Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ЗАСТОСУВАННЯ ДЛЯ ВІДСТЕЖЕННЯ ФІЛЬМІВ ДЛЯ ПЕРЕГЛЯДУ

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи

старший викладач Борозенний С.О.

_____

(підпис)

«____» _____ 2022 р.

Виконала студентка 4-го курсу

Білорус К.С.

«____» _____ 2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики,
канд. фіз-мат. наук, доц.
_____ Гороховський С.С.
(підпис)
«____»_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студентці Білорус Катерині Сергіївні факультету інформатики 4-го курсу
ТЕМА: Розробка клієнт-серверного застосування для відстеження фільмів для перегляду

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Анотація
Вступ
1 Аналіз предметної області
2 Проектування системи
3 Опис реалізації
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі «___» _____ 2022 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: **Розробка клієнт-серверного застосування для відстеження фільмів для перегляду**

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	20.10.2021	
2.	Огляд літератури за темою роботи.	24.12.2021	
3.	Проведення анкетування потенційних користувачів.	10.01.2022	
4.	Аналіз проведеного опитування, виявлення потреб користувачів.	20.01.2022	
5.	Дослідження аналогічних систем.	01.02.2022	
6.	Опис функціональних вимог до системи.	10.02.2022	
7.	Аналіз отриманих результатів з керівником.	20.02.2022	
8.	Розробка ER-моделі.	06.03.2022	
9.	Розробка реляційної моделі та створення бази даних в системі PostgreSQL.	21.03.2022	
10.	Розробка UI/UX.	10.04.2022	
11.	Створення проекту	25.04.2022	
12.	Реалізація базового функціоналу системи	16.05.2022	
13.	Подання роботи на кафедру для перевірки на плагіат.	20.05.2022	
14.	Захист курсової роботи.	травень 2022	

Студентка **Білорус К.С.**

Керівник **Борозенний С.О.**

«_____»_____

Зміст

Анотація

У цій курсовій роботі проводиться огляд існуючих сервісів для каталогізації, збір потреб цільової аудиторії. На основі отриманих при аналізі даних створюються функціональні вимоги до застосунку.

Після цього здійснюється проектування ER моделі та реляційної моделі застосунку, порівнюються різні підходи до створення архітектури застосунку та огляд обраних для його реалізації інструментів таких, як Spring Boot, Hibernate, React.

В результаті роботи отримано застосунок, який дозволяє переглядати бібліотеку фільмів та серіалів, створювати та експортувати списки для перегляду.

ВСТУП

Актуальність теми

Темп життя з кожним роком стає тільки швидшим. Кількість інформації, що нас оточує, збільшується в десятки та сотні разів. Важливою складовою нашого життя стало планування та організація всього: житла, списків покупок, розкладу прибирання та навіть власного дозвілля. Під час вибору способу проведення вільного часу, ми орієнтуємось не тільки на саму активність, але і на розташування, час затрачений на дорогу, зручність інфрастуктури. Програми, які виконують функції планувальників, є одними з найзатребуваніших в магазинах додатків.

Наразі одним з найдоступнішим і найпопулярнішим видом дозвілля є перегляд фільмів та серіалів, як в кінотеатрах, так і в домашніх умовах. Постійно постає питання, що подивитись, які фільми ви вже дивились, а які ж фільми можна порадити друзям. Вирішити цю проблему допоможе додаток, в якому ви зможете отримувати новини про прем’єри, позначати переглянуті фільми та фільми, які ви хочете переглянути в майбутньому.

Під час вибору фільму можна буде подивитись відгуки інших користувачів та їх оцінку. А також поділитися своїм списком улюблених фільмів з іншими людьми.

Мета і завдання дослідження

Аналіз сучасного ринку та потреб корстувачів стримінгових сервісів. Розробка схеми бази даних для сервісу відстеження фільмів. Розробка бібліотеки фільмів та списків для перегляду. Розробка системи імпорту та експорту списків. Основним завданням сервісу є спрощення вибору фільмів

для перегляду, відстеження переглянутих фільмів, обміну відгуками з іншими користувачами та спільного перегляду фільмів.

Об’єкт дослідження

Розробка клієнт-серверного веб-додатку на основі React, Spring Boot та PostgreSQL.

Методи дослідження

Проведення опитування цільової аудиторії. Проектування сервісу на основі отриманих даних після опитування. Розробка серверної частини застосунку мовою Java за домогою фреймворку Spring Boot та клієнтської частини мовою JavaScript на основі бібліотеки React.

Практичне значення одержаних результатів

Розроблений веб-застосунок може полегшити організацію бібліотеки фільмів для користувачів, збільшити потік користувачів для стрімінгових сервісів та може бути використаним у якості рекламного майданчику для монетизації.

Джерела дослідження

Під час розробки було досліджено різні підходи до створення архітектури клієнт-сервного застусунку, документацію фреймворку Spring Boot, інструменту Hibernate та бібліотеки React. Також були проаналізовані сучасні аналоги застосунків-планувальників у різних сферах, виявлено їх позитивні сторони та недоліки.

РОЗДІЛ 1. Аналіз предметної області

1.1. Аналіз сучасного стану питання та обгрунтування теми

Наразі існує багато стрімінгових сервіс, які конкурують між собою. Через це трансляція одного фільму чи серіалу відбувається тільки на одному з них. Таким чином одна людина має декілька різних підписок, на кожній з який є її неузгоджена з іншими бібліотека фільмів та серіалів. Зі збільшенням кількості переглянутих стрічок організація цієї інформації стає дедалі складнішою, а питання від друзів, які порадити фільми, кожен раз вводить в замішання, а всі рекомендовані іншими людьми серіали назавжди губляться в пам'яті та повідомленнях.

Кожен сервіс пропонує автоматизоване відслідковування переглянутих стрічок на їньому сервісі і рекомендації, які базуються на ваших вподобаннях. Ці рекомендації обмежені рамками одного сервісу, через що не є повними.

Питання, які постають перед кінолюбителями можна поділити на такі:

а) Який фільм обрати для перегляду?

Це призводить до декількогодинного пошуку в інтернеті, які є нові популярні фільми, на якому сервісі їх знайти або в якому кінотеатрі їх зараз траснлюють. Далі виникає питання, чи варто його взагалі дивитись, які в нього відгуки та рейтинги.

б) А чи дивився я цей фільм?

Після того, як та сама стрічка для двогодинного задоволення була обрана, постає питання, а можливо я вже бачив цей фільм? Доводиться перечитувати сюжет, псуючи враження від майбутнього перегляди. А іноді після години перегляду ви згадуєте, чим закінчить кінострічка, бо ви вже її дивились.

в) Які фільми порадити друзям?

Це питання є найскладнішим для будь-кого, хто хоча б іноді дивиться кіно. В голові одразу виникає перелік усіх переглянутих фільмів, але всі воно точно не з вашого список для рекомендацій.

Сучасні сервіси пропонують вирішення лише деяких з цих питань окремо через що вибір фільму для перегляду затягується на години або взагалі відкладається на невизначений термін.

1.2. Опитування потенційних користувачів

Для виокремлення вимог до застосунку та підтвердження актуальності проблеми було проведено опитування серед 34 студентів 17-22 років (рис.1).

1. Як часто ви дивитесь фільми або серіали?
    а) Кожен день
    б) 1-2 рази на тиждень
    в) 2-3 на місяць
    г) Декілька разів на рік

2. На що ви орієнтуєтесь під час вибору фільму або серіалу для перегляду?

3. Чи маєте ви список фільмів або серіалів, які хотіли б переглянути?
    а) так
    б) ні

4. Якщо так, в якому вигляді ви зберігаєте цей список?

5. Чи хотіли б ви користуватись застосунком для відслідковування переглянутих фільмів та мати рекомендації щодо фільмів, які варто переглянути?

Рисунок 1. Опитування цільової аудиторії

За результатами опитування було з’ясовано, що більшість опитаних, а саме 53%, дивляться фільми 1-2 рази на тиждень. Найпопулярнішим критерієм для вибору фільму було названо рейтинг IMDB, відгуки знайомих, відгуки критиків, номінації на міжнародні премії. Лише 26% серед опитаних мають список з рекомендаціями. Серед них 8 з 11 зберігають цей список у нотатках

на телефоні. З усіх опитаних студентів 93% хотіли б користуватися сервісом, який полегшував би організацію бібліотеки фільмів та серіалів.

1.3. Аналіз існуючих аналогів інформаційних систем

На сьогодні не існує подібних систем для фільмів та серіалів. Я розглянула сервіси, які або надають лише частковий функціонал, або використовуються для відстеження в інших предметних областях.

IMDb – велика бібліотека фільмів та серіалів, має понад 10 мільйонів найменувань. Включає свою рейтингову систему та систему відгуків.

Переваги:

- Велика бібліотека фільмів та серіалів;
- Власна рейтингова система;
- Можливість перегляду без реєстрації;
- Наявність можливості додавати фільму до списку для перегляду;
- Власна система рекомендацій на основі вподобань користувача;
- Користувачі можуть лишати власні відгуки до фільмів та ставити їм свою оцінку;
- Аутентифікація за допомогою сторонніх сервісів;
- Наявність власного доступного для зовнішнього використання API;
- Є мобільна версія на різних платформах.

Недоліки:

- Переповнений та складний інтерфейс;
- Погана система фільтрації бібліотеки, недостатя кількість параметрів;

- Відсутність системи для відслідковування переглянутих серій для серіалів.

MyShows – бібліотека серіалів з власним блогом, рейтингом серіалів та відслідковування статистики кожного користувача.

Переваги:

- Велика бібліотека серіалів;
- Власний рейтинг серіалів;
- Власний блог та соціальна мережа всередині сервісу;
- Відстеження переглянутих серій;
- Можливість додавати коментарі до кожної серії окремо;
- Календар виходу нових серій серіалів, які дивиться користувач;
- Зручна система позначення серіалів: дивлюсь, буду дивитись, припинив перегляд, не дивлюсь;
- Статистика переглянутих користувачем серій та годин серіалів, активність користувача по днях, система нагород за перегляди;
- Наявність мобільної версії на різних платформах;
- Аутентифікація за домогою сторонніх сервісів.

Недоліки:

- Відсутність можливості створювати власні списки;
- Відсутність бібліотеки фільмів;
- Наявність великої кількості реклами;

- Відсутність системи рекомендаці.

Goodreads - американський сервіс каталогізації від Amazon, який дозволяє шукати в його базі даних книг, анотації, цитати та огляди інших користувачів.

Переваги:

- Велика бібліотека книжок;
- Власна рейтингова система:
- Система додавання книг та створення списків;
- Система рецензій та їх відслідковування іншими користувачами;
- Система рекомендацій на основі вподобань користувача;
- Зручна система пошуку по багатьом критеріям;
- Система «друзів» зі стрічкою активністю інших користувачів, їх прочитаних книг та рецензій;
- Зручна можливість додавати книжки до списку для прочитання;
- Наявність можливість слідкувати за авторами та виходом їх нових творів;
- Наявність мобільного застосунку на різних платформах;
- Аутентифікація за допомогою сторонніх сервісів.

Недоліки:

- Складний інтерфейс;
- Незрозуміла система наповнення новинної стрічки;

- Складна система пошуку інших користувачів, надсилання запиту на дружбу та нотифікацій про отримання відповідних запитів.

Під час дослідження сучасного ринку сервісів каталогізації не було знайдено жодного додатку, який відповідав би запитам цільової аудиторії, які були сформовані після опитування. Основним функціоналом, яким має бути наділений додаток було виявлено:

- Наявність бібліотеки фільмів та серіалів;
- Можливість створювати власні списки;
- Відстеження переглянутих фільмів та серіалів;
- Рейтингова система та система відгуків;
- Система пошуку та рекомендацій.

РОЗДІЛ 2. Проектування системи

2.1. Опис користувачів

Користувач системи

Користувачем системи є людина будь-якого віку, яка полюбляє дивитись фільми та серіали. Для зручної роботи з сервісом йому потрібні два режими: авторизований та неавторизований. В авторизованому користувач хоче бачити, редагувати та зберігати свої дані. Незалежно від режиму користувач хоче бачити бібліотеку фільмів.

Адміністратор системи

Адміністратором системи є людина, яка керує наповненням бібліотеки фільмів та серіалів, створює та оновлює списки популярних стрічок.

2.2. Технічні вимоги користувача до функціональності системи

Користувач системи хоче:

- переглядати бібліотеку фільмів та серіалів;
- переглядати списки популярних фільмів та серіалів;
- шукати фільм та серіалі в бібліотеці по назві;
- переглядати інформацію про фільм або серіал;
- додавати, редагувати та видаляти власні списки фільмів та серіалів;
- додавати та видаляти фільми та серіали зі списку для перегляду;
- отримувати рекомендації щодо фільмів для перегляду на основі власних вподобань та раніше переглянутих стрічок;
- залишати власні відгуки про фільми та серіали;

- переглядати фідгуки інших користувачів;
- помічати переглянуті фільми та серії серіалів;
- експортувати списки у вигляді PDF документів;
- експортувати списки у вигляді посилання;
- додавати списки інших користувачів з посилання;
- реєструвати профіль;
- змінювати дані свого профілю;
- переглядати профіль інших користувачів.

Адміністратор системи хоче:

- переглядати, редагувати та видаляти фільми та серіалі з системи;
- створювати, редагувати та видаляти розділи фільмів та серіалів в системі.

2.3. Специфікація вимог до даних

Фільми

Фільми зберігаються як окремі об’єкти в системі. Атрибутами до цих об’єктів мають бути назва, рік випуску, жанр, країна виробництва та опис фільмів. Також фільм має посилання на списки, в які його було додано. Також мають бути сформовані різні види списків такі, як популярні, нові і т.д., які буду посилатися на об’єкти фільмів.

Серіали

Серіали зберігаються як окремі об’єкти в системі. Атрибутами до цих об’єктів мають бути назва, опис, рік випуску, жанр, країна виробництва та

кількість сезонів серіалу. Також мають зберігатися посилання на сезони та серії в них. Також має зберігатися відношення користувач – серіал, в якому буде зазначено статус серіалу («дивлюсь», «буду дивитись», «припинив дивитись» та «не дивлюсь»). Також мають бути сформовані різні види списків такі, як популярні, нові і т.д., які буду посилатися на об’єкти серіалів.

Сезони серіалів

Сезони серіалів є окремими об’єктами, є частиною серіалу. Зберігається інформація про номер сезону та серії, що входять до сезону.

Серії серіалів

Кожна серія є окремим об’єктом. Зберігається назва серії, порядковий номер в сезоні, довжина серії. Також для кожної серії має зберігатися відношення користувач – серія. В ньому зберігається статус перегляду серії (переглянуто або не переглянуто), відгук на серію, якщо він є.

Списки фільмів та серіалів

Списки є окремими об’єктами в системі, які зв’язані з користувачем. Список містить інформацію про свою назву та опис, посилається на існуючі об’єкти фільмів або серіалів.

Список для перегляду

Список для перегляду – це частковий випадок списку серіалів або фільмів, який автоматично створюється пустим під час реєстрації користувача.

Профіль користувача

Профіль користувача відповідає за особисті дані, які користувач вносить під час реєстрації, а саме: ім’я та пошта. Користувач пов’язаний зі списками фільмів та серіалів, який він створив.

Облікові дані користувача

Облікові дані користувача зберігаються окремі від профілю користувача і відповідають виключно за ідентифікацію користувача та його авторизацію в системі. Складаються з атрибуту, який однозначно ідентифікує користувача, а саме пошти та паролю.

2.4. Архітектура системи

Для вибору архітектури застосунку я ознайомилась з найбільш вживаними архітектурними шаблонами, які застосовуються в розробці програмного забезпечення.

Багатошаровий шаблон

Багатошаровий шаблон (рис.2) полягає в розділенні системи на декілька послідовних незалежних рівнів. Кожен рівень надає сервіси рівню, який знаходиться вище за нього.

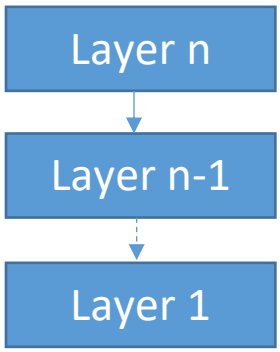


Рисунок 2. Багатошаровий шаблон

Переваги:

- Рівні ізольовані один від одного, верхній рівень викликає нижній, не знаючи про його реалізацію, розробка ведеться незалежно для кожного рівня;

- Тестування також проводиться незалежно для кожного шару, що значно пришвидшує та полегшує тестування;

Недоліки:

- Складність та ресурсозатратність при масштабуванні;
- Складність для підтримки цілої системи, зміна в одному прошарку впливає на інші;
- Неможливе розбиття на паралельне виконання.

Клієнт-серверний шаблон

Клієнт-серверний шаблон полягає у взаємодії між клієнтом та сервером за допомогою запитів від клієтнської частини та відповідей від серверної. Сервер постійно слухає клієнта, очікуючи на його запит. У той же час клієнт готовий отримати тільки відповідь на свої запити.

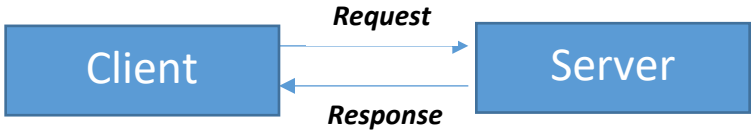


Рисунок 3. Клієнт-серверний шаблон

Переваги:

- Дані централізовано зберігаються в одному місці, що полегшує підтримку їх цілісності;
- Легкість масштабування. До одного серверу може звертатися багато клієнтів, один клієнт може звертатися до багатьох серверів;
- Легкість в отриманні потрібних даних для клієнта.

Недоліки:

- Валідація даних на обох сторонах. На стороні клієнта валідація відбувається для захисту від введення неправильних даних кінцевим користувачем. На стороні серверу валідація потрібна для перевірки даних, які могли зазнати змін під час передачі від клієнту;
- Дані, які отримує клієнт, безпосередньо залежать від доступу до серверу на момент запиту;
- Потреба в розподілі навантаження на сервери, обмежений потік трафік, який може обробляти один сервер, вразливість до атак, пов'язаних з перевантаженням серверу.

Шаблон «господар-робочий»

Шаблон «господар-робочий» полягає в розділенні ролей на дві групи: господар та робочі. Компонент-господар розподіляє задачі між однаковими робочими компонентами, збирає та ороблює отримані дані та обраховує фінальний результат.

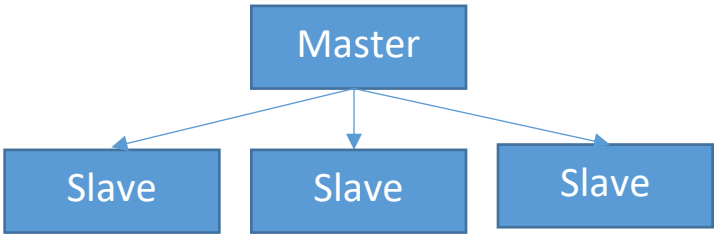


Рисунок 4. Шаблон господар-робочий

Переваги:

- Високодоступність. Дані з компоненти господаря дублюються у велику кількість компонентів робочих. Таким чином при проблемі з доступом до однієї з компонентів, її завдання може виконати інша.

Також при перекритті доступу до господарської компоненти, її роль на себе може взяти одна з робочих;

- Швидкість обробки даних, паралельні обчислення, дані розподіляються між багатьма компонентами;

Недоліки:

- Зміни в даних господарської компоненти. Дані мають оновитися на всіх робочих компонентах;
- Затратність підтримки великої кількості серверів.

Мікросервісний шаблон

При використанні цього підходу система розбивається на маленькі сервіси, кожен з яких відповідає за свою функцію і спілкується з іншими за допомогою простих протоколів передачі даних.

Переваги:

- Незалежність кожного мікросервісу. Розгортка та розробка кожного сервісу проходить незалежно від інших;
- Легкість масштабування;
- Підтримка доступності за допомогою перекриття доступу лише до одного функціоналу. Інші сервіси продовжують безперервно виконувати свою роботу.
- Кожному сервісу відповідає невелика частина проекту, що полегшує розуміння коду для розробників.

Недоліки:

- Складність розробки та комунікації між сервісами;
- Складність в пошуку помилок;

- Велика кількість API, від яких залежить робота інших сервісів через комунікацію з іншими сервісами;
- Висока вартість технічної підтримки великої кількості сервісів.

Також до найвживаніших підходів до розробки архітектури можна віднести такі шаблони:

- Шаблон посередника. При створенні великих систем, які складаються з великою кількості модулів, для полегшення взаємодії між ними створюється посередник.
- Шаблон «рівний-рівному». В такому архітектурному шаблоні кожен компонент виступає одночасно і клієнтом, і сервером для інших. Компонент може звертатись до інших компонентів як клієнт і сам надавати відповіді на запити як сервер.

Проаналізувавши використання кожного з зазначених шаблонів, для розробки свого сервісу я обрала клієнт-серверну архітектуру. Це найвживаніший підхід для розробки веб-додатку. Клієнт спілкується з сервером через API і відображає отриману інформацію кінцевому користувачу у вигляді веб-сторінки. Таким чином досягається незалежність розробки клієнта від реалізації серверу, легкість у подальшому розвитку та зміні реалізації API, наприклад, розгортки серверу у хмарі.

Для імплементації серверу я використала шаблону «сервіс-репозиторій», який лежить в основі розробки за допомогою Spring Boot. Найбільшою перевагою такого підходу є масштабованість сервісу та легкість у його підтримці.

Найнижчою компонентою шаблону є модель. Модель представляє реальний об’єкт та його характеристики, які будуть використанні для обміну

даними між всіма іншими компонентами. У розробці проекту з відслідковування перегляду фільмів цими моделями є: користувач, фільм, серіал, сезон, серія, список.

Іншим компонентом архітектурного шаблону «сервіс-репозиторій» є репозиторій. Це найнижчий рівень системи, які безпосередньо спілкується з базою даних. Надсилаючи запит до бази даних, репозиторій перетворює його на модель, яку передає сервісу.

На рівні сервісів зосереджена вся логіка серверної частини застосунку. Ця компонента відповідає за обробку отриманих від репозиторіїв даних та повертає фінальний результат контролеру.

Контролер є посередником між сервером та клієнтом. Він прослуховує запити, які надсилає сервер, розподіляє їх між сервісами та повертає результат у вигляді відповіді серверу.

2.5. Логічне проектування

Для проектування бази даних сервісу було створено ER діаграму (додаток А). На ній відображено всі сутності та відношення між ними.

Проектування користувача було вирішено розділити на дві сутності: профіль користувача та облікові дані. Це зумовлено тим, що облікові дані потрібні лише для ідентифікації та авторизації користувача. Облікові дані не входять до предметної області, а також можуть оброблятись і зберігатись окремо від основного застосунку. Для однозначної ідентифікації користувача було обрано його електронну пошту.

Сам профіль користувача складається з його облікового номеру в системі, імені та ролі в системі (користувач або адміністратор).

Користувач може створювати необмежену кількість списків фільмів та серіалів. Списки містять в собі ідентифікаційний номер, назву та необов'язковий атрибут опису. Назва має складатися лише з літер та цифр.

У свою чергу списки можуть складатися з фільмів та серіалів. Фільми зберігають в собі інформацію про свій ідентифікаційний номер, назву, опис, рейтинг та рік виходу в прокат. Сутність серіалу містить ідентичні атрибути.

Серіал складається з багатьох сезонів. Кожен сезон містить ідентифікаційний номер в системі, назву та його порядковий номер.

Кожен сезон складається з багатьох серій. Сутність серії описується ідентифікайним номером, назвою, порядковим номером та датою виходу.

Для відображення відношення між користувачем, рецензією та фільмами, серіалами, серіями було виділено окрему підсхему. Попри те, що всі рецензії мають однакові атрибути, було виділено три окремих сутності. Під час проектування цієї частини діаграми було розглянути три варіанти:

1) Створення однієї сутності «рецензія» для фільмів, серіалів та серій з 3 зовнішними різними ключами

Переваги:

- Одна таблиця в базі даних;
- Простота зображення в схемі;

Недоліки:

- Зберігання в сутності 3 зовнішніх ключів окремими атрибутами, 2 з яких завжди будуть пустими. Ускладнює індексований пошук та займає зайву пам’ять;

- Відсутність можливості зміни структури сутності лише для одного з підвидів.

2) Створення однієї сутності «рецензія» зі спільним зовнішнім ключем для всіх видів та окремим полем для визначення типу рецензії

Переваги:

- Використовується мало пам’яті для збереження;

Недоліки:

- Суттємо ускладнює пошук, в кожному запиті має враховуватися тип рецензії, по якому йде пошук;
- Складна для розуміння структура.

3) Створити окремі 3 сутності для кожного з типу рецензій

Переваги:

- Зменшення часових витрат на пошук. Менше даних зберігається в спільній таблиці;
- Зрозуміла структура;

Недоліки:

- Потреба оперувати 3 різними таблицями, а отже і створювати різні запити для кожної.

Проаналізувавши всі переваги та недоліки наведених вище варіантів, я обрала останній: створити 3 окремі сутності. Вибір зумовлений значним поліпшенням швидкості обробки запитів, логічністю структури. Операції над цими даними не мають бути складними, тому часові витрати на додаткові запити не буде суттєвим.

2.6. Проектування бази даних

Наступним етапом проектування системи є перехід від ER моделі до реляційної (додаток Б). Реляційна модель є відображенням бази даних, яка буде створена безпосередньо в застосунку.

Для проектування поділу профілю користувача від його облікових даних я виділила дві окремі сутності. Зовнішний ключ зберігається в сутності облікових даних, аби при авторизації можна було надати інформацію про користувача, при цьому обмежети сутність домену від доступу до облікових даних.

Для відношення список – користувач до сутності списку було додано зовнішній ключ, який відповідає первинному ключу користувача. Для зв’язку між серіалами та сезонами до сутності сезону було додано зовнішній ключ, що відповідає первинному ключу серіалу. Для зв’язку між серією та сезоном до сутності серії було додано зовнішній ключ, який є ідентифікаціним номером сезону. Таким чином через проміжну таблицю сезону буде можливо також дізнатися всі серії, які містить серіал.

Для відношення M до N між фільмами і серіалами та список було створено додаткову проміжну таблицю. Її первинний ключ є складеним і вміщує в себе ідентифікаціний номер фільму або серілу та ідентифікаційний номер списку. Частини первинного ключа також є зовнішніми ключами.

Для утвроення зв’язку між користувачем та рецензіями на фільм, для всіх підтипів рецензії було додано зовнішній ключ-посилання на ідентифікаційний номер користувача та зовнішній ключ, що відповідає первинному ключу фільму, серіалу або серії.

Для забезпечення функції відслідковування переглянутих фільмів та серіалів можна створити додаткову проміжну таблицю, яка буде містити зовнішні ключі користувача та фільму, які будуть складати складений первинний ключ. Таким чином, щоб позначити, що фільм був переглянутий, потрібно додати відповідний запис між користувачем та фільмом то таблиці. Якщо запису не існує, то фільм не переглянутий.

Для сутності облікових даних я обрала обмеження CASCADE для видалення та оновлення даних, так що при видаленні користувача відповідно будуть видалятися і облікові дані про нього.

Для сутності списків я обрала обмеження CASCADE для видалення та оновлення зовнішнього ключа user_id. В результаті при видаленні користувача всі його списки також будуть видалені.

Для сутностей, які є проміжними між фільмом або серіалом та список я зазначила обмеження NO ACTION на видалення та CASCADE для оновлення даних movie_id, тим самим заборонивши видаляти фільм, якщо він є хоча б в одному списку. Проте для list_id я обрала поведінку CASCADE для обох випадків, що дозволяє безперешкодно видаляти списки і відповідні записи в проміжній таблиці.

Для серій та сезонів було обрано обмеження CASCADE, адже при видаленні серіалу не має зберігатись інформація про його сезони та серії.

Для відношення між рецензіями, користувачами та фільмами для user_id було встановлено обмеження NO ACTION для видалення. Таким чином якщо користувач лишав свої відгуки інформація про профіль буде завжди зберігатися, щоб інші користувача могли і після видалення профілю передивлятись коментарі. Для зовнішніх ключів movie_id та series_id я

встановила повіденку CASCADE. В результаті усі рецензії на фільм або серіал будуть видалені у разі видалення самого фільму або серіалу.

2.7. Мапа сайту

Під час проектування системи для користувачів було виділено дві ролі: звичайни користувач та адмінстратор. Також для авторизованих та не авторизованих користувачів відрізняється кількість сторінок, доступних для використання.

В додатку В зображено мапу сайту, на якій позначені сторінки доступні різним видам користувачів. Помаранчевич кольором позначено сторінки, які доступні тільки авторизованим користувачам. Сині компомненти доступні всім відвідувачам сайту.

Приховування деяких сторінок зумовлено тим, що відображена на них інформація є змінною та залежною від користувача, наприклад список для перегляду або профіль користувача.

Шлях користувача можна описати такими етапами:

1) знайомство з головною сторінкою та бібліотекою фільмів та серіалів;
2) реєстрація у системі, заповнення особистих даних;
3) авторизація у системі;
4) створення нового списку для перегляду;
5) додавання окремих фільмів та серіалів в список для перегляду;
6) експортування списку для перегляду.

Основною метою проектування переходу між сторінками є надання простоти навігації, але в той же час і максимальної доступності переходу з будь-якої сторінки на будь-яку іншу за допомогою не більше ніж одної проміжної сторінки між ними.

РОЗДІЛ 3. Опис реалізації

3.1. Структура програми

При проектуванні системи було обрано клієнт-серверну архітектуру для розробки. Розділення на клієнтську частину та серверну припускає подальшу можливість додавати клієнта на різних платформах, нарпиклад, мобільній, використовуючи API зі спільного для всіх клієнтів сервера. При розробці структури проекту я орієнтувалась на методололгію для розробки SaaS-застосунків «Застосунок дванадцяти факторів» [1].

1. **Кодова база.** Згідно з першим пунктом методології я створила спільний GIT-репозиторій для обох частин застосунку.
2. **Залежності.** Згідно з другим пунктом методології, для визначення залежностей в серверній частині було використано Maven, де в pom.xml прописано всі залежності застосунку від зовнішних модулів. Для клієнта конфігурацією слугує файл package.json, який слугує ресурсом для управління залежностями в JavaScript проекті.
3. **Конфігурація.** Відповідно до третього пункту методології про конфігурацію, всі змінні середовища, які використовуються, наприклад, для підключення до бази даних або впливають на налаштування збірки проекту, описані в окремому файлі application.properties та не залежать від системних локальних налаштувань розробника, середовища майбутньої разгортки проекту.
4. **Сторонні служби.** У якості сторонньої служби в моєму проекті виступає база даних PostgreSQL. Під’єднання до неї описані в конфігураційному файлі у вигляді URL та даних користувача для підключення. Замінивши ці дані можна розгорнути базу даних в будь-якій іншій СКБД.

5. **Збірка, реліз, виконання.** На момент розробки застосунку сервіс не було розгорнуто у зовнішньому середовищі.
6. **Процеси.** Для полегшення масштабування сервісу та його розгортання з можливістю розготання на різних платформах, я створила два окремих проекти для клієнту та серверу. Кожна з частин запускається як окремий процес за допомогою однієї команди в командному рядку.
7. **Прив'язка портів.** Код не є прив'язаним до портів, API та навігація працюють незалежно від порту, який вказано в конфігурації та прослуховується.

Інші фактори стосуються розгортання та адміністрування застосунку, що виходить за межі цієї наукової роботи, але будуть враховані при подальшій розробці та покращенні проекту.

3.2 Серверна частина застосунку

Для розробки серверної частини застосунку я обрала мову Java та відповідні фреймворки Spring Boot та Hibernate. За допомогою додаткових бібліотек та фреймворків Java дозволяє створювати легко масштабовані застосунки. Основною задачею серверу є надання API для клієнтської частини, перевірка та захист даних.

3.2.1 База даних

Зазвичай для доступу до бази даних в Java використовується інтерфейс JDBC. Він дозволяє проводити розробку незалежно від конкретної бази даних, до якої буде звертатися застосунок.

Використання фреймворку Spring Boot полегшує роботу з JDBC, дозволяючи автоматизувати процес створення з'єднання з базою даних при

запуску програми. Для цього потрібно під’єднати драйвер за допомогою залежностей відповідної бази даних, з якою буде працювати застосунок; вказати URL та облікові дані користувача для під'єднання в конфігураційному файлі application.properties.

Спілкування між різними компонентами серверу виконується за допомогою моделей даних – Java класів, які відповідають реальним об'єктам та мають структуру відповідну до таблиць в базі даних, в яких зберігається інформація про ці об'єкти (додаток Г). До моделей також відносяться data transfer object – об’єкти, які відповідають структурі даних, що передається від клієнта, але не відповідають реальному об’єкту. Таким чином інформацію, яку надає база даних потрібно уніфіковувати до однієї з моделей.

Для полегшення та автоматизації цього процесу я використала Hibernate [2]. Співставлення Java класів з таблицями бази даних відбувається за допомогою анотацій. Hibernate дозволяє вирішити проблему об’єктно-реляційного розриву, який з’являється через різницю між представленням об’єкта в СУБД та об’єктно-орієнтованій мові програмування [3]. Окрім очевидних відмінностей пов’язаних з різними типами даних в мовах програмуванні та базами даних, існує ще ряд суттєвих відмінностей:

1. **Ідентифікація та порівняння об’єктів.** Проблема полягає у визначенні ідентичності двох об’єктів в об’єктно-орієнтованій мові та в базі даних. В базах даних два записи є одинаковими, якщо дані, які знаходяться в них також є одинаковими. В програмуванні ж ідентифікація об’єкту не є частиною його стану, а його унікальним ідентифікатором, який зазвичай є адресою в пам’яті.
2. **Об’єктна композиція.** Однієї з особливостей об’єктно-орієнтованих мов програмування є можливість поєднувати об’єкти в більш складні

за допомогою композиції та агрегації. В результаті один об’єкт може ставати складовою частиною іншого. В базах даних цей процес є неоднозначним і залежить від того, яке відношення між об’єктами: 1 до 1, 1 до N чи M до N. В будь-якому з цих випадків для зв’язку використовуються зовнішні ключі, які посилаються лише на первинний ключ відповідного об’єкту.

При співставленні об’єктів з бази даних та Java найскладнішим випадком є відношення M до N, адже на рівні бази даних воно реалізується за допомогою проміжної таблиці. Для створення цієї асоціації в коді я використала анотацію @ManyToMany. Асоціація цих об’єктів є двонаправленою. В даному прикладі головною таблицею, тобто власником асоціації є List (рис. 5).

```
@ManyToMany
@JoinTable(
        name = "movie_to_list",
        joinColumns = { @JoinColumn(name = "list_id") },
        inverseJoinColumns = { @JoinColumn(name = "movie_id") }
)
private Set<Movie> movies = new HashSet<>();
```

Рисунок 5. Поле movies в класі MovieList

Movie є інверсією цієї асоціації, яка відображається за допомогою mappedBy. В результаті при змінах в даних Hibernate буде перевіряти лише головну таблицю.

```
@JsonIgnore
@ManyToMany(mappedBy = "movies")
Set<MovieList> lists = new HashSet<>();
```

Рисунок 6. Поле lists в класі Movie

Важливим також є вибір колекції для зберігання об'єктів при асоціації багато до багатьох [4]. У разі використання List для зберігання об’єктів,

Hibernate при видаленні буде видаляти всі записи з асоційованої таблиці перед тим, як додати ті, що залишились. У той же час, якщо використовувати Set, алгоритм буде видаляти лише потрібні записи і залишити інші незмінними.

Використовуючи двонаправлену асоціацію важливо звернути також уваги на серіалізацію цих об’єктів. Оскільки об’єкти, зв’язані асоціацією багато багатьох містять один одного, звичайна серіалізація обернеться нескінченною рекурсією, що призведе до помилки. Для уникнення цієї проблеми я використала анотацію @JsonIgnore для відповідного поля.

Також треба бути обережним при використанні бібліотеки Lombok. Для перевизначення хеш-функцій об’єктів, Lombok також буде рекурсивно викликати хеш-функції об’єктів, які містять один одного. Для цих класів доведеться писати допоміжні функції без використання бібліотеки.

Для спрощення роботи з Hibernate я використала плагін JPA Buddy сумісний з IntelliJ IDEA [6]. Плагін надає додаткових три вікна:

1. **JPA Structure**. Генерування JPA Entity, репозиторіїв, таблиць з вказанням ідентифікаторів, атрибутів та їх типів, відповідних анотацій за допомогою візуального інтерфейсу. Надає ієрархічне відображення моделі бази даних і перехід між моделями.
2. **JPA Palette**. Після генерації моделі буде доступне вікно JPA Palette для генерації нових атрибутів, функцій зворотнього виклику, індексів, запитів.
3. **JPA Inspector**. Вікно JPA Inspector дозволяє керувати класами та їх полями, відображенням в базі даних за допомогою анотацій.

3.2.2 Маршрутизація запитів

Для маршрутизації запитів для кожної з моделей я створила свій контролер на базі Spring Boot @RestController [7]. Кожному класу відповідає свій шлях, який уточнюється за допомогою параметру в анотаціях, що вказують на REST метод: GET, POST, PUT або DELETE. Також за допомогою фреймворку JSON вміст вхідного запиту перетворюється на об’єкт моделі, який далі передається до сервісів.

3.2.3 Захист інформації

Spring Boot фреймворк надає доступ до Spring Security, який забезпечує автентфікацію, авторизацію та обмежує доступ користувачів до ресурсів. Реалізація процесу захисту відбувається за допомогою класів Authentication Manager, WebMvcConfigurer та WebSecurityConfigurerAdapter.

Для захисту паролів використано клас BCrypt, який надає функціонал для шифрування та дешифрування паролів.

Для перевірки авторизація користувача та його доступу до різних рівнів даних я використала JWT, який оновлюється кожні 2 години. Для покращення захисту від перехоплення токену можна використовувати дворівневу структуру токенів, що складається з оновлюваного токену та токену доступу [8]. Різниця між токенами полягає в наступному:

Токен доступу. Надається клієнту при доступі до серверу. Він є багаторазовий і має маленьку тривалість життя. Використовується при наступних зверненнях до серверу.

Токен оновлення. Має велику тривалість життя, яка досягає місяця і більше. Використовується для оновлення токену доступу та токену оновлення при закінченні часу життя токену доступу.

За допомогою цієї структури при перехопленні токенів зовнішнім пристроєм після закінчення терміну життя токену доступу одному з користувачів повернеться нова пара токенів, попередня стане невалідною. Таким чином або токени зловмисника не будуть вже давати доступ до даних, або токени справжнього користувача недійсними, йому доведеться знову авторизуватись і система видасть йому нові токени, через що токени зловмисника не будуть відповідати вимогам.

3.3 Клієнтська частина застосунку

Для написання клієнту сервісу я обрала мову JavaScript та бібліотеки React [9], Bootstrap [10] та Material UI [11].

3.3.1 Використання React Hooks

Під час реалізації клієнтської частини застосунку я використовала обидва підходи написання React компонентів: функціональний (за допомогою Reach hooks) та об’єктно-орієнтований (на базі класу React.Component). В результаті роботи з обома варіантами я виявила такі переваги та недоліки hooks.

Переваги:

- Можливість перевикористання коду поза функцією, яка описує компонент на відміну від компоненту на основі класу, де функції класу не можуть бути викликані ззовні;
- Менше коду для опису однакових компонент, звертання до змінних в компоненті без звертання до об’єкту класу;

Недоліки:

- Відсутність методів componentDidmount та componentDidUpdate. Замість них наявна функція useEffect, яка поєднує в собі їхні функції, але її застосування набагто складніше;
- Асинхронне оновлення станів.

3.3.2 Експортування списків

Для збереження списків у PDF файл я використала бібліотеку jsPDF [12]. Для додавання до файлу тексту потрібно встановити шрифт, його розмір, колір, координати початку стрічки. Для того, щоб переносити текст на інший рядок в кінці стрічки слід використати метод splitTextToSize(text, size). При переносі на новий рядок враховується висота попереднього і додається до вертикальної координати попереднього.

Для малювання доступні фігури квадрата та кола. Потрібно також обрати колір, розмір та координати розміщення фігури. Текст та фігури можна обертати.

3.4 Інтерфейс користувача

В результаті розробки застусунку користувачу доступні такі сторінки (додаток Д):

- Сторінки авторизації (логін та реєстрація);
- Головна сторінка, на якій відображено можливості сервісу та швидкий перехід до функціоналу;
- Сторінка профілю користувача;
- Бібліотека фільмів з можливістю додавати фільми до списку;
- Бібліотека списків користувача;
- Сторінка фільму з описом та можливістю додавати фільм до списку;
- Сторінка списку з відображенням всіх наявних фільмів у списку та можливістю завантажити список у форматі PDF файлу (додаток Е)

Висновки

В курсовій роботі було розглянуто підходи до проектуванян архітектури застосунку, обрано одну з них відповідно до вимог сервісу, який було розроблено. Оглянуто інструменти, які були використані під час розробки такі, як фреймворки Spring Boot, Hibernate, бібліотеки Lombok, React, Bootstrap, Material UI, jsPDF, плагін JPA Buddy.

На основі опитування цільової аудиторії було сформовано функціональні вимоги до застосунку та спроектовано шлях користувача.

Було спроектовано ER модель та реляційну модель відповідно до функціональних вимог та області використання застосунку. На основі моделей створено базу даних в СУБД PostgreSQL.

В результаті роботи отримано веб-застосунок, який допомогає користувачам відслідковувати свої переглянуті фільми та серіалі, організовувати власну бібліотеку стрічок, керувати своїм профілем та завантажувати списки.

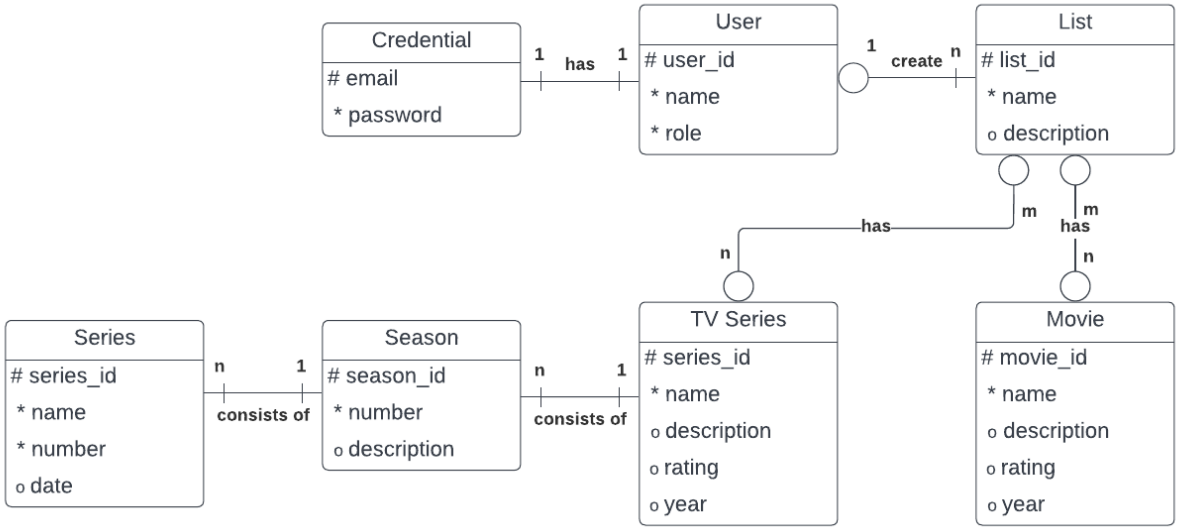
Список використаних джерел

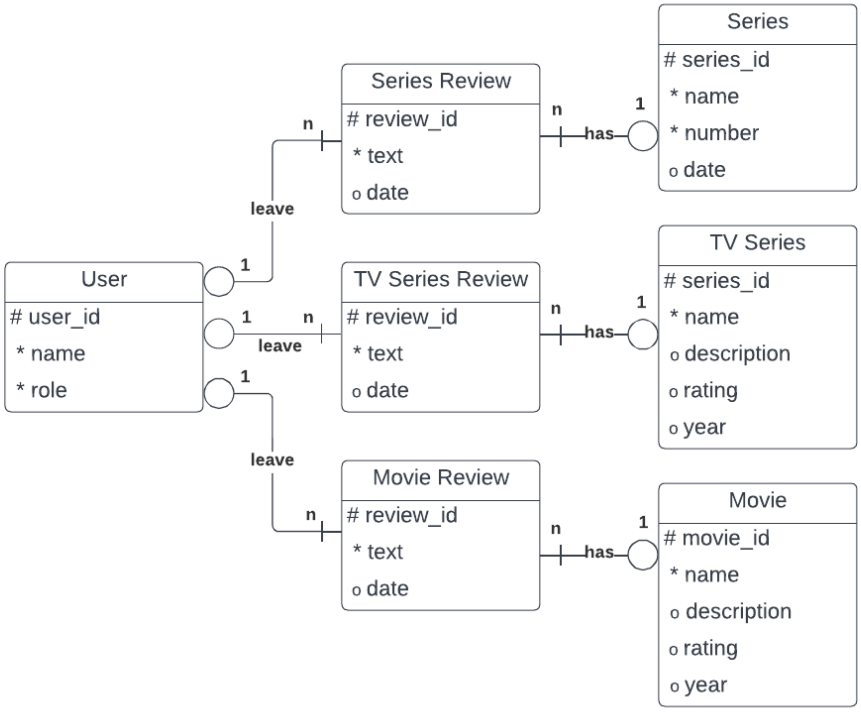
1. The Twelve-Factor App. The Twelve-Factor App. URL: https://12factor.net/
2. Hibernate. Everything data. Hibernate. URL: https://hibernate.org/
3. The Vietnam of Computer Science · Ted Neward's Blog. Ted Neward's Blog. URL: http://blogs.tedneward.com/post/the-vietnam-of-computer-science/
4. Best Practices for Many-to-Many Associations with Hibernate and JPA. Thorben Janssen. URL: https://thorben-janssen.com/best-practices-for-many-to-many-associations-with-hibernate-and-jpa/
5. JPA Buddy - IntelliJ IDEA plugin supporting Hibernate, EclipseLink, SpringData, Liquibase, Flyway and other JPA related tech. JPA Buddy - IntelliJ IDEA plugin supporting Hibernate, EclipseLink, SpringData, Liquibase, Flyway and other JPA related tech. URL: https://www.jpa-buddy.com/
6. IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains. JetBrains. URL: https://www.jetbrains.com/idea/
7. Building a Gateway. Spring | Home. URL: https://spring.io/guides/gs/gateway/
8. What Are Refresh Tokens and How to Use Them Securely. Auth0 - Blog. URL: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/
9. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.reactjs.org/

10. Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/
11. MUI: The React component library you always wanted. MUI: The React component library you always wanted. URL: https://mui.com/
12. GitHub - parallax/jsPDF: Client-side JavaScript PDF generation for everyone. GitHub. URL: https://github.com/parallax/jsPDF

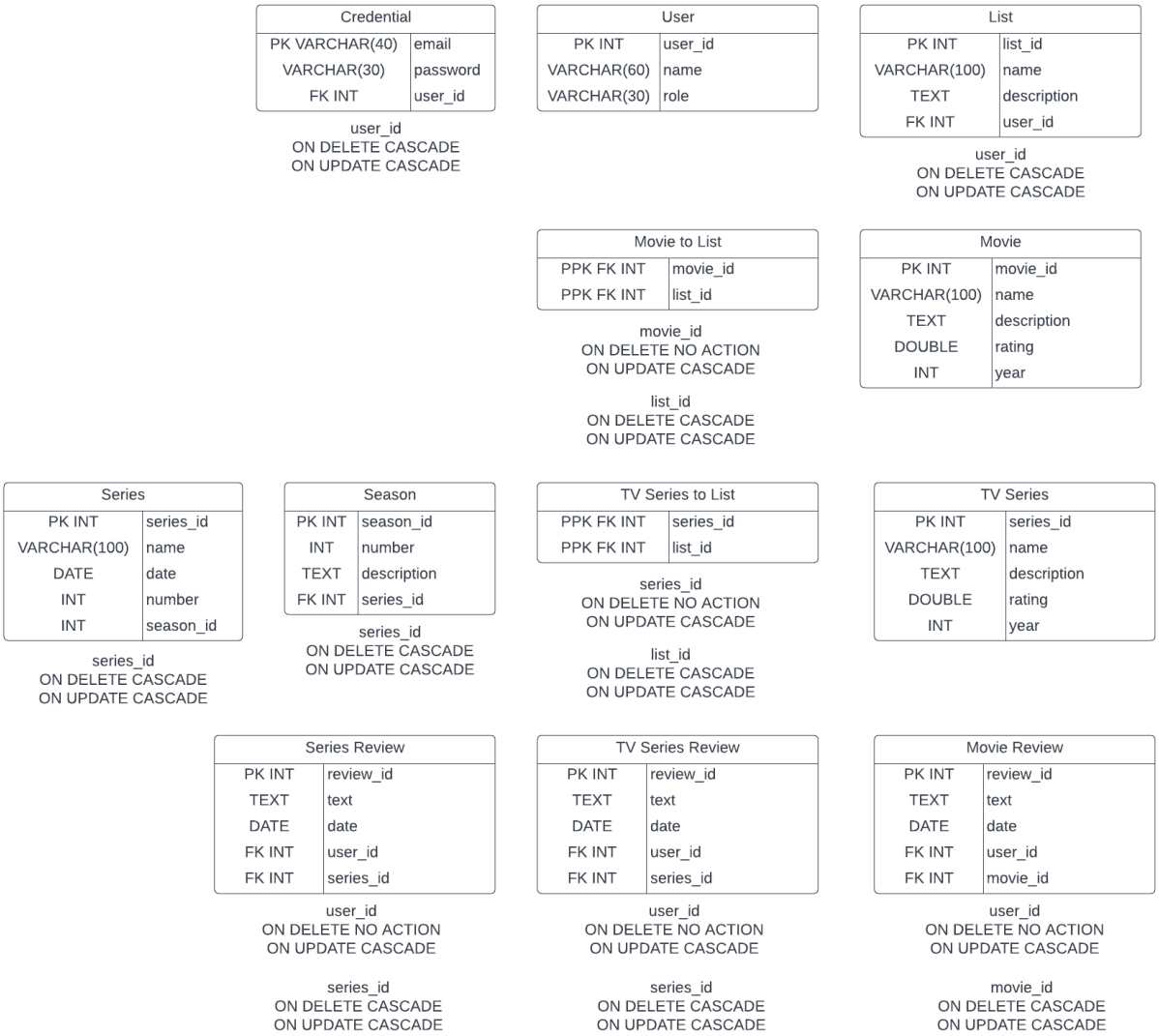
ДОДАТКИ

Додаток А – ER діаграма застосунку

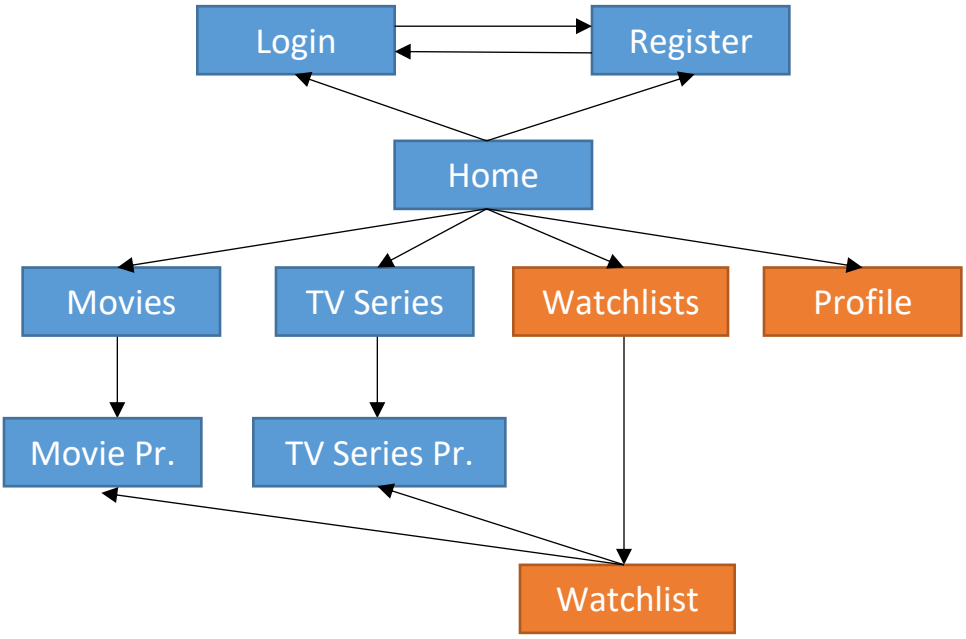




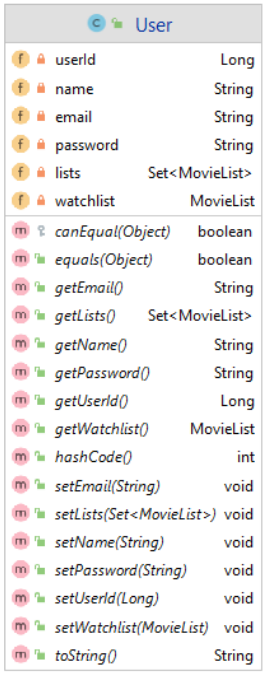
Додаток Б – реляційна модель застосунку

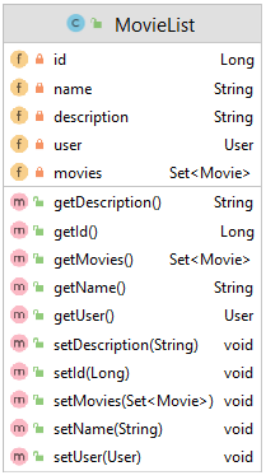


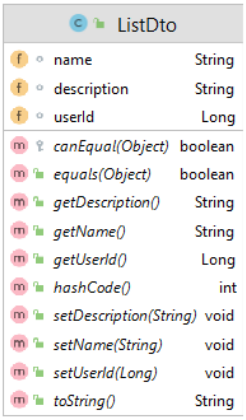
Додаток В – мапа сайту

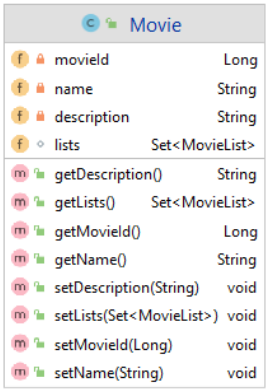


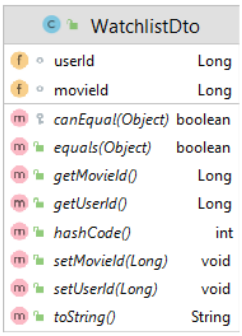
Додаток Г – структура моделей

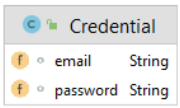




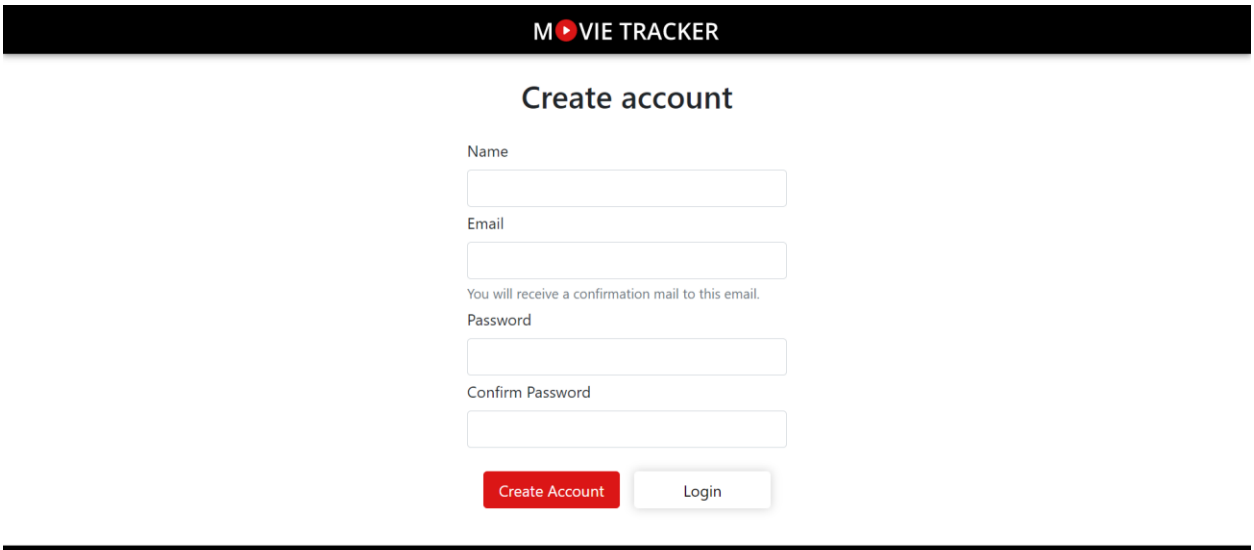


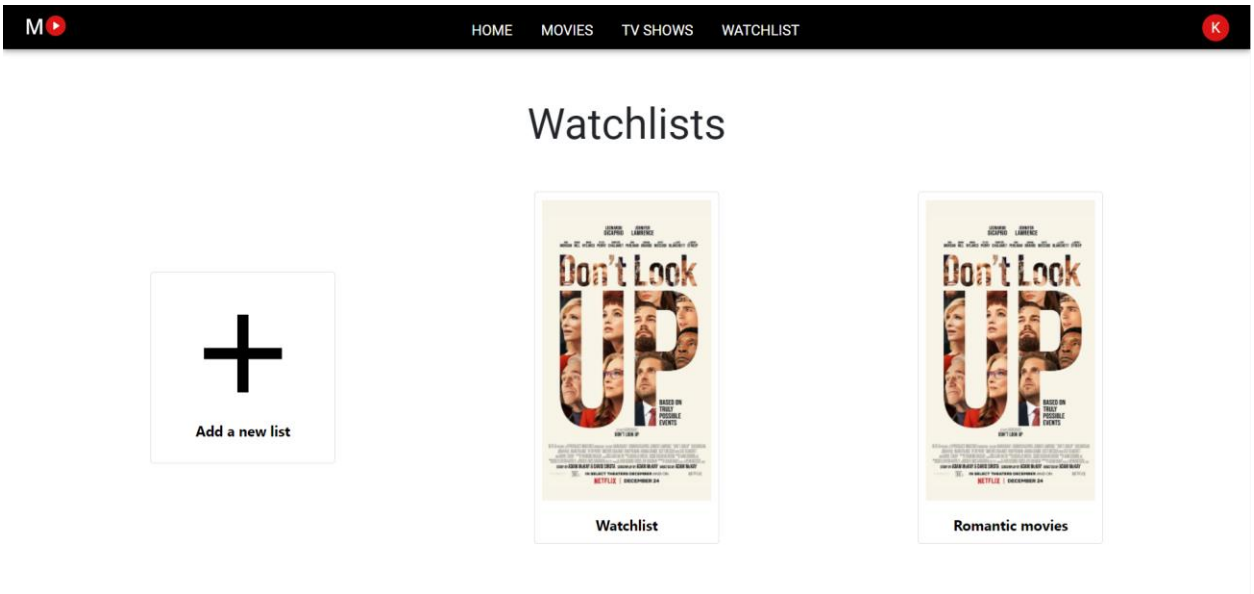


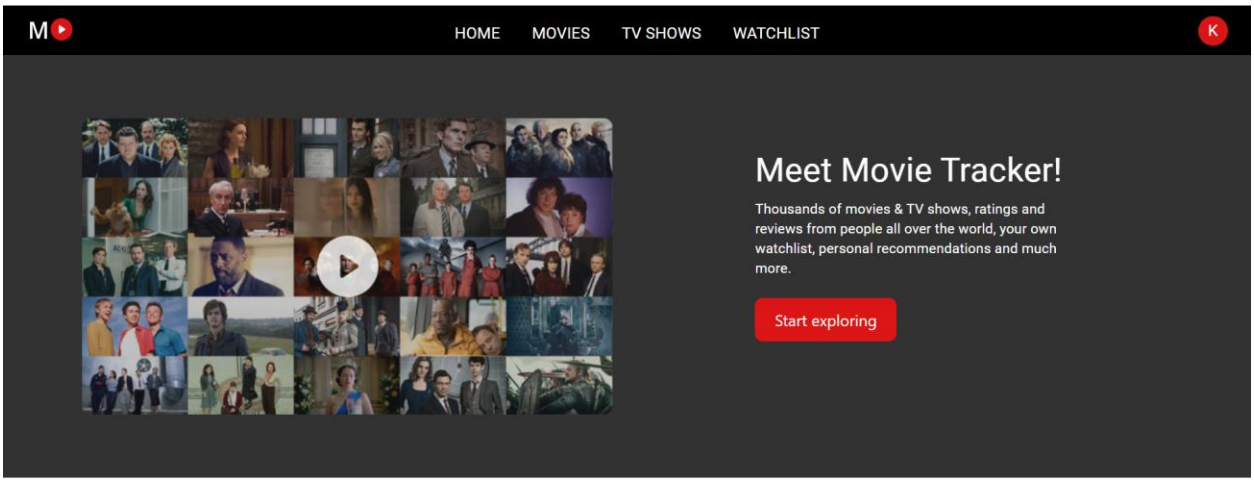


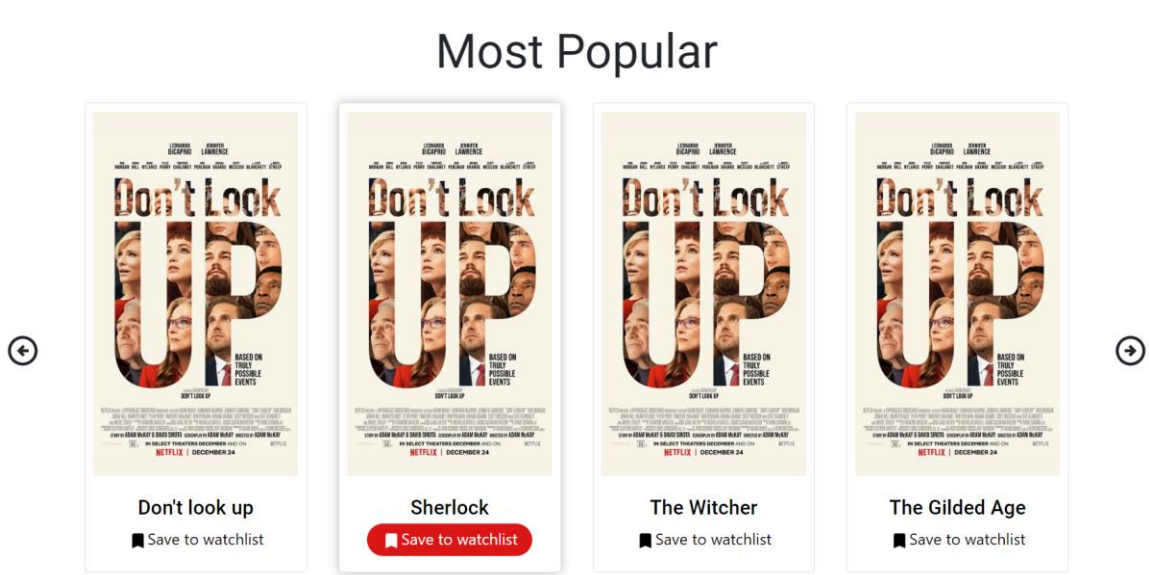


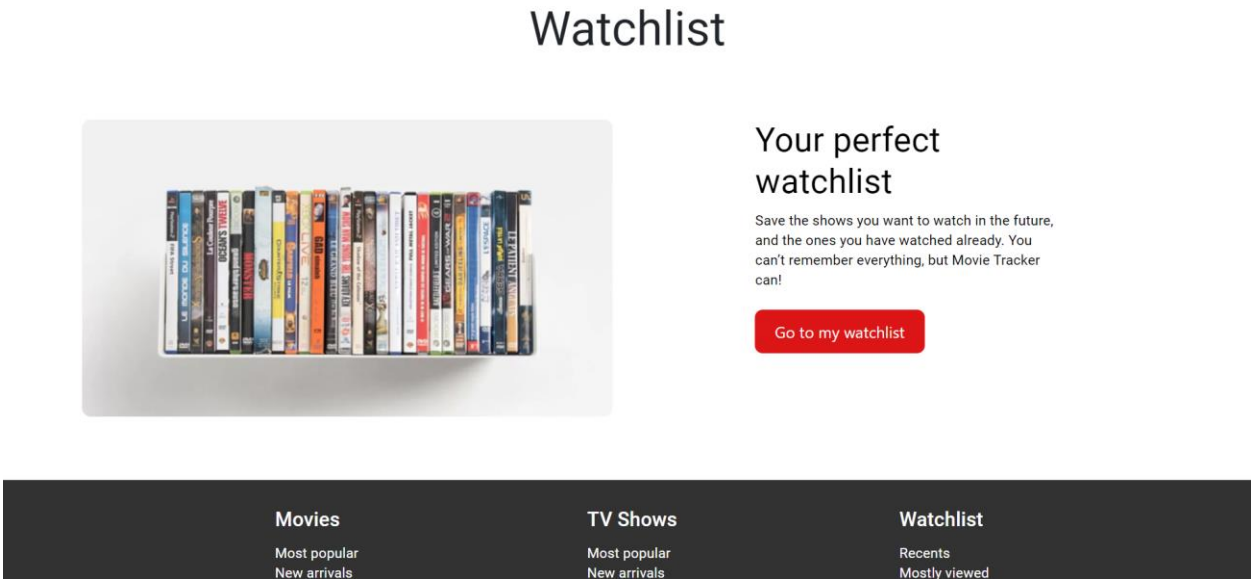
Додаток Д – інтерфейс застосунку

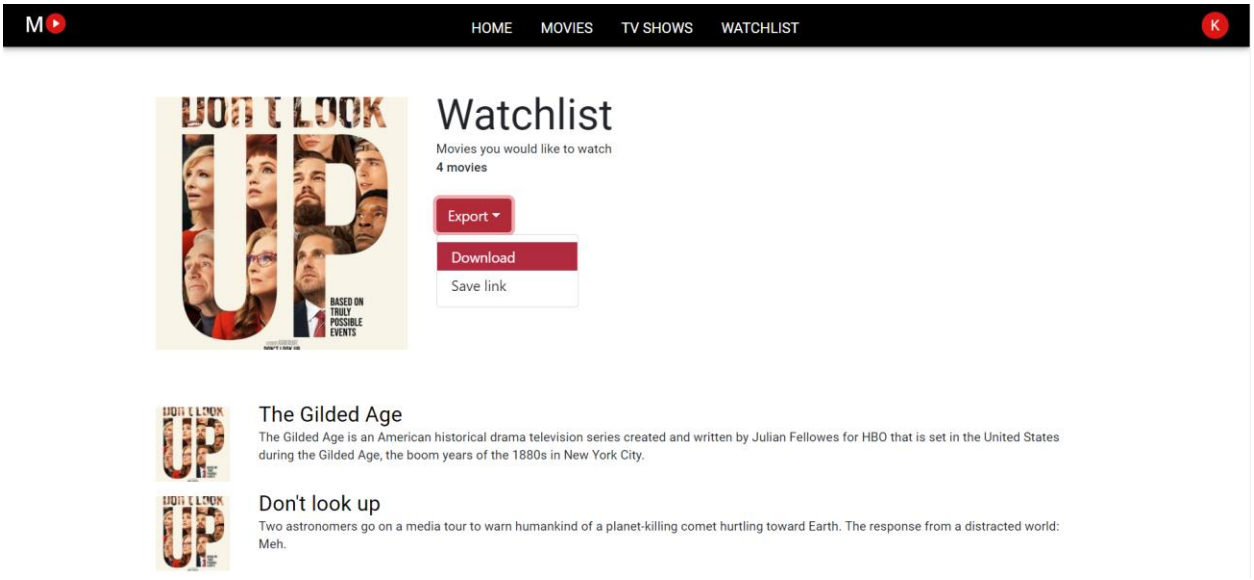


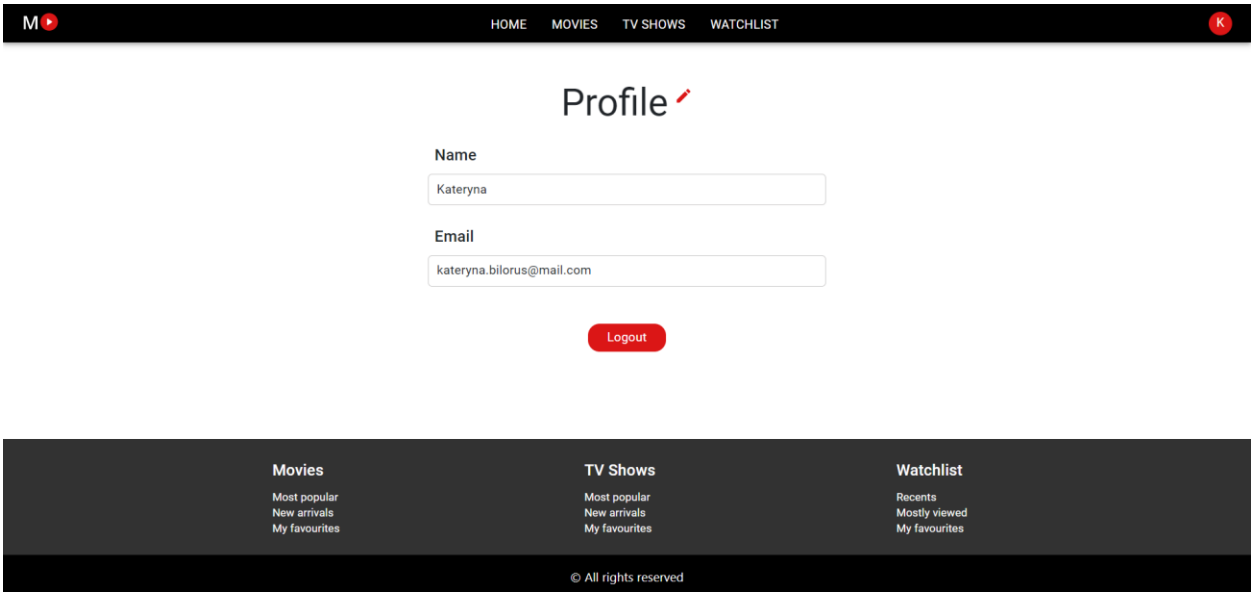



M
HOME
MOVIES
TV SHOWS
WATCHLIST
K
Meet Movie Tracker!
Thousands of movies & TV shows, ratings and reviews from people all over the world, your own watchlist, personal recommendations and much more.
Start exploring

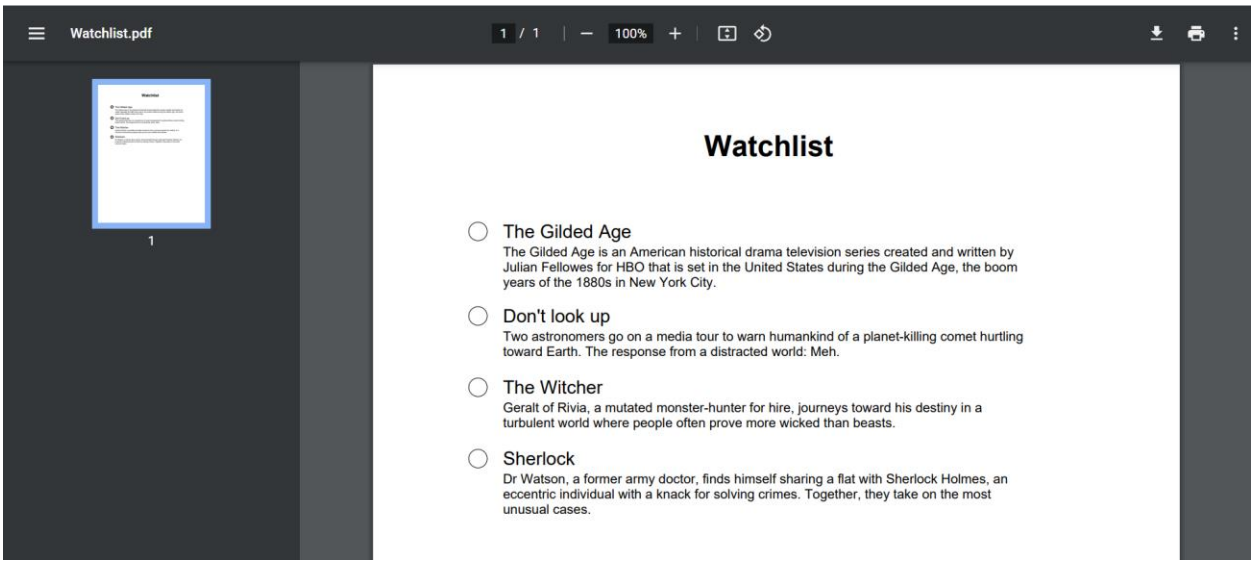

Most Popular
Don't look up
Save to watchlist
Sherlock
Save to watchlist
The Witcher
Save to watchlist
The Gilded Age
Save to watchlist


Watchlist
Your perfect watchlist
Save the shows you want to watch in the future, and the ones you have watched already. You can't remember everything, but Movie Tracker can!
Go to my watchlist
Movies
Most popular
New arrivals
TV Shows
Most popular
New arrivals
Watchlist
Recents
Mostly viewed


HOME
MOVIES
TV SHOWS
WATCHLIST
K
Watchlist
Movies you would like to watch
4 movies
Export
Download
Save link
The Gilded Age
The Gilded Age is an American historical drama television series created and written by Julian Fellowes for HBO that is set in the United States during the Gilded Age, the boom years of the 1880s in New York City.
Don't look up
Two astronomers go on a media tour to warn humankind of a planet-killing comet hurtling toward Earth. The response from a distracted world: Meh.


HOME
MOVIES
TV SHOWS
WATCHLIST
K
Profile
Name
Kateryna
Email
kateryna.bilorus@mail.com
Logout
Movies
Most popular
New arrivals
My favourites
TV Shows
Most popular
New arrivals
My favourites
Watchlist
Recents
Mostly viewed
My favourites
© All rights reserved

Додаток Е – еспортований список



Watchlist.pdf

1 / 1 100%

1

Watchlist

- The Gilded Age
  The Gilded Age is an American historical drama television series created and written by Julian Fellowes for HBO that is set in the United States during the Gilded Age, the boom years of the 1880s in New York City.
- Don't look up
  Two astronomers go on a media tour to warn humankind of a planet-killing comet hurtling toward Earth. The response from a distracted world: Meh.
- The Witcher
  Geralt of Rivia, a mutated monster-hunter for hire, journeys toward his destiny in a turbulent world where people often prove more wicked than beasts.
- Sherlock
  Dr Watson, a former army doctor, finds himself sharing a flat with Sherlock Holmes, an eccentric individual with a knack for solving crimes. Together, they take on the most unusual cases.