ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку для відстеження списків перегляду фільмів та серіалів з використанням мови Java та фреймворку Spring»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр ГУЛЕВАТИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Олександр ГУЛЕВАТИЙ

Керівник: Владислав ЯСКЕВИЧ
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гулеватому Олександру Олександровичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку для відстеження списків перегляду фільмів та серіалів з використанням мови Java та фреймворку Spring»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Владислав ЯСКЕВИЧ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література, пов’язана із розробкою веб-застосунків, веб-застосунки для відстеження списків перегляду фільмів та серіалів, мова програмування Java, фреймворк Spring, СУБД MySQL, HTTP клієнт Unirest та шаблонізатор Thymeleaf

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд та аналіз існуючих застосунків для відстеження списків перегляду.
2. Формування вимог до веб-застосунку базуючись на попередньому аналізі аналогів

3. Аналіз та вибір оптимальних існуючих технічних засобів для розробки веб-застосунку для відстеження списків перегляду фільмів та серіалів
4. Проектування застосунку для відстеження списків перегляду фільмів та серіалів.
5. Програмна реалізація та опис функціонування застосунку для відстеження списків перегляду фільмів та серіалів.
6. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання основних функцій.
5. Схема бази даних.
6. Мапа сайту.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих технічних засобів розробки	14.03-18.03.2024	
4	Проектування веб-застосунку для відстеження списків перегляду фільмів та серіалів	19.03-31.03.2024	
5	Розробка веб-застосунку	01.04-18.04.2024	
6	Тестування веб-застосунку	19.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Олександр ГУЛЕВАТИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 47 стор., 4 табл., 26 рис., 24 джерела.

Мета роботи – спрощення відстежування списків перегляду фільмів та серіалів різного формату за рахунок використання веб-застосунку, створеного на мові Java.

Об'єкт дослідження – процес відстеження перегляду фільмів та серіалів.

Предмет дослідження – веб-застосунок для відстеження списків перегляду фільмів та серіалів.

Короткий зміст роботи: В роботі проаналізовано існуючі застосунки для відстеження списків перегляду: IMDb, MyAnimeList, Anime-Planet. Розроблено веб-застосунку та програмно реалізовані ключові функціональні можливості, зокрема: взаємодія зі сторонніми API, керування списками, авторизація, статистика користувача, пошук інформації про фільми та серіали. Проведено мануальне та модульне тестування застосунку. В роботі використано фреймворк Spring та його модулі, СУБД MySQL, HTTP клієнт Unirest, мова програмування Java та Thymeleaf для генерації сторінок.

Сферою використання застосунку є керування списками перегляду фільмів та серіалів.

КЛЮЧОВІ СЛОВА: СПИСКИ ПЕРЕГЛЯДУ, АНІМЕ, SPRING, КОНТЕНТ, ВІДСТЕЖЕННЯ, КОРИСТУВАЧ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Аніме – стиль анімації, що походить з Японії, який характеризується яскравою барвистою графікою, що зображує яскравих персонажів у насичених діями сюжетах, часто з фантастичними або футуристичними темами.[1]

Мейнстрімний - те, що вважається нормальним, прийнятим або використовується більшістю людей.[2]

СУБД – Система управління базами даних.

HTTP – HyperText Transfer Protocol

API – Application Programming Interface

GraphQL – відкритий стандарт запитів та маніпуляцій з даними для API, який дозволяє клієнтам визначати структуру та обсяг необхідних даних, щоб мінімізувати кількість запитів та збільшити ефективність.

REST – Representational State Transfer

HTML – HyperText Markup Language

MVC – Model-View-Controller

MAL – MyAnimeList

IMDb – Internet Movie Database

TMDb – The Movie Database

ВСТУП

Обґрунтування вибору теми та її актуальність: У сучасному світі, кіноіндустрія та інтернет-сервіси активно розвиваються і все більше контенту стає доступним онлайн. Все частіше фільми та серіали виходять ексклюзивно на інтернет-платформах, таких як Netflix, Megogo, Disney+ та інші.[3] Це призводить до того, що все складніше стежити за переглянутим і запланованим контентом до перегляду. Крім того, багато людей дивляться кілька серіалів одночасно, і іноді важко пам'ятати, кількість переглянутих епізодів в кожному з них. [4, 5] Для оптимізації цього процесу, пропонується розробити веб-застосунок для відстеження списків перегляду фільмів та серіалів.

Цей веб-застосунок дозволить користувачам створювати власні списки фільмів та серіалів, які вони планують переглянути, а також відстежувати свій прогрес у перегляді.

Об'єктом дослідження є процес відстеження перегляду фільмів та серіалів.

Предмет дослідження є веб-застосунок для відстеження списків перегляду фільмів та серіалів.

Метою роботи є спрощення відстежування списків перегляду фільмів та серіалів різного формату за рахунок використання веб-застосунку, створеного на мові Java.

Задачі дипломної роботи:

- провести огляд та аналіз схожих веб-застосунків для відстеження списків перегляду
- розробити вимоги до веб-застосунку для відстеження списків перегляду зважаючи на переваги та недоліки проаналізованих аналогів
- проаналізувати технічні засоби та обрати оптимальні для розробки веб-застосунку для відстеження списків перегляду
- спроектувати та розробити веб-застосунок з використанням обраних технічних засобів

- протестувати застосунок використовуючи мануальне та модульне тестування

Практична значущість результатів: Розробка такого веб-застосунку може надати багато переваг користувачам, допомагаючи їм в систематизації інформації про статус перегляду контенту. Застосунок може допомогти користувачам зберігати всю необхідну інформацію про фільми та серіали в одному місці, забезпечуючи легкий доступ до неї в будь-який час.

Крім того, застосунок може допомогти мінімізувати витрати часу на обирання контенту для перегляду. Завдяки фільтру, користувачі зможуть швидко знайти цікаві фільми та серіали, які присутні в їхніх списках, без необхідності проводити години в пошуках чогось нового для перегляду.

Апробація результатів та публікації: Робота пройшла апробацію на науково-технічній конференції: Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» – Київ: ДУІКТ, 2024. За результатами апробації опубліковано тези доповідей [6].

1 АНАЛІЗ АНАЛОГІВ ТА ТЕХНІЧНИХ ЗАСОБІВ

1.1 Аналіз існуючих застосунків для відстеження списків перегляду

1.1.1 IMDb

IMDb (Internet Movie Database) - це сервіс, який містить базу даних про фільми та серіали, а також надає інформацію про акторів, режисерів та інших представників кіноіндустрії.[7] IMDb був заснований у 1990 році як проект фанатів кіно, а у 1998 році був придбаний компанією Amazon.

IMDb надає інформацію про фільми, серіали, акторів, режисерів та інших представників кіноіндустрії, включаючи біографії, фільмографії, оцінки, відгуки, новини, огляди, трейлери та інше. IMDb також має власний рейтинг фільмів та серіалів, який базується на думках мільйонів користувачів з усього світу. Окрім інформації пов’язаної з фільмами та серіалами, IMDb також пропонує можливість створення списків перегляду, що дозволяє користувачам відстежувати фільми та серіали, які вони хочуть подивитися. Користувачі можуть створювати власні списки, редагувати їх та ділитися ними з друзями. Приклад списку перегляду користувача наведено на рисунку 1.2.

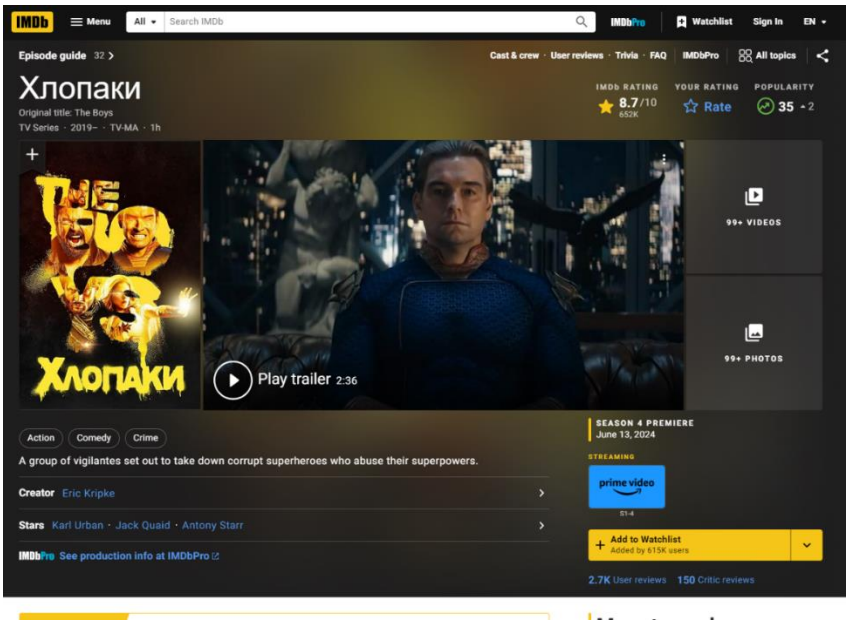


Рис. 1.1 Зовнішній вигляд сторінки IMDb з інформацією про серіал

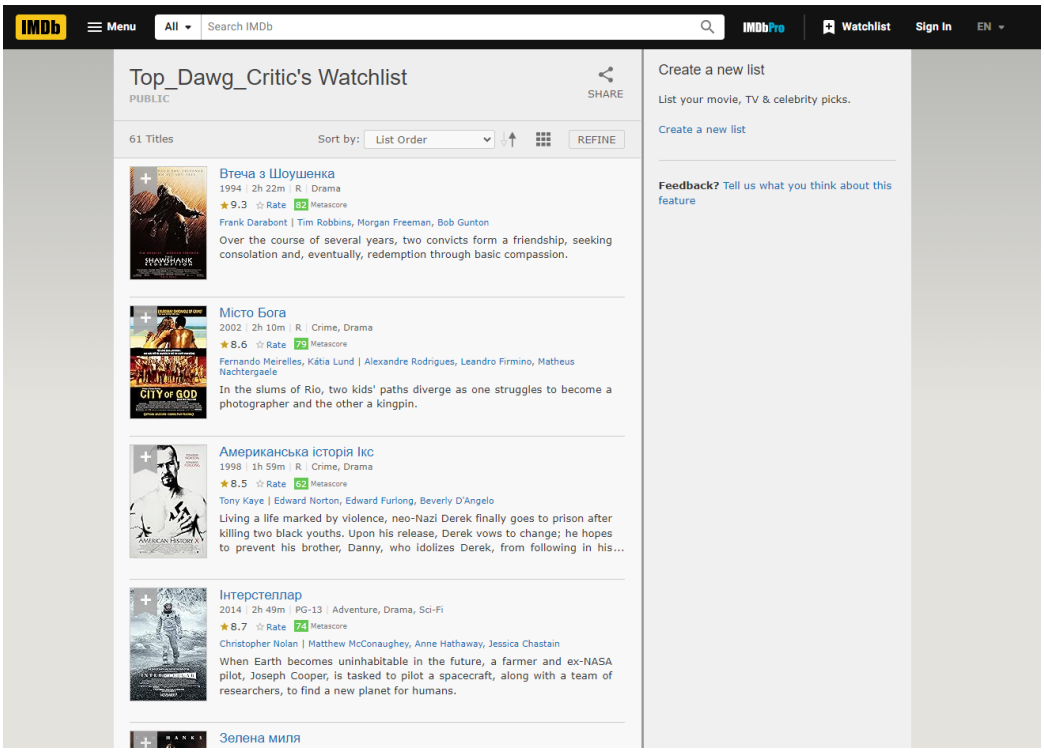


Рис. 1.2 Список перегляду користувача на IMDb

Переваги IMDb:

- багата база даних, яка містить інформацію про більшість фільмів та серіалів;
- можливість створювати власні списки фільмів та серіалів, які ви хочете переглянути, переглядаєте або вже переглянули;
- можливість оцінювати фільми та серіали, а також читати та писати огляди;
- можливість ділитися своїми списками з друзями та іншими користувачами;
- можливість перегляду трейлерів, кліпів та інших відеоматеріалів;
- наявність мобільного застосунку;

Недоліки IMDb:

- можливість керування списків сильно обмежена (присутній лише список запланованого перегляду);
- відсутність можливості відстежувати переглянуті епізоди;

- відсутність користувацької статистики по спискам перегляду;
- відсутність великої кількості аніме контенту;

1.1.2 MyAnimeList

MyAnimeList (MAL) - це веб-сайт та застосунок, який був створений як проект фанатів аніме контенту.[8] Головна задача сервісу полягає у можливості користувачів створювати власні списки аніме, які вони переглядали, а також відстежувати свій прогрес.

MyAnimeList має велику базу даних, яка містить інформацію про тисячі аніме, включаючи описи, жанри, рейтинги, відгуки та інше. Також на MyAnimeList є можливість дізнатися про новинки та майбутні релізи контенту. MyAnimeList має власну систему рейтингу аніме контенту, яка базується на думках мільйонів користувачів з усього світу. Користувачі можуть ставити оцінки та залишати відгуки про переглянуті серіали та прочитану мангу, що допомагає іншим користувачам визначитися з вибором контенту для перегляду.

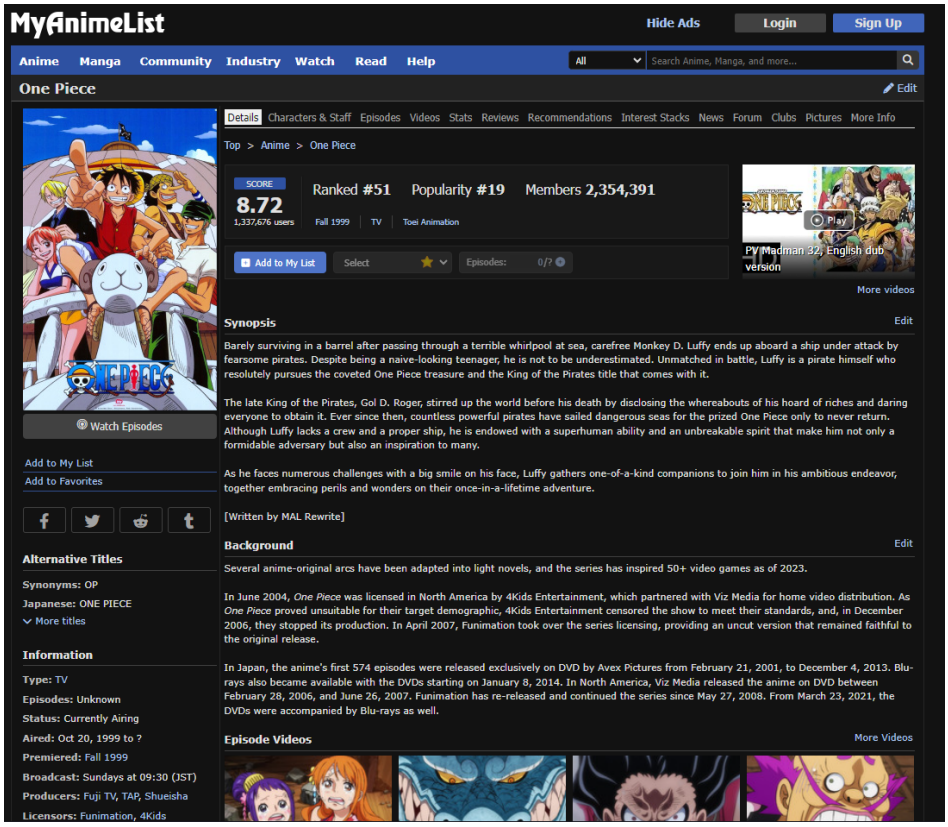


Рис. 1.3 Сторінка з інформацією про контент на MyAnimeList

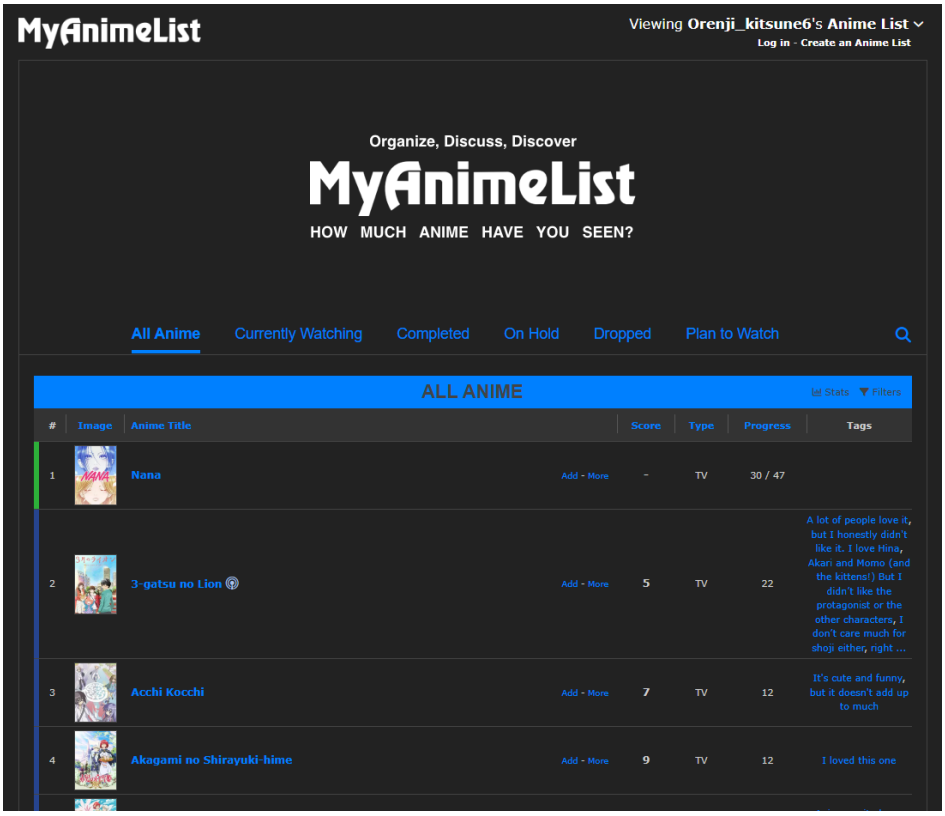


Рис. 1.4 Список перегляду користувача MyAnimeList

Переваги MyAnimeList:

- багата база даних, яка містить інформацію про більшість аніме контенту;
- можливість створювати власні списки перегляду, які ви хочете переглянути, переглядаєте або вже переглянули;
- можливість оцінювати аніме контент, а також читати та писати огляди;
- можливість ділитися своїми списками та оглядами з друзями та іншими користувачами;
- наявність мобільного застосунку;
- можливість стежити за графіком виходу нових серій аніме серіалів;
- можливість перегляду трейлерів, кліпів та інших відеоматеріалів.

Недоліки MyAnimeList:

- myanimelist націлений лише на аніме контент, тому тут повністю відсутні фільми та серіали інших типів;

- хоча мобільний застосунок і присутній, але його можливості обмежені в порівнянні з веб-застосунком;

1.1.3 Anime-Planet

Anime-Planet є популярним онлайн-сервісом для фанатів аніме, який надає широкий спектр функцій для відстеження перегляду аніме.[9] Anime-Planet має велику базу даних, яка містить інформацію про більш ніж 45 тисяч аніме контенту включаючи описи, жанри, рейтинги, відгуки та інше. Користувачі можуть додавати аніме контент до своїх списків перегляду, вказувати свій статус перегляду та ставити оцінки.

Також Anime-Planet має функцію рекомендацій, яка допомагає користувачам знайти нові аніме для перегляду на основі їхніх попередніх переглядів та оцінок. Anime-Planet має можливість додавання користувацьких рекомендацій до аніме, що допомагає іншим користувачам знайти нові цікаві серіали та мангу для перегляду.

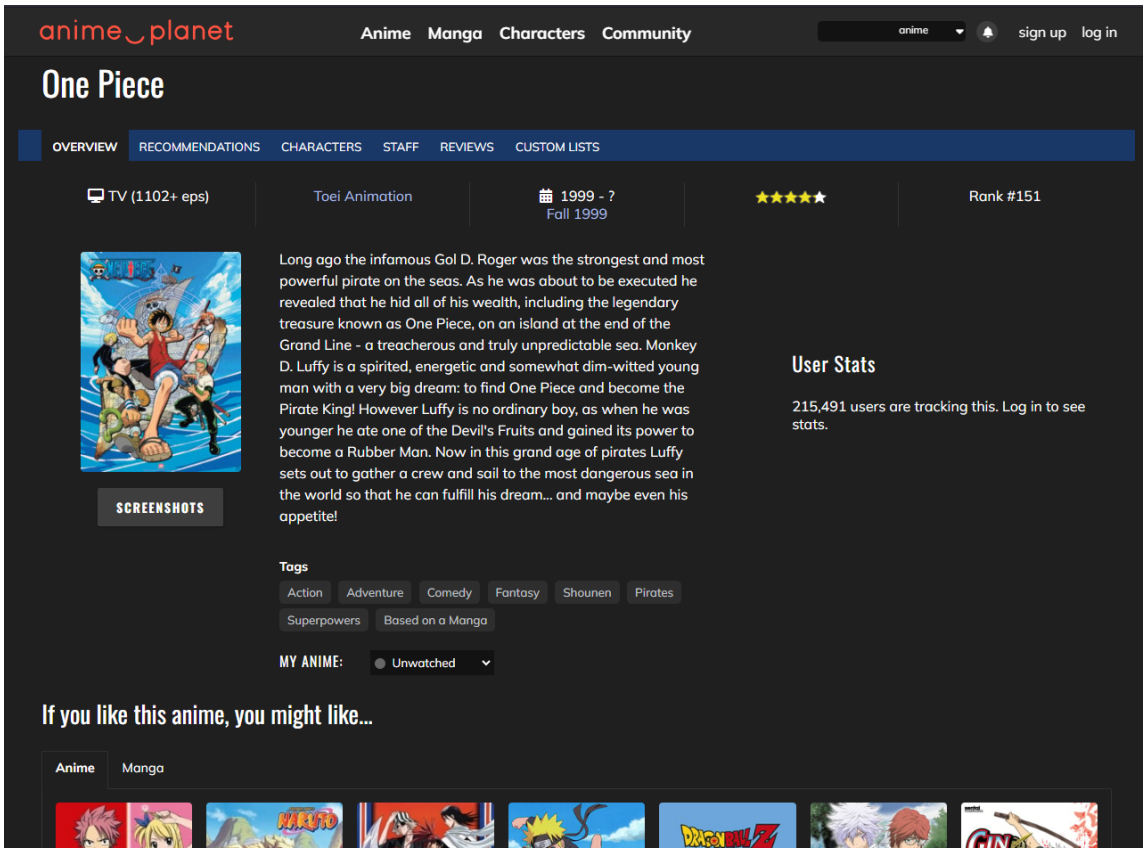


Рис. 3.5 Сторінка з інформацією про серіал на Anime-Planet

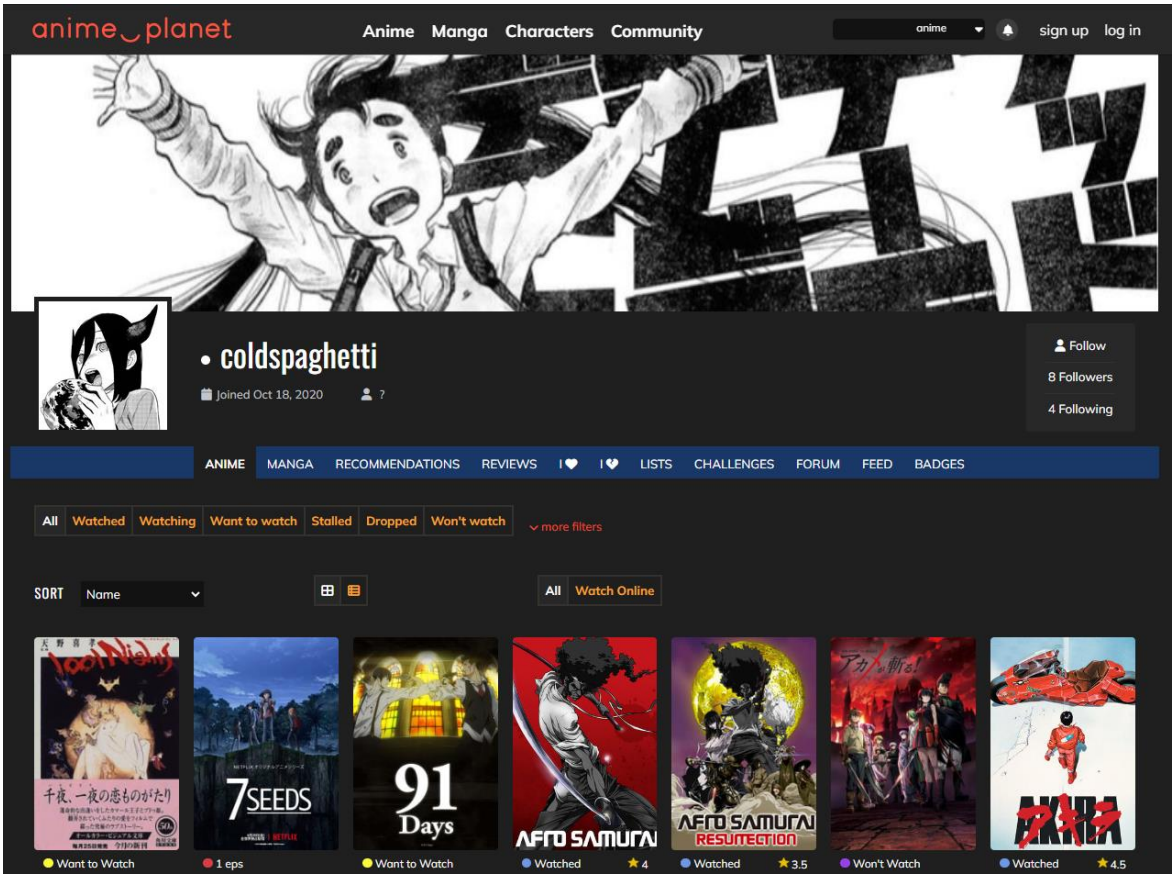


Рис. 1.6 Список перегляду користувача на Anime-Planet

Переваги Anime-Planet:

- багата база даних, яка містить інформацію про аніме контент;
- можливість створювати власні списки перегляду аніме контенту;
- можливість оцінювати аніме контент, а також читати та писати огляди;
- можливість ділитися своїми списками та оглядами з друзями та іншими користувачами;

Недоліки Anime-Planet:

- Anime-Planet націлений лише на аніме, тому тут повністю відсутні фільми та серіали іншого типу;
- менша кількість контенту для відслідковування у порівнянні з MyAnimeList;

1.1.4 Таблиця порівняння аналогів

Таблиця 1.1

Порівняння існуючих схожих застосунків

Характеристика	IMDb	MAL	Anime-Planet	BisonFun
Наявність аніме контенту	Обмежена	+	+	+
Наявність не аніме контенту	+	-	-	+
Керування списками перегляду	примітивне	+	+	+
Відстеження епізодів	-	+	+	+
Платформи	Web, iOS, Android	Web, iOS, Android	Web	Web
Можливість ставити власну оцінку	+	+	+	+
Статистика списків перегляду користувача	-	+	+	+

1.2 Аналіз технічних засобів розробки

1.2.1 Мова програмування Java

Java - це об'єктно-орієнтована мова програмування високого рівня, яка була розроблена компанією Sun Microsystems (тепер належить Oracle Corporation). Мова програмування Java відноситься до C-подібних мов, але вона має декілька важливих відмінностей, які роблять її більш простою та безпечною для використання.

Однією з ключових переваг Java є її дистрибутивність. Код, написаний на Java, може бути виконаний на будь-якому пристрої, який має встановлену віртуальну машину Java (JVM). Це досягається завдяки платформо-незалежному байт-коду, який генерується після компіляції вихідного коду Java. Віртуальна машина Java (JVM) забезпечує середовище для виконання цього байт-коду, незалежно від операційної системи чи апаратного забезпечення.[10, с.21]

Для розробки на Java необхідно Java Development Kit (JDK), який включає в себе Java Runtime Environment (JRE) та інструменти для розробки. JRE забезпечує всі необхідні бібліотеки та JVM для виконання Java-програм, тоді як JDK включає в себе інструменти, необхідні для створення та тестування програм. Коли код Java компілюється, він перетворюється на байт-код, який потім інтерпретується віртуальною машиною Java під час виконання програми.

Java також має автоматичне управління пам'яттю, яке звільняє розробників від необхідності вручну керувати виділенням та звільненням пам'яті. Завдяки цьому зменшується ймовірність виникнення помилок, пов'язаних з пам'яттю, таких як витоки або пошкодження пам'яті.

Іншою важливою особливістю Java є її безпека. Java має вбудовані механізми безпеки, які дозволяють створювати надійні та захищені програми. Наприклад, Java використовує модель безпеки на основі політик, яка контролює доступ до системних ресурсів та захищає від виконання шкідливого коду. Крім того, вбудована система управління виключеннями забезпечує ефективне оброблення помилок, що дозволяє створювати більш стабільні та надійні програми.

Java широко використовується в різних сферах, включаючи веб-розробку, мобільні застосунки, серверні рішення та вбудовані системи. Її популярність обумовлена потужною екосистемою бібліотек і фреймворків, а також активною спільнотою розробників. Завдяки цьому Java залишається однією з найпопулярніших мов програмування у світі, забезпечуючи розробникам ефективні та надійні інструменти для створення сучасних програмних рішень.

1.2.2 Фреймворк Spring

Spring - це популярна платформа для розробки програмного забезпечення на мові програмування Java. Вона включає в себе набір інструментів, бібліотек та фреймворків, які допомагають розробникам швидко та ефективно створювати програми для різних платформ.

Разом з Spring використовувались Spring Boot, Spring MVC, Spring Cache, Spring Data та Spring Security, які є модулями платформи Spring, які допомагають

розробникам швидко та ефективно створювати програми для різних платформ та забезпечувати їх належну роботу та безпеку.

Spring Boot є модулем, який допомагає швидко та легко створювати повністю функціональні веб-застосунки на основі платформи Spring. Він включає в себе набір готових конфігурацій, бібліотек та інструментів, які необхідні для створення веб-застосунків, що дозволяє розробникам зосередитися на бізнес-логіці та функціональності застосунку, а не на конфігурації та налаштуванні інструментів.

Spring MVC є модулем фреймворку Spring для мови програмування Java, який реалізує шаблон MVC. Він включає в себе набір інструментів та бібліотек, які допомагають розробникам реалізувати шаблон MVC у своїх веб-застосунках, та спрощують роботу з HTTP-запитами, обробкою даних, та відображенням інтерфейсу користувача. [11]

Spring MVC також тісно інтегрується з іншими модулями фреймворку Spring, такими як Spring Data та Spring Security, що дозволяє розробникам легко використовувати ці інструменти у своїх веб-застосунках.

Spring Data є модулем, який допомагає розробникам працювати з базами даних та іншими системами зберігання даних. Він включає в себе набір бібліотек та інструментів, які спрощують доступ до даних, маніпуляцію ними та їх зберігання. Spring Data підтримує багато популярних баз даних, включаючи MySQL, PostgreSQL, MongoDB та інші.[12, с.11]

Spring Security є модулем, який допомагає розробникам забезпечувати безпеку та захист веб-застосунків. Він включає в себе набір бібліотек та інструментів, які дозволяють розробникам легко реалізувати такі функції, як автентифікація користувачів, авторизація доступу до ресурсів, шифрування даних та інші функції, необхідні для забезпечення безпеки веб-застосунків.[13]

Spring Cache є модулем, який додає можливості кешування. Кешування дозволяє зберігати часто використовувані дані в пам'яті, що значно підвищує продуктивність застосунку, знижуючи необхідність повторних обчислень або доступу до бази даних.

1.2.3 Система управління базою даних

MySQL - це відкрита система управління базами даних, яка використовується для зберігання, пошуку та обробки даних у багатьох програмних застосунках. Вона є однією з найпопулярніших СУБД через свою простоту, надійність та швидкість роботи. MySQL підтримує стандартний SQL-синтаксис, що дозволяє розробникам легко створювати та обробляти дані в базах даних, забезпечуючи зручність у розробці та підтримці програмних рішень.

Одна з ключових переваг MySQL - її багата функціональність. Вона включає підтримку транзакцій, які дозволяють виконувати групи операцій як єдине ціле, забезпечуючи цілісність даних. Крім того, MySQL підтримує зберігання процедур, що дозволяють виконувати складні бізнес-логіки безпосередньо в базі даних. Тригери забезпечують автоматичне виконання певних дій у відповідь на зміни в даних, що дозволяє реалізувати додаткову логіку контролю та обробки даних. Повнотекстовий пошук надає можливість ефективно здійснювати пошук текстових даних, що є важливим для багатьох застосунків, таких як пошукові системи та електронна комерція.

MySQL може працювати на багатьох платформах, включаючи Windows, Linux, macOS та інші, що робить її універсальною та зручною для використання в різних середовищах. Це забезпечує гнучкість у виборі платформи для розгортання бази даних, що є важливим фактором для багатьох організацій.[14]

MySQL широко використовується у багатьох галузях, включаючи веб-розробку, електронну комерцію, фінансові послуги, телекомунікації та інші. Її популярність пояснюється не тільки технічними характеристиками, але й активною підтримкою спільноти розробників, яка забезпечує розвиток та підтримку численних інструментів і бібліотек для роботи з MySQL. Серед клієнтів MySQL присутні такі відомі компанії, як Sony, Spotify, YouTube, Boeing та Bayer, що підкреслює її надійність та ефективність у великих масштабах.

Завдяки своїй простоті у використанні, високій продуктивності та надійності, MySQL залишається одним з основних виборів для розробників та організацій, що шукають ефективне рішення для управління базами даних. Вона надає всі необхідні

інструменти для створення складних програмних застосунків, забезпечуючи при цьому високу продуктивність та безпеку даних.

1.2.4 Thymeleaf

Thymeleaf - це сучасний шаблонізатор для мови програмування Java, який використовується для створення веб-сторінок та веб-застосунків. Він дозволяє розробникам створювати динамічні веб-сторінки, використовуючи звичайні HTML-файли як шаблони, до яких можна додавати динамічний контент за допомогою спеціальних атрибутів та виразів. Це забезпечує можливість створювати семантично вірні HTML-сторінки, які краще індексуються пошуковими системами і краще сприймаються користувачами.

Thymeleaf забезпечує високу продуктивність та швидку роботу, що дозволяє розробникам створювати ефективні веб-застосунки. Використовуючи простий та зрозумілий синтаксис, розробники можуть легко інтегрувати динамічний контент у HTML-шаблони. Це досягається через використання спеціальних атрибутів, що дозволяють вставляти дані, додавати посилання, умовно відображати елементи та багато іншого.

Thymeleaf відзначається своєю здатністю працювати в режимі "природного шаблону", що означає, що шаблони можуть бути відображені як у браузері, так і на стороні сервера без жодних змін. Це забезпечує чудовий досвід розробки, оскільки дозволяє дизайнерам та розробникам працювати з одними й тими ж файлами.

Інтеграція з Spring Boot та Spring MVC є однією з ключових переваг Thymeleaf. У Spring Boot Thymeleaf використовується як стандартний шаблонізатор, що дозволяє легко налаштовувати та використовувати його у Spring-застосунках. Розробники можуть скористатися всіма можливостями Spring MVC для побудови потужних веб-застосунків з мінімальними зусиллями.

Окрім цього, Thymeleaf має велику та активну спільноту користувачів, що забезпечує наявність численних ресурсів, таких як документація, навчальні матеріали та приклади. Це сприяє швидкому освоєнню та ефективному використанню Thymeleaf у різних проектах.

Загалом, Thymeleaf є потужним інструментом для створення динамічних веб-сторінок у Java-застосунках, забезпечуючи простоту, продуктивність та гнучкість у розробці.

1.2.5 Unirest

Unirest - це бібліотека для мови програмування Java, яка полегшує роботу з HTTP-запитами та відповідями. Unirest дозволяє розробникам швидко створювати та відправляти HTTP-запити, такі як GET, POST, PUT, DELETE та інші, та обробляти отримані відповіді.

Однією з ключових переваг Unirest є її легка установка та конфігурація. Для початку роботи з цією бібліотекою достатньо додати відповідну залежність у проект, після чого можна одразу використовувати її можливості. Unirest забезпечує простий і зрозумілий інтерфейс для роботи з HTTP-запитами, що дозволяє швидко інтегрувати її у будь-який Java-проект.

Unirest підтримує багато форматів даних, включаючи JSON, XML, HTML та інші. Це дозволяє розробникам легко обробляти різні типи відповідей від веб-серверів без необхідності додаткового перетворення даних. Бібліотека автоматично керує заголовками HTTP-запитів, додаючи необхідні заголовки для правильного формування запитів та обробки відповідей.

Unirest також підтримує обробку помилок та виключень, що дозволяє розробникам ефективно обробляти різні сценарії відмов та несправностей при роботі з HTTP-запитами.

Завдяки своїм перевагам, Unirest стала популярною бібліотекою для роботи з HTTP у Java. Вона широко використовується у багатьох галузях та сферах діяльності, включаючи веб-розробку, інтеграцію з зовнішніми сервісами, автоматизацію завдань та багато іншого. Unirest дозволяє розробникам швидко та ефективно взаємодіяти з веб-сервісами, забезпечуючи надійну та зручну роботу з HTTP-запитами у Java-проектах.

2 ВИМОГИ ТА ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

2.1 Визначення вимог до системи

Після проведеного аналізу переваг та недоліків існуючих аналогів веб-застосунків для відстеження списків перегляду фільмів та серіалів було виставлено наступні вимоги до розробки застосунку. Застосунок допоможе користувачам шукати та відстежувати свої улюблені фільми та серіали незалежно від типу цього контенту, а також отримувати інформацію про них. Крім того, застосунок повинен мати статистику користувацьких списків для більшого залучення користувачів.

2.2 Функціональні вимоги

2.2.1 Пошук фільмів та серіалів різного типу

Користувачі повинні мати можливість пошуку фільму або серіалу за назвою та отримувати список результатів, які відповідають їхньому запиту. Результат повинен включати в себе наступну інформацію: назва, рік випуску, постер (при наявності) та посилання на більш детальну інформацію.

2.2.2 Інформація про обраний фільм чи серіал

Повинен бути доступ до інформації про обраний фільм та серіал, включаючи назву, жанри, рік випуску, студії та компанії, що причетні до виробництва, опис та інші дані.

2.2.3 Реєстрація та авторизація користувача

Користувачі повинні мати можливість зареєструватися та увійти в свій особистий профіль. Це дозволяє користувачам зберігати та керувати своїми списками перегляду.

2.2.4 Керування списками перегляду

Користувачі повинні мати можливість створювати, редагувати та видаляти свої списки перегляду фільмів та серіалів. Застосунок повинен дозволяти користувачам додавати фільми та серіали до списків перегляду, змінювати статус перегляду, відстежувати кількість переглянутих епізодів та ставити оцінку. Крім того, повинна бути передбачена можливість ділитися своїми списками перегляду з друзями та іншими користувачами.

2.2.5 Фільтр по назві контенту в списках перегляду

Користувачі можуть мати велику кількість фільмів та серіалів у своєму списку перегляду, тому важливо, щоб вони могли легко знаходити та фільтрувати контент, який їх цікавить. Фільтр по назві контенту дозволяє користувачам вводити назву фільму або серіалу, та отримувати список результатів, які відповідають їхньому запиту.

2.2.6 Користувацька статистика перегляду

Користувачі повинні мати можливість переглядати свою статистику перегляду фільмів та серіалів, включаючи кількість переглянутих фільмів та серіалів, кількість переглянутих епізодів та середню оцінку користувача.

2.3 Нефункціональні вимоги

2.3.1 Модульність застосунку

Веб-застосунок повинен бути розмежований на окремі модулі або компоненти, які виконують певні функції. Модульність дозволяє легко додавати або видаляти компоненти, не впливаючи на інші частини веб-застосунку.

2.3.2 Сумісність застосунку з усіма популярними браузерами

Користувачі можуть використовувати різні браузери, такі як Google Chrome, Mozilla Firefox, Microsoft Edge та інші, тому важливо, щоб веб-застосунок був сумісний з усіма популярними браузерами.

2.4 Архітектура

MVC (Model-View-Controller) - це архітектурний шаблон, який використовується для розробки програмного забезпечення, зокрема веб-застосунків. MVC розділяє програму на три основні компоненти: Model, View та Controller.[15]

Model - це компонент, який відповідає за зберігання та обробку даних у програмі. Model включає в себе бази даних, файли та інші джерела даних. Крім того, Model містить бізнес-логіку, яка обробляє ці дані, виконуючи всі необхідні операції для забезпечення коректної роботи програми.

View - це компонент, який відповідає за відображення даних користувачеві. View може включати в себе HTML-сторінки, шаблони, та інші елементи інтерфейсу користувача, які відображають дані, отримані від Model. Основна функція View - це презентація інформації у зручному для користувача вигляді, забезпечуючи інтуїтивно зрозумілий і привабливий інтерфейс.

Controller - це компонент, який відповідає за обробку запитів користувача та керування роботою програми. Controller включає маршрутизацію запитів, обробку вхідних даних і керування роботою Model та View. Він отримує запити від користувачів, обробляє їх, взаємодіє з Model для отримання або оновлення даних, а потім передає ці дані до View для відображення.

Загалом, використання архітектурного шаблону MVC та модуля Spring MVC дозволяє створювати масштабовані, легко підтримувані та ефективні веб-застосунки, забезпечуючи чітке розділення обов'язків між різними компонентами системи.

При розробці веб-застосунку, для реалізації цього шаблону буде використано модуль Spring MVC, який забезпечує модульність, маршрутизацію запитів, підтримку шаблонів, валідацію та легкість інтеграцій.

2.5 Діаграма використання основних функцій застосунку

На базі вимог було створено діаграму використання ключових функцій застосунку. Вирішено, що не весь функціонал буде доступний лише для авторизованих користувачів. Деякі можливості, такі як пошук і перегляд інформації про фільми та серіали, а також перегляд списків користувачів, будуть доступні для анонімних користувачів. Це забезпечить зручність і доступність базових функцій для всіх відвідувачів сайту, незалежно від того, чи вони зареєстровані.

Такий підхід дозволяє забезпечити баланс між доступністю функціоналу та мотивацією користувачів реєструватися. Анонімні користувачі можуть ознайомитися з можливостями застосунку, що може спонукати їх до створення облікового запису для отримання додаткових можливостей. Авторизовані користувачі, у свою чергу, отримують доступ до повного набору функцій, що підвищує їх залученість та задоволення від використання застосунку.

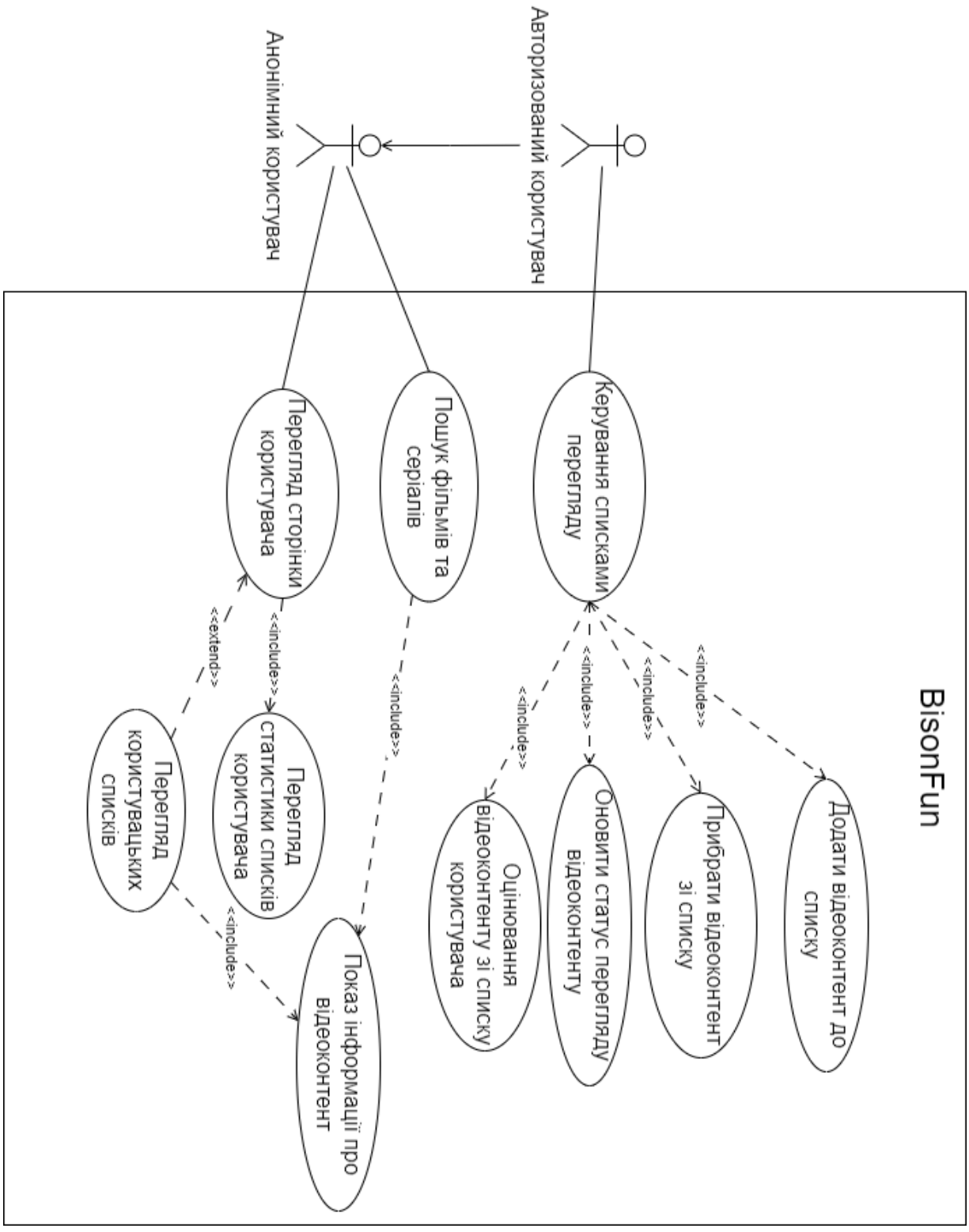


Рис. 2.1 Діаграма використання основних функцій

2.6 Сторонні API

Однією з основних вимог до веб-застосунку є доступ до інформації різного типу. Для цього, веб-застосунок, що розроблюється буде взаємодіяти з сторонніми сервісами які виступають як бази даних з інформацією про фільми та серіали різного типу контенту. Часто вони спеціалізуються лише на конкретному типі контенту, тому використовуватимуться декілька сервісів одразу.

2.6.1 AniList API

AniList - це веб-сайт та онлайн-спільнота для любителів аніме. [16]

AniList надає потужні інструменти для пошуку аніме за різними критеріями, такими як жанр, рік випуску, студія, автор тощо. Система рекомендацій, що базується на вподобаннях користувачів, допомагає знаходити новий контент, що може зацікавити. Користувачі можуть ділитися своїми думками та коментарями щодо аніме контенту, слідкувати за іншими користувачами, створювати групи та брати участь у спільних обговореннях.

AniList API - це інтерфейс програмування застосунків, який дозволяє розробникам отримувати доступ до даних AniList та використовувати їх у своїх програмах. Використання AniList API не вимагає отримання ключа API для доступу до відкритої інформації, що спрощує процес інтеграції.

AniList API використовує GraphQL, що дозволяє отримувати тільки ті дані, які потрібні, за допомогою одного запиту. Це значно зменшує кількість запитів та збільшує швидкість роботи програм. GraphQL надає гнучкість у запитах, дозволяючи специфікувати точні поля даних, які потрібно отримати. За допомогою AniList API розробники можуть отримувати назви, описи, жанри, рейтинги, дати випуску та інші дані про аніме та мангу, а також інформацію про популярний контент. API дозволяє шукати аніме контент за різними параметрами.

Таким чином, AniList API є потужним інструментом для розробників, який дозволяє легко та ефективно отримувати дані про аніме контент, створювати інтерактивні та корисні застосунки.

2.6.2 The Movie Database API

The Movie Database (TMDb) - це онлайн-база даних, яка містить інформацію про фільми, телевізійні шоу та акторів. TMDb дозволяє користувачам шукати фільми та телевізійні шоу, переглядати трейлери, читати рецензії та ділитися своїми думками з іншими користувачами. Користувачі можуть отримувати детальну інформацію про будь-який фільм чи серіал, включаючи акторський склад, режисерів, жанри, рік випуску та рейтинги. TMDb також надає платформу для обговорень, де користувачі можуть обмінюватися своїми враженнями та рекомендаціями.[17]

TMDb має REST API, який дозволяє розробникам отримувати доступ до даних TMDb та використовувати їх у своїх програмах. API TMDb використовує формат JSON для обміну даними, що забезпечує легкість інтеграції та обробки інформації у різних застосунках.

Для використання TMDb API розробникам потрібно отримати ключ API, який надається після реєстрації на платформі. Ключ API забезпечує контрольований доступ до ресурсів TMDb, дозволяючи відстежувати та керувати використанням API. З цим ключем розробники можуть інтегрувати базу даних TMDb у свої застосунки, забезпечуючи користувачам доступ до великої кількості інформації про фільми та телевізійні шоу.

TMDb API надає широкий спектр можливостей для розробників, включаючи доступ до останніх релізів, популярних фільмів та серіалів, трейлерів, зображень, рецензій користувачів та багато іншого. Це робить його важливим інструментом для створення різноманітних застосунків, пов'язаних з кіноіндустрією, від простих каталогів фільмів до складних систем рекомендацій.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

3.1 Реалізація модуля взаємодії з сторонніми API

При подальшій розробці застосунку, який має обробляти велику кількість інформації та даних, розроблений модуль, спрямований на взаємодію зі сторонніми API та передачу необхідної інформації.[18]

Враховуючи специфіку роботи зі сторонніми сервісами, важливо мати на увазі їх можливі технічні проблеми та недоступність на деякий час. Щоб ефективно впоратися з цими сценаріями, в розробленому модулі було включено обробку винятків.

Класи, що взаємодіють зі сторонніми сервісами, можуть повертати виключення ContentNotFoundException та NoAccessException у випадках, коли не вдається отримати необхідну інформацію або відмовляється доступ до ресурсів.

Це стратегічне рішення допоможе забезпечити відповідний контроль за ситуаціями, коли сторонні сервіси не можуть відповісти на запити через технічні або доступові обмеження. Такий підхід дозволить застосунку гнучко реагувати на негативні сценарії та забезпечить користувачів надійністю та безперебійною роботою.

3.1.1 Взаємодія з AniList API

Клас AniListResponse відповідає за запити до AniList API і повертає інформацію у форматі JSON. Оскільки AniList API функціонує на основі протоколу GraphQL, для кожного конкретного запиту створюється схема даних, які зберігаються в окремих файлах у ресурсах застосунку.[19] Це забезпечує зручність у керуванні та зміні запитів, дозволяючи легко модифікувати та розширювати функціональність застосунку. Як приклад, на рисунку 3.1 зображено метод AniListResponse що звертається до API для інформації про

AniList API має обмеження на кількість запитів на проміжок часу (наразі обмеження становить 90 запитів на хвилину)[20]. Це означає, що у випадку перевищення цього ліміту AniListResponse може повертати виключення TooManyAnimeRequestsException (рис. 3.1). Це виключення містить інформацію про час, після якого можна відправляти нові запити, що дозволяє застосунку адекватно реагувати на перевищення ліміту і уникати частих повторних запитів.

```
@ResponseStatus(value = HttpStatus.TOO_MANY_REQUESTS, reason = "Too many requests to Anilist.co")
public class TooManyAnimeRequestsException extends Exception{
    4 usages
    private int seconds;

    Alexander
    public TooManyAnimeRequestsException(String errorMessage, Throwable throwable, int seconds){
        super(errorMessage, throwable);
        this.seconds = seconds;
    }
    Alexander
    public TooManyAnimeRequestsException(String errorMessage, int seconds){
        super(errorMessage);
        this.seconds = seconds;
    }

    Codeium: Refactor | Explain | Docstring | ×
    1 usage   Alexander
    public int getSeconds() { return seconds; }

    Codeium: Refactor | Explain | Docstring | ×
    no usages   Alexander
    public void setSeconds(int seconds) { this.seconds = seconds; }
}
```

Рис. 3.1 Реалізація виключення при перевищенні ліміту запитів

Для оптимізації запитів до сервісу використовується кешування відповідей. Кешування дозволяє зберігати результати запитів та повторно використовувати їх у разі повторного звернення за тією ж інформацією. Це значно зменшує кількість запитів до API, покращуючи продуктивність та ефективність використання ресурсу.

Клас AniListClient використовує AniListResponse для виконання запитів до API. AniListClient відповідає за обробку інформації та можливих виключень, а також повертає повноцінні Java-об’єкти, які можуть бути легко використані в іншій

частині застосунку. Якщо провести аналогію з базами даних, то AniListClient виступає як клас сервісу, що взаємодіє з класом репозиторієм AniListResponse. Це дозволяє розділити відповідальність між різними компонентами системи, що сприяє більш зрозумілій та підтримуваній архітектурі.

3.1.2 The Movie Database

Клас TmdbApiResponse відповідає за запити до The Movie Database API і повертає інформацію у форматі JSON. Для взаємодії з TMDb API потрібно мати ключ, який можна отримати на персональній сторінці користувача у вкладці розробника. Незважаючи на те, що у 2019 році TMDb API скасували обмеження на кількість запитів за певний проміжок часу[21], вони все ще застерігають від надмірного використання сервісу, рекомендуючи обмежити навантаження до приблизно 50 запитів на секунду. Це означає, що при значних навантаженнях можливе введення обмежень.

З метою оптимізації роботи та зменшення навантаження на сервіс, TmdbApiResponse використовує кешування відповідей на запити. Це дозволяє зберігати результати запитів і повторно використовувати їх при наступних зверненнях за тією ж інформацією. Таким чином, кешування допомагає знизити кількість запитів до API, покращуючи продуктивність застосунку та забезпечуючи стабільність роботи.

Клас TmdbClient використовує TmdbApiResponse для виконання запитів до TMDb API, обробки отриманих даних та повернення їх у вигляді Java-об’єктів. TmdbClient виконує функції, аналогічні до AniListClient, забезпечуючи розділення відповідальностей між різними компонентами системи. Це позитивно впливає на підтримування та розширювання архітектури застосунку.

TmdbClient відповідає за взаємодію з TmdbApiResponse, опрацювання інформації та обробку можливих виключень, що можуть виникати під час запитів.

3.2 Проектування бази даних MySQL

Для бази даних було обрано схему з семи таблиць, яка зображена на рисунку 3.2. Вирішено розділити контент на мейнстрімні фільми, мейнстрімні серіали, а також аніме контент. Це рішення було прийнято з кількох причин.

По-перше, розділення контенту на окремі категорії дозволяє легше оновлювати поля інформації для кожного типу контенту без необхідності втручання в інші частини бази даних. Це спрощує процес підтримки та оновлення даних, забезпечуючи гнучкість у керуванні різними типами контенту.

По-друге, така структура сприяє модульності застосунку, що полегшує його подальше розширення. Якщо в майбутньому виникне потреба в додаванні нових типів контенту, це можна буде зробити без значних змін у наявній архітектурі бази даних. Модульний підхід дозволяє розробникам зосередитися на окремих частинах системи, не впливаючи на інші її компоненти.

Таблиці контенту зберігають ключову інформацію, таку як назва, рік релізу, посилання на постер, формат контенту (фільм, серіал), а також ідентифікатори сторонніх сервісів, таких як IMDb або MAL. Це дозволяє зберігати унікальну інформацію для кожного типу контенту, що полегшує доступ і керування даними.

Згідно з таблицями контенту, існує три різні набори списків, що зроблено для зручності користувачів, які можуть використовувати не всі типи списків. Також це забезпечує відмежованість різних типів контенту. Усі списки зберігають інформацію про стан перегляду, кількість переглянутих епізодів, а також користувацьку оцінку цього контенту. Такий підхід дозволяє користувачам легко керувати своїми списками перегляду та отримувати необхідну інформацію про свій прогрес і вподобання.

Загалом, обрана схема бази даних забезпечує гнучкість, модульність та ефективність в управлінні різними типами контенту, що є важливими аспектами для розробки зручного та функціонального застосунку.

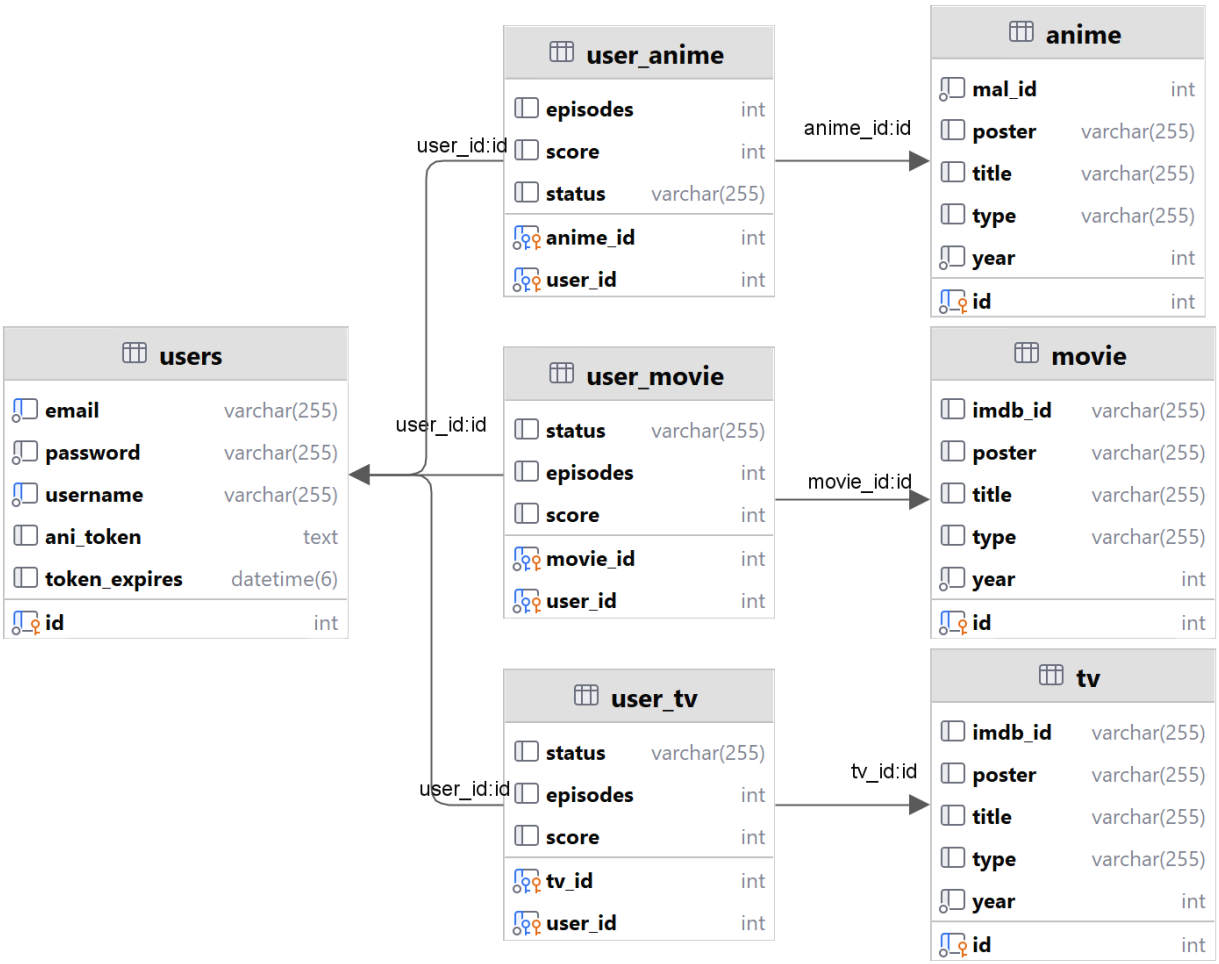


Рис. 3.2 Схема бази даних застосунку

3.3 Розробка користувацького інтерфейсу

Сторінки веб-застосунку розроблені з використанням HTML, CSS та JS. Після готових концептів, використовується Thymeleaf для відображення потрібної наявної інформації.

Розроблені сторінки:

- головна сторінка веб-застосунку;
- профіль користувача;
- списки перегляду користувача;
- інформація про контент;
- результат пошукового запиту;
- авторизація \ реєстрація;

- фільтр списків користувача для вибору для перегляду;

Головна сторінка веб-застосунку існує як початкова точка взаємодії, де користувач може почати шукати інформацію, авторизуватися або переглянути інформацію про популярні фільми та серіали.

Головна сторінка (рисунок 3.3) має наступні елементи:

- поле пошуку з можливістю обрати тип контенту;
- кнопки для реєстрації \ авторизації, якщо користувач не авторизований;
- кнопка для перегляду профіля користувача та виходу з профіля, якщо користувач авторизований;
- перемикач між 3 групами списків;
- горизонтальний список трендових фільмів та серіалів;

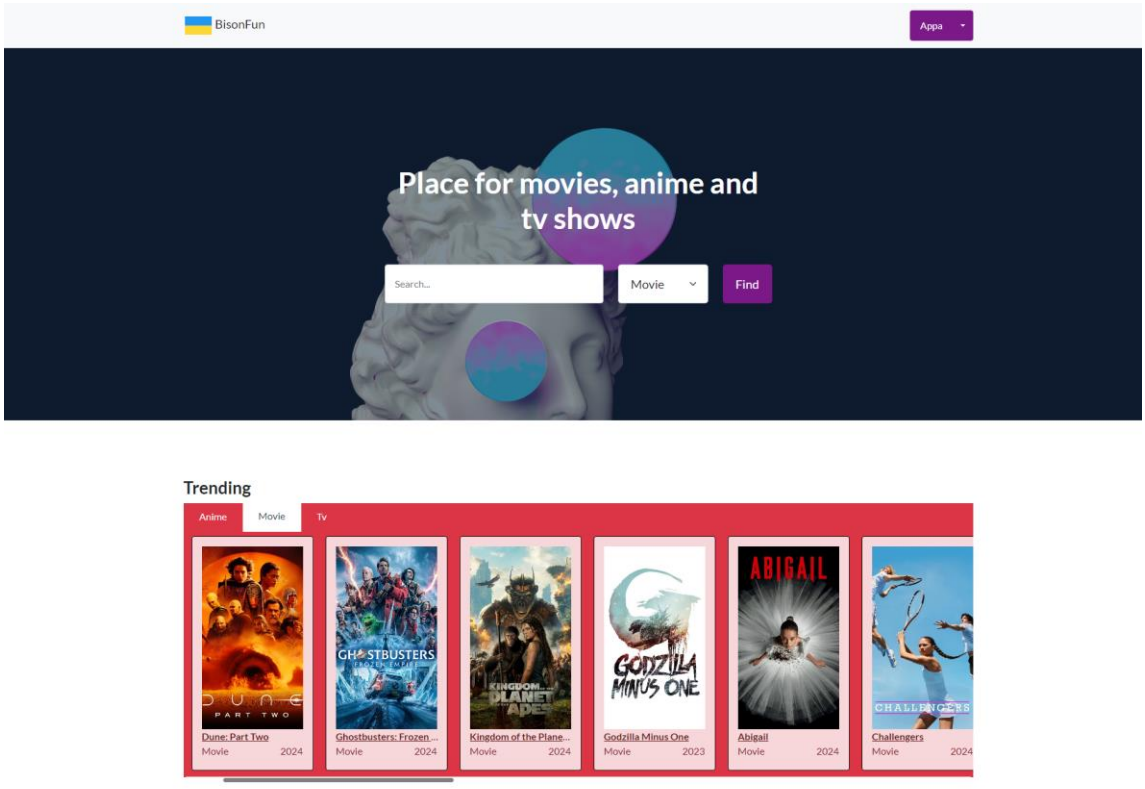


Рис. 3.3 Головна сторінка веб-застосунку

Профіль користувача (рис. 3.4) використовується для доступу до перегляду списків користувача, а також для перегляду користувацької статистики.

Містить в собі наступні елементи:

- поле пошуку з можливістю обрати тип контенту;
- кнопки для реєстрації \ авторизації, якщо користувач не авторизований;
- кнопка для перегляду профіля користувача та виходу з профіля, якщо користувач авторизований;
- псевдонім користувача;
- профільне зображення, що згенероване на базі псевдоніму користувача;
- посилання на набори списків перегляду;
- діаграми зі статистикою списків;
- діаграма зі статистикою переглянутих епізодів;

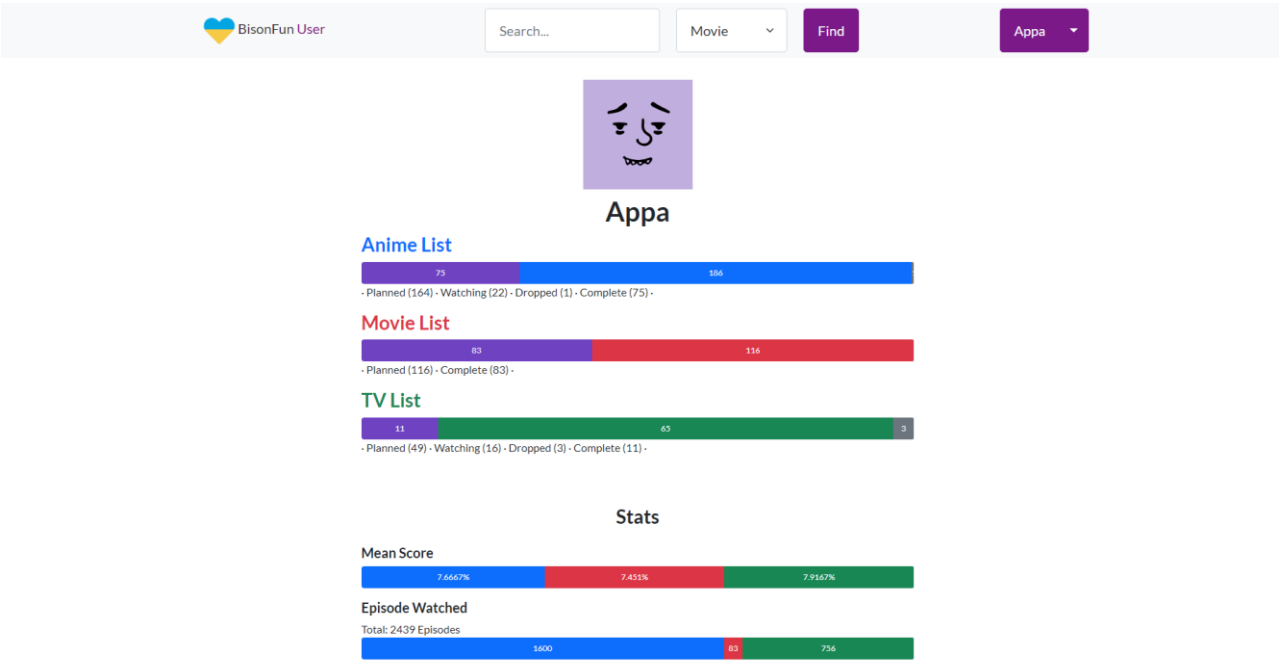


Рис. 3.4 Профіль користувача

Сторінка списків користувача (рис. 3.5) містить в собі усі списки перегляду одного з типів, які можна згорнути для зручності. Для великих списків також існує можливість фільтру контенту по назві.

Сторінка списків користувача містить в собі наступні елементи:

- поле пошуку з можливістю обрати тип контенту;
- кнопки для реєстрації \ авторизації, якщо користувач не авторизований;

- кнопка для перегляду профіля користувача та виходу з профіля, якщо користувач авторизований;
- псевдонім користувача та тип списків;
- списки у вигляді таблиці з наступними полями: ідентифікаційний номер, постер, назва контенту з посиланням на сторінку з інформацією про контент, користувацький рейтинг та кількість переглянутих епізодів;
- поле введення для фільтрації контенту по назві;
- кнопки для згортання списків;

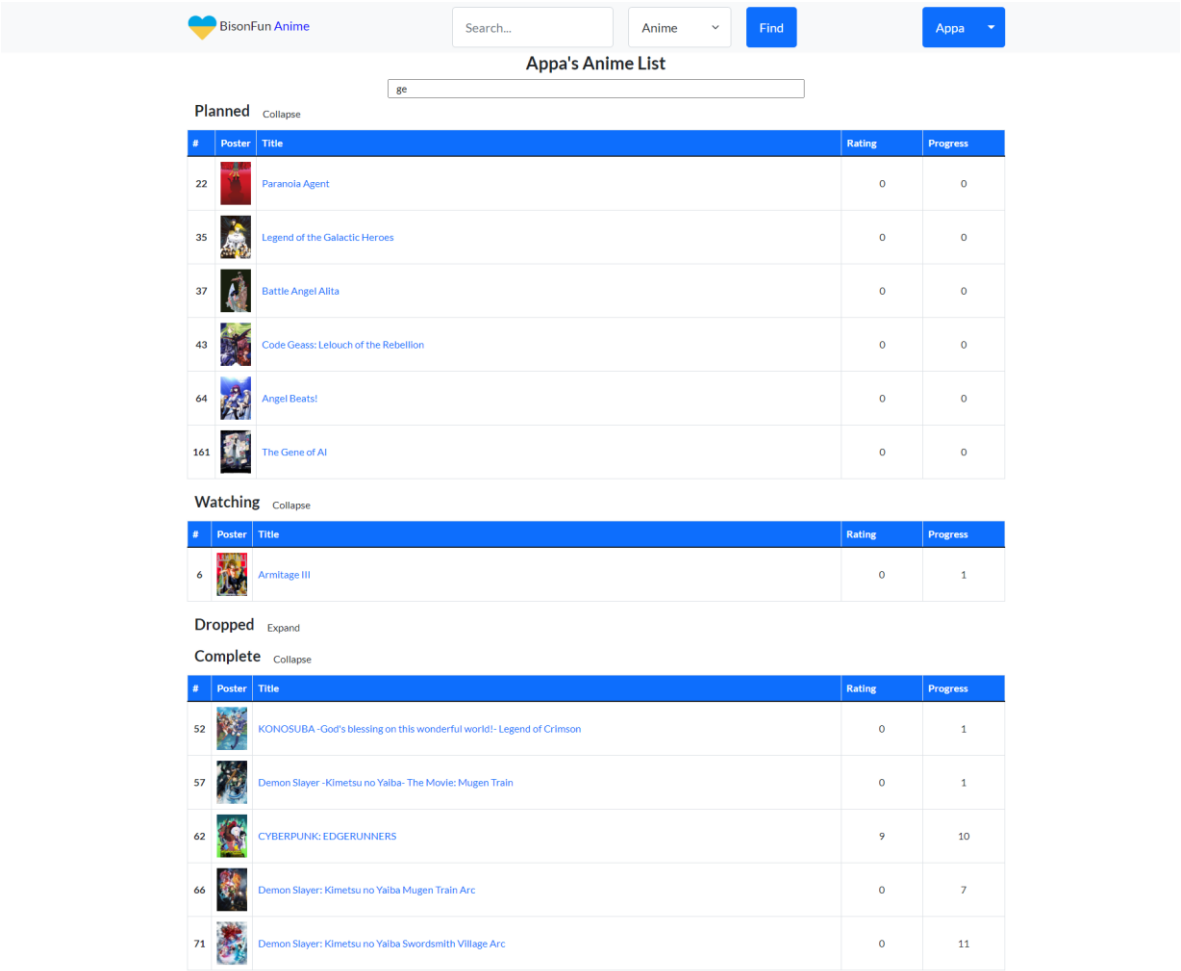


Рис. 3.5 Списки перегляду аніме користувача

Сторінка інформації про контент (рис. 3.6) містить в собі ключову інформацію (назва, дата випуску, кількість епізодів тощо), щодо контенту, а також посилання на сторонні джерела у випадку потреби більшої кількості інформації.

Для авторизованих користувачів є можливість взаємодіяти зі списками користувача (змінити статус перегляду, кількість переглянутих епізодів, оцінку).

Сторінка інформації про контент містить в собі наступні елементи:

- поле пошуку з можливістю обрати тип контенту;
- кнопки для реєстрації \ авторизації, якщо користувач не авторизований;
- кнопка для перегляду профіля користувача та виходу з профіля, якщо користувач авторизований;
- постер контенту;
- назва контенту;
- опис контенту;
- кількість епізодів;
- кількість сезонів;
- тривалість серії (фільму);
- статус виходу;
- дата виходу (якщо серіал, то і дата завершення, якщо серіал закінчився);
- жанри;
- студії які брали участь у створенні контенту;
- при наявних рекомендаціях, горизонтальний список зі схожим контентом;
- посилання на сторонні сервіси;
- кнопка взаємодії контенту зі списком;
- кнопка зміни списку для контенту;

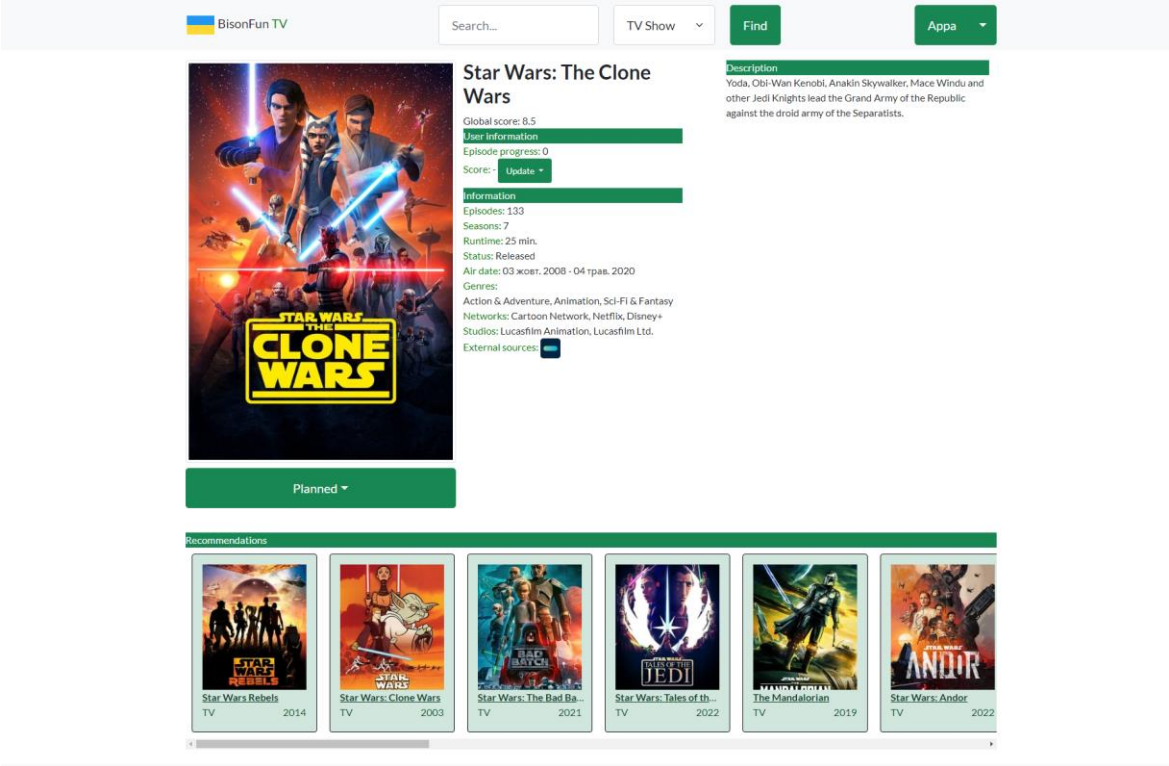


Рис. 3.6 Сторінка інформації про контент

Сторінка результатів пошуку (рис. 3.7) містить в собі посилання на сторінки з інформацією про контент, який підходить під запит. Якщо результатів більше 20, то можна перейти на наступну сторінку результатів.

Сторінка результатів пошуку містить в собі «картки» з інформацією, що в собі мають наступні елементи:

- постер контенту;
- назва разом з посиланням на повноцінну сторінку;
- формат контенту;
- рік випуску;

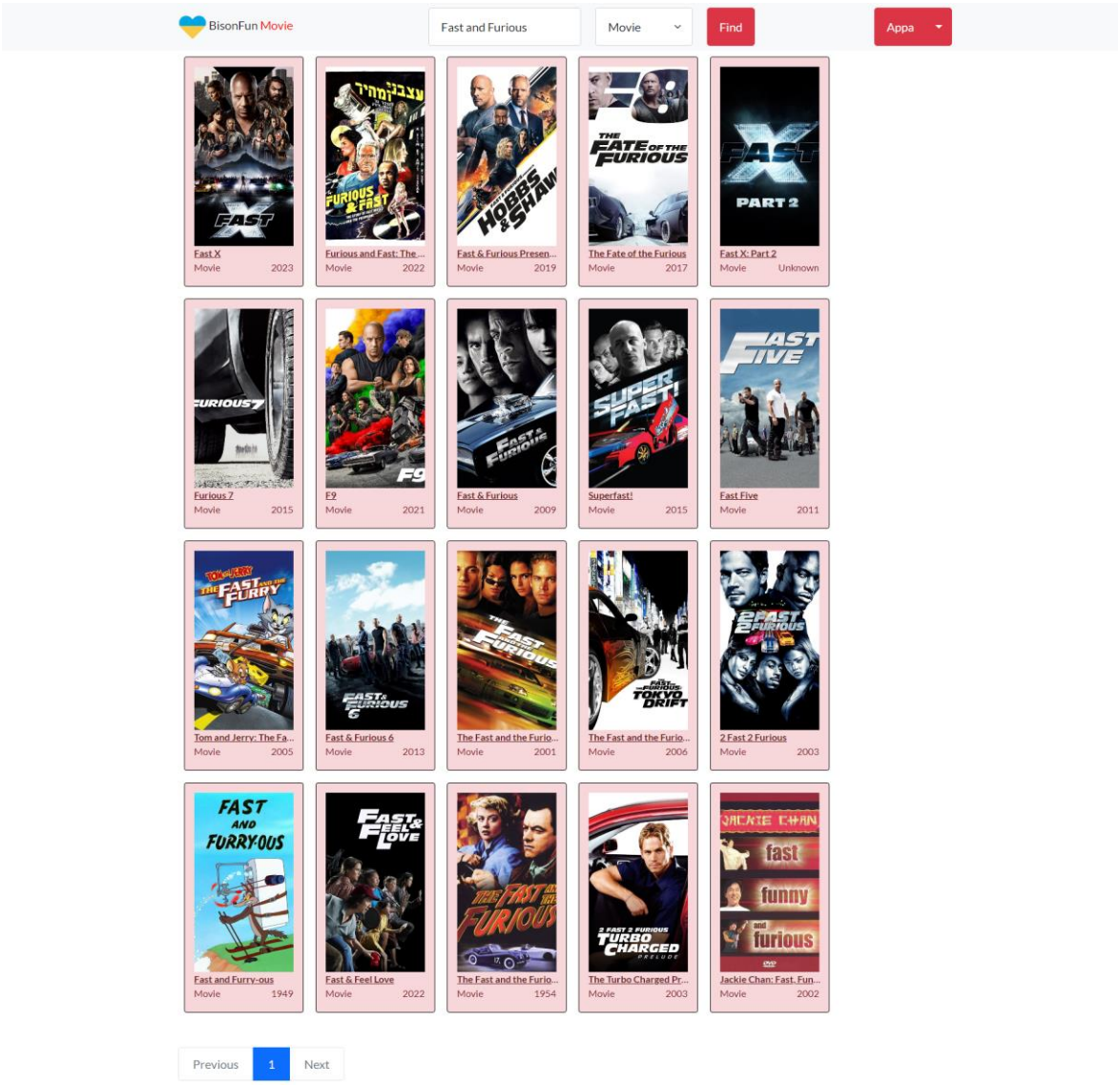


Рис. 3.7 Результат пошуку контенту

Сторінка авторизації \ реєстрації (рис. 3.8) містить в собі наступні елементи:

- текстове привітання користувача;
- поле для введення псевдоніму;
- поле для введення електронної пошти (при реєстрації);
- поле для введення паролю;
- кнопка для авторизації \ реєстрації;
- посилання на реєстрацію\ авторизацію;

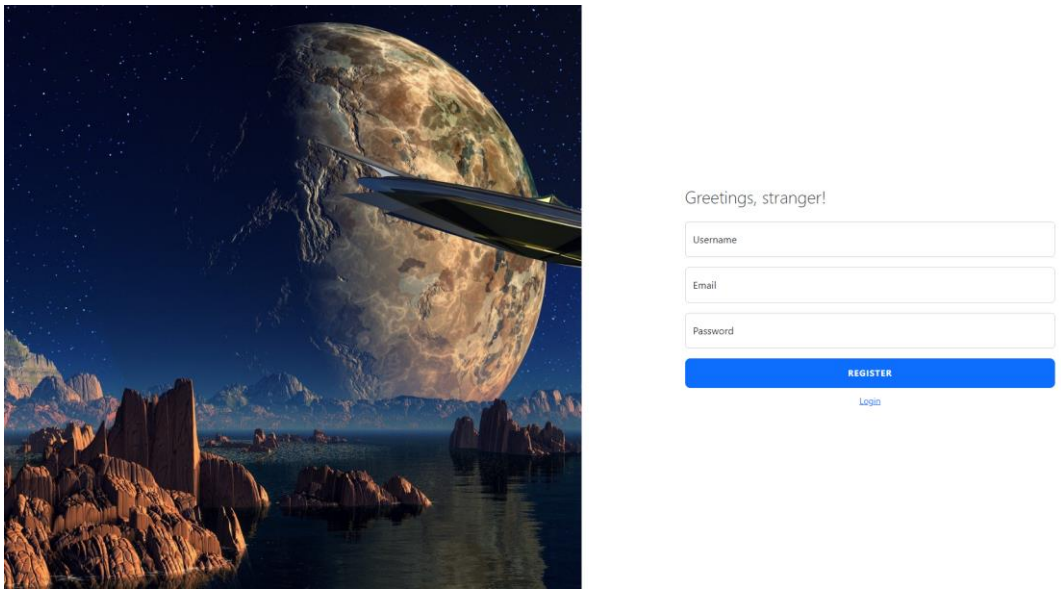


Рис. 3.8 Сторінка реєстрації

Сторінка з фільтром (рис. 3.9) виконує функцію допомоги при виборі для перегляду контенту зі списків користувача.

Сторінка з фільтром містить в собі наступні елементи:

- перемикач між персональним фільтром та глобальним;
- перемикач для списків з аніме контентом, фільмами та серіалами;
- поле введення даних формату checkbox для обирання з яких списків брати контент;
- кнопка, що відправляє запит;
- постер з обраним контентом;
- назва контенту та посилання на його сторінку з інформацією;
- формат контенту;
- рік випуску;

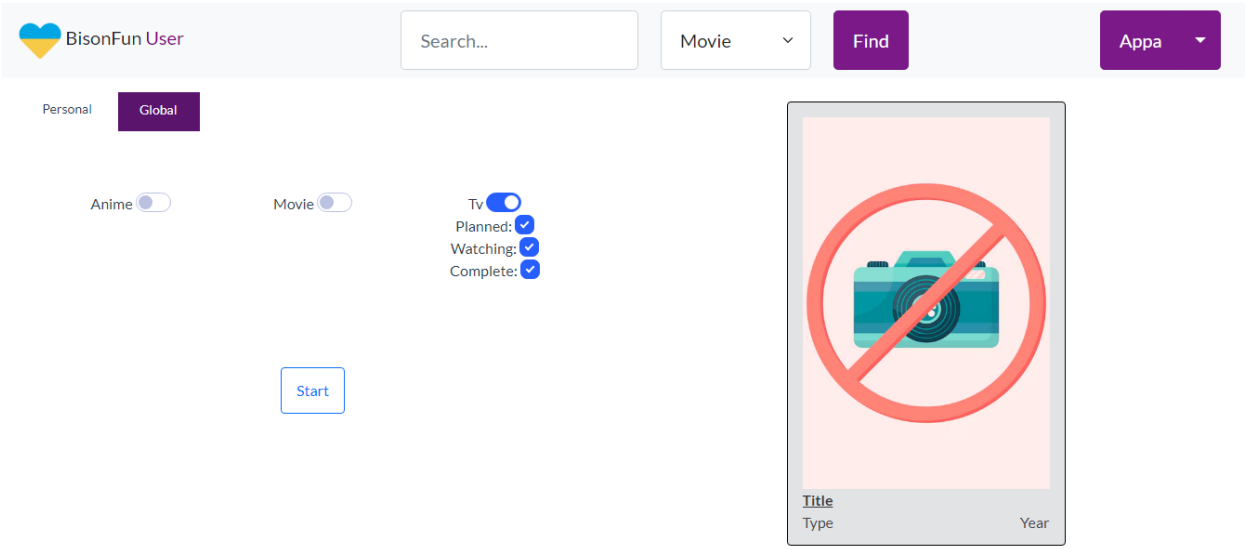


Рис. 3.9 Сторінка з фільтром

3.4 Мапа веб-застосунку

Для веб-застосунку була розроблена мапа (рис. 3.10), яка відображує зв’язки між головними сторінками.

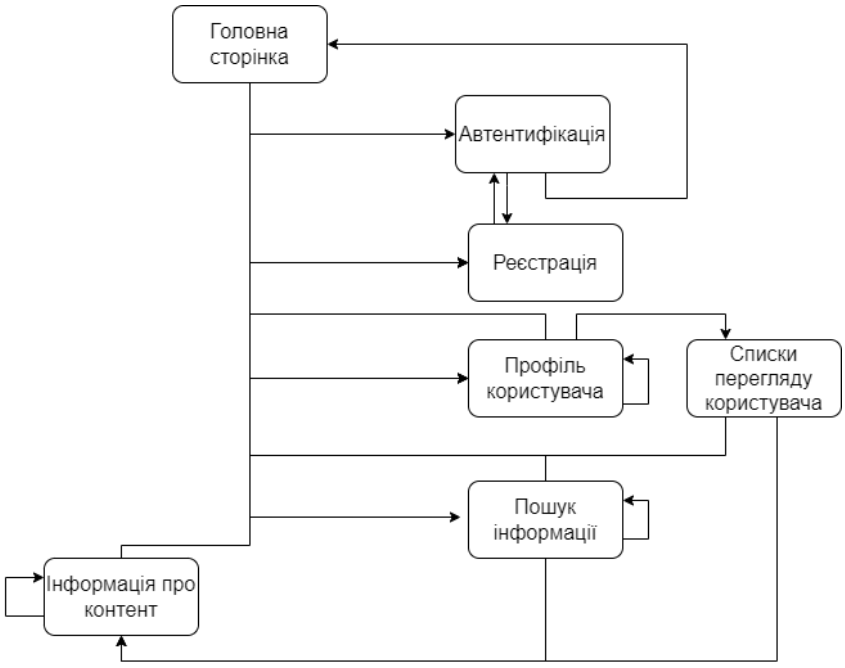


Рис. 3.10 Мапа веб-застосунку

3.5 Реалізація архітектури

MVC шаблон складається з 3 компонентів: модель, вигляд та контролер.

Завдяки функціоналу Spring Data, описано класи сутності що представляють таблиці бази даних, а також створено класи для взаємодії з базою даних. За модель в застосунку відповідають класи взаємодії з базою даних, а також класи взаємодії зі сторонніми API.

Вигляд в застосунку - це HTML сторінки що працюють разом з шаблонізатором Thymeleaf.

Після цього, використовуючи Spring MVC розроблено класи контролери для кожної зі сторінок, в тому числі і контролер, що відповідає за обробку різних HTTP помилок (400, 403, 404, 429 та 500).[11]

Як приклад, метод, що відповідає за відображення сторінки з інформацією про фільм зображено на рисунку 3.11. Контролер отримує інформацію з TMDb API про фільм. Потім контент перевіряється, на приналежність до аніме контенту. Якщо він відноситься до аніме контенту, то відбувається пошук в AniList API по назві і при наявності такого контенту відправляє користувача на сторінку з інформацією про аніме контент. Якщо ж контент відноситься до мейнстрімних фільмів, то інформація про нього, та його приналежність до списків користувача (якщо він авторизований) передається до шаблонізатора Thymeleaf який вже генерує повноцінну сторінку.

```
@GetMapping("/movie/{id}")
public String moviePage(Model model, @PathVariable int id, Principal principal) throws JSONException {
    log.info("User {} get Movie {}", principal == null ? "Unknown" : principal.getName(), id);
    TMDBMovie movie = tmdbClient.parseMovieById(id);

    if (movie.isAnime()) {
        VideoEntertainment anime;
        try {
            anime = aniListClient.parseAnimeByName(movie.getTitle());
        } catch (ContentNotFoundException e) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        } catch (TooManyAnimeRequestsException e) {
            throw new ResponseStatusException(HttpStatus.TOO_MANY_REQUESTS);
        }
        String animeLink = "/anime/"+anime.getId();
        return "redirect:" + animeLink;
    }

    UserMovie userMovie = principal == null ? new UserMovie() : userMovieService.getUserMovieByUsernameAndId(principal.getName(), movie.getId());

    List<TMDBMovie> movieRecommendations = tmdbClient.parseMovieRecommendations(id);

    model.addAttribute( attributeName: "content", movie);
    model.addAttribute( attributeName: "recommendations", movieRecommendations);

    model.addAttribute( attributeName: "actions", asList(VideoConsumingStatus.values()));
    model.addAttribute( attributeName: "userMovie", userMovie);

    return "content_movie";
}
```

Рис. 3.11 Метод сторінки інформації про фільм

3.6 REST контролер

Виключенням з усіх контролерів у нашому застосунку є контролер RandomContentController, який повертає не HTML сторінку, а JSON об’єкт. Цей контролер призначений для отримання запитів від сторінки з фільтром контенту списків користувача, опрацювання цих запитів (рис. 3.12) і повернення навмання обраного контенту з обраних списків користувача. Це дозволяє користувачам отримувати випадково вибраний контент відповідно до своїх вподобань та заданих фільтрів.

Рис. 3.12 Обробка запиту від сторінки з фільтром контенту на базі списків перегляду користувача

3.6 Налаштування безпеки та кешування

3.6.1 Кешування

Хоча Spring Boot Caching Starter і надає менеджер кешу, він зберігає кеш на протязі усього часу роботи застосунку, що не підходить для збереження інформації про контент, який може змінюватися протягом часу. Для розв'язання проблеми збереження кешу, що може змінюватися протягом часу, було вирішено використати бібліотеку Caffeine Cache. Ця бібліотека надає багато опцій для налаштування, дозволяючи встановлювати термін життя кешу та інші параметри. Налаштовується кеш таким чином, щоб він зберігався в застосунку протягом 80 хвилин.

Клас конфігурації (рис. 3.13) включає метод для створення менеджера кешу, який використовує Caffeine. Використання Caffeine Cache дозволяє ефективно керувати кешем, налаштовуючи його відповідно до потреб застосунку. Це забезпечує актуальність даних та підвищує продуктивність застосунку, зменшуючи навантаження на базу даних та інші джерела даних за рахунок повторного використання кешованих результатів.

```
@Configuration
@EnableCaching
public class CacheConfiguration {
    Codeium: Refactor | Explain | Docstring | ×
    Alexander
    @Bean
    public Caffeine<Object, Object> caffeineConfig() {
        return Caffeine.newBuilder().expireAfterWrite( duration: 80, TimeUnit.MINUTES);
    }
    Codeium: Refactor | Explain | Docstring | ×
    Alexander
    @Bean
    public CacheManager cacheManager(Caffeine<Object, Object> caffeine){
        CaffeineCacheManager caffeineCacheManager = new CaffeineCacheManager();
        caffeineCacheManager.setCaffeine(caffeine);
        return caffeineCacheManager;
    }
}
```

Рис. 3.13 Клас налаштування кешування

3.6.2 Безпека

Для забезпечення безпеки даних користувачів, а також реалізації механізмів авторизації у проекті використовується Spring Security.[13] При налаштуванні модуля Spring Security (рис. 3.14) визначаються сторінки, доступні лише для авторизованих користувачів, обробка успішної авторизації та сама перевірка авторизації.

```
@Bean
public SecurityFilterChain securityChain(HttpSecurity http) throws Exception {
    http
            .exceptionHandling() ExceptionHandlingConfigurer <HttpSecurity >
            .authenticationEntryPoint(loginWithTargetUrlAuthenticationEntryPoint)

            .and().formLogin(loginForm -> loginForm
                    .loginPage(LOGIN_FORM_URL).permitAll()
                    .successHandler(redirectToOriginalUrlAuthenticationSuccessHandler)
            ) HttpSecurity
            .authorizeHttpRequests(request -> request
                    .requestMatchers(LOGIN_FORM_URL).permitAll()
                    .requestMatchers("/cabinet").authenticated()
                    .requestMatchers("/wtw").authenticated()
                    .anyRequest().permitAll()
            )
            .authenticationProvider(authenticationProvider);

    return http.build();
}
```

Рис. 3.14 Налаштування механізмів безпеки та авторизації

3.7 Тестування

Тестування веб-застосунку є критично важливим етапом у процесі розробки, оскільки воно допомагає забезпечити якість і стабільність коду. Використано декілька інструментів і підходів для проведення різних типів тестування, включаючи модульне тестування та мануальне тестування.[22]

Модульне тестування передбачає тестування окремих компонентів застосунку, ізольовано від решти системи. Це дозволяє перевірити коректність

роботи кожного компонента незалежно від інших. Для модульного тестування у Spring Boot часто використовуються JUnit та Mockito.[23, с.7]

Мануальне тестування передбачає ручну перевірку функціональності веб-застосунку без використання автоматизованих інструментів. Це дозволяє виявляти дефекти, які можуть бути пропущені автоматизованими тестами, зокрема ті, що пов'язані з користувацьким інтерфейсом, зручністю використання та іншими суб'єктивними аспектами.

3.7.1 Модульне тестування

Під час розробки програмного забезпечення, яке взаємодіє зі сторонніми API, важливо переконатися, що воно працює належним чином і не виникає несподіваних помилок або збоїв. Завдяки ретельному підходу до модульного тестування класів, що зв'язані зі сторонніми API, була досягнута висока ступінь стабільності та надійності програмного забезпечення.

Низка модульних тестів була розроблена для охоплення різних сценаріїв та випадків використання. Ці тести глибоко перевіряли функціональні можливості програми, включаючи взаємодію зі сторонніми сервісами, обробку даних та зберігання інформації.

Як приклад, наведено декілька модульних тестів, які наявні в проекті.

На рисунку 3.15 представлений тест, який перевіряє взаємодію між класами TmdbClient та TmdbApiResponse. Цей тест виконує перевірку на коректну конвертацію об'єктів JSON у Java об'єкти. Це допомагає переконатися, що дані зі стороннього сервісу правильно обробляються та інтегруються у програму.

На рисунку 3.16 показаний тест, який випробовує зберігання інформації, отриманої з AniList API, в кеші. Це важливо для переконання в тому, що дані зберігаються та використовуються ефективно, зменшуючи час доступу до сторонніх сервісів і підвищуючи продуктивність програми.

Всі ці тести разом допомогли забезпечити високу якість програмного забезпечення, зменшивши ризик виникнення непередбачених помилок або збоїв та забезпечивши безперебійну роботу для користувачів.

```
@Test
public void testParseMovieById(){
    TmdbClient tmdbClient = new TmdbClient(tmdbApiResponse, gson);

    when(tmdbApiResponse.getMovieById(872585)).thenReturn(new JSONObject(oppenheimer));

    TMDBMovie movie = tmdbClient.parseMovieById(872585);

    System.out.println(movie.toString());
    Assertions.assertEquals( expected: 872585, movie.getId());
    Assertions.assertEquals( expected: "Oppenheimer", movie.getTitle());
    Assertions.assertEquals(VideoContentStatus.RELEASED, movie.getStatus());
    Assertions.assertEquals( expected: 8.11, movie.getScore(), delta: 0.5);
    Assertions.assertEquals( expected: "tt15398776", movie.getImdbId());
    Assertions.assertEquals(
            expected: "The story of J. Robert Oppenheimer's role in the development of the atomic bomb during World War II.",
            movie.getDescription()
    );
    Assertions.assertEquals( expected: 2, movie.getGenres().length);
    Assertions.assertEquals( expected: "Drama", movie.getGenres()[0]);
    Assertions.assertEquals( expected: "History", movie.getGenres()[1]);
}
```

Рис. 3.15 Тестування одного з методів TmdbClient

```
@Test
public void animeParseCachingTest(){
    Optional<JSONObject> jsonAnime = Optional.empty();
    try {
        jsonAnime = Optional.ofNullable(aniListApiResponse.getAnimeById(450));
    } catch (TooManyAnimeRequestsException | ContentNotFoundException e) {
        e.printStackTrace();
    }

    assertEquals(jsonAnime, getCachedJSONObject(JSON_ANIME, id: 450));
}
```

Рис. 3.16 Тестування кешування інформації з AniList API

3.7.2 Мануальне тестування

Для перевірки функціонування веб-застосунку було сформовано тест-кейси. Тест-кейси являють собою сценарії, які визначають послідовність дій, необхідних для перевірки роботи певних функцій веб-застосунку. Вони допомагають

забезпечити, що всі частини застосунку працюють відповідно до вимог і очікувань. Ці тест-кейси охоплюють різні аспекти функціонування, включаючи вхідні дані, очікувані результати, і можливі помилки.

Далі наведено таблиці тест-кейсів для деяких сторінок веб-застосунку. У таблицях міститься інформація про конкретні дії, які необхідно виконати на кожній зі сторінок, разом з описом очікуваних результатів. Це дозволяє чітко визначити, які функції перевіряються, і дає змогу легко відслідковувати успішність або невдачі тестів. У разі виявлення помилок або невідповідностей, ці тест-кейси також служать засобом для відтворення проблем і полегшення їх виправлення.

Таблиця 3.1

Тест-кейси для сторінки авторизації

Сторінка авторизації		
Кроки	Дані тесту	Очікуваний результат
Авторизація користувача	Вірна інформація	Успішна авторизація, перехід на головну сторінку
Авторизація користувача	Невірна інформація	Повідомлення про неправильні дані авторизації

Таблиця 3.2

Тест-кейси для сторінки з інформацією про контент

Сторінка з інформацією про контент		
Кроки	Дані тесту	Очікуваний результат
Зміна статусу перегляду через кнопку на сторінці	Неавторизований користувач	Відображення сторінки авторизації
Зміна статусу перегляду через кнопку на сторінці	Авторизований користувач	Зміна статусу перегляду
Натискання на посилання на сторонні сервіси		Перехід на сторонній сервіс з інформацією про обраний контент
Натискання на посилання рекомендованого контенту		Перехід на сторінку інформації про обраний контент

Таблиця 3.3

Тест-кейси для сторінки пошуку

Сторінка пошуку		
Кроки	Дані тесту	Очікуваний результат
Введення в пошуковий рядок назви фільму	Назва існуючого фільму	Відображення списку результату з посиланням на інформацію про існуючий фільм
Введення в пошуковий рядок назви фільму	Пробіл	Відображення пустого списку результату

3.7.3 Тестування сумісності

Однією з вимог до веб-застосунку є сумісність з популярними браузерами. До найбільш вживаних браузерів на 2024 рік відноситься Google Chrome, Safari та Edge.

Використовуючи сервіси для тестування веб-застосунків, такі як LambdaTest[24] та Browshot[25] проведено перевірки, на правильне функціонування та зображення сторінок. На рисунках 3.17, 3.18 та 3.19 зображено приклади зовнішнього вигляду головної сторінки в різних браузерах та системах.

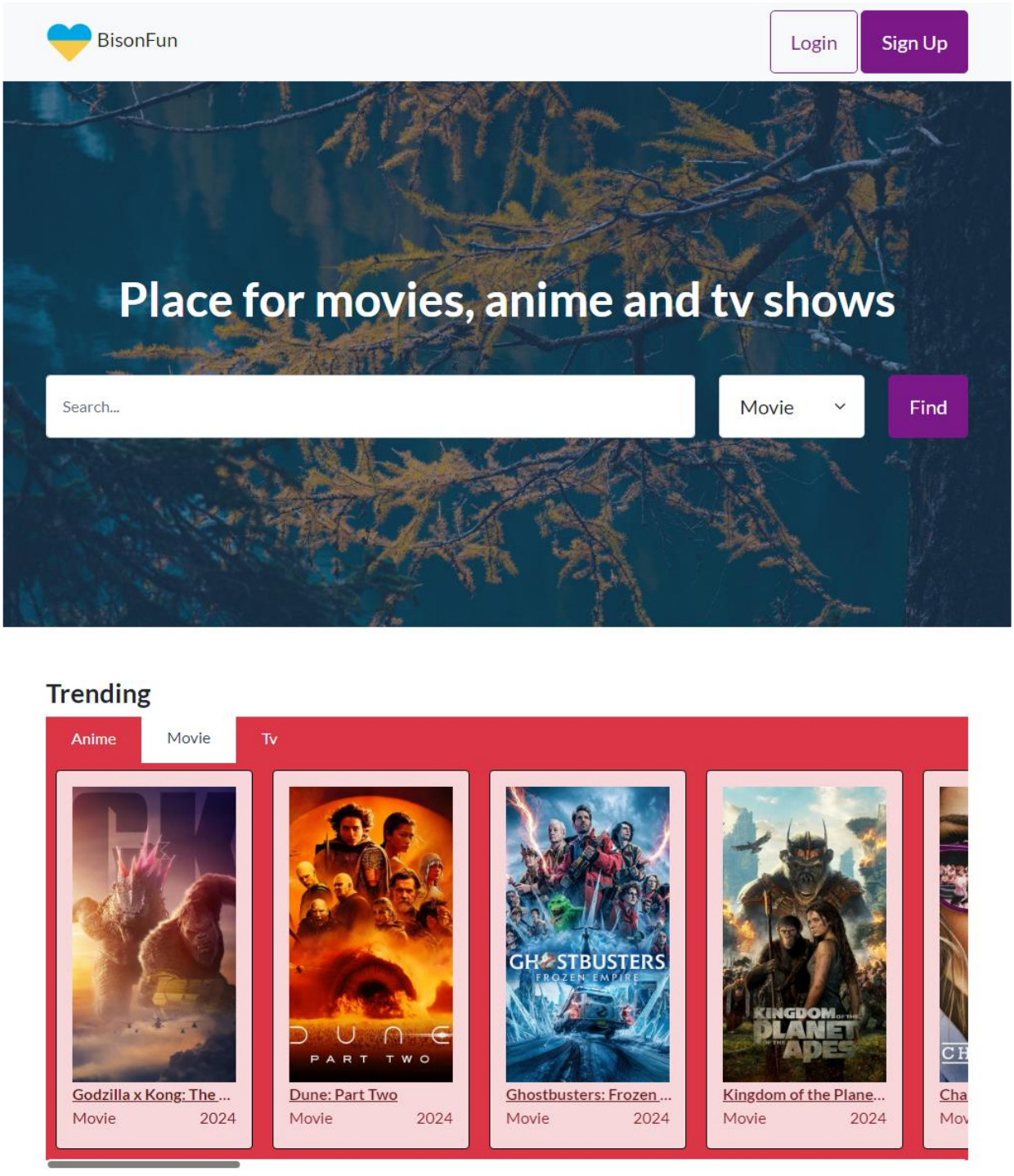


Рис. 3.17 Головна сторінка, Google Chrome, Windows 10, 963x1153

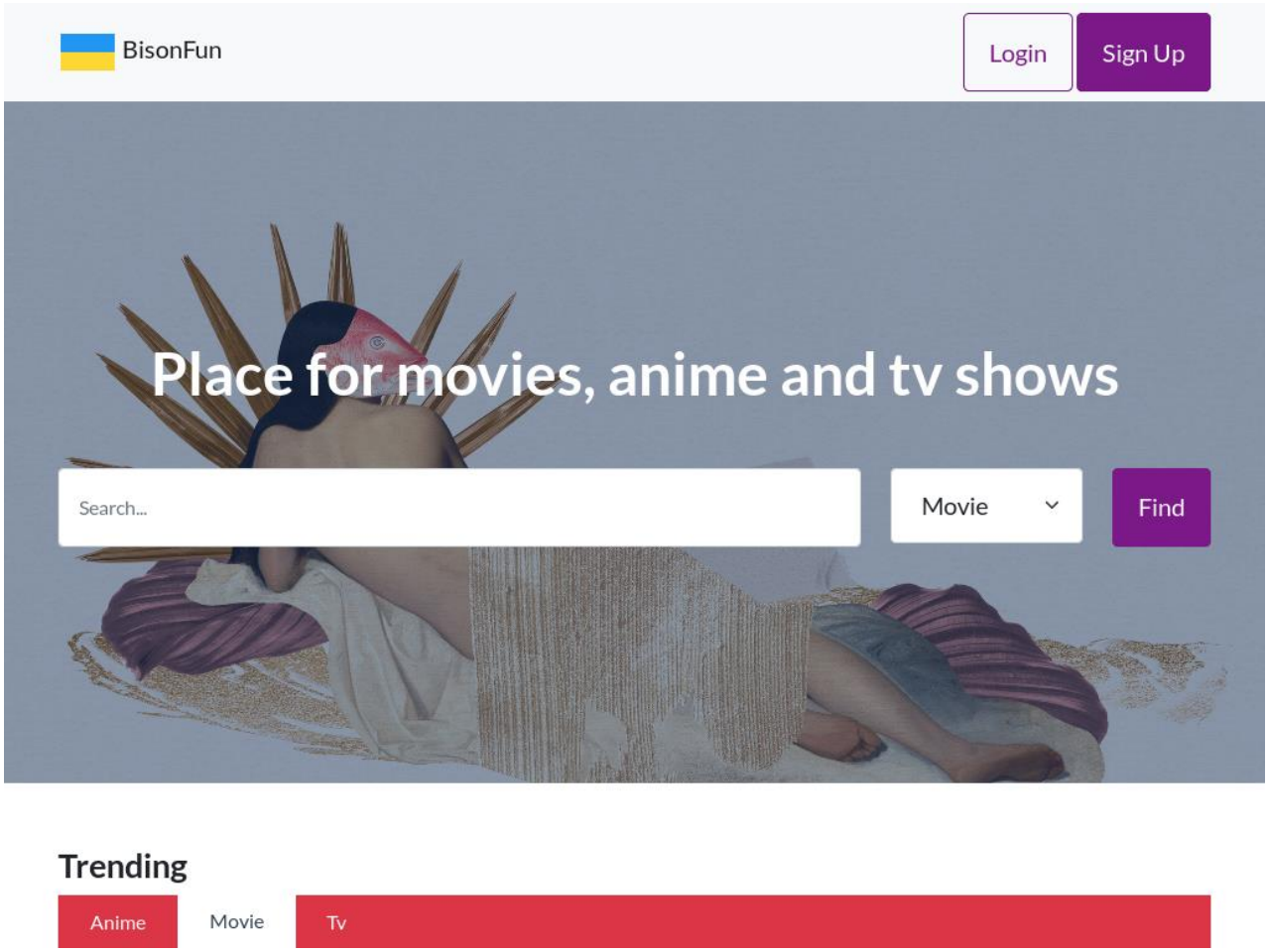


Рис. 3.18 Головна сторінка, Edge, Windows 7, 1024x768

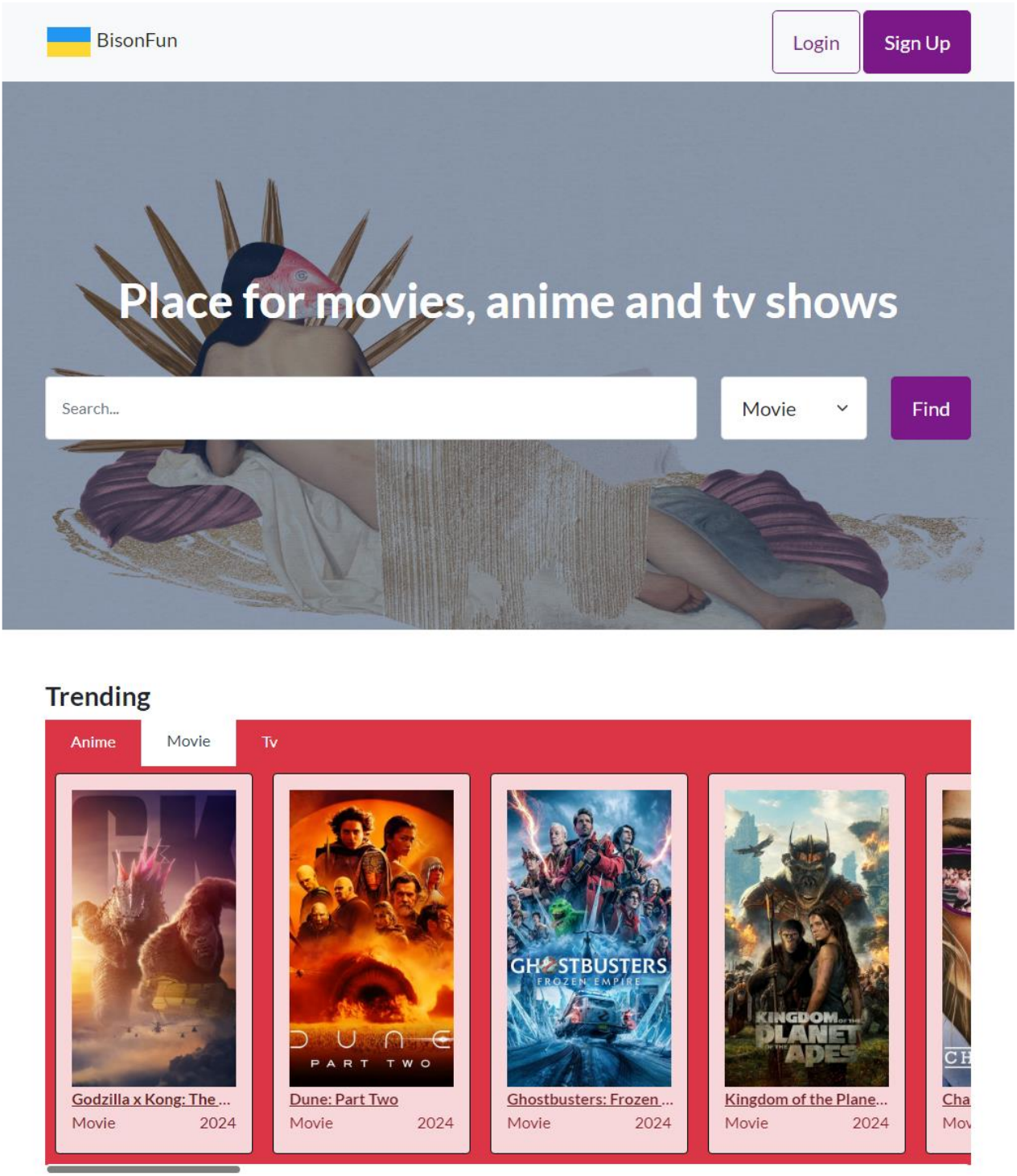


Рис. 3.19 Головна сторінка, Safari, MacOS Ventura, 963x1153

ВИСНОВКИ

У результаті виконання кваліфікаційної бакалаврської роботи було спрощено відстежування списків перегляду фільмів та серіалів різного формату за рахунок використання веб-застосунку, створеного на мові Java.

Проведено огляд та проаналізовано існуючі веб-застосунки, основна проблема яких базується на відсутності великої частки інформації та контенту, а також обмежених можливостях керування списками перегляду.

Проаналізовано та обрано наступні технічні засоби реалізації веб-застосунку: мова програмування Java, фреймворк Spring з модулями, HTTP клієнт Unirest, система управління базами даних MySQL та шаблонізатор Thymeleaf.

На базі сформованих вимог було спроектовано веб-застосунок. Створено та використано Use case та ER діаграми.

Розроблено веб-застосунок для керування списками перегляду фільмів та серіалів.

Проведено модульне тестування функціоналу застосунку використовуючи бібліотеки JUnit та Mockito. Проведено мануальне тестування сторінок веб-застосунку, а також тестування сумісності з іншими браузерами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Definition of ANIME. Merriam-Webster: america's most trusted dictionary. URL: https://www.merriam-webster.com/dictionary/anime (дата звернення: 20.05.2024).

2. Mainstream. Cambridge dictionary | english dictionary, translations & thesaurus. URL: https://dictionary.cambridge.org/dictionary/english/mainstream (дата звернення: 20.05.2024).

3. Evens T., Henderickx A., Conradie P. Technological affordances of video streaming platforms: why people prefer video streaming platforms over television. European journal of communication. 2023. С. 026732312311557. URL: https://doi.org/10.1177/02673231231155731 (дата звернення: 20.05.2024).

4. AL-Zoubi A. Digital technology and changes in media consumption: a case study of smartphone and app usage. Technology: toward business sustainability. Cham, 2024. С. 433–444. URL: https://doi.org/10.1007/978-3-031-54019-6_39 (дата звернення: 20.05.2024).

5. Mohammad Alzub A. Navigating the disruption of digital and conventional media in changing media consumption landscape in digital era. Journal of engineering, technology, and applied science (JETAS). 2023. Т. 5, № 1. С. 38–48. URL: https://doi.org/10.36079/lamintang.jetas-0501.517 (дата звернення: 20.05.2024).

6. Гулеватий О.О., Яскевич В.О. Вибір оптимального протоколу API для веб-сервісів. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. с.98-101.

7. IMDb: ratings, reviews, and where to watch the best movies & TV shows. IMDb. URL: https://www.imdb.com/ (дата звернення: 20.05.2024).

8. MyAnimeList.net - anime and manga database and community. MyAnimeList.net. URL: https://myanimelist.net/ (дата звернення: 20.05.2024).

9. About us. About us | Anime-Planet. URL: https://www.anime-planet.com/about (дата звернення: 20.05.2024).

10. Schildt H. Java : the complete reference, twelfth edition: comprehensive coverage of the java language. McGraw-Hill Education, 2021. 1248 с.

11. Downey T. Spring MVC. Texts in computer science. Cham, 2021. С. 171–225. URL: https://doi.org/10.1007/978-3-030-62274-9_5 (дата звернення: 20.05.2024).

12. Spring data JPA reference. Spring. URL: https://docs.spring.io/spring-data/jpa/docs/1.10.x/reference/pdf/spring-data-jpa-reference.pdf (дата звернення: 20.05.2024).

13. Spilca L. Spring security in action. Manning Publications Co. LLC, 2020. 560 с.

14. Silva R. MySQL crash course. No Starch Press, Incorporated, 2023. 352 с.

15. Sarcar V. MVC pattern. Java design patterns. Berkeley, CA, 2022. С. 593–617. URL: https://doi.org/10.1007/978-1-4842-7971-7_27 (дата звернення: 20.05.2024).

16. AniList. AniList. URL: https://anilist.co (дата звернення: 20.05.2024).

17. The movie database. The Movie Database (TMDB). URL: https://www.themoviedb.org (дата звернення: 20.05.2024).

18. Brito G., Valente M. T. REST vs graphql: a controlled experiment. 2020 IEEE international conference on software architecture (ICSA), м. Salvador, Brazil, 16–20 берез. 2020 р. 2020. URL: https://doi.org/10.1109/icsa47634.2020.00016 (дата звернення: 20.05.2024).

19. Marek A., Zhou D. GraphQL with java and spring. 2023. 481 с.

20. Rate Limiting | AniList APIv2 Docs. Introduction | AniList APIv2 Docs. URL: https://anilist.gitbook.io/anilist-apiv2-docs/overview/rate-limiting (дата звернення: 20.05.2024).

21. Rate limiting. The Movie Database (TMDB). URL: https://developer.themoviedb.org/docs/rate-limiting (дата звернення: 20.05.2024).

22. Refactoring test smells / E. Soares та ін. SAST 20: 5th brazilian symposium on systematic and automated software testing, м. Natal Brazil. New York, NY, USA, 2020. URL: https://doi.org/10.1145/3425174.3425212 (дата звернення: 20.05.2024).

23. JUnit 5 user guide. JUnit. URL: https://junit.org/junit5/docs/5.8.0-M1/user-guide/junit-user-guide-5.8.0-M1.pdf (дата звернення: 20.05.2024).

24. About lambdatest inc | cross browser testing cloud. Next-Generation Mobile Apps and Cross Browser Testing Cloud | LambdaTest. URL: https://www.lambdatest.com/about (дата звернення: 20.05.2024).

25. Service for real time website screenshots - Browshot. Service for real time website screenshots - Browshot. URL: https://browshot.com/ (дата звернення: 20.05.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку для відстеження списків перегляду фільмів та серіалів з використанням мови Java та фреймворку Spring

Виконав студент 4 курсу
групи ПД-41
Гулеватий Олександр Олександрович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**: спрощення відстежування списків перегляду фільмів та серіалів різного формату за рахунок використання веб-застосунку, створеного на мові Java.
- **Об'єкт дослідження:** процес відстеження перегляду фільмів та серіалів.
- **Предмет дослідження:** веб-застосунок для відстеження списків перегляду фільмів та серіалів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд та аналіз схожих веб-застосунків для відстеження списків перегляду
2. Розробити вимоги до веб-застосунку для відстеження списків перегляду зважаючи на переваги та недоліки проаналізованих аналогів
3. Проаналізувати технічні засоби та обрати оптимальні для розробки веб-застосунку для відстеження списків перегляду
4. Спроектувати та розробити веб-застосунок з використанням обраних технічних засобів
5. Протестувати застосунок за допомогою бібліотек JUnit та Mockito

3

АНАЛІЗ АНАЛОГІВ

	IMDb	MAL	Anime-Planet	BisonFun
Наявність аніме контенту	обмежена	+	+	+
Наявність не аніме контенту	+	-	-	+
Керування списками перегляду	примітивне	+	+	+
Відстеження епізодів	-	+	+	+
Платформи	Web, iOS, Android	Web, iOS, Android	Web	Web
Можливість ставити власну оцінку	+	+	+	+
Статистика списків перегляду користувача	-	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Пошук фільмів та серіалів різного типу
2. Сторінки з інформацією про обраний фільм чи серіал
3. Реєстрація та автентифікація користувача
4. Керування списками перегляду
5. Фільтр по назві контенту в списках перегляду
6. Користувацька статистика перегляду

Нефункціональні вимоги:

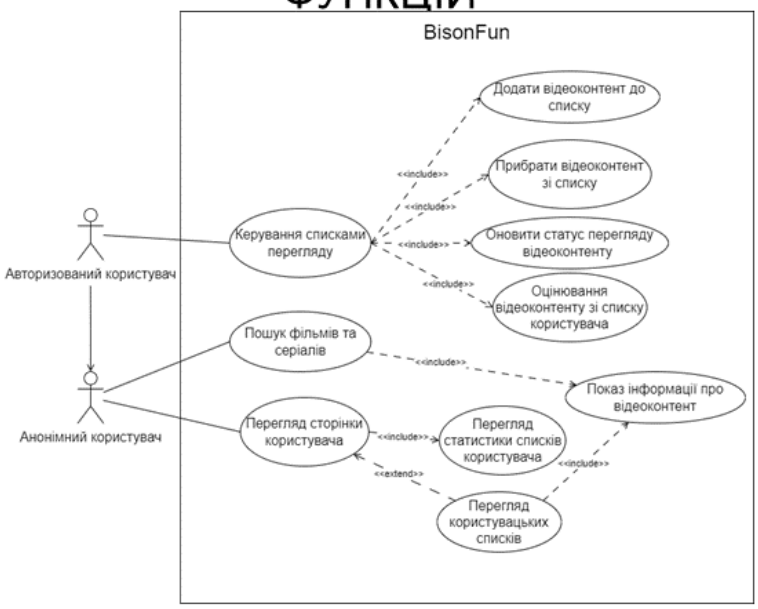
1. Модульність застосунку
2. Сумісність застосунку з популярними браузерами

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

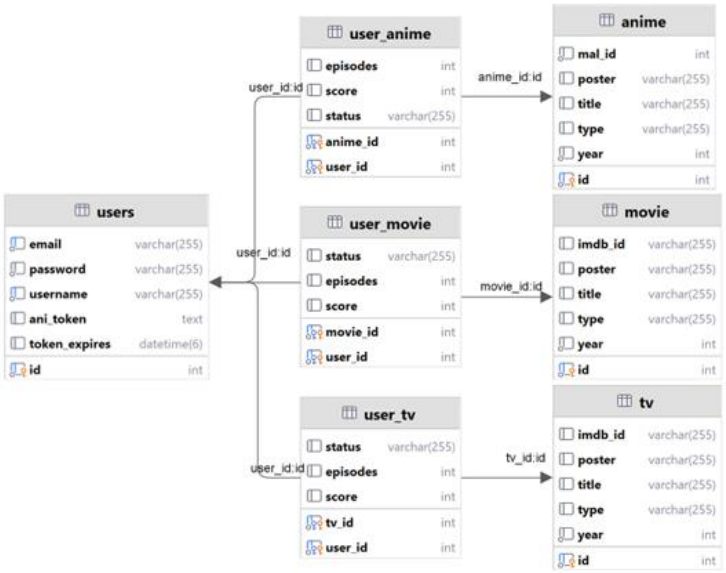
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ ОСНОВНИХ ФУНКЦІЙ



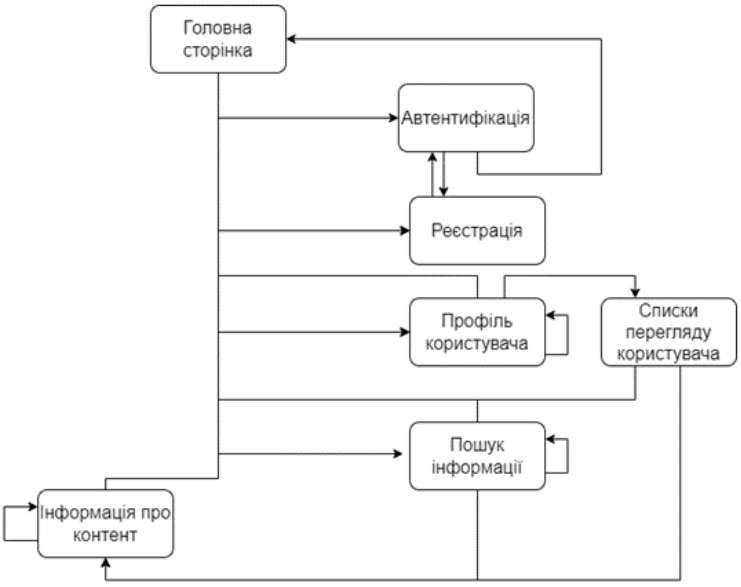
7

СХЕМА БАЗИ ДАНИХ



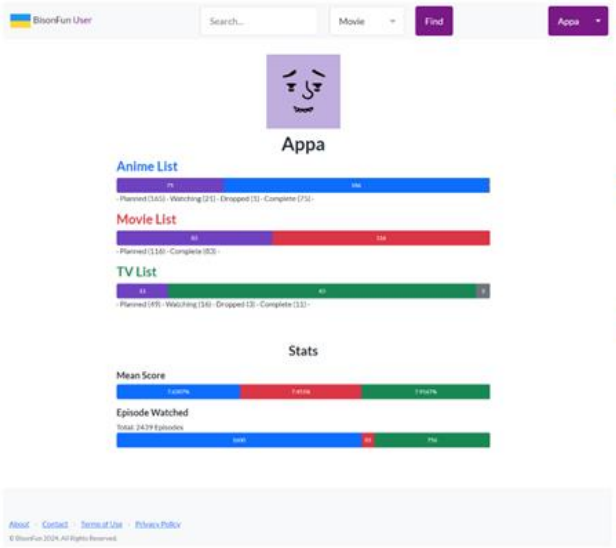
8

МАПА САЙТУ

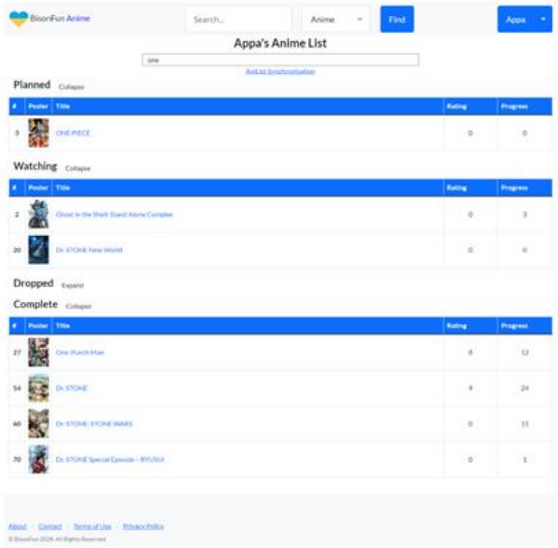


9

ЕКРАННІ ФОРМИ



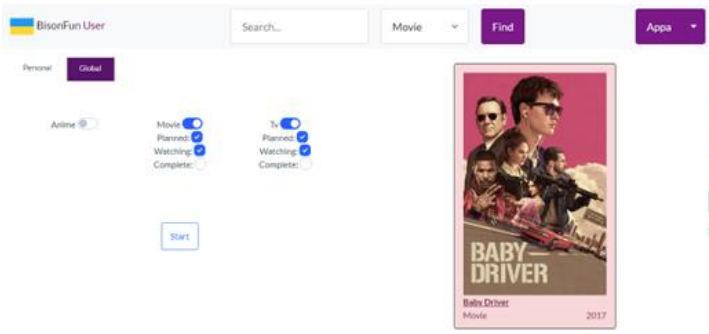
Профіль користувача



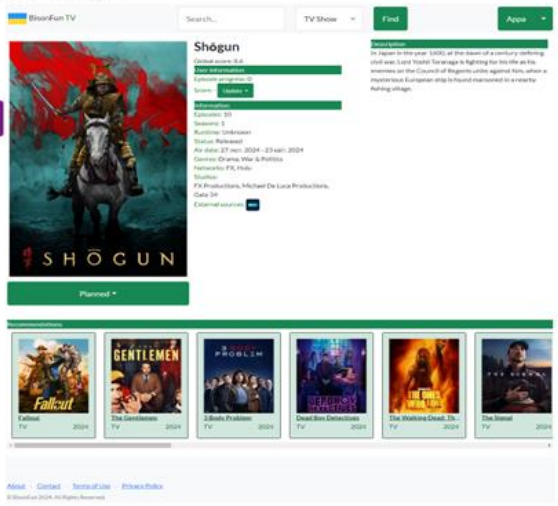
Списки перегляду користувача

10

ЕКРАННІ ФОРМИ



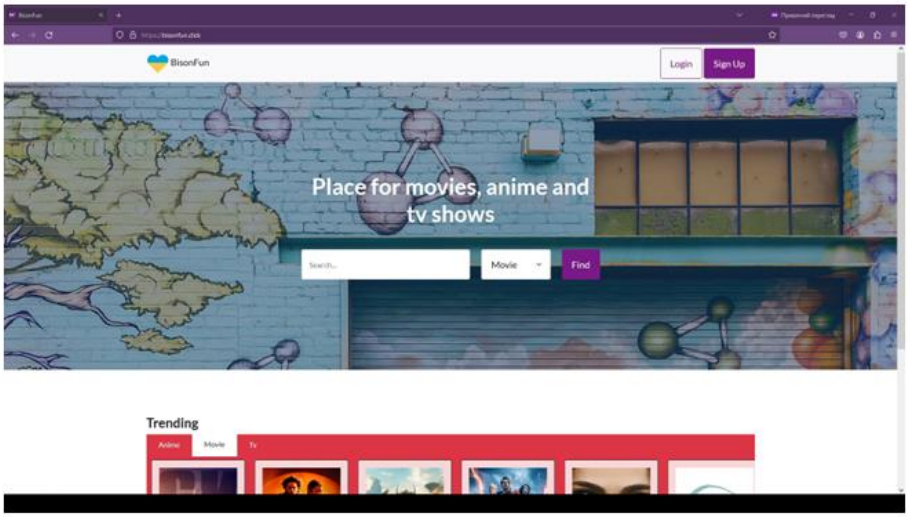
Фільтри для обрання контенту для перегляду на базі списків користувача



Інформація про контент

11

ДЕМОНСТРАЦІЯ РОБОТИ ВЕБ-ЗАСТОСУНКУ



12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Гулеватий О.О. Вибір оптимального протоколу API для веб-сервісів. Актуальні проблеми забезпечення інформаційної та кібернетичної безпеки: Матеріали Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. с.98

ВИСНОВКИ

1. Спрощено відстежування списків перегляду фільмів та серіалів різного формату за рахунок використання веб-застосунку, створеного на мові Java.
2. Проведено огляд та проаналізовано існуючі веб-застосунки, основна проблема яких базується на відсутності великої частки інформації та контенту, а також обмежених можливостях керування списками перегляду.
3. Проаналізовано та обрано наступні технічні засоби реалізації веб-застосунку: мова програмування Java, фреймворк Spring з модулями, HTTP клієнт Unirest, система управління базами даних MySQL та шаблонізатор Thymeleaf.
4. На базі сформованих вимог спроектовано веб-застосунок. Створено та використано Use case та ER діаграми.
5. Розроблено веб-застосунок для керування списками перегляду фільмів та серіалів.
6. Проведено модульне тестування функціоналу застосунку використовуючи бібліотеки JUnit та Mockito. Проведено мануальне тестування сторінок веб-застосунку, а також тестування сумісності з іншими браузерами.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.18</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com</groupId>
    <artifactId>bisonfun</artifactId>
    <version>0.8.2-ALPHA</version>
    <name>bisonfun</name>
    <description>bisonfun</description>
    <packaging>jar</packaging>
    <properties>
        <java.version>11</java.version>
        <org.mapstruct.version>1.5.5.Final</org.mapstruct.version>
        <spring-security.version>5.8.11</spring-security.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-cache</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <!-- Exclude the default Jackson dependency -->
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-json</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.thymeleaf.extras</groupId>
            <artifactId>thymeleaf-extras-springsecurity5</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
```

```
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${org.mapstruct.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>com.vaadin.external.google</groupId>
            <artifactId>android-json</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>

<dependency>
    <groupId>com.github.ben-manes.caffeine</groupId>
    <artifactId>caffeine</artifactId>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20240303</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
```

```
            <version>2.10.1</version>
        </dependency>
        <dependency>
            <groupId>com.konghq</groupId>
            <artifactId>unirest-java</artifactId>
            <version>3.14.2</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${artifactId}</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.mapstruct</groupId>
                            <artifactId>mapstruct-processor</artifactId>
                            <version>${org.mapstruct.version}</version>
                        </path>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                            <version>1.18.22</version>
                        </path>
                        <dependency>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok-mapstruct-binding</artifactId>
                            <version>0.2.0</version>
                        </dependency>
                    </annotationProcessorPaths>
                    <compilerArgs>
                        <compilerArg>
                            -Amapstruct.defaultComponentModel=spring
                        </compilerArg>
                    </compilerArgs>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
@Component
@Slf4j
public class AniListApiResponse {

    final
    Environment environment;

    @Autowired
    public AniListApiResponse(Environment environment) {
        this.environment = environment;
    }

    /**
```

```
     * Get anime list by search query.
     * @param search string query to get list.
     * @param page current page of list
     * @return "Page" JSONObject which have page info(number, count, etc.) and
JSONArray of anime "Media" JSONObjects.
     * @throws TooManyAnimeRequestsException if Anilist.co have got more requests
than limit from app's account.
     * */
    public JSONObject getAnimeList(String search, int page) throws
TooManyAnimeRequestsException{
        log.info("Get list of Anime by \"{}\" Page: {}", search, page);

        String variables = "{\n" +
                "  \"query\": \""+search+"\",\n" +
                "  \"page\": "+page+"\n" +
                "}";


        //get anime list from AniList API
        HttpResponse<String> result = Unirest.post(AniList.GRAPHQL.link)
                .queryString("query", AniListQuery.SEARCH.getQuery())
                .queryString("variables", variables)
                .asString();

        log.debug(result.getBody());

        if(result.getStatus() == 429){
            String secs = result.getHeaders().getFirst("Retry-After");
            int seconds = Integer.parseInt(secs);
            log.warn("delay in {} seconds", seconds);
            throw new TooManyAnimeRequestsException("Too many requests to
Anilist.co. will be available after " + seconds + "seconds", seconds);
        }else if(result.getStatus() == 404){
            log.error("Anime weren't found("+search+");");
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }else if(result.getStatus() == 400){
            log.error("Something went wrong:\n"+
AniList.GRAPHQL+"\n"+AniListQuery.ANIME_BY_ID+"\n"+variables);
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        return new
JSONObject(result.getBody()).getJSONObject("data").getJSONObject("Page");
    }
    /**
     * Get list of anime trends. Cacheable as "animeTrends".
     * @return "Page" JSONObject which have page info(number, count, etc.) and
JSONArray of anime "Media" JSONObjects.
     * @throws TooManyAnimeRequestsException if Anilist.co have got more requests
than limit from app's account.
     * @throws NoAccessException if app can't access to Anilist.co API.
     */
    @Cacheable("animeTrends")
    public JSONObject getAnimeTrends() throws TooManyAnimeRequestsException,
NoAccessException {
        log.info("Get anime trends");
        HttpResponse<String> result;
        try {
            result = Unirest.post(AniList.GRAPHQL.link)
                    .queryString("query", AniListQuery.ANIME_TRENDING.getQuery())
                    .asString();
        }catch (Exception e){
            throw new NoAccessException("No Access to Anilist.co");
```

```
        }
        log.debug(result.getBody());

        if(result.getStatus() == 429){
            String secs = result.getHeaders().getFirst("Retry-After");
            int seconds = Integer.parseInt(secs);
            log.warn("delay in {} seconds", seconds);
            throw new TooManyAnimeRequestsException("Too many requests to
Anilist.co. will be available after " + seconds + "seconds", seconds);
        }else if(result.getStatus() == 400){
            log.error("Something went wrong:\n"+
AniList.GRAPHQL+"\n"+AniListQuery.ANIME_BY_ID+"\n");
            throw new NoAccessException("No Access to Anilist.co");
        }

        return new
JSONObject(result.getBody()).getJSONObject("data").getJSONObject("Page");
    }
    /**
     * Get anime by id. Cacheable as "jsonAnime".
     * @param id identification number from Anilist database.
     * @return "Media" JSONObject with info about anime (id, title, description,
etc.).
     * @throws TooManyAnimeRequestsException if Anilist.co have got more requests
than limit from app's account.
     * @throws ContentNotFoundException if there's no such anime in database.
     */
    @Cacheable("jsonAnime")
    public JSONObject getAnimeById(int id) throws TooManyAnimeRequestsException,
ContentNotFoundException {
        log.info("Get Anime: {}", id);
        String variables = "{\n" +
                "  \"id\": "+id+"\n" +
                "}";

        //get anime from AniList API
        HttpResponse<String> anime = Unirest.post(AniList.GRAPHQL.link)
                .queryString("query", AniListQuery.ANIME_BY_ID.getQuery())
                .queryString("variables", variables)
                .asString();

        log.debug(anime.getBody());

        if(anime.getStatus() == 429){
            int seconds = Integer.parseInt(anime.getHeaders().getFirst("Retry-
After"));
            log.warn("delay in {} seconds", seconds);
            throw new TooManyAnimeRequestsException("Too many requests to
Anilist.co, will be available after " + seconds + "seconds", seconds);
        }else if(anime.getStatus() == 404){
            log.error("Anime {} not found", id);
            throw new ContentNotFoundException("Anime #"+id+" not found");
        }else if(anime.getStatus() == 400){
            log.error("Something went wrong:\n"+
AniList.GRAPHQL+"\n"+AniListQuery.ANIME_BY_ID+"\n"+variables);
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        return new
JSONObject(anime.getBody()).getJSONObject("data").getJSONObject("Media");
    }
    /**
     * Get anime by name. Cacheable as "jsonAnime".
```

```
     * @param name title from Anilist database
     * @return "Media" JSONObject with info about anime (id, title, description,
etc.).
     * @throws TooManyAnimeRequestsException if Anilist.co have got more requests
than limit from app's account.
     * @throws ContentNotFoundException if there's no such anime in database.
     */
    @Cacheable("jsonAnime")
    public JSONObject getAnimeByName(String name) throws
TooManyAnimeRequestsException, ContentNotFoundException {
        log.info("Get Anime: {}", name);
        String variables = "{\n" +
                "  \"name\": \""+name+"\"\n" +
                "}";

        //get anime from AniList API
        HttpResponse<String> anime = Unirest.post(AniList.GRAPHQL.link)
                .queryString("query", AniListQuery.ANIME_BY_NAME.getQuery())
                .queryString("variables", variables)
                .asString();

        log.debug(anime.getBody());

        if(anime.getStatus() == 429){
            int seconds = Integer.parseInt(anime.getHeaders().getFirst("Retry-
After"));
            log.warn("delay in {} seconds", seconds);
            throw new TooManyAnimeRequestsException("Too many requests to
Anilist.co, will be available after " + seconds + "seconds", seconds);
        }else if(anime.getStatus() == 404){
            log.error("Anime \"{}\" not found", name);
            throw new ContentNotFoundException("Anime '"+name+"' not found");
        }else if(anime.getStatus() == 400){
            log.error("Something went wrong:\n"+
AniList.GRAPHQL+"\n"+AniListQuery.ANIME_BY_NAME+"\n"+variables);
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        return new
JSONObject(anime.getBody()).getJSONObject("data").getJSONObject("Media");
    }
    /**
     * Get token from Anilist.co by code.
     * @param code granted from authorised Anilist user.
     * @return JSONObject with token to get user info.
     */
    public JSONObject getAniToken(String code){
        log.info("Get Anilist Token");
        String body = "{" +
                "\"grant_type\":\"authorization_code\","+

"\"client_id\":\""+environment.getProperty("bisonfun.anilist.client.id")+"\",\n"+

"\"client_secret\":\""+environment.getProperty("bisonfun.anilist.client.secret")+"
\","+

"\"redirect_uri\":\""+environment.getProperty("bisonfun.anilist.redirect_uri")+"\"
,"+
                "\"code\":\""+code+"\"}";

        HttpResponse<String> response = Unirest.post(AniList.TOKEN.link)
                .header("Content-Type", "application/json")
                .header("Accept", "application/json")
```

```
                .body(body).asString();
        if(response.getStatus() == 400){
            log.error("Something went wrong: {}", response.getBody());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return new JSONObject(response.getBody());
    }
    /**
     * Get Anilist user info by token.
     * @param token granted from Anilist by code from logged Anilist user.
     * @return JSONObject with Anilist user info (id, name).
     */
    public JSONObject getUserByToken(String token){

        log.info("Get Anilist User By Token");

        HttpResponse<String> userResponse = Unirest.post(AniList.GRAPHQL.link)
                .header("Authorization", "Bearer "+token)
                .queryString("query", AniListQuery.VIEWER.getQuery())
                .asString();
        if(userResponse.getStatus() == 400){
            log.error("Something went wrong: {}", userResponse.getBody());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return new JSONObject(userResponse.getBody()).getJSONObject("data");
    }
    /**
     * Get Anilist user lists by user's id.
     * @param userId identification number of Anilist user.
     * @param page page of the user's list
     * @param status status of the user's list
     * @return "Page" JSONObject which have page info(number, count, etc.) and
JSONArray of anime "Media" JSONObjects.
     * @throws TooManyAnimeRequestsException if Anilist.co have got more requests
than limit from app's account.
     */
    public JSONObject getUserMediaList(int userId, int page, MediaListStatus
status) throws TooManyAnimeRequestsException {
        log.info("Get User {} {} anime list. Page:{}", userId, status, page);

        String variables = "{\n" +
                "  \"page\": "+page+",\n" +
                "  \"userId\": "+userId+",\n" +
                "  \"status\": \""+status.getString()+"\"\n" +
                "}";

        //get anime list from AniList API
        HttpResponse<String> result = Unirest.post(AniList.GRAPHQL.link)
                .queryString("query", AniListQuery.USER_LIST.getQuery())
                .queryString("variables", variables)
                .asString();

        log.debug(result.getBody());

        if(result.getStatus() == 429){
            String secs = result.getHeaders().getFirst("Retry-After");
            int seconds = Integer.parseInt(secs);
            log.warn("delay in {} seconds", seconds);
            throw new TooManyAnimeRequestsException("Too many requests to
Anilist.co. will be available after " + seconds + "seconds", seconds);
        }else if(result.getStatus() == 400){
            log.error("Something went wrong: {}", result.getBody());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
```

```
        }
        return new
JSONObject(result.getBody()).getJSONObject("data").getJSONObject("Page");
    }
}
@Component
@Slf4j
public class TmdbApiResponse {
    final
    Environment environment;
    @Value("#{environment['bisonfun.tmdb.key']}")
    private String tmdbKey;

    @Autowired
    public TmdbApiResponse(Environment environment) {
        this.environment = environment;
    }

    /**
     * Get movie by id. Cacheable as "jsonMovie".
     * @param id identification number from TheMovieDB database.
     * @return JSONObject with info about movie (id, title, description, etc.).
     */
    @Cacheable("jsonMovie")
    public JSONObject getMovieById(int id){
        log.info("Get Movie: {}", id);
        //Get JSON of movie by id from TMDB
        HttpResponse<String> result = Unirest.get(TMDB.MOVIE.link)
                .routeParam("movie_id", Integer.toString(id))
                .queryString("api_key", tmdbKey)
                .queryString("language", "en-US")
                .asString();

        log.debug(result.getBody());

        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }else if(result.getStatus() == 404){
            log.error("Movie couldn't be found: "+id);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }

        return new JSONObject(result.getBody()).put("keywords",
getMovieKeywords(id));
    }
    /**
     * Get tv by id. Cacheable as "jsonShow".
     * @param id identification number from TheMovieDB database.
     * @return JSONObject with info about tv (id, title, description, etc.).
     */
    @Cacheable("jsonShow")
    public JSONObject getShowById(int id){
        log.info("Get TV: {}", id);
        // get JSON of tv-show by id from TMDB
        HttpResponse<String> result = Unirest.get(TMDB.TV.link)
                .routeParam("tv_id", Integer.toString(id))
                .queryString("api_key", tmdbKey)
                .queryString("language", "en-US")
                .asString();

        log.debug(result.getBody());
```

```
        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }else if(result.getStatus() == 404){
            log.error("TV Show couldn't be found: "+id);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }

        return new JSONObject(result.getBody())
                .put("keywords", getShowKeywords(id))
                .put("external_ids", getTvExternalId(id));
    }
    /**
     * Get movie keywords by movie id. Cacheable as "movieKeywords".
     * @param id movie identification number from TheMovieDB database.
     * @return JSONArray of keywords connected to movie with specific id.
     */
    @Cacheable("movieKeywords")
    public JSONArray getMovieKeywords(int id){
        log.info("Get Keywords of Movie: {}", id);
        HttpResponse<String> result = Unirest.get(TMDB.KEYWORDS_MOVIE.link)
                .routeParam("movie_id", Integer.toString(id))
                .queryString("api_key", tmdbKey)
                .asString();

        log.debug(result.getBody());

        if(result.getStatus() == 401 || result.getStatus() == 404){
            log.error("Something went wrong, got status {} in getMovieKeywords",
result.getStatus());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        JSONObject root = new JSONObject(result.getBody());
        return root.getJSONArray("keywords");
    }
    /**
     * Get tv keywords by tv id. Cacheable as "tvKeywords".
     * @param id tv identification number from TheMovieDB database.
     * @return JSONArray of keywords connected to tv with specific id.
     */
    @Cacheable("tvKeywords")
    public JSONArray getShowKeywords(int id){
        log.info("Get Keywords of TV Show: {}", id);
        HttpResponse<String> result = Unirest.get(TMDB.KEYWORDS_TV.link)
                .routeParam("tv_id", Integer.toString(id))
                .queryString("api_key", tmdbKey)
                .asString();

        log.debug(result.getBody());

        if(result.getStatus() == 401 || result.getStatus() == 404){
            log.error("Something went wrong, got status {} in getMovieKeywords",
result.getStatus());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        JSONObject root = new JSONObject(result.getBody());
        return root.getJSONArray("results");
    }
    /**
     * Get movie\tv list by search query.
     * @param query string query to find movie\tv.
```

```
     * @param contentType (Movie or TV).
     * @param page current page of list.
     * @param year year of release
     * @return JSONObject with page info(current page, etc.) and JSONArray with
movie\tv JSONObjects.
     */
    public JSONObject getTMDBList(String query, VideoContentType contentType, int
page, Integer year){
        log.info("Get list of {} by \"{}\" Page: {}", contentType, query, page);
        //get list of content from TMDB
        GetRequest request = Unirest.get(contentType == VideoContentType.MOVIE ?
TMDB.SEARCH_MOVIE.link : TMDB.SEARCH_TV.link)
                .queryString("api_key", tmdbKey)
                .queryString("language", "en-US")
                .queryString("query", query)
                .queryString("page", page)
                .queryString("include_adult", false);
        if (year != null){
            request.queryString("year", year);
        }
        HttpResponse<String> result = request.asString();

        log.debug(result.getBody());

        if(result.getStatus() == 404){
            log.error("Content couldn't be found: "+query);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }else if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        return new JSONObject(result.getBody());
    }

    public JSONObject getTMDBList(String query, VideoContentType contentType, int
page){
        return getTMDBList(query, contentType, page, null);
    }
    /**
     * Get list of movie trends. Cacheable as "movieTrends".
     * @return JSONObject with page info(current page, etc.) and JSONArray with
movie JSONObjects.
     * @throws NoAccessException if app can't access to TheMovieDB API.
     */
    @Cacheable("movieTrends")
    public JSONObject getMovieTrends() throws NoAccessException {
        log.info("Get movie trends");
        HttpResponse<String> result;
        try {
            result = Unirest.get(TMDB.TRENDS_MOVIE.link)
                    .queryString("api_key", tmdbKey)
                    .asString();
        }catch (Exception e){
            log.error("Caught exception {}", e.getMessage());
            throw new NoAccessException("Can't access to TheMovieDB");
        }
        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new NoAccessException("Can't access to TheMovieDB (wrong API
key)");
        }
        return new JSONObject(result.getBody());
```

```
    }
    /**
     * Get list of tv trends. Cacheable as "tvTrends".
     * @return JSONObject with page info(current page, etc.) and JSONArray with tv
JSONObjects.
     * @throws NoAccessException if app can't access to TheMovieDB API.
     */
    @Cacheable("tvTrends")
    public JSONObject getTvTrends() throws NoAccessException {
        log.info("Get tv trends");
        HttpResponse<String> result;
        try {
            result = Unirest.get(TMDB.TRENDS_TV.link)
                    .queryString("api_key", tmdbKey)
                    .asString();
        }catch (Exception e){
            log.error("Caught exception {}", e.getMessage());
            throw new NoAccessException("Can't access to TheMovieDB");
        }
        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new NoAccessException("Can't access to TheMovieDB (wrong API
key)");
        }
        return new JSONObject(result.getBody());
    }
    /**
     * Get object with external ids to current tv. Cacheable as "tvExternalIds".
     * @return JSONObject with external ids ( tmdb id, imdb_id, freebase_mid,
freebase_id, tvdb_id, tvrage_id, wikidata_id, facebook_id, instagram_id,
twitter_id ).
     */
    @SneakyThrows
    @Cacheable("tvExternalIds")
    public JSONObject getTvExternalId(int id){
        log.info("Get tv external ids");
        HttpResponse<String> result;
        try {
            result = Unirest.get(TMDB.EXTERNAL_ID_TV.link)
                    .routeParam("tv_id", String.valueOf(id))
                    .queryString("api_key", tmdbKey)
                    .asString();
        }catch (Exception e){
            log.error("Caught exception {}", e.getMessage());
            throw new NoAccessException("Can't access to TheMovieDB");
        }
        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new NoAccessException("Can't access to TheMovieDB (wrong API
key)");
        }
        return new JSONObject(result.getBody());
    }
    /**
     * Get recommendations based on movie. Cacheable as "movieRec"
     * @param id movie identification number from TheMovieDB API.
     * @return JSONObject with page info (current page, etc.) and JSONArray with
movie JSONObject.
     * @throws NoAccessException if app can't access to TheMovieDB API.
     */
    @Cacheable("movieRec")
    public JSONObject getMovieRecommendations(int id) throws NoAccessException {
        log.info("Get movie recommendations by {} movie", id);
```

```
        HttpResponse<String> result = Unirest.get(TMDB.RECOMMENDATIONS_MOVIE.link)
                .routeParam("movie_id", String.valueOf(id))
                .queryString("api_key", tmdbKey)
                .asString();

        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new NoAccessException("Can't access to TheMovieDB (wrong API
key)");
        }
        return new JSONObject(result.getBody());
    }
    /**
     * Get recommendations based on tv show. Cacheable as "tvRec"
     * @param id tv show identification number from TheMovieDB API.
     * @return JSONObject with page info (current page, etc.) and JSONArray with
tv JSONObject.
     * @throws NoAccessException if app can't access to TheMovieDB API.
     */
    @Cacheable("tvRec")
    public JSONObject getTvRecommendations(int id) throws NoAccessException {
        log.info("Get tv recommendations by {} tv show", id);

        HttpResponse<String> result = Unirest.get(TMDB.RECOMMENDATIONS_TV.link)
                .routeParam("tv_id", String.valueOf(id))
                .queryString("api_key", tmdbKey)
                .asString();

        if(result.getStatus() == 401){
            log.error("Invalid API key (TMDB API key)");
            throw new NoAccessException("Can't access to TheMovieDB (wrong API
key)");
        }
        return new JSONObject(result.getBody());
    }
}
@Controller
@Slf4j
public class AnimeController{

    private final UserService userService;
    private final AnimeService animeService;
    private final UserAnimeService userAnimeService;
    private final AniListClient aniListClient;

    @Autowired
    public AnimeController(UserService userService, AnimeService animeService,
UserAnimeService userAnimeService, AniListClient aniListClient) {
        this.userService = userService;
        this.animeService = animeService;
        this.userAnimeService = userAnimeService;
        this.aniListClient = aniListClient;
    }

    @GetMapping("/anime/{id}")
    public String animePage(@PathVariable int id, Model model, Principal
principal) throws JSONException, TooManyAnimeRequestsException {
        log.info("User {} get Anime {}", principal == null ? "Unknown" :
principal.getName(), id);
        AniAnime anime;
        try {
            anime = aniListClient.parseById(id);
```

```
        } catch (ContentNotFoundException e) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }
        UserAnime userAnime;
        if (principal != null) {
            User user = userService.getUserByUsername(principal.getName());
            userAnime = userAnimeService.getUserAnimeById(user.getId(),
anime.getId());
        } else {
            userAnime = new UserAnime();
        }

        model.addAttribute("anime", anime);
        model.addAttribute("actions", asList(VideoConsumingStatus.values()));
        model.addAttribute("userAnime", userAnime);


        return "content_anime";
    }

    @PostMapping("/anime/{animeId}")
    public String updateAnimeList(UserAnime userAnime, @PathVariable int animeId,
Principal principal) throws ContentNotFoundException,
TooManyAnimeRequestsException {
        if (principal == null) {//if user isn't logged in
            String loginLink = "/login";
            return "redirect:"+loginLink;
        }
        if (userAnime.getStatus() == null) {
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        log.info("Update Anime {} in User {} List", animeId, principal.getName());
        Anime dbAnime = animeService.updateAnime(animeId);

        User user = userService.getUserByUsername(principal.getName());
        userAnimeService.createUserAnime(userAnime, user, dbAnime);

        String redirectLink = "/anime/"+animeId;
        return "redirect:"+redirectLink;
    }

    @DeleteMapping("/anime/{animeId}")
    public String deleteAnimeFromList(@PathVariable int animeId, Principal
principal){
        User user = userService.getUserByUsername(principal.getName());
        log.info("Delete Anime {} from User {} List", animeId,
user.getUsername());
        userAnimeService.deleteAnimeFromUserList(new UserAnimeKey(user.getId(),
animeId));
        String redirectLink = "/anime/"+animeId;
        return "redirect:"+redirectLink;
    }
}
@Controller
public class HomeController {

    final
    AniListClient aniListClient;
    final
    TmdbClient tmdbClient;
    final
    ImageService imageService;
```

```
    @Autowired
    public HomeController(AniListClient aniListClient, TmdbClient tmdbClient,
ImageService imageService) {
        this.aniListClient = aniListClient;
        this.tmdbClient = tmdbClient;
        this.imageService = imageService;
    }

    @GetMapping("/")
    public String home(Model model) throws TooManyAnimeRequestsException {
        Random random = new Random();

        List<String> backgrounds = imageService.getMainPageBackground();

        List<VideoEntertainment> animeTrends = aniListClient.parseAnimeTrends();
        List<VideoEntertainment> movieTrends = tmdbClient.parseMovieTrends();
        List<VideoEntertainment> tvTrends = tmdbClient.parseTVTrends();

        model.addAttribute("animeTrends", animeTrends);
        model.addAttribute("movieTrends", movieTrends);
        model.addAttribute("tvTrends", tvTrends);
        model.addAttribute("background",
backgrounds.get(random.nextInt(backgrounds.size())));

        return "main";
    }
}
<!DOCTYPE html>
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=1024" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <title>BisonFun</title>
    <!-- Favicon-->
    <link rel="icon" type="image/x-icon" th:href="@{~/images/logo.png}" />
    <!-- Bootstrap icons-->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-
icons.css" rel="stylesheet"
        type="text/css" />
    <!-- Google fonts-->
    <link
href="https://fonts.googleapis.com/css?family=Lato:300,400,700,300italic,400italic
,700italic" rel="stylesheet"
        type="text/css" />
    <!-- Core theme CSS (includes Bootstrap)-->
    <link th:href="@{~/css/mainstyles.css}" rel="stylesheet" />
    <link th:href="@{~/css/maincustoms.css}" rel="stylesheet" />
    <link th:href="@{~/css/usercustoms.css}" rel="stylesheet" />
    <script th:src="@{~/js/iconscript.js}"></script>
    <script th:src="@{~/js/tabscript.js}"></script>
</head>

<body onload="seasonIcon(); tabs()">
    <!-- Navigation-->
    <nav class="navbar navbar-light bg-light static-top">
        <div class="container">
            <a class="navbar-brand" href="/" ><img src="" id="icon"> BisonFun</a>
```

```
            <!--Authenticated buttons-->
            <div sec:authorize="isAnonymous()">
                <a class="btn btn-lg btn-outline-bisonfun"
th:href="@{~/login}">Login</a>
                <a class="btn btn-lg btn-bisonfun" th:href="@{~/register}">Sign
Up</a>
            </div>
            <div sec:authorize="isAuthenticated()" class="btn-group">
                <a th:href="@{~/cabinet}" sec:authentication="name" class="btn
btn-bisonfun">Username</a>
                <button type="button" class="btn btn-bisonfun dropdown-toggle
dropdown-toggle-split"
                        data-bs-toggle="dropdown" aria-expanded="false">
                    <span class="visually-hidden"></span>
                </button>
                <ul class="dropdown-menu">
                    <li><a class="dropdown-item" th:href="@{~/cabinet}">
                        <i class="bi bi-person-circle"></i>
                        User page</a></li>
                    <li><a class="dropdown-item" th:href="@{~/wtw}">
                        <i class="bi bi-collection-play"></i>
                        What To Watch</a></li>
                    <li><form action="/logout" method="post">
                        <button type="submit" class="dropdown-item">
                            <i class="bi bi-box-arrow-left"></i>
                            Logout</button>
                    </form></li>
                </ul>
            </div>
        </div>
    </nav>
    <!-- Masthead-->
    <header class="masthead" th:style="'background-image:url(' + ${background} +
');background-size: cover;'" id="background-header">
        <div class="container position-relative">
            <div class="row justify-content-center">
                <div class="col-xl-6">
                    <div class="text-center text-white">
                        <!-- Page heading-->
                        <h1 class="mb-5">Place for movies, anime and tv shows</h1>
                        <form action="/search" method="get" class="form-
subscribe">
                            <!-- Search input-->
                            <div class="row">
                                <div class="col input-group">
                                    <input class="form-control" name="query"
id="query" type="text"
                                           placeholder="Search..." required />
                                </div>
                                <div class="col-auto">
                                    <select id="types" class="form-select form-
select-lg types" name="type">
                                        <option value="movie">Movie</option>
                                        <option value="tv">TV Show</option>
                                        <option value="anime">Anime</option>
                                    </select>
                                </div>
                                <div class="col-auto"><button class="btn btn-
bisonfun btn-lg" id="searchButton"
                                        type="submit">Find</button></div>
                            </div>
                        </form>
```

```
                    </div>
                </div>
            </div>
        </div>
    </header>

    <div th:if="${(animeTrends.size()+movieTrends.size()+tvTrends.size()) > 0}"
class="container" style="margin-top: 5%; margin-bottom: 5%;">
        <h3>
            Trending
        </h3>
        <div class="tabs" id="tabs">
            <div th:if="${animeTrends.size() > 0}" class="tab">
                <input type="radio" name="css-tabs" id="anime" value="#0d6efd"
class="tab-switch">
                <label for="anime" class="tab-label">Anime</label>
                <div class="tab-content overflow-auto d-flex flex-row flex-
nowrap">
                    <div th:each="anime : ${animeTrends}" class="alert alert-
primary">
                        <img th:src="${anime.poster}" alt="">
                        <div class="title">
                            <a th:href="@{/anime/{id}(id=${anime.id})}"
                               th:text="${anime.title}" class="alert-
link">Title</a>
                        </div>
                        <div class="type">
                            <span th:text="${anime.type.string}">Type</span>
                            <span th:text="${anime.getReleaseYear() < 0 ?
'Unknown' : anime.getReleaseYear()}"></span>
                        </div>
                    </div>
                </div>
            </div>
            <div th:if="${movieTrends.size() > 0}" class="tab">
                <input type="radio" name="css-tabs" checked  id="movie"
value="#dc3545" class="tab-switch">
                <label for="movie" class="tab-label">Movie</label>
                <div class="tab-content overflow-auto d-flex flex-row flex-
nowrap">
                    <div th:each="movie : ${movieTrends}" class="alert alert-
danger">
                        <img th:src="${movie.poster}" alt="">
                        <div class="title">
                            <a th:href="@{/movie/{id}(id=${movie.id})}"
                               th:text="${movie.title}" class="alert-
link">Title</a>
                        </div>
                        <div class="type">
                            <span>Movie</span>
                            <span th:text="${movie.getReleaseYear() < 0 ?
'Unknown' : movie.getReleaseYear()}"></span>
                        </div>
                    </div>
                </div>
            </div>
            <div th:if="${tvTrends.size() > 0}" class="tab">
                <input type="radio" name="css-tabs" id="tv" value="#198754"
class="tab-switch">
                <label for="tv" class="tab-label">Tv</label>
                <div class="tab-content overflow-auto d-flex flex-row flex-
nowrap">
                    <div th:each="tv : ${tvTrends}" class="alert alert-success">
```

```
                        <img th:src="${tv.poster}" alt="">
                        <div class="title">
                            <a th:href="@{/tv/{id}(id=${tv.id})}"
                               th:text="${tv.title}" class="alert-link">Title</a>
                        </div>
                        <div class="type">
                            <span>TV</span>
                            <span th:text="${tv.getReleaseYear() < 0 ? 'Unknown' :
tv.getReleaseYear()}"></span>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Bootstrap core JS-->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
></script>
    <!-- Core theme JS-->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
</body>

</html>
```