ефективного доступу та оптимізації обробки цих даних. База даних дозволяє структурувати дані, забезпечувати їх цілісність та безпеку, робити запити та звіти на основі цих даних, та забезпечувати можливість одночасної роботи з ними багатьох користувачів.

Для більш ефективної роботи з базою даних існують спеціальні програми, які дозволяють здійснювати роботу з даними на більш високому рівні абстракції, що дозволяє розробникам та адміністраторам дістати більше користі з бази даних. Такі програми називаються Системами управління базами даних (СУБД), і найбільш поширеними СУБД є MySQL, PostgreSQL, Microsoft SQL Server та Oracle.

Загалом, база даних є важливим компонентом багатьох інформаційних систем та додатків, які збирають, зберігають та обробляють велику кількість даних. Вона дозволяє ефективно організувати та управляти цими даними, що забезпечує більш продуктивну та надійну роботу програмного забезпечення.

2.4.1. Вибір бази даних веб-застосунку

Існує багато різних типів баз даних, кожен з яких має свої унікальні особливості та підходи до зберігання та обробки даних. Для кожної конкретної ситуації необхідно вибрати оптимальний тип бази даних, залежно від потреб бізнесу та характеру даних, які необхідно зберігати.

Реляційні бази даних - це найбільш поширений тип баз даних, який забезпечує надійність, стійкість до помилок та можливість запитів за допомогою мови SQL. Вони базуються на теорії реляцій та складаються з таблиць, що містять рядки та стовпці з даними. Кожна таблиця має назву та структуру, яка складається з набору полів, які визначають тип даних, які зберігаються в цьому полі.

Для доступу до даних у реляційній базі даних використовується мова запитів SQL (Structured Query Language). SQL є стандартом для взаємодії з реляційними базами даних та надає можливість створювати, модифікувати, видаляти та запитувати дані з бази даних.

Основними перевагами реляційних баз даних є їхній високий рівень структуризації та стандартизації, що дозволяє легко розуміти їх структуру та

зміст. Реляційні бази даних також забезпечують можливість обмежувати доступ до даних та забезпечують високий рівень безпеки.

Проте, реляційні бази даних мають певні недоліки. Наприклад, вони не дуже ефективні для зберігання та обробки великих обсягів даних, що містять багато пов'язаних даних, які потрібно швидко опрацьовувати. Крім того, модифікація складних структур даних може бути досить складною та часоємкою процедурою.

Нереляційні бази даних, або NoSQL бази даних - це тип баз даних, які не використовують традиційну реляційну структуру таблиць та SQL мову запитів. Вони були створені для роботи з великими обсягами невпорядкованих даних, які не легко зберігати та обробляти за допомогою реляційної моделі.

NoSQL бази даних забезпечують гнучкість та можливість зберігати дані у різних форматах, таких як JSON, XML, документи та інші. Це дозволяє зберігати неструктуровані дані, такі як зображення, відео, аудіо та тексти, що важко зберігати в традиційних реляційних базах даних.

NoSQL бази даних розділяються на кілька типів:

- Документ-орієнтовані бази даних (Document-oriented databases) - це бази даних, де дані зберігаються у вигляді документів, що містять ключі та значення. Кожен документ може мати різні поля та структуру, що дозволяє зберігати дані з відмінними структурами та форматами. Прикладами документ-орієнтованих баз даних є MongoDB, Couchbase, CouchDB.
- Ключ-значення бази даних (Key-value databases) - це бази даних, де кожен запис зберігається у вигляді пари ключ-значення. Ці бази даних мають дуже просту структуру та є найшвидшими з усіх NoSQL баз даних. Прикладами ключ-значення баз даних є Redis, Riak, Amazon DynamoDB.

- Колоночні бази даних (Column-family databases) - це бази даних, де дані зберігаються у вигляді колонок, а не рядків, як у реляційних базах даних. Вони забезпечують швидкий доступ до даних та можливість зберігати дані з великою кількістю колонок. Прикладами колоночних баз даних є Apache Cassandra, HBase, Amazon SimpleDB.

- Графові бази даних (Graph databases) - це бази даних, де дані зберігаються у вигляді графів, які складаються з вершин та зв'язків між ними. Вони ідеально підходять для зберігання та обробки даних, що мають складні зв'язки між собою, такі як соціальні мережі, географічні мережі та багато інших. Прикладами графових баз даних є Neo4j, OrientDB, ArangoDB.

NoSQL бази даних дозволяють зберігати та обробляти дані на великих обсягах та високій швидкості, що дозволяє їх використовувати в різних сферах, таких як IoT, мережі соціальних мереж, медіа та інше. Однак, їх використання потребує певного рівня експертизи та знань з NoSQL баз даних, щоб ефективно зберігати та обробляти дані.

Оскільки наш веб-застосунок має 2 головних об’єкта - користувач та фільм, він буде мати структуровану схему. Тому для веб-застосунку буде доцільно використовувати реляційну базу даних.

2.4.2. Вибір СУБД веб-застосунку

СУБД - це програмне забезпечення, яке дозволяє зберігати, організовувати та забезпечувати доступ до даних у базі даних. Існують різні види СУБД, кожен з яких використовує певні методи та алгоритми для зберігання та обробки даних. MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL та SQLite - це п'ять

найпопулярніших Систем Управління Базами Даних (СУБД) для реляційних баз даних. Кожна з них має свої особливості, переваги та недоліки[27].

MySQL - це безкоштовна та відкрита реляційна СУБД, яка часто використовується у веб-розробці. Вона працює на багатьох операційних системах та є однією з найшвидших СУБД. MySQL має простий та зрозумілий синтаксис SQL, підтримує транзакції та має велику спільноту розробників, які постійно покращують її.

Oracle Database - це комерційна СУБД, яка має дуже великий функціонал та використовується для зберігання та обробки великих обсягів даних. Вона працює на багатьох операційних системах та має високу надійність та безпеку. Oracle має дуже потужний оптимізатор запитів, що дозволяє працювати з великими обсягами даних швидко та ефективно.

Microsoft SQL Server - це комерційна СУБД, яка розроблена для операційних систем Windows. Вона має широкий функціонал та може бути використана для зберігання та обробки даних будь-якої складності. Microsoft SQL Server має інтуїтивний інтерфейс та простий синтаксис SQL, що дозволяє легко вивчити її для початківців. Вона також має хорошу інтеграцію з іншими продуктами Microsoft, такими як Excel та SharePoint.

PostgreSQL - це безкоштовна та відкрита СУБД, яка часто використовується для розробки веб-додатків. PostgreSQL має велику кількість функцій, підтримує транзакції, має високу надійність та безпеку. Вона підтримує багато типів даних, включаючи географічні дані, що робить її ідеальною для веб-розробки. PostgreSQL також має добру масштабованість та може обробляти великі обсяги даних.

SQLite - це безкоштовна та відкрита СУБД, яка має дуже маленький розмір та може бути використана як вбудована база даних для мобільних та настільних додатків. SQLite має дуже простий та зрозумілий синтаксис SQL, підтримує транзакції та має добру продуктивність для невеликих баз даних.

Вона також підтримує багато типів даних та має вбудовану підтримку для багатої функціональності, такої як повнотекстовий пошук.

Для проекту з Node.js найбільш поширеним варіантом є використання СУБД PostgreSQL, вона має велику кількість бібліотек для роботи та найбільше ком'юніті розробників, тому для проекту було обрано саме його.

2.4.3. Схема бази даних

База має наступні моделі:

1) Users - модель користувача застосунку.

Таблиця 2.10 — Схема моделі Users

Найменування поля	Тип	Призначення
id	UUID	Унікальний ключ
name	String	ім'я користувача
email	String	пошта
password	Number	пароль
createdAt	Date	дата створення
updatedAt	Date	дата оновлення

2) Movies - модель фільму.

Таблиця 2.11 — Схема моделі Movies

Найменування поля	Тип	Призначення
id	UUID	Унікальний ключ

title	String	Назва фільму
genres	String	Жанри фільму
budget	Number	Бюджет фільму
description	String	Опис фільму

Продовження таблиці 2.11

image	String	Картинка фільму
thumbnail	String	Маленька картинка фільму
director	String	Режисер фільму
rating	Number	Рейтинг фільму на IMDB
trailer	String	посилання на трейлер фільму
writers	String	Сценаристи фільму
year	Number	рік випуску фільму
createdAt	Date	дата створення
updatedAt	Date	дата оновлення

3) Likes - таблиця вподобань юзерів.

Таблиця 2.12 — Схема моделі Likes

Найменування поля	Тип	Призначення

user_id	UUID	зовнішній ключ, що посилається на id користувача

Продовження таблиці 2.12

movie_id	UUID	зовнішній ключ, що посилається на id фільму
createdAt	Date	дата створення
updatedAt	Date	дата оновлення

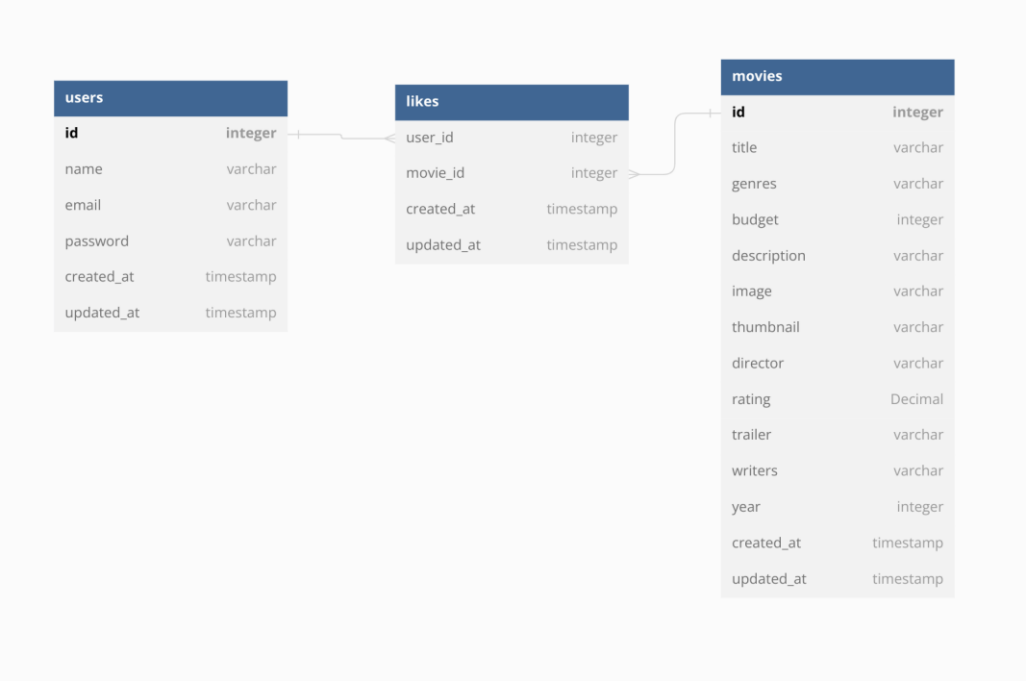


Рисунок 2.1 – Схема зв’язків бази даних

2.5. Реалізація рекомендаційної системи на основі вподобань

Система рекомендацій веб-застосунку для підбору фільмів побудована на алгоритмах колаборативного та контент фільтрах. Огляд колаборативного фільтру.

Реалізація знаходиться у папці core/RecommendationEngine. Папка має 2 файли:

1) algorithms.ts - файл з алгоритмами для реалізації системи рекомендацій, наприклад Коефіцієнт Жаккара;

2) index.ts - головний файл, що має класс RecomendationEngine, який реалізує функціонал колаборативного фільтру.

Реалізація алгоритму наступна:

1) знаходимо найбільш схожих за вподобаннями користувачів за допомогою коефіцієнту Жаккара, його сенс полягає у тому, щоб міру спільної частини поділити на міру об'єднання множин. Тобто, кількість однакових вподобаних фільмів поділити на загальну кількість фільмів двох користувачів. Якщо коефіцієнт дорівнює нулю, тобто немає спільних вподобаних фільмів, ми ігноруємо користувача і не додаємо його до массиву найбільш схожих за вподобаннями користувачів;

2) робимо пустий Мап-об’єкт рейтингу фільму, чим вище рейтинг, тим вище його позиція в списку рекомендацій;

3) в циклі проходимося по кожному користувачу серед найбільш схожих і робимо цикл з кожного вподобаного фільму користувача;

4) в циклі фільмів проходимося по кожному фільму і додаємо до його нинішнього рейтингу(або 0, якщо рейтингу ще не існує) і сумуємо з коефіцієнтом Жаккара користувача, записуємо значення у Мап-об’єкт рейтингу фільм;

5) сортуємо наш Мап-об'єкт від найбільш рекомендованого фільму до найменш.

Контент фільтр реалізується у модулі MovieRecommendationSystem і його алгоритм доволі простий:

1) З об’єкта фільму, для якого хочемо знайти схожі, витягнути необхідні поля для порівняння, в нашій реалізації це поля: genres, director, writers та year.

2) Масив всіх фільмів за допомогою map функції перетворити на масив рейтинг фільму, чим вищий рейтинг, тим вище його позиція в списку рекомендацій. Рейтинг рахуємо наступним чином: витягуємо з поточного фільму необхідні поля та знаходимо коефіцієнт Жаккара між фільмом, для якого хочемо знайти схожі, та поточним фільмом, у нас утворюється масив коефіцієнтів, який ми сумуємо.

3) Фільтруємо фільми, щоб його рейтинг був вище 0.

4) Сортуємо фільми від найбільш схожого фільму до найменш.

Коли користувач натискає кнопку вподобання, фільм потрапляє у список вподобаних фільмів користувача та генерує рекомендації наступним чином: за оновленим списком вподобаних фільмів знаходить рекомендації за допомогою колаборативного фільтру, після чого знаходить рекомендації до вподобаного

фільму за допомогою контент фільтру і об’єднує рекомендації у один масив.

2.6. Контейнеризація та запуск веб-застосунку

Контейнеризація - це технологія, яка дозволяє упаковувати та виконувати додатки з їх залежностями та конфігурацією в ізольованих середовищах, які називають контейнерами. Контейнери забезпечують однакове середовище виконання для додатків незалежно від конфігурації та інфраструктури на яких вони запускаються. Це дозволяє розробникам зберігати і розгортати додатки з мінімальними втратами часу і грошей[30].

Docker - це платформа для контейнеризації додатків. Вона дозволяє розробникам створювати, запускати та управляти контейнерами. Docker використовує технологію контейнеризації для забезпечення ізольованих середовищ виконання додатків, які можуть бути легко перенесені між різними інфраструктурами.

Для контейнеризації додатків у Docker використовуються Docker-образи. Образ - це шаблон, на основі якого створюються контейнери. В образі містяться всі необхідні файли та залежності додатка, а також інструкції для створення контейнера. Образ можна зберегти в репозиторії Docker та легко переносити між різними середовищами.

Одна з основних переваг Docker - це те, що він дозволяє розробникам створювати однакові середовища виконання додатків на різних машинах, що дозволяє легко переносити та розгортати додатки на будь-яких серверах. Крім того, Docker забезпечує ізольоване середовище виконання додатків, що знижує ризики взаємодії з іншими додатками, що працюють на тій самій машині.

Docker також дозволяє розробникам швидко та ефективно розгортати та масштабувати додатки. За допомогою Docker Compose можна легко запускати та керувати групою контейнерів, які працюють разом у складі додатку. Docker також підтримує автоматизовані засоби для побудови та тестування образів

додатків, що дозволяє розробникам зосередитись на розробці додатків, а не на управлінні інфраструктурою.

Крім того, Docker дозволяє розробникам використовувати різні інструменти та технології для розробки додатків, такі як мови програмування, бази даних, веб-сервери та інші. Розробники можуть використовувати свої улюблені інструменти та середовища розробки, що дозволяє їм бути більш продуктивними та ефективними у розробці додатків.

Загалом, Docker - це потужний інструмент для контейнеризації додатків, який дозволяє розробникам швидко та ефективно створювати, розгортати та управляти додатками в ізольованих середовищах. Він забезпечує стандартизацію середовищ виконання додатків, зменшує ризики та забезпечує ефективність у розробці та управлінні додатками.

В нашому проекті ми використовуємо Docker для контейнеризації нашої клієнтської та серверної частини, а також для створення Бази Даних PostgreSQL. Запускаємо проект за допомогою нашого docker-compose.yml файлу і команди docker-compose up.

РОЗДІЛ 3

ОПИС РОБОТИ РЕАЛІЗОВАНОГО ВЕБ-ЗАСТОСУНКУ ДЛЯ ПІДБОРУ ФІЛЬМІВ З УРАХУВАННЯМ ВПОДОБАНЬ

3.1. Опис інтерфейсу користувача

Веб-застосунок для підбору фільмів з урахуванням вподобань складається з наступних сторінок:

1) Головна сторінка
2) Сторінка фільмів
3) Сторінка детального опису фільму
4) Сторінка вподобань
5) Сторінка авторизації
6) Сторінка реєстрації.

3.2. Опис інтерфейсу авторизації користувача

Також застосунок має 2 стани, від якого залежить і його зовнішній вигляд:

1) авторизований;
2) не авторизований.

Розглянемо інтерфейс авторизації та реєстрації

1) Сторінка авторизації - сторінка з формою, що складається з двох полів: пошти та пароля. Обидва поля мають валідацію на незаповнене поле, також поле пошти має валідацію на правильність пошти, а поле пароля на мінімальну довжину з 6 символів. За відсутності зареєстрованого акаунта, користувач має можливість перейти до сторінки реєстрації за допомогою посилання над кнопкою авторизації. При натисканні на кнопку, користувач авторизується та переходить на головну сторінку веб-застосунку.

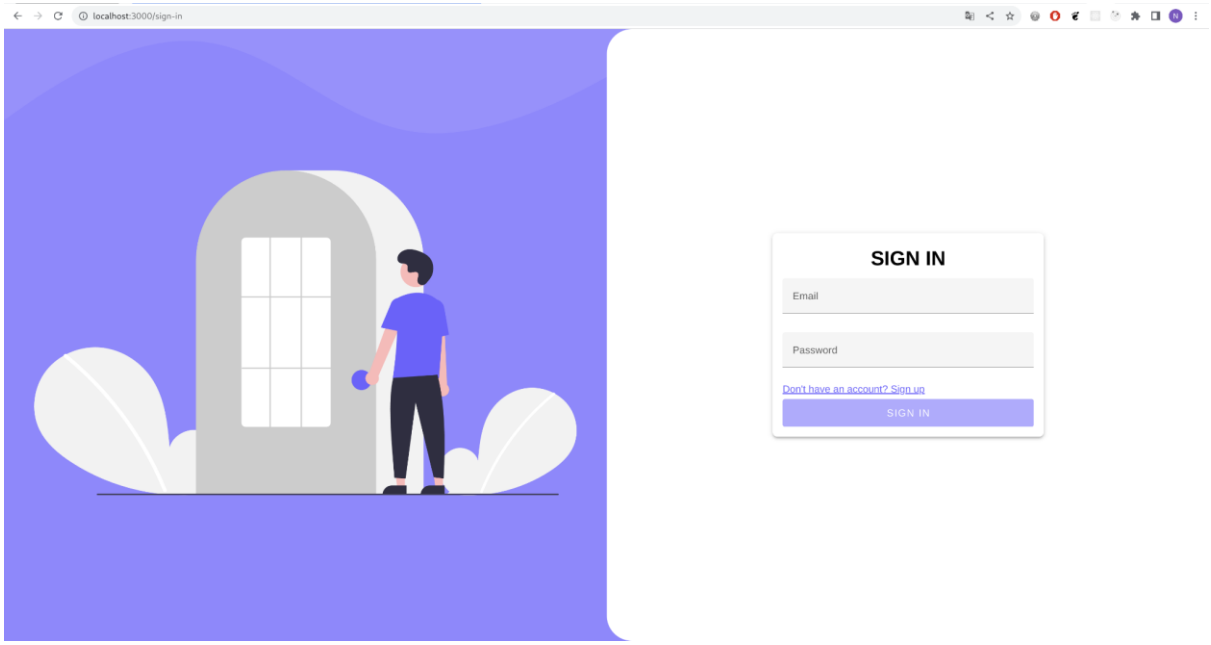


Рисунок 3.1 — Сторінка авторизації користувача

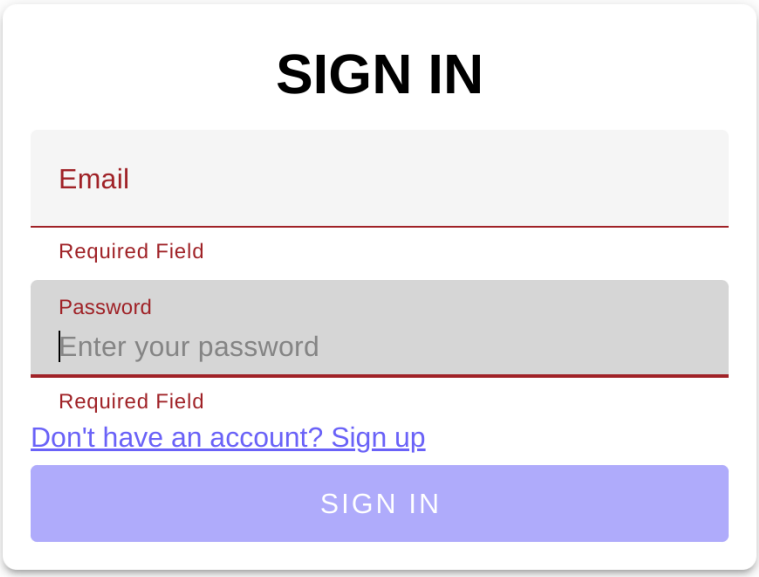


Рисунок 3.2 — Валідація форми авторизації на незаповнене поле

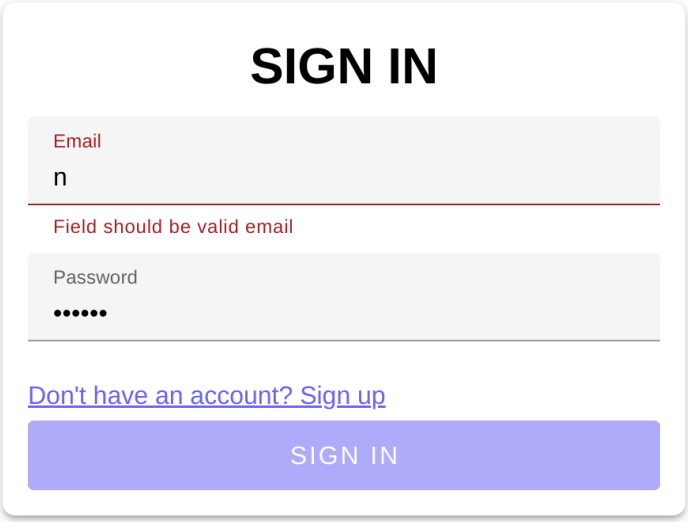


Рисунок 3.3 — Валідація поля пошти на правильність пошти для форми авторизації

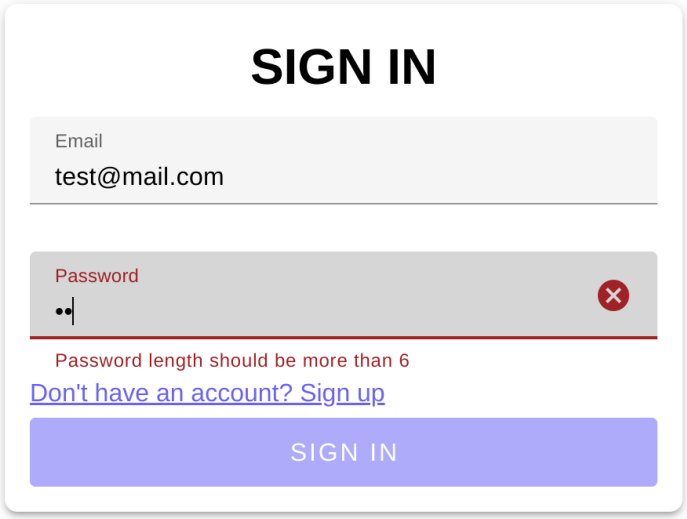


Рисунок 3.4 — Валідація поля пароля на мінімальну довжину пароля для форми авторизації

2) Сторінка реєстрації - сторінка з формою, що складається з трьох полів: ім’я, пошти та пароля. Всі поля мають валідацію на незаповнене поле, також поле пошти має валідацію на правильність пошти, а поле пароля на мінімальну довжину з 6 символів. За наявності зареєстрованого акаунта, користувач має можливість перейти до сторінки авторизації за допомогою посилання над кнопкою реєстрації. При натисканні на кнопку, користувач реєструється та переходить на головну сторінку веб-застосунку.

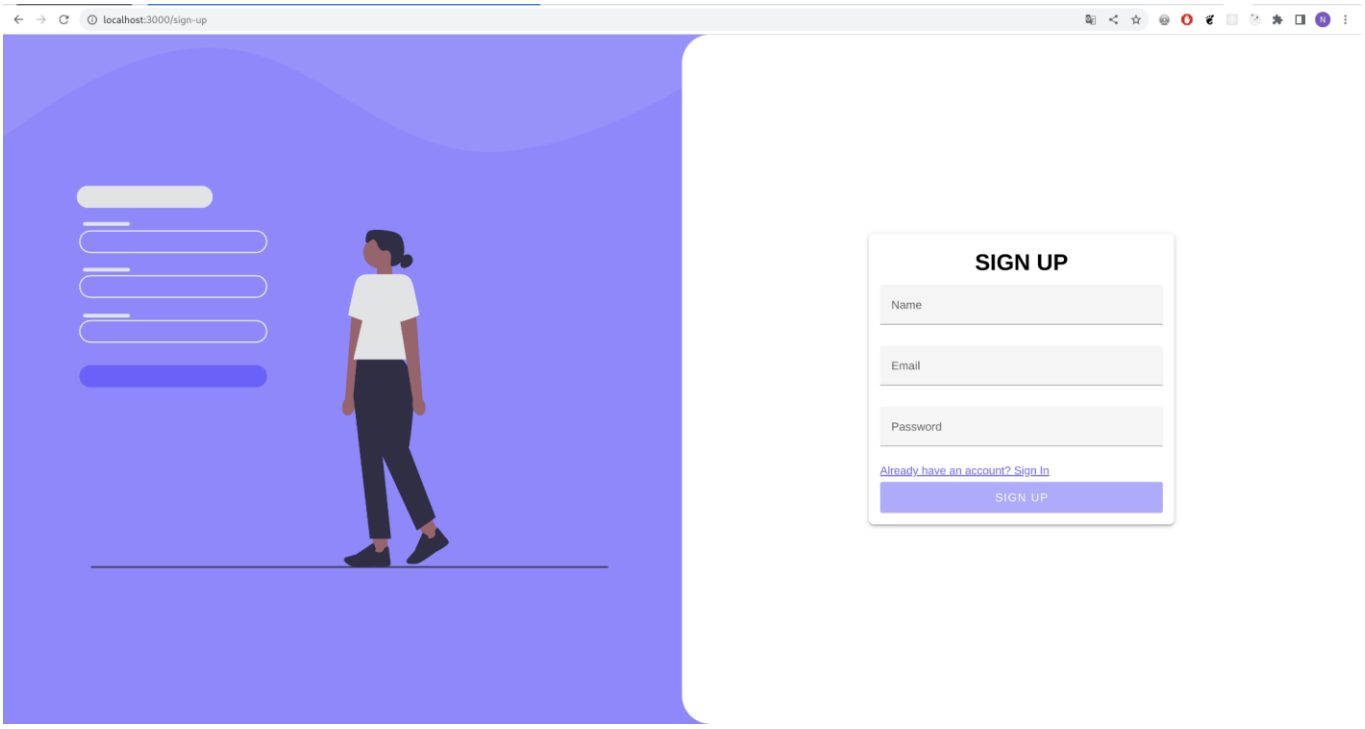


Рисунок 3.5 — Сторінка реєстрації користувача

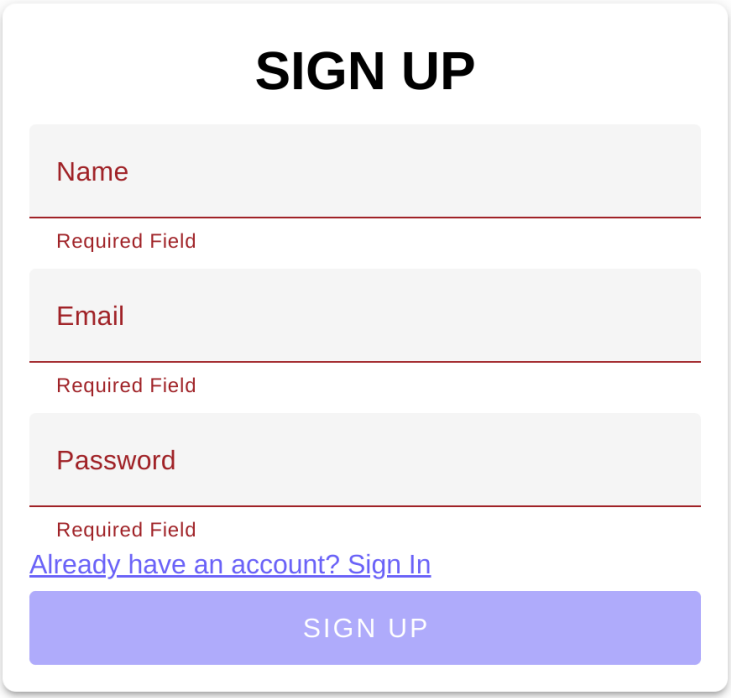


Рисунок 3.6 — Валідація форми реєстрації на незаповнене поле

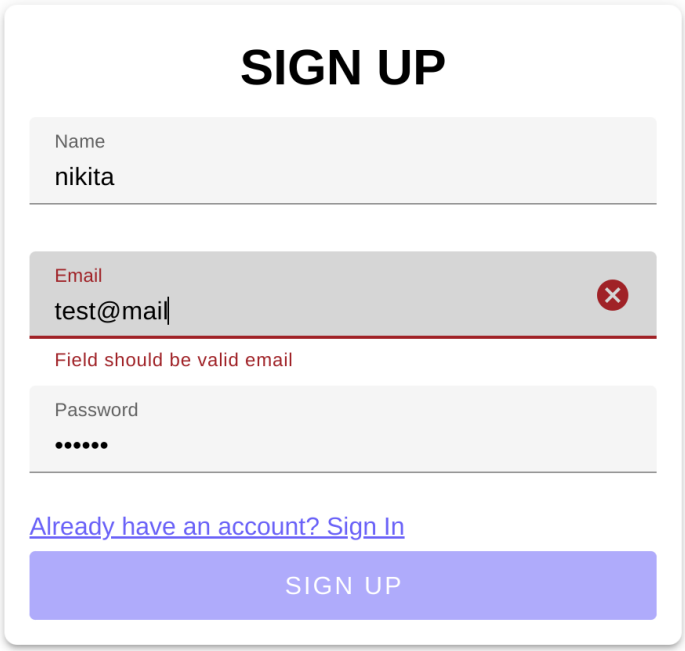


Рисунок 3.7 — Валідація поля пошти на правильність поля для форми реєстрації

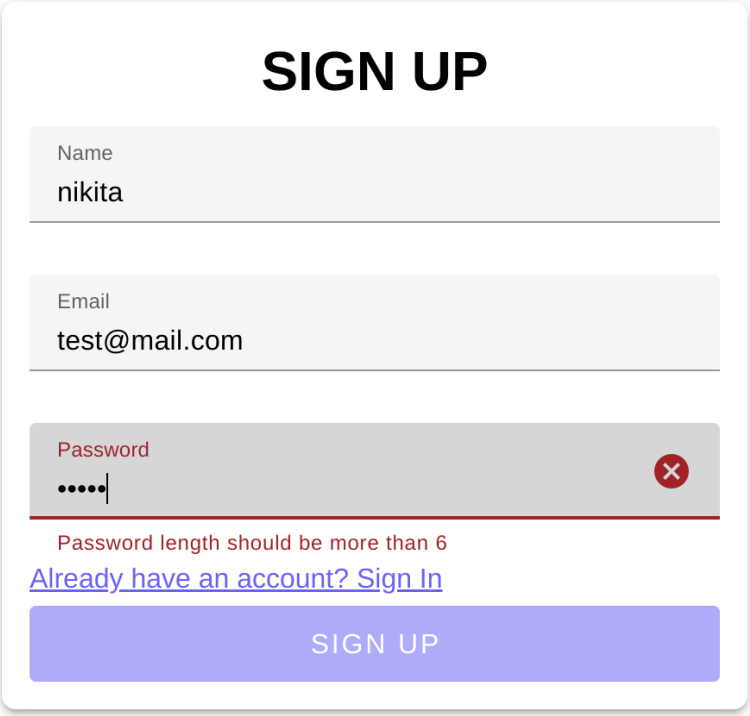


Рисунок 3.8 — Валідація поля пароля на мінімальну довжину пароля для форми реєстрації

3.3. Опис головного інтерфейсу користувача

Головна сторінка - основна сторінка нашого веб-застосунку. Саме на неї ми потрапляємо після успішної авторизації, сторінка також доступна і для неавторизованих користувачів, але буде мати певні відмінності. Основні блоки сторінки:

1) Хедер - компонент, завдяки якому здійснюється навігація. Має посилання на: головну сторінку, сторінку фільмів, сторінку вподобань(за умови авторизації користувача), також містить кнопку для логіну/логауту (в залежності від стану авторизації користувача).

Рисунок 3.9 — Хедер для неавторизованого користувача

Рисунок 3.10 — Хедер для неавторизованого користувача

2) Банер - компонент, яким ми зустрічаємо користувача і заохочуємо до перегляду фільмів, однаковий при будь-якому станові юзера.

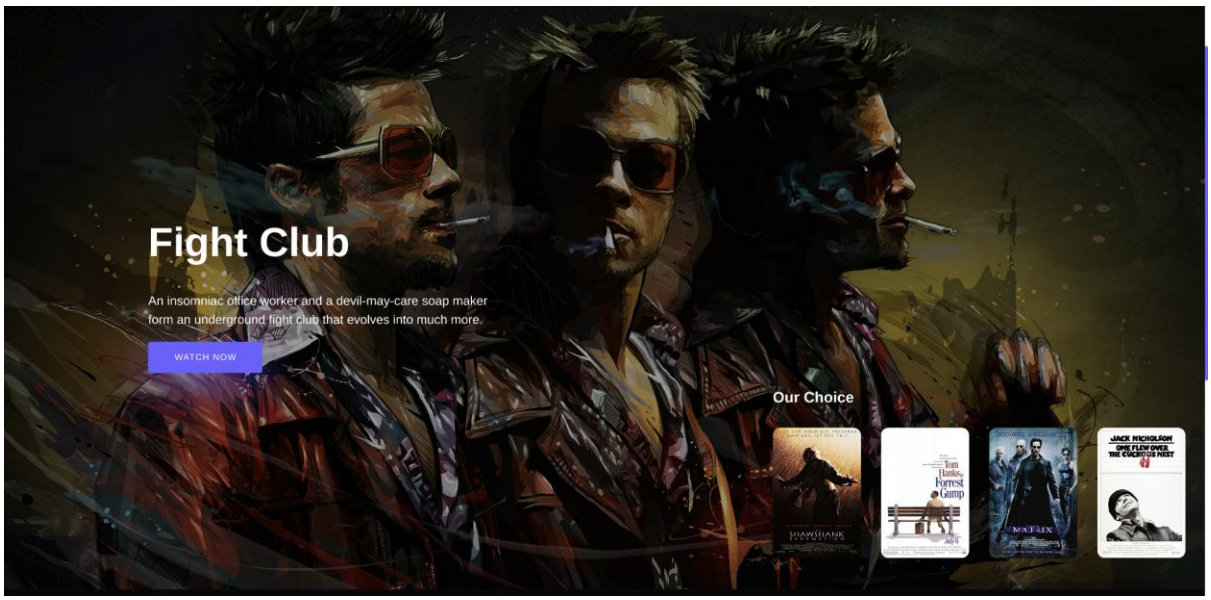


Рисунок 3.11 — Банер головної сторінки

3) Карусель найбільш популярних фільмів, однаковий при будь-якому станові юзера.

Рисунок 3.12 — Карусель найбільш популярних фільмів

4) Masonry блок з фільмами, що можуть подобатися юзеру на основі його вподобань(колаборативний та контент фільтри), є лише у авторизованого користувача.

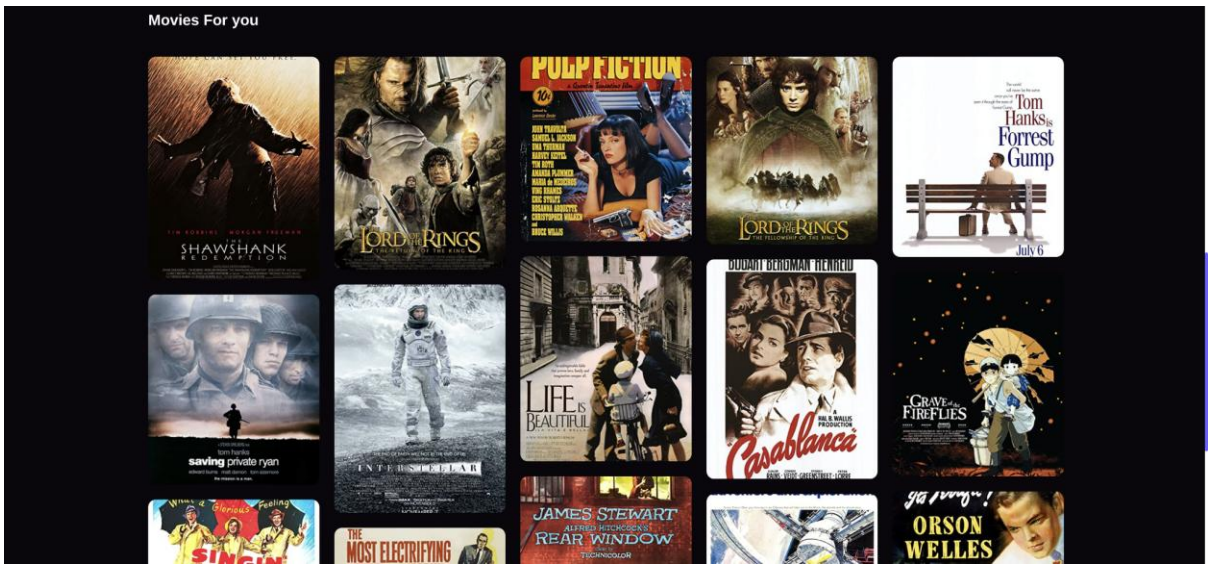


Рисунок 3.13 — Masonry блок з фільмами, що можуть подобатися юзеру на основі його вподобань

5) Карусель найбільш рейтингових фільмів за версією IMDb, однаковий при будь-якому станові юзера.

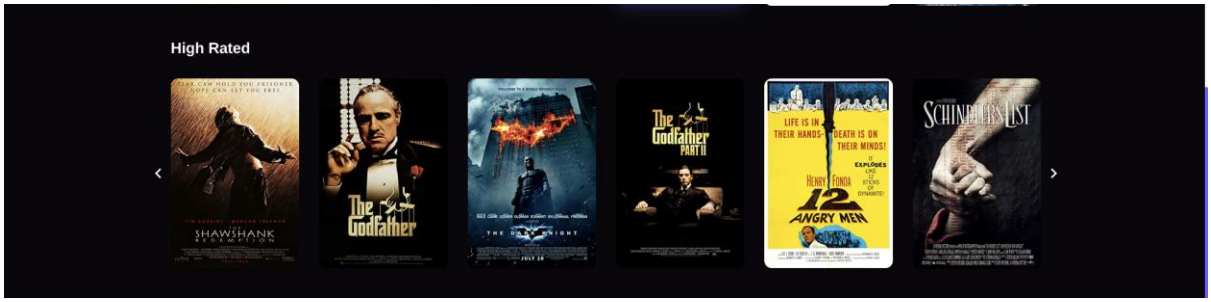


Рисунок 3.14 — Карусель найбільш популярних фільмів

6) Футер з посиланнями на соціальні мережі, однаковий при будь-якому станові юзера.

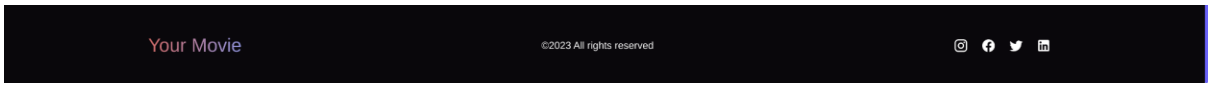


Рисунок 3.15 — Футер веб-застосунку

На рисунках вище можна побачити картки фільмів, при натисканні на них, ми потрапляємо на сторінку детального опису фільму, сторінка має: трейлер фільму, заголовок з назвою фільму, деталі фільми, кнопку вподобання(за умови авторизації користувача) та схожі фільми, що були знайдені за допомогою контент фільтру.

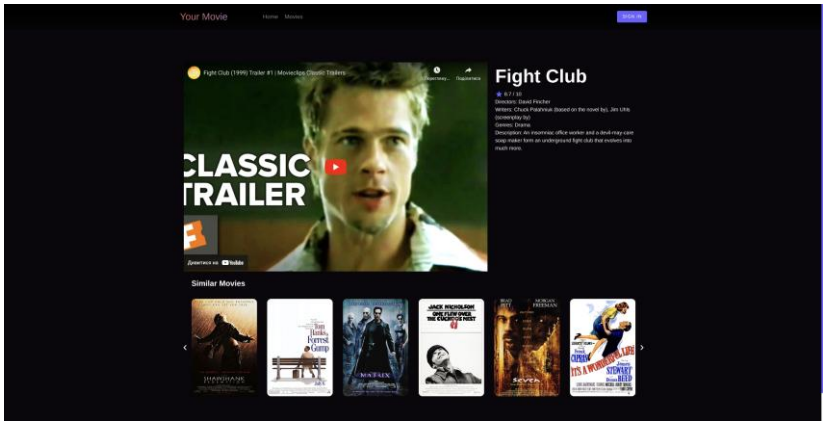


Рисунок 3.16 — Сторінка детального опису фільму для неавторизованого користувача

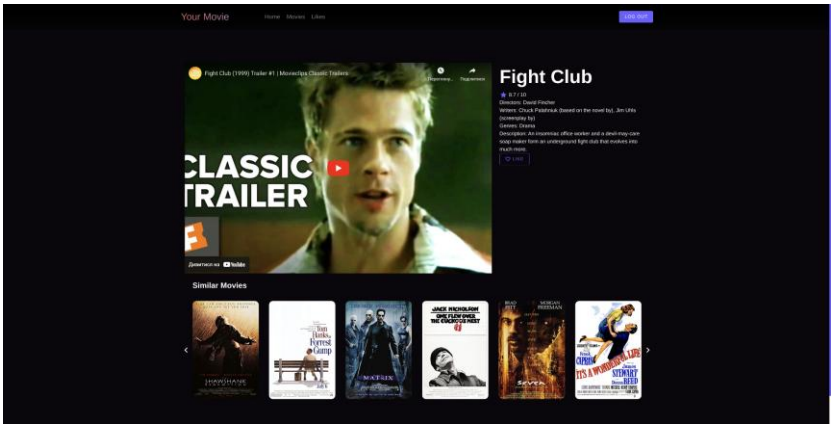


Рисунок 3.17 — Сторінка детального опису фільму для авторизованого користувача(є кнопка для вподобання)

Сторінка фільмів - на цій сторінці користувач може побачити всі доступні фільми, а також відфільтрувати їх за жанром, роком випуску та назвою, також є можливість відсортувати за зростанням/спаданням рейтингу або року випуску. Сторінка однакова при будь-якому станові юзера.

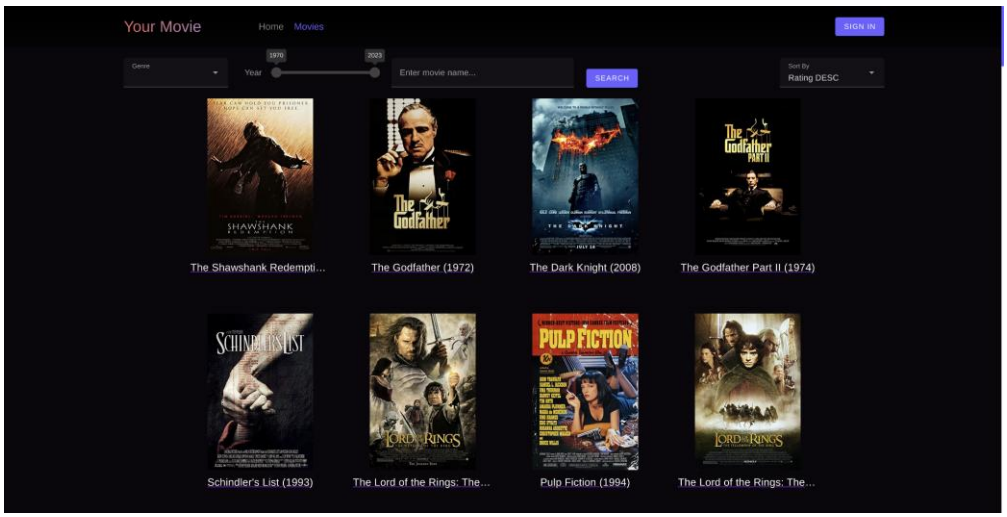


Рисунок 3.18 — Сторінка фільмів

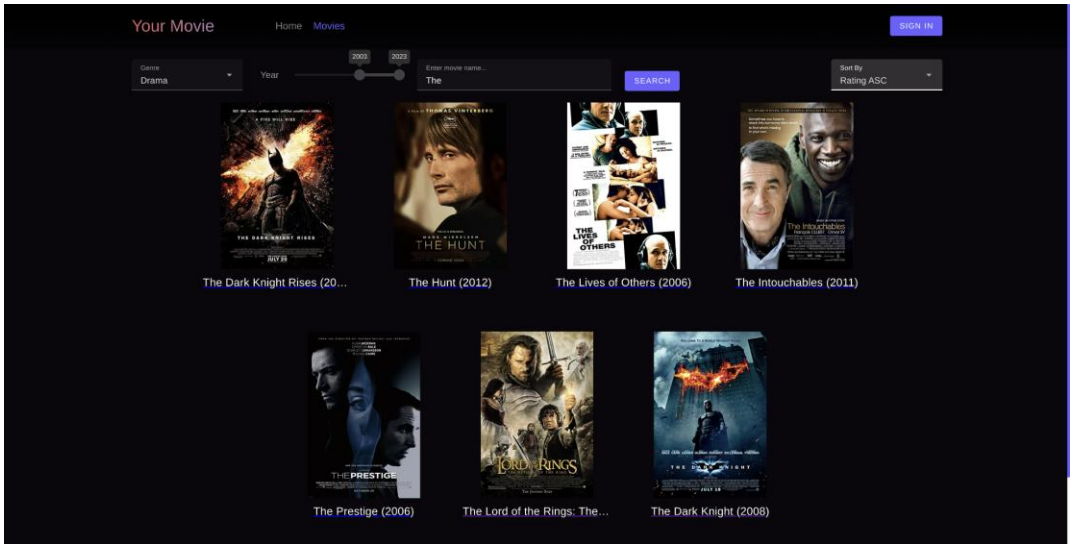


Рисунок 3.19 — Приклад пошуку фільмів та їх сортування на сторінці фільмів

Сторінка вподобань - сторінка, що є лише в авторизованих користувачів, на цій сторінці користувач може переглянути список вподобаних фільмів з їх описами, може перейти на сторінку детального опису фільму при натисканні на картинку фільму, а також прибрати своє вподобання з фільму.

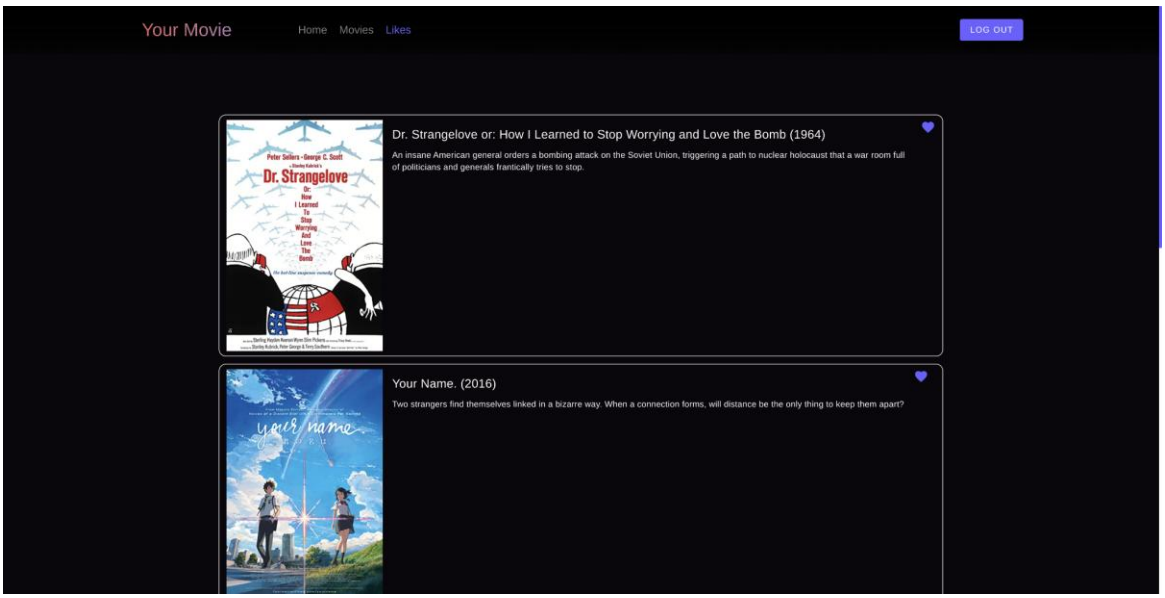


Рисунок 3.20 — Сторінка вподобань

ВИСНОВОК

У результаті виконання кваліфікаційної роботи бакалавра розроблено веб-застосунок для підбору фільмів з урахуванням вподобань, зокрема:

- досліджено загальнотеоретичні засади побудови веб-застосунків для підбору фільмів з урахуванням вподобань;
- знайдено програмно-технологічні рішення для реалізації веб-застосунку;
- реалізовано на практиці веб-застосунок для підбору фільмів з урахуванням вподобань згідно вимог.

Були досліджені особливості, стан та тренди розвитку індустрії, принципи роботи подібних веб-застосунків, проведено аналіз існуючих програм підбору та виведення загальних засад, необхідних для створення веб-застосунку для підбору фільмів з урахуванням вподобань

Також було проведено роботу з пошуку програмно-технологічних рішень, які дозволили ефективно та надійно побудувати модулі системи, такі як: клієнт, сервер, базу даних та алгоритми.

Базуючись на отриманих результатах досліджень, було створено прототип веб-застосунку для підбору фільмів з урахуванням вподобань на основі всіх виведених раніше рішень з застосуванням мови програмування JavaScript, фреймворком Vue для UI частини, платформою Node.js та фреймворком Express для серверу, базою даних PostgreSQL та контейнеризацією за допомогою Docker.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Current and Future State of Recommender Systems. *Medium*. URL: https://medium.com/compassred-data-blog/current-and-future-state-of-recommender-systems-ea3c4669c6ba (дата звернення 03.02.2023)
2. The Future of Video Streaming Apps. *Cyfuture*. URL: https://cyfuture.com/blog/future-of-video-streaming-apps-and-websites/ (дата звернення 03.02.2023)
3. Introduction to recommender systems. *Towardsdatascience*. URL: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada (дата звернення 05.02.2023)
4. What are Recommendation Systems? *Medium.* URL: https://medium.com/@khang.pham.exxact/what-are-recommendation-systems-6bb5036042db (дата звернення 05.02.2023)
5. An In-Depth Guide to How Recommender Systems Work. *Builtin*. URL: https://builtin.com/data-science/recommender-systems (дата звернення 07.02.2023)
6. McKendrick R. Mastering Docker: Enhance your containerization and DevOps skills to deliver production-ready applications : навч. посіб., 2020. 546 с.
7. Godbolt M. Frontend Architecture for Design Systems : навч. посіб., 2016. 185 с.
8. Flanagan, D. JavaScript. The Definitive Guide. 6th Edition : навч. посіб., 2011. 1100 с.
9. Duckett, J. HTML and CSS: Design and Build Websites : навч. посіб., 2020. 490 с.
10. Mardan A. Practical Node.js, 2nd edition : навч. посіб., 2018. 529 с.
11. Doglio F. Scaling Your Node.js Apps : навч. посіб., 2018. 162 с.
12. Casciaro M, Mammino L. Node.js Design Patterns : навч. посіб., 2020. 631 с.

13. Frontend vs Backend. *Geeksforgeeks*. URL: https://www.geeksforgeeks.org/frontend-vs-backend/ (дата звернення 08.02.2023)
14. Документація Javascript. *MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 11.02.2023)
15. Документація HTML. *MDN*. URL: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics (дата звернення 11.02.2023)
16. Документація CSS. *MDN*. URL: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics (дата звернення 11.02.2023)
17. Документація SCSS. *Sass-lang*. URL: https://sass-lang.com/documentation/ (дата звернення 11.02.2023)
18. Документація Typescript. *Typescriptlang*. URL: https://www.typescriptlang.org/docs/handbook/2/basic-types.html (дата звернення 13.02.2023)
19. Angular vs React vs Vue: Core Differences. *Browserstack*. URL: https://www.browserstack.com/guide/angular-vs-react-vs-vue (дата звернення 18.02.2023)
20. Документація Vue. *Vuejs*. URL: https://vuejs.org/guide/introduction.html. (дата звернення 23.02.2023)
21. Vue Style Guide. *Vuejs*. URL: https://v2.vuejs.org/v2/style-guide/?redirect=true (дата звернення 23.02.2023)
22. Vue Composition API. *Vuejs*. URL: https://vuejs.org/guide/extras/composition-api-faq.html (дата звернення 23.02.2023)
23. Документація Vite. *Vite*. URL: https://vitejs.dev/guide/ (дата звернення 23.02.2023)

24. Документація Pinia. *Pinia*. URL: https://pinia.vuejs.org/getting-started.html (дата звернення 23.02.2023)
25. A different approach to frontend architecture. *Dev.to*. URL: https://dev.to/itshugo/a-different-approach-to-frontend-architecture-38d4 (дата звернення 02.03.2023)
26. Документація Node.js. *Nodejs*. URL: https://nodejs.org/docs/latest-v18.x/api (дата звернення 05.03.2023)
27. 5 Alternatives to Express.js. *Makeuseof*. URL: https://www.makeuseof.com/expressjs-alternatives/ (дата звернення 08.03.2023)
28. Документація Express. *Expressjs*. URL: https://expressjs.com/en/guide/routing.html (дата звернення 10.03.2023)
29. Clean architecture with Node.js. *Medium*. URL: https://javascript.plainenglish.io/clean-code-with-node-js-994e9b6b7e56 (дата звернення 04.03.2023)
30. PostgreSQL Tutorial. *Postgresqltutorial*. URL: https://www.postgresqltutorial.com (дата звернення 02.03.2023)
31. Difference between SQL and NoSQL. *Geeksforgeeks*. URL: https://www.geeksforgeeks.org/difference-between-sql-and-nosql/ (дата звернення 17.03.2023)
32. SQL vs. NoSQL: The Differences Explained + When to Use Each. *Coursera*. URL: https://www.coursera.org/articles/nosql-vs-sql (дата звернення 21.03.2023)
33. Database Management System. *Techopedia*. URL: https://www.techopedia.com/definition/24361/database-management-systems-dbms (дата звернення 21.03.2023)
34. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others. *Altexsoft*. URL: https://www.altexsoft.com/blog/business/comparing-database-management-

systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others (дата звернення 25.03.2023)

35. Документація Prisma. *Prisma*. URL: https://www.prisma.io/docs/getting-started (дата звернення 07.03.2023)
36. Документація Docker. *Docker*. URL: https://docs.docker.com/get-started (дата звернення 07.03.2023)
37. Containerization using Docker. *Geeksforgeeks.* URL: https://www.geeksforgeeks.org/containerization-using-docker/ (дата звернення 12.03.2023)

ДОДАТКИ

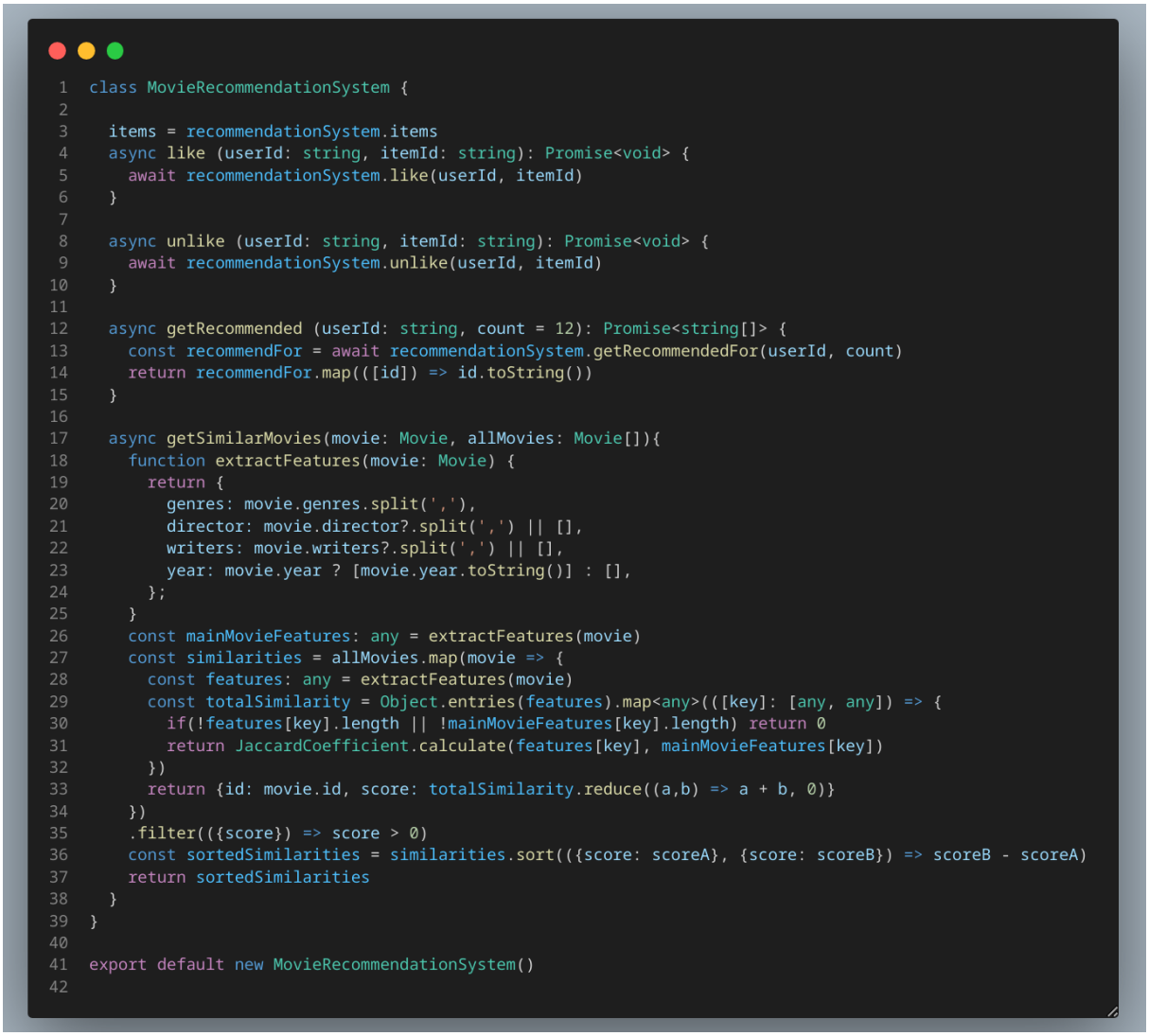
ДОДАТОК А

```ts
class JaccardCoefficient {
  static union<T>(a: T[], b: T[]){
    return new Set([...a, ...b])
  }

  static intersection<T>(a: T[], b: T[]){
    const intersection = a.filter((el) => b.includes(el))
    return new Set(intersection)
  }

  static calculate<T>(a: T[], b: T[]){
    return this.intersection(a, b).size / this.union(a, b).size
  }
}

export default JaccardCoefficient
```

Рисунок А1 – Реалізація алгоритму коефіцієнту Жаккара

```ts
class RecomendationEngine<T, K> implements RecommendationSystemI {
  public items = new Map<T, Set<K>>()

  like = (userId: T, itemId: K) => {
    const likes = this.items.get(userId) || new Set()
    likes.add(itemId)
    this.items.set(userId, likes)
  }

  unlike = (userId: T, itemId: K) => {
    const likes = this.items.get(userId) || new Set()
    likes.delete(itemId)
    this.items.set(userId, likes)
  }

  getSimilarUsers(userId: T){
    const similarities = new Map()
    const userItems = Array.from(this.items.get(userId) || [])
    if(!userItems?.length) return []

    for(const currentUserId of this.items.keys()){
      if(currentUserId === userId) continue
      const currentItems = Array.from(this.items.get(currentUserId) || [])
      const score = +JaccardCoefficient.calculate<K>(userItems, currentItems).toFixed(5)
      if(score > 0) similarities.set(currentUserId, score)
    }
    const mostSimilar = Array.from(similarities).sort(([_, scoreA], [__, scoreB]) => scoreB - scoreA)
    return new Map(mostSimilar)
  }

  getRecommendedFor = (userId: T, count = 10) => {
    const mostSimilarUsers = this.getSimilarUsers(userId)
    const movieRatings = new Map()
    const userItems = Array.from(this.items.get(userId) || [])
    for(const [similarUserId, score] of mostSimilarUsers.entries()){
      const movies = this.items.get(similarUserId)
      if(!movies) continue
      for(const movieId of movies) {
        if(userItems.includes(movieId)) continue
        const currentRating = movieRatings.get(movieId) || 0
        const newRating = currentRating + score
        movieRatings.set(movieId, newRating)
      }
    }
    const mostSimilar = Array.from(movieRatings).sort(([_, scoreA], [__, scoreB]) => scoreB - scoreA)
    return mostSimilar.slice(0, count)
  }
}
```

Рисунок А2 – Реалізація класу RecommendationEngine, що реалізує колаборативний фільтр



```
class MovieRecommendationSystem {

  items = recommendationSystem.items
  async like (userId: string, itemId: string): Promise<void> {
    await recommendationSystem.like(userId, itemId)
  }

  async unlike (userId: string, itemId: string): Promise<void> {
    await recommendationSystem.unlike(userId, itemId)
  }

  async getRecommended (userId: string, count = 12): Promise<string[]> {
    const recommendFor = await recommendationSystem.getRecommendedFor(userId, count)
    return recommendFor.map(([id]) => id.toString())
  }

  async getSimilarMovies(movie: Movie, allMovies: Movie[]){
    function extractFeatures(movie: Movie) {
      return {
        genres: movie.genres.split(','),
        director: movie.director?.split(',') || [],
        writers: movie.writers?.split(',') || [],
        year: movie.year ? [movie.year.toString()] : [],
      };
    }
    const mainMovieFeatures: any = extractFeatures(movie)
    const similarities = allMovies.map(movie => {
      const features: any = extractFeatures(movie)
      const totalSimilarity = Object.entries(features).map<any>(([key]: [any, any]) => {
        if(!features[key].length || !mainMovieFeatures[key].length) return 0
        return JaccardCoefficient.calculate(features[key], mainMovieFeatures[key])
      })
      return {id: movie.id, score: totalSimilarity.reduce((a,b) => a + b, 0)}
    })
    .filter(({score}) => score > 0)
    const sortedSimilarities = similarities.sort(({score: scoreA}, {score: scoreB}) => scoreB - scoreA)
    return sortedSimilarities
  }
}

export default new MovieRecommendationSystem()

```

Рисунок А3 – Реалізація класу MovieRecommendationSystem, що знаходить рекомендовані фільми для користувача та схожі фільми

ДОДАТОК Б

```
generator client {
  provider      = "prisma-client-js"
  binaryTargets = ["native", "debian-openssl-1.1.x", "debian-openssl-3.0.x", "linux-musl", "linux-musl-openssl-3.0.x"]
}

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

model User {
  id        String   @id @default(uuid())
  email     String   @unique
  name      String
  password  String
  createdAt DateTime @default(now())
  updatedAt DateTime @default(now())
  likes     Like[]

  @@map("users")
}

model Movie {
  id          String   @id @default(uuid())
  title       String
  genres      String
  budget      Int?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @default(now())
  description String
  image       String?
  thumbnail   String?
  director    String?
  rating      Decimal?
  trailer     String?
  writers     String?
  year        Int?
  likes       Like[]

  @@map("movies")
}

model Like {
  user_id   String   @map("user_id")
  movie_id  String   @map("movie_id")
  createdAt DateTime @default(now())
  updatedAt DateTime @default(now())
  movie     Movie    @relation(fields: [movie_id], references: [id])
  user      User     @relation(fields: [user_id], references: [id])

  @@id([user_id, movie_id])
  @@map("likes")
}

```

Рисунок Б1 – Налаштування ORM Prisma та створення моделей БД

ДОДАТОК В

```yaml
version: "3"
services:
  server:
    build:
      context: ./server
      dockerfile: Dockerfile
    depends_on:
      - db
      - cache
    env_file:
      - ./server/.env
    ports:
      - "8080:8080"
    volumes:
      - ./server:/app

  client:
    build:
      context: ./client
      dockerfile: Dockerfile
    depends_on:
      - server
    ports:
      - "3000:3000"
    volumes:
      - ./client:/app

  db:
    image: postgres:14
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: movie-recommendation

volumes:
  db-data:
    driver: local
  cache:
    driver: local
```

Рисунок В1 – Налаштування Docker Compose для контейнеризації та запуску веб-застосунку