

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

На тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ КІНОТЕАТРУ
CINEMA WEB APPLICATION DEVELOPMENT

Виконав: студент(ка) 4 курсу, групи КНТ-110сп

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ЖУК Є.В.

(прізвище та ініціали)

Керівник ЗЕЛЕНЬОВА І.Я.

(прізвище та ініціали)

Рецензент КАЗУРОВА А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЖУКА Євгена Владіславовича
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка вебзастосунку для кінотеатру Cinema Web Application Development

Керівник проекту (роботи) ЗЕЛЕНЬОВА Ірина Яківна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від 04 квітня 2023 року №87

2. Строк подання студентом проекту (роботи) 01 червня 2023 року

3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Керівництво програміста. 5. Керівництво оператора

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЗЕЛЕНЬОВА І.Я., доцент		
Нормоконтролер	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 04 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4тиждень	Розділ 3
5	Тестування та експериментальне дослідження програми	5 тиждень	Розділ 4
6	Створення керівництва оператора програми	6 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
8	Захист роботи	8 тиждень	

Студент(ка)

Євген ЖУК
(підпис) (прізвище та ініціали)

Керівник проекту (роботи)

Ірина ЗЕЛЕНЬОВА
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра:
83 с., 25 рис., 2 дод., 15 джерел., 17 табл.

ВЕБСАЙТ, РЕЄСТРАЦІЯ, АВТОРИЗАЦІЯ, КІНОФІЛЬМ, ВІДГУК,
КОМЕНТАР, НОВИНА, ТРЕЙЛЕР, ОБРАНЕ.

Об'єкт дослідження – програмні засоби для отримання інформації про фільми, анімаційні фільми та телесеріали.

Предмет дослідження – комп'ютерні програми для перегляду фільмів, анімаційних фільмів та телесеріалів.

Мета роботи – створення програмної системи для перегляду інформації про фільми, анімаційні фільми та телесеріали.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, технологія та пристрій WebStorm, JetBrains Rider, мови програмування Typescript, Html, CSS, персональний комп'ютер з процесором Ryzen 5 5600G під управлінням операційної системи Microsoft Windows 11 Pro.

Результати. Створено програмний вебсайт, використовуючи реактивний підхід за рахунок бібліотеки React.

Висновки. Розроблено програмну систему для перегляду інформації що до фільмів, анімаційних фільмів та телесеріалів з додатковим функціоналом.

Галузь використання – перегляд актуальних новин, інформації про фільми, анімаційні фільми, телесеріали, а також їх трейлери.

Економічна ефективність. При використанні програми збільшена швидкість завантаження даних, в порівнянні з іншими програмами.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 83 pp., 25 figures, 2 appendices, 15 sources, 17 tables.

WEBSITE, REGISTRATION, AUTHORIZATION, MOVIE, REVIEW, COMMENT, NEWS, TRAILER, FAVORITES.

Research subject - software tools for obtaining information about movies, animated films, and TV series.

Research object - computer programs for viewing movies, animated films, and TV series.

Research goal - development of a software system for accessing information about movies, animated films, and TV series.

Materials, methods, and technical tools: Structural and object-oriented programming, WebStorm technology and device, JetBrains Rider, programming languages TypeScript, HTML, CSS, personal computer with Ryzen 5 5600G processor running Microsoft Windows 11 Pro operating system.

Results - a web-based software application has been created using a reactive approach with the React library.

Conclusions - a software system has been developed for accessing information about computer games with additional functionality.

Field of application - viewing current news, information about movies, animated films, TV series, as well as their trailers.

Economic efficiency - the program demonstrates improved data loading speed compared to other applications when in use.

ЗМІСТ

ВСТУП	9
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	11
1.1 Огляд аналогів.....	11
1.1.1 Аналоги інших сайтів	11
1.1.2 Аналіз переваг та цікавих вирішень аналогів	14
1.2 Аналіз інструментів для розробки	14
1.2.1 Підбір стеку технологій.....	14
1.2.2 Технічна підготовка до завдання.....	15
2. АНАЛІЗ ТЕХНОЛОГІЇ РОЗРОБКИ.....	16
2.1 Angular.....	16
2.2 TypeScript.....	17
2.3 Asp net core.....	19
2.4 Entity Framework.....	21
2.5 C#.....	24
2.6 Вибір засобів розробки.....	25
3. ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ СИСТЕМИ.....	27
3.1 Розробка бази даних.....	27
3.2 Алгоритм функціонування програми.....	33
3.3 Проектування бази даних.....	34
4. ЕКСПЛУАТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ.....	37
4.1 Характеристики програми.....	37
4.2 Запуск застосування.....	37
4.3 Загальні відомості про програму.....	38
4.4 Структура програми.....	38
4.5 Тестування програми.....	39

5 КЕРІВНИЦТВО ОПЕРАТОРА.....	40
5.1 Призначення програми.....	40
5.2 Умови виконання програми.....	40
5.3 Експлуатація програми.....	40
Висновки.....	46
Перелік джерел посилань.....	47
Додаток А Текст програми.....	49
Додаток Б Слайди презентації.....	78

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
EF	– Entity Framework
IDE	– Integrated Development Environment;
IMDb	– Internet Movie Database;
LINQ	– Language-Integrated Query
MPA	– Multi Page Application;
TS	– Typescript;
SEO	– Search Engine Optimization;
SPA	– Single Page Application;
UI	– User Interface;
WEB	– World Wide Web.

ВСТУП

Портали для огляду відгуків про фільми є корисним інструментом для тих, хто шукає інформацію про фільми. Ось декілька причин, чому вони можуть бути корисні:

Допомагають зробити вибір: на порталах для огляду відгуків про фільми зазвичай зібрані відгуки від різних людей, які вже переглянули фільм. Це дає можливість отримати різноманітні думки та погляди на фільм, що допомагає визначитися з вибором фільму, який може сподобатися.

Допомагають оцінити якість фільму: на порталах для огляду відгуків про фільми можна знайти професійні оцінки та рейтинги фільмів. Це дозволяє оцінити якість фільму та його потенційну привабливість для перегляду.

Допомагають знайти нові фільми: Портали для огляду відгуків про фільми часто містять рекомендації щодо нових фільмів, які можуть бути цікавими для перегляду. Це дозволяє знайти нові фільми, які можуть не виявитися в обробці.

Допомагають зрозуміти сюжет: на порталах для огляду відгуків про фільми можна знайти резюме та описи сюжетів фільмів, що допомагає зрозуміти, про що фільм.

Різнноманітність джерел відгуків: на порталах для огляду відгуків про фільми можна знайти різнноманітні джерела відгуків, включаючи професійних критиків, журналістів, любителів кіно та інших експертів. Це дозволяє отримати широку палітру думок та оцінок про фільм.

Розширені функції пошуку: Багато порталів для огляду відгуків про фільми мають розширені функції пошуку, які дозволяють знайти фільми за категоріями, жанрам, режисерами, акторами, роках виходу та іншими параметрами.

Коментарі та обговорення: Багато порталів для огляду відгуків про фільми також мають функції коментування та обговорення фільмів. Це

дозволяє користувачам обмінюватися думками та враженнями про фільм, обговорювати його сюжет та характери.

Рейтинги та підсумки року: Багато порталів для огляду відгуків про фільми мають щорічні рейтинги та підсумки року, які включають найкращі фільми за рік, кращих акторів та актрис, кращі сценарії та режисерів.

Можливість залишати відгуки: Багато порталів для огляду відгуків про фільми дозволяють користувачам залишати свої власні відгуки про фільми. Це дозволяє людям поділитися своїми враженнями та оцінками про фільм, що може бути корисно для інших користувачів.

Отже, портали для огляду відгуків про фільми можуть бути корисними для тих, хто хоче знайти новий фільм для перегляду або просто оцінити якість та популярність вже відомих фільмів[1].

Критичний аналіз: Багато порталів для огляду фільмів пропонують детальні аналітичні огляди, де професійні критики аналізують різні аспекти фільму, включаючи акторську гру, режисуру, сценарій, операторську роботу, монтаж та багато іншого. Це допомагає краще розуміти і оцінювати різні аспекти фільму.

Ретроспективні огляди: Деякі портали для огляду фільмів також пропонують ретроспективні огляди, де аналізуються класичні фільми або фільми, що вийшли у минулому. Це може бути корисно для тих, хто хоче відкрити для себе старіші твори кінематографу або переглянути їх з новою перспективою.

Рекомендації засновані на вподобаннях: Деякі портали для огляду фільмів використовують алгоритми рекомендацій, які засновані на вподобаннях користувача. Вони аналізують історію переглядів та оцінки користувача, щоб рекомендувати фільми, які йому можуть сподобатися на основі подібних смаків.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд аналогів

Перегляд аналогів дає можливість для розбору переваг та недоліків сайтів різних розробників та різних пристроїв.

1.1.1 Аналоги інших сайтів

В інтернеті дуже велика кількість сайтів з відгуками про фільми, кожне з них чимось виділяється, але більшість із сайтів не зручні та нудні для перегляду.

Rotten Tomatoes - це один з найбільш відомих та популярних порталів для огляду відгуків про фільми. Він має велику базу даних фільмів, а також зібрані відгуки від критиків та глядачів. На порталі можна знайти професійні оцінки, рейтинги, резюме та коментарі(рис. 1.1) [2].

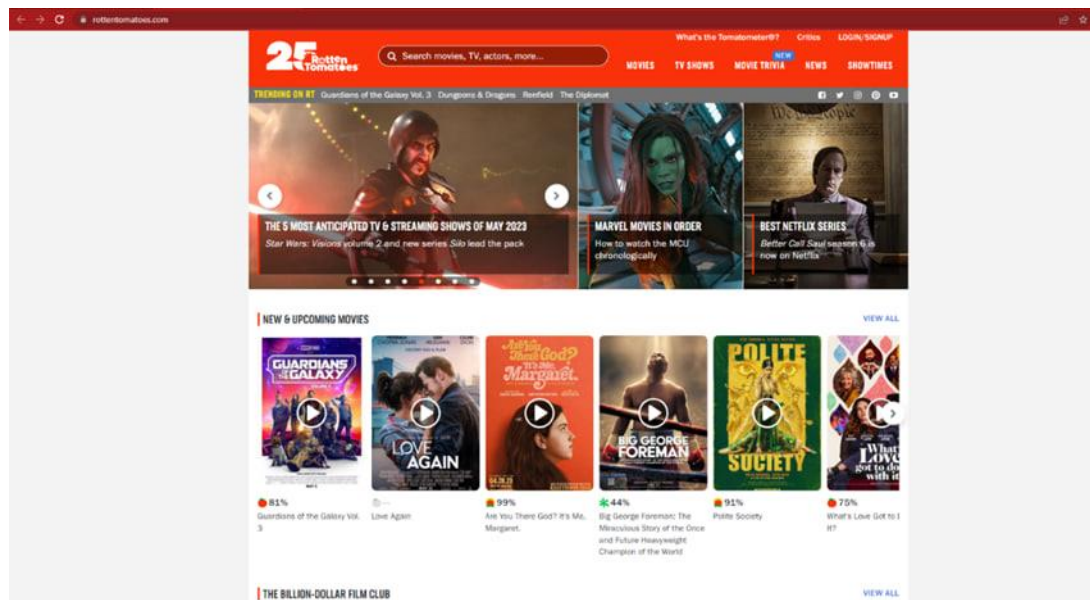


Рис. 1.1 - Головна сторінка сайту Rotten Tomatoes

IMDb - це ще один дуже популярний портал для огляду відгуків про фільми. Він містить велику базу даних фільмів, серіалів та телешоу, а також зібрані відгуки від критиків та глядачів. На порталі можна знайти інформацію про акторів, режисерів, бюджет та касові збори фільмів(рис. 1.2) [3].

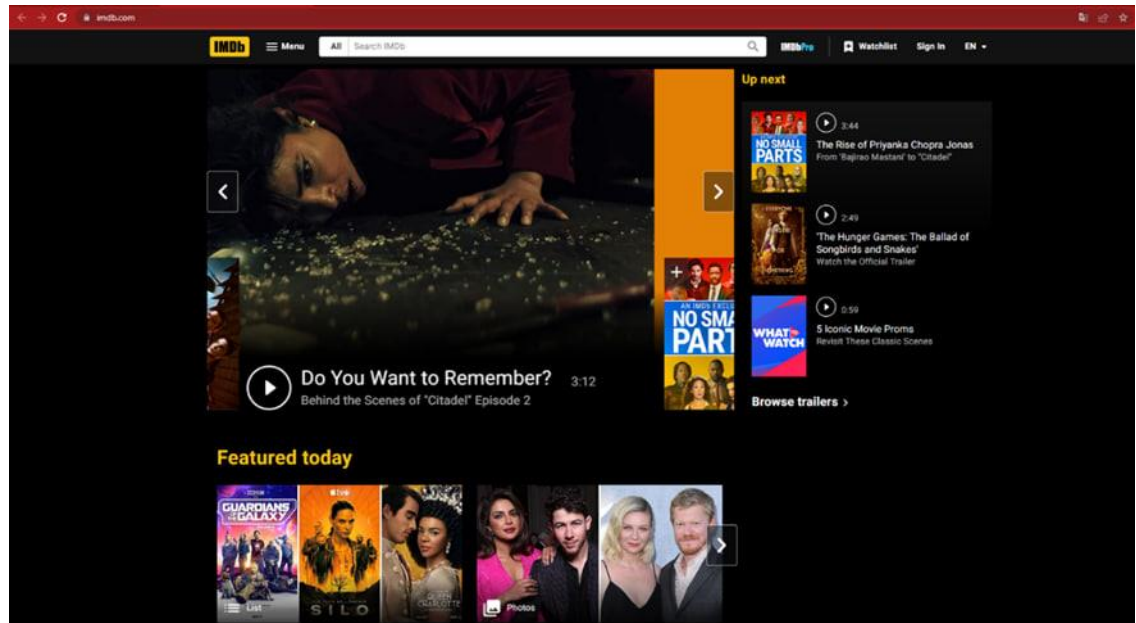


Рис. 1.2 - Головна сторінка сайту IMDb

Metacritic - це портал, який зібрав огляди та рейтинги від критиків з різних джерел, включаючи видання, журналісти та інтернет-сайти. На порталі можна знайти рейтинги для фільмів, телешоу, музики та відеоігор, а також резюме та коментарі(рис. 1.3) [4].

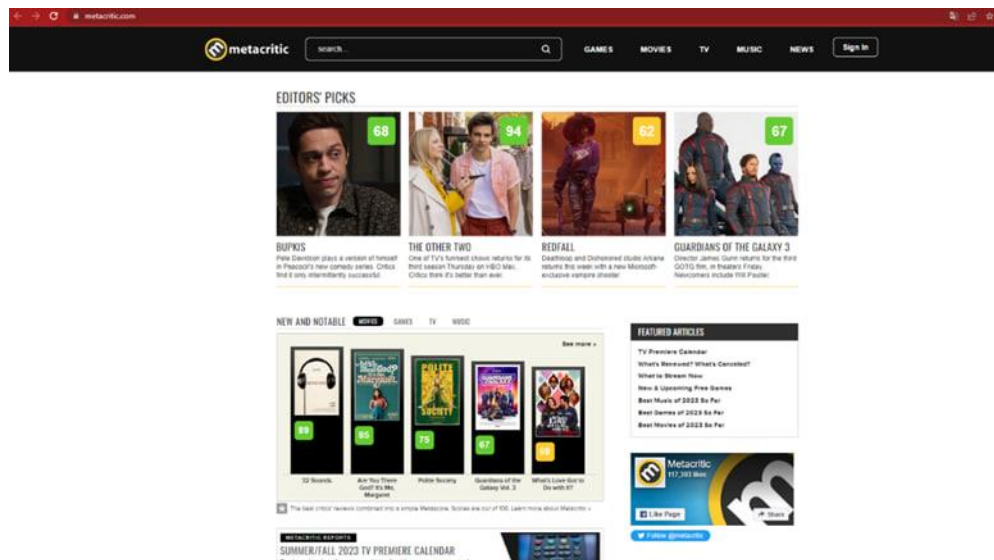


Рис. 1.3 - Головна сторінка сайту Metacritic

Letterboxd - це соціальна мережа для кінофілів, де користувачі можуть ділитися своїми відгуками про фільми та серіали, створювати списки переглянутих фільмів та рекомендації для інших користувачів. На порталі можна знайти відгуки, рейтинги та інші враження про фільми від любителів кіно(рис. 1.4) [5].

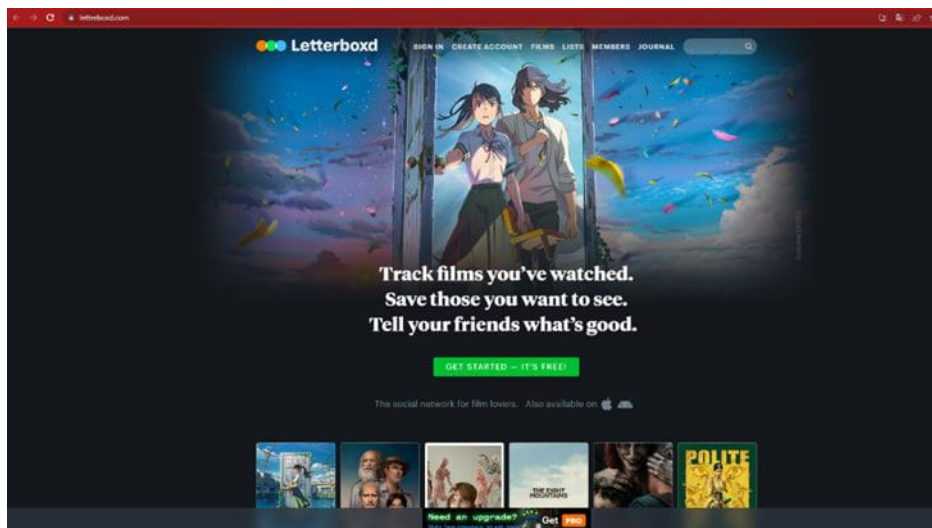


Рис. 1.4 - Головна сторінка сайту Letterboxd

1.1.2 Аналіз переваг та цікавих вирішень аналогів

Найкраще з побачених аналогів – це швидкість та зручність пошуку інформації про фільми, що вас цікавить, гарний дизайн, а також популярність цих сайтів, завдяки чому є багато цікавих відгуків та рецензій – що для мене є найбільш суттєвим та виділяє їх серед усіх. Такі сайти або програми красиві та зручні для перегляду.

Також важливо було дізнатися джерела, звідки вони беруть дані. Більшість з них беруть дані з API-сервісів і дуже мало зі своїх досліджень, які володіють та ведуть власні сайти. Багато з API-сервісів є платними і не мають безкоштовних рішень, але один з найпопулярніших moviedb має безкоштовний доступ з варіативними даними інформації щодо популярних фільмів.

1.2 Аналіз інструментів для розробки

Аналіз включає перегляд інструментів сайтів аналогів та їх вирішення у верстці сторінок. Підбір стеку технологій та технічна підготовка до завдання.

1.2.1 Підбір стеку технологій

Написання сайту на чистому html, css, js зараз не актуальний, програма які перезавантажують сторінку втратили актуальність, так як є жахливою практикою для розробника:

- коли додаток почне розширюватися – подібна техніка додасть проблем майбутнім розробкам;
- повторюваний функціонал для схожих блоків у яких відрізняється лише мала частина;
- велика збитковість ресурсів при багатосторінковому додатку.

З цього стає зрозуміло, що стек технологій потребує сучасних вирішень, а саме вибір фреймворку, що підходить для всіх рішень сайту “ShowTime”.

1.2.2 Технічна підготовка до завдання

Сайт “ ShowTime” потребує у таких технологіях:

- динамічні зміни даних та їх централізоване зберігання
- фреймворк який не вимагає перезавантаження сторінок для багатосторінковості, отримання, відправлення, оновлення даних;
- технологія відправлення та отримання даних через API;
- можливість повторного використання компонентів.

2 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Angular

Angular - це вебфреймворк, розроблений компанією Google, який дозволяє розробляти високопродуктивні односторінкові додатки (SPA) та додатки з багатьма сторінками (MPA). Angular був розроблений на базі мови TypeScript та спрощує процес розробки вебдодатків за допомогою компонентної архітектури та декларативного програмування[6].

Основні переваги Angular:

- модульність: Angular дозволяє розбити додаток на невеликі, незалежні компоненти, що полегшує розробку та підтримку;
- компонентно-орієнтований підхід: Angular використовує компонентно-орієнтований підхід, що дозволяє розробникам легко зосереджуватися на створенні окремих частин додатку та їх підтримці;
- декларативне програмування: Angular використовує декларативний підхід до програмування, що дозволяє розробникам описувати бажаний стан додатку та залишати Angular відповідальним за його реалізацію;
- підтримка TypeScript: Angular підтримує мову програмування TypeScript, що дозволяє розробникам створювати складні додатки з більшою читабельністю та зменшенням кількості помилок;
- підтримка серверного рендерингу: Angular підтримує серверний рендеринг, що дозволяє зменшити час завантаження додатку та поліпшити SEO;
- розширюваність: Angular має велику кількість доповнень (plugins), що дозволяє розширити можливості фреймворку;
- тестова підтримка: Angular має вбудовану підтримку для тестування коду та взаємодії з користувачем;
- автоматичне виявлення змін: Angular має вбудований механізм автоматичного виявлення змін в даних, що дозволяє автоматично оновлювати

інтерфейс користувача при зміні даних;

- підтримка мобільних платформ: Angular підтримує розробку мобільних додатків для iOS та Android;
- широкі можливості: Angular має багато можливостей, включаючи роботу з формами, валідацію даних, роботу з HTTP-запитами та інші.

Вибір Angular зумовлено його порівнянням з аналогами(табл. 2.1)

Таблиця 2.1 Порівняння мов Angular, React та Vue.js

Мова	Angular	React	Vue.js
Тип	Фреймворк	Бібліотека	Фреймворк
Виділення	Широкий функціонал	Відображення UI	Простота використання
Виробник	Google	Facebook	Команда розробників
Управління станом	Стандартний підхід через RxJS та NgRx	Рекомендується використання Redux або MobX	Вбудована система управління станом
Компоненти	Angular компоненти	React компоненти	Vue компоненти
Складність	Висока	Середня	Низька

2.2 TypeScript

TypeScript - це мова програмування, яка є розширенням JavaScript, розроблена компанією Microsoft. TypeScript дозволяє розробникам створювати більш складні та структуровані додатки, забезпечуючи додаткові можливості, які не доступні у JavaScript. TypeScript компілюється до чистого JavaScript, що дозволяє виконувати його в будь-якому середовищі, яке підтримує JavaScript[7].

Основні переваги TypeScript:

- строга типізація: TypeScript має строгу типізацію, що дозволяє зменшити кількість помилок в коді та підвищити його надійність;
- орієнтація на об'єктно-орієнтоване програмування: TypeScript має

підтримку об'єктно-орієнтованого програмування, що дозволяє розробникам створювати складні додатки зі структурованою архітектурою;

- підтримка модульності: TypeScript підтримує модульність, що дозволяє розробникам легко організовувати код та підтримувати його;
- підтримка інструментів розробки: TypeScript має велику кількість інструментів розробки, таких як редактори коду, IDE та засоби відлагодження, що дозволяє розробникам зосередитися на створенні додатків замість вирішення проблем з інфраструктурою;
- підтримка розробки вебдодатків та мобільних додатків: TypeScript підтримує розробку вебдодатків та мобільних додатків на платформах, таких як Angular, React та Vue;
- вбудована підтримка нових стандартів ECMAScript: TypeScript підтримує нові стандарти ECMAScript, такі як ES6, ES7, ES8 та ESNext, що дозволяє використовувати нові функції та можливості JavaScript;
- підтримка мови програмування C#: TypeScript був розроблений з використанням підходу, схожого на мову програмування C#, тому розробники, які володіють C#, можуть швидко оволодіти TypeScript;
- підтримка вбудованої документації: TypeScript дозволяє створювати вбудовану документацію, що полегшує розробку та підтримку коду;
- розширені можливості для рефакторингу коду: TypeScript дозволяє використовувати розширені можливості для рефакторингу коду, такі як перейменування, знаходження посилань на змінні та інші;
- підтримка сторонніх бібліотек: TypeScript підтримує сторонні бібліотеки та має велику кількість оголошень типів для них, що дозволяє розробникам легко використовувати сторонні бібліотеки в своїх проектах[7].

Вибір TypeScript надано його порівнянням з аналогами(табл. 2.2)

Таблиця 2.2 Порівняння мов TypeScript, JavaScript та Flow

Мова	TypeScript	JavaScript	Flow
Типізація	Статична	Динамічна	Статична
Розширення	Надмножина JavaScript	-	-
Розробка	Забезпечує розширену функціональність, типізацію, інтелектуальні функції	Базова мова для веброзробки	Забезпечує статичну типізацію JavaScript
Компіляція	Так	Ні	Ні
Сумісність	Сумісний з JavaScript	Сумісний з TypeScript	Сумісний з JavaScript
Інтеграція	Вбудована в Visual Studio Code та інші IDE	Всюди	Вбудована в Flow IDE
Розробник	Microsoft	ECMA International	Facebook
Спільнота	Велика	Дуже велика	Значна

2.3 Asp net core

Asp net core - це відкритий фреймворк, що використовується для розробки вебдодатків та служб на мові програмування C#. Asp net core розроблений компанією Microsoft та є наступником Asp net core. Asp net core підтримує багато платформ, включаючи Windows, macOS та Linux[8].

Основні переваги Asp net core:

- кросплатформенність: Asp net core підтримує багато платформ, включаючи Windows, macOS та Linux, що дозволяє розробникам створювати вебдодатки, які працюють на будь-якій платформі;

- висока продуктивність: ASP.NET Core дозволяє створювати високопродуктивні вебдодатки та служби, що можуть обслуговувати велику кількість запитів;
- інтеграція з Azure: Asp net core добре інтегрується з хмарним сервісом Azure, що дозволяє розробникам швидко розгорнути свої вебдодатки та служби на Azure;
- підтримка сторонніх бібліотек: Asp net core підтримує сторонні бібліотеки, такі як Entity Framework, AutoMapper та інші;
- вбудована безпека: Asp net core має вбудовану підтримку безпеки, що дозволяє розробникам забезпечити безпеку своїх вебдодатків та служб;
- швидке виконання: Asp net core має оптимізований процес виконання, що дозволяє розробникам створювати високопродуктивні вебдодатки та служби;
- підтримка різних протоколів: Asp net core підтримує різні протоколи, такі як HTTP, WebSocket та SignalR, що дозволяє розробникам створювати різноманітні вебдодатки та служби;
- широкий вибір засобів для фронтенду: Asp net core підтримує різні засоби для фронтенду, такі як Angular, React та Vue, що дозволяє розробникам використовувати засоби, які найбільше підходять для їхніх потреб;
- можливість розгортання на різних серверах: Asp net core дозволяє розробникам розгорнути вебдодатки та служби на різних серверах, включаючи IIS, Apache та Nginx;
- широкий вибір засобів для розгортання: Asp net core підтримує різні засоби для розгортання, такі як Docker та Kubernetes, що дозволяє розробникам швидко та легко розгорнути свої вебдодатки та служби на будь-якій інфраструктурі[8].

Вибір Asp net core продемонстровано його порівнянням з аналогами(табл. 2.3)

Таблиця 2.3 Порівняння фреймворків Asp net core, Express.js та Django

Фреймворк	Asp net core	Express.js	Django
Мова	C#, F#	JavaScript	Python
Виділення	Повний стек фреймворк	Мінімалістичний фреймворк для Node.js	Повний стек фреймворк
Розробка	Орієнтована на Microsoft-платформи	Орієнтована на Node.js	Орієнтована на Python
Масштабованість	Висока	Залежить від розробника	Висока
Спільнота	Велика	Велика	Велика
Швидкодія	Висока	Висока	Висока
Безпека	Вбудована система безпеки та авторизації	Залежить від розробника	Вбудована система безпеки та авторизації
Шаблони	Razor, MVC, Web API	Залежить від розробника	Django шаблони
Документація	Широка	Доступна, але менш повна	Широка

2.4 Entity Framework

Entity Framework (EF) - це фреймворк, що використовується для роботи з базами даних на платформі .NET. EF дозволяє розробникам працювати з базами даних на рівні об'єктів, замість прямої роботи з SQL-запитами. EF є частиною платформи .NET та підтримується компанією Microsoft[9].

Основні переваги Entity Framework:

- простота використання: EF дозволяє розробникам працювати з базами даних на рівні об'єктів, що дозволяє спростити роботу з базами даних та зменшити кількість коду.

- кроссплатформенність: EF підтримує багато баз даних, включаючи Microsoft SQL Server, MySQL та PostgreSQL, що дозволяє розробникам працювати з базами даних на різних платформах;
- швидкість розробки: EF дозволяє розробникам швидко розробляти та змінювати моделі баз даних без необхідності написання складних SQL-запитів;
- можливість роботи зі сторонніми бібліотеками: EF підтримує роботу зі сторонніми бібліотеками, такими як AutoMapper, Ninject та інші;
- підтримка LINQ: EF підтримує мову запитів LINQ, що дозволяє розробникам писати запити до бази даних на мові C#;
- можливість міграції баз даних: EF дозволяє розробникам використовувати міграції баз даних, що дозволяє зберігати та контролювати структуру баз даних та її зміни в ході розробки;
- можливість роботи з багатьма джерелами даних: EF дозволяє розробникам працювати з багатьма джерелами даних, такими як бази даних, вебслужби та інші;
- підтримка транзакцій: EF підтримує транзакції, що дозволяє розробникам працювати з базами даних з більшою безпекою та надійністю;
- підтримка зв'язків між таблицями: EF дозволяє розробникам працювати зі зв'язками між таблицями, що дозволяє зменшити кількість коду та спростити роботу з базою даних;
- відкритий код: EF є відкритим фреймворком, що дозволяє розробникам вносити внески до коду та використовувати його у власних проектах.

Вибір Entity Framework приведено його порівнянням з аналогами(табл. 2.4)

Таблиця 2.4 Порівняння фреймворків Entity Framework, NHibernate та Dapper

Фреймворк	Entity Framework	NHibernate	Dapper
Тип	ORM (Object-Relational Mapping)	ORM (Object-Relational Mapping)	Мікро-ORM (Micro-ORM)
Компанія-розробник	Microsoft	Hibernate community	StackExchange
Мова	C#	C#, Java	C#, .NET
Спрощена робота з базою даних	Так	Так	Так
Продуктивність	Висока	Висока	Дуже висока
Розширені можливості мапування	Так	Так	Обмежені
Контроль над SQL-запитами	Середній	Високий	Високий
Швидкодія	Порівняно середня	Порівняно висока	Порівняно висока
Документація	Широка	Широка	Задokumentована
Спільнота	Широка	Велика	Середня

2.5 C#

C# - це мова програмування, розроблена компанією Microsoft. Вона є частиною .NET-платформи та використовується для розробки різноманітних програм, від настільних до вебдодатків та мобільних додатків[10].

Основні переваги мови програмування C#:

- безпека: C# дозволяє розробникам створювати безпечні програми завдяки вбудованим механізмам обробки помилок та контролю пам'яті;
- об'єктно-орієнтований підхід: C# є мовою з об'єктно-орієнтованим підходом, що дозволяє створювати складні програми з використанням класів та об'єктів;
- інтеграція зі засобами розробки: C# може використовуватися з різними засобами розробки, такими як Visual Studio, Visual Studio Code та інші;
- платформонезалежність: програми на C# можуть працювати на різних платформах, включаючи Windows, Linux та macOS;
- широкий спектр застосування: C# може використовуватися для розробки різноманітних програм, включаючи вебдодатки, мобільні додатки, настільні додатки, ігри та інші;
- легкість вивчення: C# є досить легкою мовою для вивчення програмування, оскільки вона має простий синтаксис та багато інструментів для підтримки;
- підтримка мультипоточності: C# має вбудовану підтримку мультипоточності, що дозволяє створювати багатопоточні програми з легкістю;
- підтримка LINQ: C# має вбудовану підтримку LINQ, що дозволяє виконувати запити до бази даних та колекцій даних без необхідності писати SQL-запити;
- інтероперабельність: C# підтримує інтероперабельність з іншими мовами програмування, такими як C++ та Python.

Вибір C# зумовлено його порівнянням з аналогами(табл. 2.5)

Таблиця 2.5 Порівняння мов C#, C++ та C

Мова	C#	C++	C
Тип	Об'єктно-орієнтована	Об'єктно-орієнтована	Процедурна
Статична типізація	Є	Є	Ні
Підтримка мультипарадигми	Є	Є	Ні
Використання пам'яті	Управляється	Вручну керується	Вручну керується
Компіляція	Високорівнева	Високорівнева	Низькорівнева

2.6 Вибір засобів розробки

Розробку вебзастосунку заплановано на основі мови TypeScript для фронтенду через її придатність для вирішення широкого кола задач і зокрема створення вебзастосунків, які за рахунок використання вебфреймворку придатні для використання у бізнесах різного рівня, хоча на рівень впливає і вибір фреймворка[11]. Для бекенду було обрано мову C# через її швидкодійність, можливість працювати в декілька потоків, гнучкість при використанні шаблонів проектування, та орієнтацію на високонавантажені застосунки[10]. Тому тепер потрібно порівняти (табл. 2.6) два найпопулярніші фреймворки веброботи для TypeScript: Angular та React[6][12]. Також необхідно порівняти (табл. 2.7) дві найпопулярніші технології мови C#: ASP.Net.Core[13], а саме WebApi та Blazor[14].

Таблиця 2.6 – Вибір вебфреймворку розробки мовою TypeScript

Показник	Angular	React
Високорівневий API	+	–
Висока оптимізація	+	–
Кросплатформеність	–	+
Формування динамічних вебсторінок	+	+
Використовувана мова програмування	TypeScript	TypeScript

Таблиця 2.7 – Вибір технології розробки мовою C#

Показник	ASP.Net.CoreWeb API	ASP.Net.CoreBlazor
Висока оптимізація	+	+
Кросплатформеність	+	+
Використання стороннього фреймворку для фронтенду	+	–
Використовувана мова програмування	C#	C#

Вибір у підсумку фреймворку Angular обґрунтований його основними перевагами, що включають високорівневий API, завдяки чому спрощено початок розробки вебзастосунку через відсутність необхідності самостійного встановлення великої кількості бібліотек. Високий рівень оптимізації також є важливим показником для розробки високо навантаженої системи. Враховуючи вибір цього фреймворку, за основу було взято компонентну архітектуру для фронтенду[15].

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Розробка бази даних

Розроблювана БД має містити в собі інформацію про користувача, новини, скріншоти, коментарі, тегі, платформи та улюблені фільми користувача. Тобто ми можемо виділити наступні сутності, які будуть зберігатися у БД:

- Movies;
- Comments;
- News;
- NewsComments;
- Actors;
- ActorMovie;
- Genres;
- GenresMovie;
- Users;
- UserMovies.

Таблиця Movies містить інформацію про фільми, анімаційні фільми та телесеріали(табл. 3.1).

Таблиця 3.1 – Атрибути таблиці Movies

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
id	bigint	+	Bigint max	Первинний ключ
Name	text	+	255	
PosterPath	text	+	255	

Продовження таблиці 3.1

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Overview	text	+	255	
TrailerPath	text	+	255	

Таблиця Comments містить коментарі користувачів щодо фільмів (табл. 3.2).

Таблиця 3.2 – Атрибути таблиці Comments

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Id	bigint	+	Bigint max	Первинний ключ
CommentText	text	+	255	
MovieId	bigint	+	Bigint max	Зовнішній ключ
UserId	bigint	+	Bigint max	Зовнішній ключ

Таблиця News містить інформацію про новини (табл. 3.3).

Таблиця 3.3 – Атрибути таблиці News

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Id	bigint	+	Bigint max	Первинний ключ
Title	text	+	255	
Description	text	+	255	
ImgPath	text	+	255	
MovieId	bigint	+	Bigint max	Зовнішній ключ

Таблиця NewsComments містить коментарі користувачів щодо новин(табл. 3.4).

Таблиця 3.4 – Атрибути таблиці NewsComments

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
id	bigint	+	Bigint max	Первинний ключ
Comment	text	+	255	
NewsId	bigint	+	Bigint max	Зовнішній ключ
UserId	bigint	+	Bigint max	Зовнішній ключ

Таблиця Actors містить інформацію щодо акторів(табл. 3.5).

Таблиця 3.5 – Атрибути таблиці Actors

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Id	bigint	+	Bigint max	Первинний ключ
FirstName	text	+	255	
LastName	text	+	255	

Таблиця ActorMovie містить інформацію щодо акторів, які грають чи озвучують ролі у фільмах, анімаційних фільмах або телесеріалах(табл. 3.6).

Таблиця 3.6 – Атрибути таблиці ActorMovie

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
CastId	bigint	+	Bigint max	Первинний ключ
MoviesId	bigint	+	Bigint max	Зовнішній ключ

Таблиця Genres містить інформацію про жанри(табл. 3.7).

Таблиця 3.7 – Атрибути таблиці Genres

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Id	bigint	+	Bigint max	Первинний ключ
Name	bigint	+	Bigint max	

Таблиця GenreMovie містить інформацію про те, до якого жанру або жанрів відноситься фільм(табл. 3.8).

Таблиця 3.8 – Атрибути таблиці GenreMovie

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
GenresId	bigint	+	bigint max	Первинний ключ
MoviesId	bigint	+	bigint max	Зовнішній ключ

Таблиця Users містить інформацію про користувача(табл. 3.9).

Таблиця 3.9 – Атрибути таблиці Users

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Id	bigint	+	Bigint max	Первинний ключ
Email	text	+	255	
Password	text	+	255	
NickName	text	+	255	

Таблиця UserMovies містить інформацію про обрані фільми користувача, його оцінку, та його відгук про фільм(табл. 3.10).

Таблиця 3.10 – Атрибути таблиці UserMovie

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
Id	bigint	+	Bigint max	Первинний ключ
MovieId	bigint	+	Bigint max	
UserRating	integer	+	max	
UserDescription	text	+	255	

Розроблювана БД була спроектована за допомогою -PostGre EntityFramework(рис. 3.1).

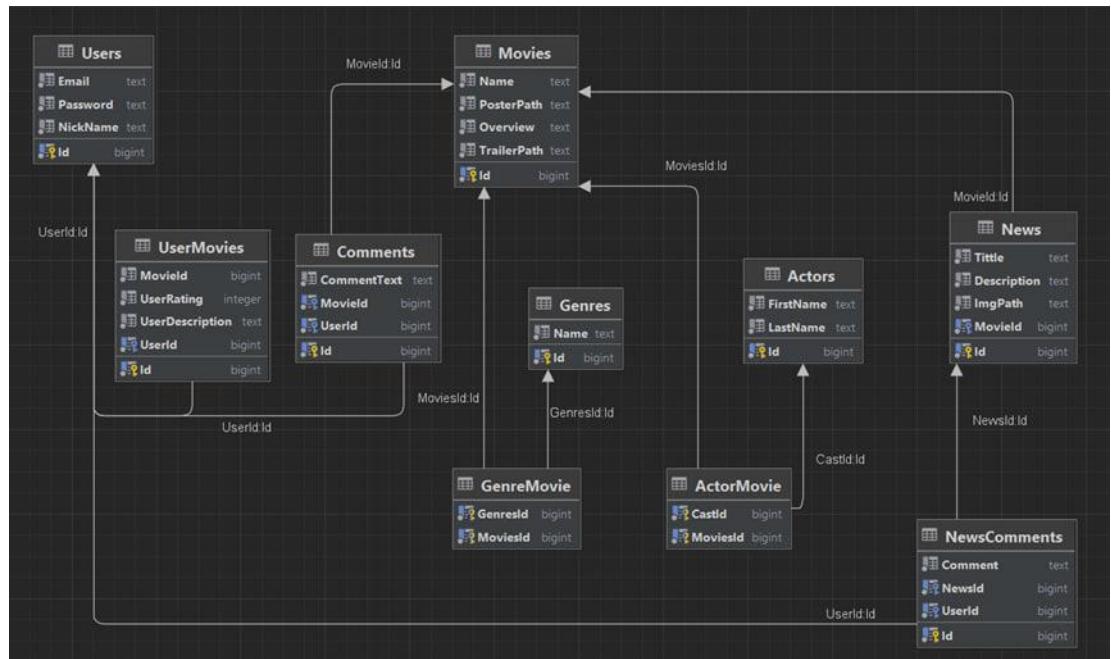


Рисунок 3.1 – Макет створеної БД.

3.2 Алгоритм функціонування програми

Процес роботи користувача виконано таким чином щоб йому було просто та зручно користуватись програмою. Інтерфейс був створений таким чином щоб кожний користувач міг інтуїтивно зрозуміти яка панель за що відповідає. Сайт складається з декількох сторінок, а саме:

- головна сторінка;
- сторінка реєстрації;
- сторінка входу;
- сторінка з фільмами;
- сторінка з новинами;
- сторінка з більш детальною інформацією що до новини.

Нижче показано схему функціонування програми(рис. 3.1):

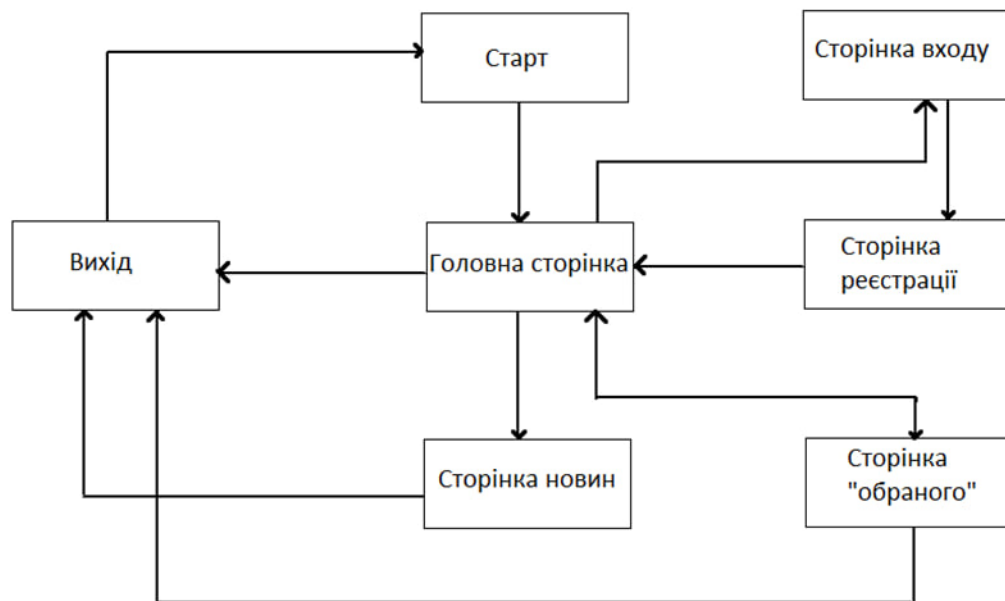


Рисунок 3.1 – Схема функціонування програми

3.3 Проектування бази даних

Проектування БД є важливою частиною проектування застосунку. Для створення БД було обрано EntityFramework[13] (EF) на базі технології .Net. EF має декілька підходів[14] для проектування БД:

- Code-first[15] – вперше з'явився в EntityFramework 4.1, зазвичай використовується, коли у вас вже є існуюча програма, що містить модель даних. Ця модель зазвичай описується за допомогою декількох класів і коду взаємодії між цими класами;

- Model-First – вперше з'явився у версії EntityFramework 4, застосовується розробниками, які не хочуть використовувати інструменти СУБД для створення та управління базами даних, а також не хочуть вручну налаштовувати класи моделі EDM. Фактично це найпростіший підхід під час роботи з EntityFramework. Проектування моделі відбувається у графічному дизайнері EDM середовища VisualStudio;

- Database-First – робочий процес створення моделі починається зі створення та проектування бази даних. Після створення сутнісних класів

моделі з існуючої бази даних, робота з EntityFramework аналогічна підходам Code-First і Model-First. Це означає створення об'єкта класу контексту та використання цього об'єкта для виконання необхідних завдань.

Для розробки було обрано підхід Code-first через відсутність необхідності роботи зі сторонніми засобами окрім середи розробки та кращу сумісність створенної БД з кодом, що було написано через відсутність конфліктів типів даних.

Нижче приведені список полів у БД та що вони виконують.

Таблиця Movies містить поля:

- Name – назва фільму;
- PosterPath – постер фільму;
- Overview – опис фільму;
- Rating – рейтинг гри;
- TrailerPath – трейлер фільму;
- Id – ідентифікатор фільму.

Таблиця Comments містить поля:

- CommentText – текст коментаря від користувача;
- MovieId – ідентифікатор фільму;
- UserId – ідентифікатор користувача;
- Id – ідентифікатор коментаря.

Таблиця News містить поля:

- Tittle – назва фільму для якого додано новину;
- Description – опис новини;
- ImgPath – фото новини;
- MovieId – ідентифікатор фільму на який додана новина;
- Id – ідентифікатор новини.

Таблиця NewsComments містить поля:

- Comment – текст коментаря;
- NewsId – ідентифікатор новини для якої додано коментар;
- UserId – ідентифікатор користувача який додав коментар;

- Id – ідентифікатор коментаря.

Таблиця Actors містить поля:

- FirstName – ім'я актора;
- LastName – прізвище актора;
- Id – ідентифікатор актора.

Таблиця ActorMovie містить поля:

- CastId – ідентифікатор актора;
- MoviesId – ідентифікатор фільму.

Таблиця Genres містить поля:

- Name – назва жанру;
- Id – ідентифікатор жанру.

Таблиця GenreMovie містить поля:

- GenresId – ідентифікатор жанру;
- MoviesId – ідентифікатор фільму.

Таблиця Users містить поля:

- Email – пошта користувача;
- Password – пароль користувача;
- NickName – ім'я користувача;
- Id – ідентифікатор користувача.

Таблиця UserMovies містить поля:

- MovieId – ідентифікатор фільму;
- UserRating – рейтинг фільму який виставив користувач;
- UserDescription – опис фільму який додав собі користувач;
- UserId – ідентифікатор користувача;
- Id – ідентифікатор «обраного».

4 ЕКСПЛУАТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Характеристики програми

У розробленому вебзастосунку було реалізовано усі потрібні складові для коректної роботи сервісу. Основні серед них є наступні:

- Реєстрація та авторизація користувачів;
- можливість додавання коментарів ;
- перегляд інформації о новинах та списку фільмів;
- додавання інформації до списку обраного;
- додавання новин;
- пошук інформації.

4.2 Запуск застосування

Запуск програми відбувається на будь-якій операційній системі, що має підтримку бібліотеки пакетів dot-net. Перед запуском необхідно використати файл prelaunch.bat, що автоматично вставить усі необхідні для роботи програми бібліотеки, та налаштує середовище перед запуском. Назви та версії усіх необхідних бібліотек описані у файлі package.json в обох проектах.

Щоб запустити програму потрібно окремо запустити проекти Assets.WebApi та Frontend. Проект Assets.WebApi запускається за допомогою консольної команди dot-netstart. Запуск проекту Frontend відбувається за допомогою команди ngstart.

4.3 Загальні відомості про програму

Програма призначена для спрощення пошуку інформації що до телеіндустрії.

Додаток було створено за допомогою середовищ розробки JetBrainsRider, JetBrainsWebStorm та JetBrainsIntelliJ IDEA.

Базу даних було створено за допомогою EntityFramework.

Підключення таблиць бази даних на форму проекту було виконано за допомогою EntityFramework та HTTP.

Підключення до бази даних було організоване за допомогою строки з'єднання.

4.4 Структура програми

Програма складається з двох проектів:

–Assets.WebAPI – проект бекенду на базі Asp.Net CoreWeb API, що містить в собі основні контролери та функції програми, створення БД, міграції та допоміжні бібліотеки;

–Frontend – проект на базі Angular 14, що містить в собі основні компоненти програми з відображення даних.

Зв'язок між проектами Assets.WebAPI та Frontend відбувається за допомогою HTTPпротоколу.

4.5 Тестування програми

Тестування програми дозволило перевірити роботу програми щодо виконання функціональних можливостей, які передбачались у процесі розробки. Тестування спочатку виконувалось під час реалізації цих можливостей і дозволяло виявляти проблеми і вирішувати їх. У подальшому, після того, як всі такі проблеми було усунуто, було проведено окремий етап тестування з перевіркою виконання всіх функцій, тому успішність реалізації була підтверджена результатами тестування на персональному комп'ютері під керуванням операційної системи Windows 11 Pro. Та за допомогою unit-тестування, інтеграційного тестування та тестування збірки.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Загалом призначенням програми є створення інструменту для пошуку та ознайомлення ресурсів телеіндустрії з відстеженням новин про неї.

Інформаційний сайт про фільми часто публікує огляди та рецензії на нові фільми. Це дозволяє глядачам отримати об'єктивну інформацію про фільми, їх якість, сюжет, графіку та інші аспекти перед переглядом або вирішенням, чи варто йти до кінотеатру або купляти доступ до перегляду на різних сайтах.

Поради та стратегії сайтів про фільми надає поради, стратегії та вказівки для глядачів. Це допомагає глядачам поліпшити свої знання у світі телеіндустрії, та розрізняти гарні фільми від поганих, цим самим зберегти свій час.

5.2 Умови виконання програми

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять об'ємом 16 Гб, жорсткий диск об'ємом 120 Гб.

5.3 Експлуатація програми

Робота користувача з програмою є достатньо простою. Кожен користувач має абсолютно однаковий інтерфейс. Для того щоб отримати доступ до системи, йому потрібно зареєструватися, а потім авторизуватися.

На будь який елемент можна натиснути та переглянути інформацію, будь це фільм, новини, обране або реєстрація.

Одним із перших і головних вікон програми є головне вікно зі списком фільмів та навігаційна панель (рис. 5.1).

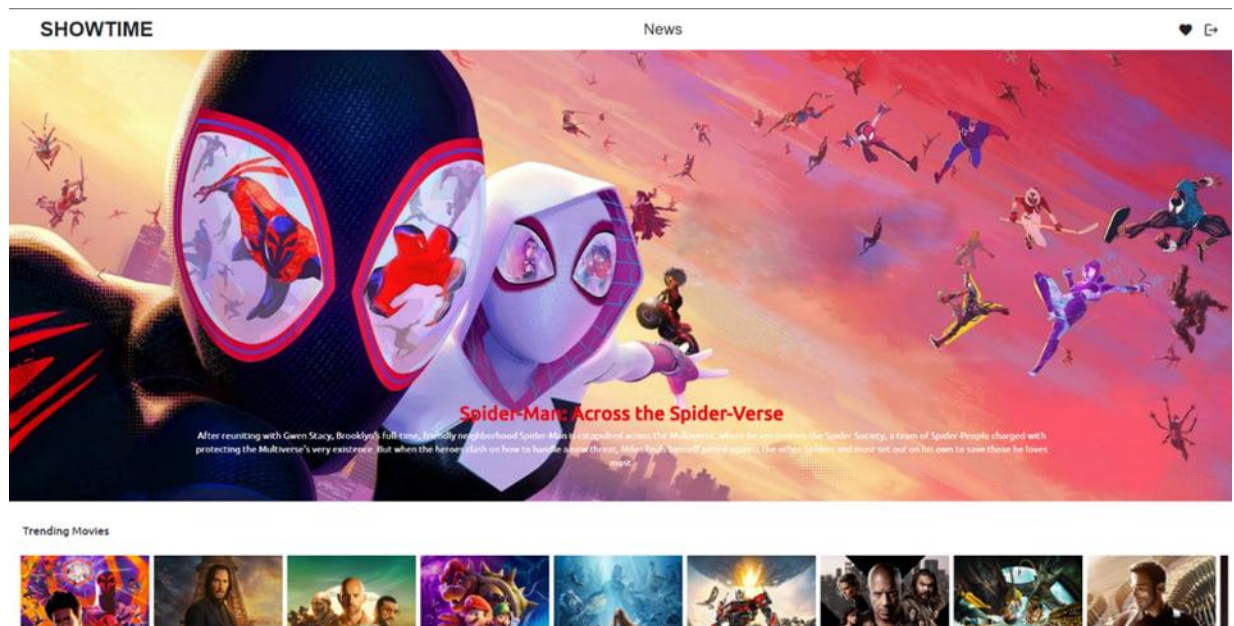


Рисунок 5.1 – Вікно головної сторінки

При натисканні на кнопку News можна побачити список новин про телеіндустрію та додати ще новину(рис 5.2).

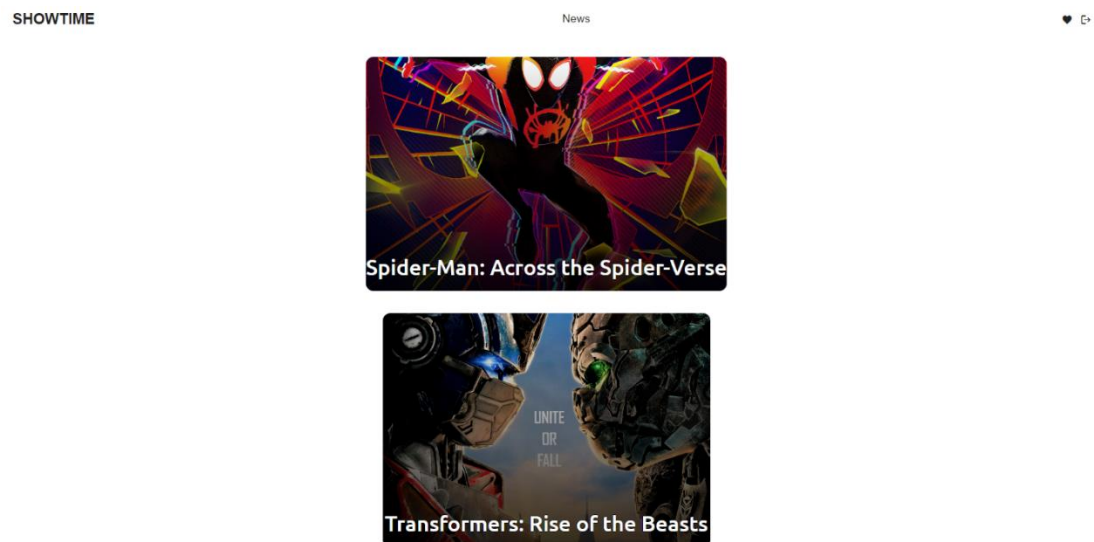


Рисунок 5.2 – Вікно сторінки новин

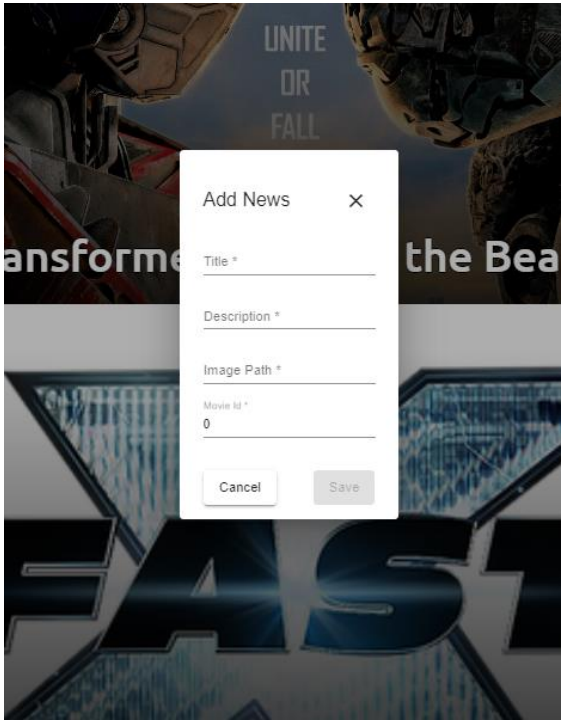


Рисунок 5.3 – Діалогове вікно додавання новини

При натисканні на фільм можна побачити детальну інформацію про нього(рис 5.4).

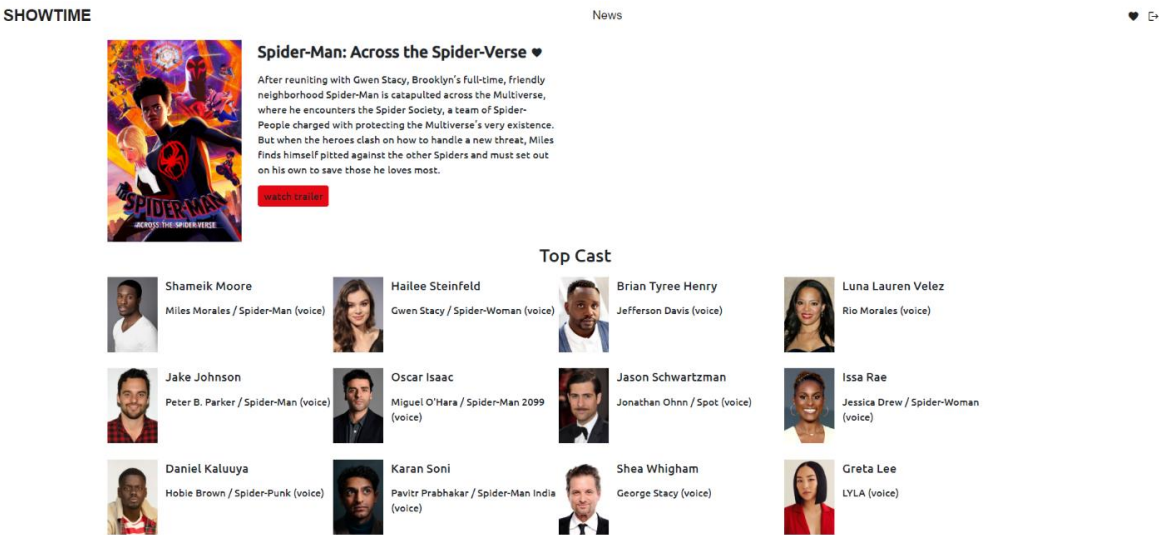


Рисунок 5.4 – Вікно інформації про фільм

Під кожною новиною або фільмом можна додати коментар(рис. 5.5).

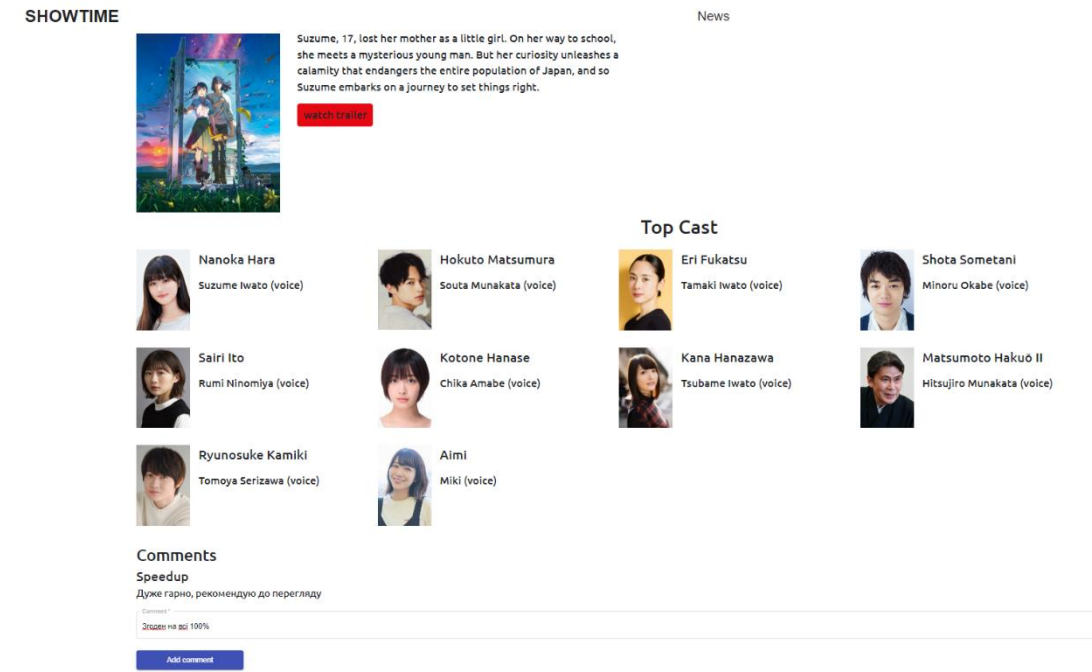


Рисунок 5.5 – Вікно коментарів фільму

При натисканні на блок новини можна побачити інформацію про фільм та розділ коментарів(рис 5.6).

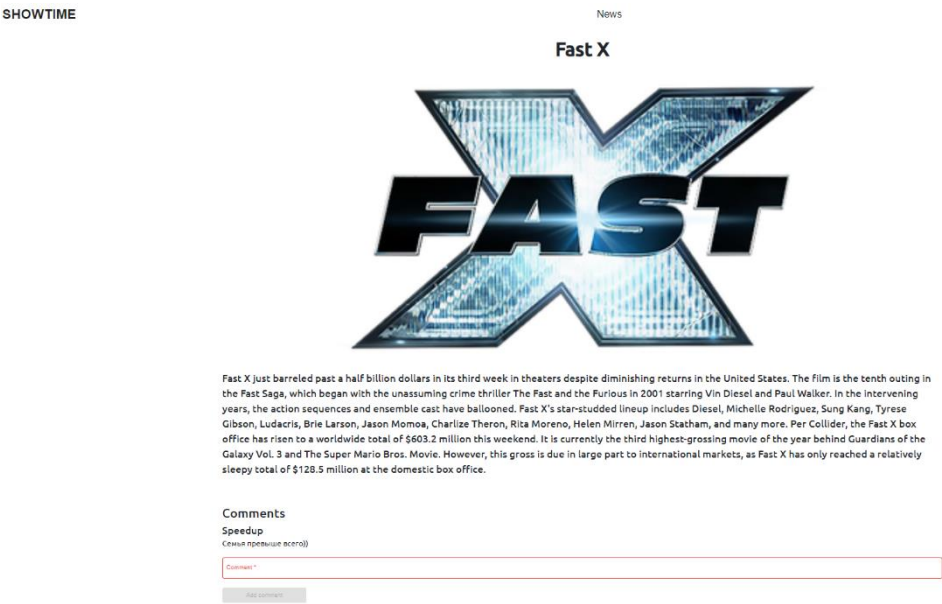
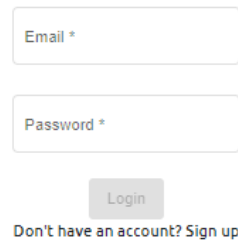


Рисунок 5.6 – Вікно новин фільму

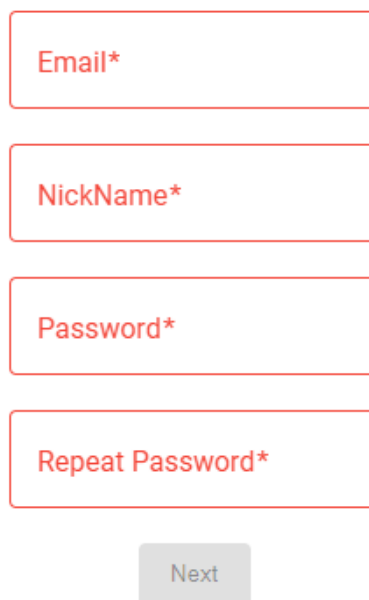
При натисканні на кнопку логіну можна перейти на сторінку входу до аккаунту або реєстрації якщо ви не маєте аккаунту(рис. 5.7).



The login form consists of two text input fields: 'Email *' and 'Password *'. Below these fields is a grey 'Login' button. At the bottom, there is a link that reads 'Don't have an account? Sign up'.

Рисунок 5.7 – Вікно входу

При натисканні на текст Sign up відкриється вікно реєстрації(рис. 5.8).



The registration form contains four text input fields: 'Email*', 'NickName*', 'Password*', and 'Repeat Password*'. Below these fields is a grey 'Next' button.

Рисунок 5.8 – Вікно реєстрації

При натискані на кнопку «серце» можна побачити список обраних фільмів, ваш коментар та рейтинг який ви вказали(рис. 5.9).



Рисунок 5.9 – Вікно «обраного»

ВИСНОВКИ

В результаті розробки було створено програмне забезпечення вебсайту з інформацією про фільми, анімаційні фільми та телесеріали, які допомагають глядачам отримати доступ до важливої інформації, такої як огляди, рецензії, новини та поради щодо фільмів. За результатами аналізу проблеми виконано аналіз аналогів і їх порівняння з розробкою, визначено її особливості.

За результатами проектування представлено результати порівняння фреймворків для розробки вебзастосунків мовою TypeScript та C#, обрано фреймворк Angular у цій якості, що визначило структуру програми на основі використання шаблону проектування MVC, представлено функціональні вимоги до програми, діаграму варіантів використання, схему бази даних і описано її відповідні таблиці і зв'язки між ними.

За результатами реалізації програми подано схему її функціонування, детально описано створені одиниці програми, включаючи класи моделей і уявлень, та загальну логіку роботи користувача. Під час виконання розробки та за результатами її виконання реалізовано тестування програми, що дозволило отримати результат, який відповідає запланованим для реалізації у технічному завданні вимогам.

Розроблена програма дозволила в результаті створити інструмент для зручного та швидкого огляду інформації щодо фільмів, анімаційних фільмів, телесеріалів та телеіндустрії у цілому, реалізовано додавання фільмів до «обраних», додавання новин та коментарів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Review of the films documentanion [Electronic resource] – Access mode:
<https://www.thebalancecareers.com>
2. Rotten Tomatoes [Electronic resource] – Access mode:
<https://www.rottentomatoes.com/>
3. IMDb [Electronic resource] – Access mode: <https://www.imdb.com/>
4. Metacritic [Electronic resource] – Access mode:
<https://www.metacritic.com/>
5. Letterboxd [Electronic resource] – Access mode: <https://letterboxd.com/>
6. Angular Офіційна документація [Electronic resource] – Access mode:
<https://angular.io/>
7. Typescript documentation [Electronic resource] – Access mode:
<https://www.typescriptlang.org>
8. ASP.Net Core documentation [Electronic resource] – Access mode:
<https://dotnet.microsoft.com/apps/aspnet>
9. Entity Framework documentation [Electronic resource] – Access mode:
<https://docs.microsoft.com/en-us/ef/>
10. C# Офіційна документація [Electronic resource] – Access mode:
<https://docs.microsoft.com/en-us/dotnet/csharp/>
11. Стаття "Введение в TypeScript". [Electronic resource] – Access mode:
<https://metanit.com/web/typescript/1.1.php>
12. React documentation [Electronic resource] – Access mode:
<https://react.dev/>
13. ASP.Net Core documentation [Electronic resource] – Access mode:
<https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0>
- 14 Blazor for ASP.NET Core documentation [Electronic resource] – Access mode: <https://docs.microsoft.com/en-us/aspnet/core/blazor>

15. Code First to a New Database using Entity Framework [Electronic resource] – Access mode: <https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database>

ДОДАТОК А
Текст програми

```

using Imdb.WebApi.Dtos;
using Imdb.WebApi.Persistence.Entities;
using Imdb.WebApi.Persistence.Repository;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace Imdb.WebApi.Controllers;

[ApiController]
[Route("[controller]")]
public class AuthController : ControllerBase
{
    private readonly IImdbRepository _repository;

    public AuthController(
        ILogger<AuthController> logger,
        IImdbRepository repository,
        IConfiguration configuration)
    {
        _repository = repository;
    }

    [HttpPost("login")]
    public async Task<IActionResult> Login(LoginRequestDto requestDto)
    {
        var user = await _repository.DbContext.Users.FirstOrDefaultAsync(x => x.Email
== requestDto.Email);
        if (user == null)
        {
            return BadRequest("User not found");
        }
        if (user.Password == requestDto.Password)
        {
            return Ok(new { user.Email, userId = user.Id, user.NickName });
        }

        return BadRequest("Email or password not correct");
    }

    [HttpPost("signup")]
    public async Task<IActionResult> SignUp(SignupRequestDto requestDto)
    {
        var user = await _repository.DbContext.Users.FirstOrDefaultAsync(x => x.Email
== requestDto.Email);
        if (user != null)
        {
            return BadRequest("User already exist");
        }
        await _repository.CreateAsync(new User()
        {
            Email = requestDto.Email,
            Password = requestDto.Password,

```

```

        NickName = requestDto.NickName
    });
    return Ok("User successfully registered");
}
}
using Imdb.WebApi.Dtos;
using Imdb.WebApi.Persistence.Entities;
using Imdb.WebApi.Persistence.Repository;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace Imdb.WebApi.Controllers;

[ApiController]
[Route("[controller]")]
public class MovieController : ControllerBase
{
    private readonly IImdbRepository _repository;

    public MovieController(
        ILogger<MovieController> logger,
        IImdbRepository repository,
        IConfiguration configuration)
    {
        _repository = repository;
    }

    [HttpPost("addToFavourite")]
    public async Task<IActionResult> AddToFavourite(AddToFavouriteRequestDto
requestDto, [FromQuery] long userId)
    {
        var movie = await _repository.DbContext.UserMovies
            .FirstOrDefaultAsync(x => x.MovieId == requestDto.MovieId && x.UserId ==
userId);

        if (movie != null)
        {
            return BadRequest("already in fav");
        }

        var user = await _repository.DbContext.Users.FirstOrDefaultAsync(x => x.Id ==
userId);

        if (user != null)
        {
            await _repository.CreateAsync(new UserMovie()
            {
                UserId = userId,
                MovieId = requestDto.MovieId,
                UserRating = requestDto.Rating,
                MovieName = requestDto.MovieName,
                UserDescription = requestDto.Description,
                User = user
            });
        }
    }
}

```

```

        return Ok("ooh yeah");
    }

    return BadRequest("cannot find user");
}

[HttpGet("getFavourites")]
public async Task<IActionResult> GetFavourites([FromQuery] long userId)
{
    var movies = await _repository.DbContext.UserMovies.Where(x => x.UserId ==
userId).ToListAsync();
    return Ok(movies);
}

[HttpPost("createMovieComment")]
public async Task<IActionResult> CreateMovieComment([FromQuery] long userId,
[FromBody] CreateCommentRequestDto dto)
{
    var user = await _repository.DbContext.Users.FirstOrDefaultAsync(x => x.Id ==
userId);
    if (user != null)
    {
        var comment = new Comment()
        {
            User = user,
            UserId = user.Id,
            CommentText = dto.CommentText,
            MovieId = dto.TargetId
        };
        var created = await _repository.CreateAsync(comment);
        return Ok(created);
    }

    return NoContent();
}

[HttpPost("createNewsComment")]
public async Task<IActionResult> CreateNewsComment([FromQuery] long userId,
[FromBody] CreateCommentRequestDto dto)
{
    var user = await _repository.DbContext.Users.FirstOrDefaultAsync(x => x.Id ==
userId);
    if (user == null)
    {
        return BadRequest();
    }

    var news = await _repository.DbContext.News.FirstOrDefaultAsync(x => x.Id ==
dto.TargetId);
    if (news == null)
    {
        return BadRequest();
    }

```

```

    }

    var comment = new NewsComment()
    {
        User = user,
        UserId = user.Id,
        Comment = dto.CommentText,
        NewsId = news.Id,
        News = news
    };
    var created = await _repository.CreateAsync(comment);

    return Ok(created);
}

[HttpPost("getMovieComments")]
public async Task<IActionResult> GetMovieComments([FromBody] long movieId)
{
    var comments = await _repository.DbContext.Comments.Include(x =>
x.User).Where(x => x.MovieId == movieId).ToListAsync();
    return Ok(comments.Select(x => new CommentResponse()
    {
        Comment = x.CommentText,
        NickName = x.User.NickName
    }));
}

[HttpPost("getNewsComments")]
public async Task<IActionResult> GetNewsComments([FromBody] long newsId)
{
    var comments = await _repository.DbContext.NewsComments.Include(x =>
x.User).Where(x => x.NewsId == newsId).ToListAsync();
    return Ok(comments.Select(x => new CommentResponse()
    {
        Comment = x.Comment,
        NickName = x.User.NickName
    }));
}

[HttpPost("createNews")]
public async Task<IActionResult> CreateNews([FromBody]
CreateNewsRequestDto dto)
{
    var news = new News()
    {
        Title = dto.Title,
        Description = dto.Description,
        ImgPath = dto.ImgPath,
        MovieId = dto.MovieId,
    };
    var created = await _repository.CreateAsync(news);

```

```

        return Ok(created);
    }

    [HttpGet("getNews")]
    public async Task<IActionResult> GetNews()
    {
        var news = await _repository.DbContext.News.Include(x =>
x.NewsComments).ThenInclude(x => x.User).ToListAsync();
        return Ok(news.Select(x => new NewsResponseDto()
        {
            Id = x.Id,
            Description = x.Description,
            Tittle = x.Tittle,
            ImgPath = x.ImgPath,
            MovieId = x.MovieId,
            Comments = x.NewsComments.Select(z => new CommentResponse()
            {
                Comment = z.Comment,
                NickName = z.User.NickName
            }).ToList()
        }));
    }

    [HttpPost("getNewsDetails")]
    public async Task<IActionResult> GetNews([FromBody] long newsId)
    {
        var news = await _repository.DbContext.News.Include(x =>
x.NewsComments).ThenInclude(x => x.User).FirstOrDefaultAsync(x => x.Id == newsId);
        if (news == null)
        {
            return NoContent();
        }
        return Ok(new NewsResponseDto()
        {
            Id = news.Id,
            Description = news.Description,
            Tittle = news.Tittle,
            ImgPath = news.ImgPath,
            MovieId = news.MovieId,
            Comments = news.NewsComments.Select(z => new CommentResponse()
            {
                Comment = z.Comment,
                NickName = z.User.NickName
            }).ToList()
        }));
    }
}

using Microsoft.AspNetCore.Mvc;

namespace Imdb.WebApi.Controllers;

[ApiController]

```

```

[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private static readonly string[] Summaries = new[]
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot",
        "Sweltering", "Scorching"
    };

    private readonly ILogger<WeatherForecastController> _logger;

    public WeatherForecastController(ILogger<WeatherForecastController> logger)
    {
        _logger = logger;
    }

    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
    {
        return Enumerable.Range(1, 5).Select(index => new WeatherForecast
        {
            Date = DateTime.Now.AddDays(index),
            TemperatureC = Random.Shared.Next(-20, 55),
            Summary = Summaries[Random.Shared.Next(Summaries.Length)]
        })
        .ToArray();
    }
}

namespace Imdb.WebApi.Dtos;

public class AddToFavouriteRequestDto
{
    public string Description { get; set; }
    public int Rating { get; set; }
    public long MovieId { get; set; }
    public string MovieName { get; set; }
}

namespace Imdb.WebApi.Dtos;

public class CommentResponse
{
    public string NickName { get; set; }
    public string Comment { get; set; }
}

namespace Imdb.WebApi.Dtos;

public class CreateCommentRequestDto
{
    public string CommentText { get; set; }
    public long TargetId { get; set; }
}

namespace Imdb.WebApi.Dtos;

```

```

public class CreateNewsRequestDto
{
    public string Tittle { get; set; }
    public string Description { get; set; }
    public string ImgPath { get; set; }
    public long MovieId { get; set; }
}
namespace Imdb.WebApi.Dtos;

public class LoginRequestDto
{
    public string Email { get; set; }
    public string Password { get; set; }
}
namespace Imdb.WebApi.Dtos;

public class NewsResponseDto
{
    public long Id { get; set; }
    public string Tittle { get; set; }
    public string Description { get; set; }
    public string ImgPath { get; set; }
    public long MovieId { get; set; }
    public ICollection<CommentResponse> Comments { get; set; } = new
List<CommentResponse>();
}
namespace Imdb.WebApi.Dtos;

public class SignupRequestDto
{
    public string Email { get; set; }
    public string Password { get; set; }
    public string NickName { get; set; }
}
using Microsoft.EntityFrameworkCore.Migrations;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace Imdb.WebApi.Migrations
{
    public partial class InititalMigration : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Actors",
                columns: table => new
                {
                    Id = table.Column<long>(type: "bigint", nullable: false)

```

```

        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        FirstName = table.Column<string>(type: "text", nullable: false),
        LastName = table.Column<string>(type: "text", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Actors", x => x.Id);
    });

migrationBuilder.CreateTable(
    name: "Genres",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        Name = table.Column<string>(type: "text", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Genres", x => x.Id);
    });

migrationBuilder.CreateTable(
    name: "Movies",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        Name = table.Column<string>(type: "text", nullable: false),
        PosterPath = table.Column<string>(type: "text", nullable: false),
        Overview = table.Column<string>(type: "text", nullable: false),
        TrailerPath = table.Column<string>(type: "text", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Movies", x => x.Id);
    });

migrationBuilder.CreateTable(
    name: "Users",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
        .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        Email = table.Column<string>(type: "text", nullable: false),
        Password = table.Column<string>(type: "text", nullable: false)
    },
    constraints: table =>

```

```

    {
        table.PrimaryKey("PK_Users", x => x.Id);
    });

migrationBuilder.CreateTable(
    name: "ActorMovie",
    columns: table => new
    {
        CastId = table.Column<long>(type: "bigint", nullable: false),
        MoviesId = table.Column<long>(type: "bigint", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_ActorMovie", x => new { x.CastId, x.MoviesId });
        table.ForeignKey(
            name: "FK_ActorMovie_Actors_CastId",
            column: x => x.CastId,
            principalTable: "Actors",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_ActorMovie_Movies_MoviesId",
            column: x => x.MoviesId,
            principalTable: "Movies",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "GenreMovie",
    columns: table => new
    {
        GenresId = table.Column<long>(type: "bigint", nullable: false),
        MoviesId = table.Column<long>(type: "bigint", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_GenreMovie", x => new { x.GenresId, x.MoviesId

});

        table.ForeignKey(
            name: "FK_GenreMovie_Genres_GenresId",
            column: x => x.GenresId,
            principalTable: "Genres",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_GenreMovie_Movies_MoviesId",
            column: x => x.MoviesId,
            principalTable: "Movies",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

```

```

migrationBuilder.CreateTable(
    name: "News",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
            .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        Title = table.Column<string>(type: "text", nullable: false),
        Description = table.Column<string>(type: "text", nullable: false),
        ImgPath = table.Column<string>(type: "text", nullable: false),
        MovieId = table.Column<long>(type: "bigint", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_News", x => x.Id);
        table.ForeignKey(
            name: "FK_News_Movies_MovieId",
            column: x => x.MovieId,
            principalTable: "Movies",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "Comments",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
            .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        CommentText = table.Column<string>(type: "text", nullable: false),
        MovieId = table.Column<long>(type: "bigint", nullable: false),
        UserId = table.Column<long>(type: "bigint", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Comments", x => x.Id);
        table.ForeignKey(
            name: "FK_Comments_Movies_MovieId",
            column: x => x.MovieId,
            principalTable: "Movies",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_Comments_Users_UserId",
            column: x => x.UserId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

```

```

migrationBuilder.CreateTable(
    name: "UserMovies",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
            .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        MovieId = table.Column<long>(type: "bigint", nullable: false),
        UserRating = table.Column<int>(type: "integer", nullable: false),
        UserDescription = table.Column<string>(type: "text", nullable: false),
        UserId = table.Column<long>(type: "bigint", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_UserMovies", x => x.Id);
        table.ForeignKey(
            name: "FK_UserMovies_Users_UserId",
            column: x => x.UserId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "NewsComments",
    columns: table => new
    {
        Id = table.Column<long>(type: "bigint", nullable: false)
            .Annotation("Npgsql:ValueGenerationStrategy",
NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
        Comment = table.Column<string>(type: "text", nullable: false),
        NewsId = table.Column<long>(type: "bigint", nullable: false),
        UserId = table.Column<long>(type: "bigint", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_NewsComments", x => x.Id);
        table.ForeignKey(
            name: "FK_NewsComments_News_NewsId",
            column: x => x.NewsId,
            principalTable: "News",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_NewsComments_Users_UserId",
            column: x => x.UserId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateIndex(

```

```

        name: "IX_ActorMovie_MoviesId",
        table: "ActorMovie",
        column: "MoviesId");

migrationBuilder.CreateIndex(
    name: "IX_Comments_MovieId",
    table: "Comments",
    column: "MovieId");

migrationBuilder.CreateIndex(
    name: "IX_Comments_UserId",
    table: "Comments",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_GenreMovie_MoviesId",
    table: "GenreMovie",
    column: "MoviesId");

migrationBuilder.CreateIndex(
    name: "IX_News_MovieId",
    table: "News",
    column: "MovieId");

migrationBuilder.CreateIndex(
    name: "IX_NewsComments_NewsId",
    table: "NewsComments",
    column: "NewsId");

migrationBuilder.CreateIndex(
    name: "IX_NewsComments_UserId",
    table: "NewsComments",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_UserMovies_UserId",
    table: "UserMovies",
    column: "UserId");
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "ActorMovie");

    migrationBuilder.DropTable(
        name: "Comments");

    migrationBuilder.DropTable(
        name: "GenreMovie");

    migrationBuilder.DropTable(

```

```

        name: "NewsComments");

migrationBuilder.DropTable(
    name: "UserMovies");

migrationBuilder.DropTable(
    name: "Actors");

migrationBuilder.DropTable(
    name: "Genres");

migrationBuilder.DropTable(
    name: "News");

migrationBuilder.DropTable(
    name: "Users");

migrationBuilder.DropTable(
    name: "Movies");
    }
}
}
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Imdb.WebApi.Migrations
{
    public partial class AddedNickNameForUser : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<string>(
                name: "NickName",
                table: "Users",
                type: "text",
                nullable: false,
                defaultValue: "");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "NickName",
                table: "Users");
        }
    }
}
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

```

```

namespace Imdb.WebApi.Migrations
{
    public partial class AddedMovieNameToUserMovies : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<string>(
                name: "MovieName",
                table: "UserMovies",
                type: "text",
                nullable: false,
                defaultValue: "");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "MovieName",
                table: "UserMovies");
        }
    }
}
using Microsoft.EntityFrameworkCore.Migrations;

```

#nullable disable

```

namespace Imdb.WebApi.Migrations
{
    public partial class DeletingWrongNavigatios : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropForeignKey(
                name: "FK_News_Movies_MovieId",
                table: "News");

            migrationBuilder.DropIndex(
                name: "IX_News_MovieId",
                table: "News");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateIndex(
                name: "IX_News_MovieId",
                table: "News",
                column: "MovieId");

            migrationBuilder.AddForeignKey(
                name: "FK_News_Movies_MovieId",
                table: "News",
                column: "MovieId",

```

```

        principalTable: "Movies",
        principalColumn: "Id",
        onDelete: ReferentialAction.Cascade);
    }
}
}
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Imdb.WebApi.Migrations
{
    public partial class DeletingWrongNavigatios2 : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropForeignKey(
                name: "FK_Comments_Movies_MovieId",
                table: "Comments");

            migrationBuilder.DropIndex(
                name: "IX_Comments_MovieId",
                table: "Comments");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateIndex(
                name: "IX_Comments_MovieId",
                table: "Comments",
                column: "MovieId");

            migrationBuilder.AddForeignKey(
                name: "FK_Comments_Movies_MovieId",
                table: "Comments",
                column: "MovieId",
                principalTable: "Movies",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        }
    }
}
// <auto-generated />
using Imdb.WebApi.Persistence;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace Imdb.WebApi.Migrations

```

```

{
  [DbContext(typeof(ImdbDbContext))]
  partial class ImdbDbContextModelSnapshot : ModelSnapshot
  {
    protected override void BuildModel(ModelBuilder modelBuilder)
    {
#pragma warning disable 612, 618
      modelBuilder
        .HasAnnotation("ProductVersion", "6.0.12")
        .HasAnnotation("Relational:MaxIdentifierLength", 63);

```

```

NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

```

```

      modelBuilder.Entity("ActorMovie", b =>
      {
        b.Property<long>("CastId")
          .HasColumnType("bigint");

        b.Property<long>("MoviesId")
          .HasColumnType("bigint");

        b.HasKey("CastId", "MoviesId");

        b.HasIndex("MoviesId");

        b.ToTable("ActorMovie");
      });

```

```

      modelBuilder.Entity("GenreMovie", b =>
      {
        b.Property<long>("GenresId")
          .HasColumnType("bigint");

        b.Property<long>("MoviesId")
          .HasColumnType("bigint");

        b.HasKey("GenresId", "MoviesId");

        b.HasIndex("MoviesId");

        b.ToTable("GenreMovie");
      });

```

```

      modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.Actor", b =>
      {
        b.Property<long>("Id")
          .ValueGeneratedOnAdd()
          .HasColumnType("bigint");

```

```

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

```

```

        b.Property<string>("FirstName")
            .IsRequired()
            .HasColumnType("text");

        b.Property<string>("LastName")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");

        b.ToTable("Actors");
    });

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.Comment", b =>
{
    b.Property<long>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("bigint");

```

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

```

        b.Property<string>("CommentText")
            .IsRequired()
            .HasColumnType("text");

        b.Property<long>("MovieId")
            .HasColumnType("bigint");

        b.Property<long>("UserId")
            .HasColumnType("bigint");

        b.HasKey("Id");

        b.HasIndex("UserId");

        b.ToTable("Comments");
    });

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.Genre", b =>
{
    b.Property<long>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("bigint");

```

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

```

        b.Property<string>("Name")
            .IsRequired()
            .HasColumnType("text");

```

```

        b.HasKey("Id");

        b.ToTable("Genres");
    });

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.Movie", b =>
{
    b.Property<long>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("bigint");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

    b.Property<string>("Name")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("Overview")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("PosterPath")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("TrailerPath")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("Id");

    b.ToTable("Movies");
});

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.News", b =>
{
    b.Property<long>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("bigint");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

    b.Property<string>("Description")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("ImgPath")
        .IsRequired()
        .HasColumnType("text");

```

```

        b.Property<long>("MovieId")
            .HasColumnType("bigint");

        b.Property<string>("Title")
            .IsRequired()
            .HasColumnType("text");

        b.HasKey("Id");

        b.ToTable("News");
    });

```

```

=> modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.NewsComment", b
    {
        b.Property<long>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("bigint");

```

```

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

```

```

        b.Property<string>("Comment")
            .IsRequired()
            .HasColumnType("text");

        b.Property<long>("NewsId")
            .HasColumnType("bigint");

        b.Property<long>("UserId")
            .HasColumnType("bigint");

        b.HasKey("Id");

        b.HasIndex("NewsId");

        b.HasIndex("UserId");

        b.ToTable("NewsComments");
    });

```

```

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.User", b =>
    {
        b.Property<long>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("bigint");

```

```

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

```

```

        b.Property<string>("Email")

```

```

        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("NickName")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("Password")
        .IsRequired()
        .HasColumnType("text");

    b.HasKey("Id");

    b.ToTable("Users");
});

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.UserMovie", b =>
{
    b.Property<long>("Id")
        .ValueGeneratedOnAdd()
        .HasColumnType("bigint");

    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<long>("Id"));

    b.Property<long>("MovieId")
        .HasColumnType("bigint");

    b.Property<string>("MovieName")
        .IsRequired()
        .HasColumnType("text");

    b.Property<string>("UserDescription")
        .IsRequired()
        .HasColumnType("text");

    b.Property<long>("UserId")
        .HasColumnType("bigint");

    b.Property<int>("UserRating")
        .HasColumnType("integer");

    b.HasKey("Id");

    b.HasIndex("UserId");

    b.ToTable("UserMovies");
});

modelBuilder.Entity("ActorMovie", b =>
{
    b.HasOne("Imdb.WebApi.Persistence.Entities.Actor", null)

```

```

        .WithMany()
        .HasForeignKey("CastId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.HasOne("Imdb.WebApi.Persistence.Entities.Movie", null)
        .WithMany()
        .HasForeignKey("MoviesId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

modelBuilder.Entity("GenreMovie", b =>
{
    b.HasOne("Imdb.WebApi.Persistence.Entities.Genre", null)
        .WithMany()
        .HasForeignKey("GenresId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.HasOne("Imdb.WebApi.Persistence.Entities.Movie", null)
        .WithMany()
        .HasForeignKey("MoviesId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.Comment", b =>
{
    b.HasOne("Imdb.WebApi.Persistence.Entities.User", "User")
        .WithMany("Comments")
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.Navigation("User");
});

=> modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.NewsComment", b
{
    b.HasOne("Imdb.WebApi.Persistence.Entities.News", "News")
        .WithMany("NewsComments")
        .HasForeignKey("NewsId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();

    b.HasOne("Imdb.WebApi.Persistence.Entities.User", "User")
        .WithMany("NewsComments")
        .HasForeignKey("UserId")
        .OnDelete(DeleteBehavior.Cascade)
        .IsRequired();
});

```

```

        b.Navigation("News");

        b.Navigation("User");
    });

    modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.UserMovie", b =>
    {
        b.HasOne("Imdb.WebApi.Persistence.Entities.User", "User")
            .WithMany("UserMovies")
            .HasForeignKey("UserId")
            .OnDelete(DeleteBehavior.Cascade)
            .IsRequired();

        b.Navigation("User");
    });

    modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.News", b =>
    {
        b.Navigation("NewsComments");
    });

    modelBuilder.Entity("Imdb.WebApi.Persistence.Entities.User", b =>
    {
        b.Navigation("Comments");

        b.Navigation("NewsComments");

        b.Navigation("UserMovies");
    });
#pragma warning restore 612, 618
    }
}

using System.Text.Json.Serialization;
using Imdb.WebApi.Persistence;
using Imdb.WebApi.Persistence.Repository;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers().AddJsonOptions(x =>
x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddCors();
builder.Services.AddDbContext<ImdbDbContext>(opts =>
{

```

```

        //todo change pass to real one
        opts.UseNpgsql("Host=localhost;Port=5432;Database=imdb-
diploma;Username=postgres;Password=allinme");
    });
    builder.Services.AddScoped<IImdbRepository, ImdbRepository>();

    var app = builder.Build();

    app.UseCors(x =>
    {
        x.AllowAnyHeader();
        x.AllowAnyMethod();
        x.AllowAnyOrigin();
    });
    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    app.UseHttpsRedirection();

    app.UseAuthorization();

    app.MapControllers();

    app.Run();
    namespace Imdb.WebApi;

    public class WeatherForecast
    {
        public DateTime Date { get; set; }

        public int TemperatureC { get; set; }

        public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

        public string? Summary { get; set; }
    }
    import { Injectable } from '@angular/core';
    import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from
'@angular/router';
    import { AuthService } from "../AuthService";

    @Injectable({
        providedIn: 'root',
    })
    export class AuthGuard implements CanActivate {
        constructor(private authService: AuthService, private router: Router) {}

```

```

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
  if (this.authService.isAuthenticated()) {
    return true;
  } else {
    this.router.navigate(['/login']);
    return false;
  }
}
}
import {Injectable} from "@angular/core";
import {HttpClient} from "@angular/common/http";
import {environment} from "../../environments/environment";
import {BehaviorSubject, map, Observable} from "rxjs";

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private currentUserSubject: BehaviorSubject<any>;
  public currentUser: Observable<any>;

  constructor(private http: HttpClient) {
    this.currentUserSubject = new
BehaviorSubject<any>(JSON.parse(localStorage.getItem('currentUser')));
    this.currentUser = this.currentUserSubject.asObservable();
  }

  public get currentUserValue(): any {
    return this.currentUserSubject.value;
  }

  getUser() {
    return this.currentUserValue;
  }

  private controllerUrl = environment.apiUrl + '/Auth';

  login(email: string, password: string): Observable<any> {
    return this.http.post<any>(`${this.controllerUrl}/login`, {email, password})
      .pipe(map(user => {
        localStorage.setItem('currentUser', JSON.stringify(user)); //todo change to this
authGuard
        this.currentUserSubject.next(user);
        return user;
      }));
  }

  signup(email: string, password: string, nickname: string): Observable<any> {
    return this.http.post<any>(`${this.controllerUrl}/signup`, {email, password,
nickname})
      .pipe(map(user => {
        return user;
      }));
  }
}

```

```

    }));
  }

  logout() {
    // remove user from local storage to log user out
    localStorage.setItem('currentUser', null);
    this.currentUserSubject.next(null);
  }

  isAuthenticated(): boolean {
    const token = localStorage.getItem('currentUser');
    if (!token) {
      return false;
    }
    return true;
  }
}
import { Component, HostListener } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'showtime';
  navbg:any;
  @HostListener('document:scroll') scrollover(){
    console.log(document.body.scrollTop,'scrolllength#');

    if(document.body.scrollTop > 0 || document.documentElement.scrollTop > 0)
    {
      this.navbg = {
        'background-color':'#000000'
      }
    }else
    {
      this.navbg = {}
    }
  }
}
export const environment = {
  production: true
};
// This file can be replaced during build by using the `fileReplacements` array.
// `ng build` replaces `environment.ts` with `environment.prod.ts`.
// The list of file replacements can be found in `angular.json`.

export const environment = {
  production: false,
  apiUrl: 'http://localhost:5048'
};

```

```

/*
 * For easier debugging in development mode, you can import the following file
 * to ignore zone related error stack frames such as `zone.run`,
`zoneDelegate.invokeTask`.
 *
 * This import should be commented out in production mode because it will have a
negative impact
 * on performance if an error is thrown.
 */
// import 'zone.js/plugins/zone-error'; // Included with Angular CLI.
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

function bootstrap() {
  platformBrowserDynamic().bootstrapModule(AppModule)
    .catch(err => console.error(err));
};

if (document.readyState === 'complete') {
  bootstrap();
} else {
  document.addEventListener('DOMContentLoaded', bootstrap);
}
// This file is required by karma.conf.js and loads recursively all the .spec and framework
files

import 'zone.js/testing';
import { getTestBed } from '@angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';

declare const require: {
  context(path: string, deep?: boolean, filter?: RegExp): {
    <T>(id: string): T;
    keys(): string[];
  };
};

// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,

```

```

    platformBrowserDynamicTesting(),
  );

  // Then we find all the tests.
  const context = require.context('./', true, /\.spec\.ts$/);
  // And load the modules.
  context.keys().forEach(context);
  // Karma configuration file, see link for more information
  // https://karma-runner.github.io/1.0/config/configuration-file.html

  module.exports = function (config) {
    config.set({
      basePath: "",
      frameworks: ['jasmine', '@angular-devkit/build-angular'],
      plugins: [
        require('karma-jasmine'),
        require('karma-chrome-launcher'),
        require('karma-jasmine-html-reporter'),
        require('karma-coverage'),
        require('@angular-devkit/build-angular/plugins/karma')
      ],
      client: {
        jasmine: {
          // you can add configuration options for Jasmine here
          // the possible options are listed at
https://jasmine.github.io/api/edge/Configuration.html
          // for example, you can disable the random execution with `random: false`
          // or set a specific seed with `seed: 4321`
        },
        clearContext: false // leave Jasmine Spec Runner output visible in browser
      },
      jasmineHtmlReporter: {
        suppressAll: true // removes the duplicated traces
      },
      coverageReporter: {
        dir: require('path').join(__dirname, './coverage/showtime'),
        subdir: '.',
        reporters: [
          { type: 'html' },
          { type: 'text-summary' }
        ]
      },
      reporters: ['progress', 'kjhtml'],
      port: 9876,
      colors: true,
      logLevel: config.LOG_INFO,
      autoWatch: true,
      browsers: ['Chrome'],
      singleRun: false,
      restartOnFileChange: true
    });
  };

```

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <!-- Primary Meta Tags -->
  <title>Showtime - watch movies</title>
  <meta name="title" content="Showtime - watch movies">
  <meta name="description" content="watch movies online">
  <meta name="author" content="numofcode">

  <!-- Open Graph / Facebook -->
  <meta property="og:type" content="website">
  <meta property="og:url" content="https://metatags.io/">
  <meta property="og:title" content="Showtime - watch movies">
  <meta property="og:description" content="watch movies online">
  <meta property="og:image" content="src/assets/showtimemovie.png">
  <link
    href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
  <!-- Twitter -->
  <meta property="twitter:card" content="summary_large_image">
  <meta property="twitter:url" content="https://metatags.io/">
  <meta property="twitter:title" content="Showtime - watch movies">
  <meta property="twitter:description" content="watch movies online">
  <meta property="twitter:image" content="src/assets/showtimemovie.png">

  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <!-- CSS only -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"
crossorigin="anonymous">
  <!-- google fonts -->
  <link
href="https://fonts.googleapis.com/css2?family=Ubuntu:wght@500&display=swap"
rel="stylesheet">
</head>
<body>
  <app-root></app-root>
  <!-- JavaScript Bundle with Popper -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
kenU1KFdBIe4zVF0s0G1M5b4hcxpyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4"
crossorigin="anonymous"></script>
</body>
</html>

```

Додаток Б
Слайди презентації

Національний університет «Запорізька політехніка»
кафедра програмних засобів

ДИПЛОМНА КВАЛІФІКАЦІЙНА РОБОТА

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ КІНОТЕАТРУ

Виконав: студент групи КНТ-110сп

Євген ЖУК

Керівник: к.т.н., доцент

Ірина ЗЕЛЕНЬОВА

2023

Рисунок Б.1 – Слайд № 1

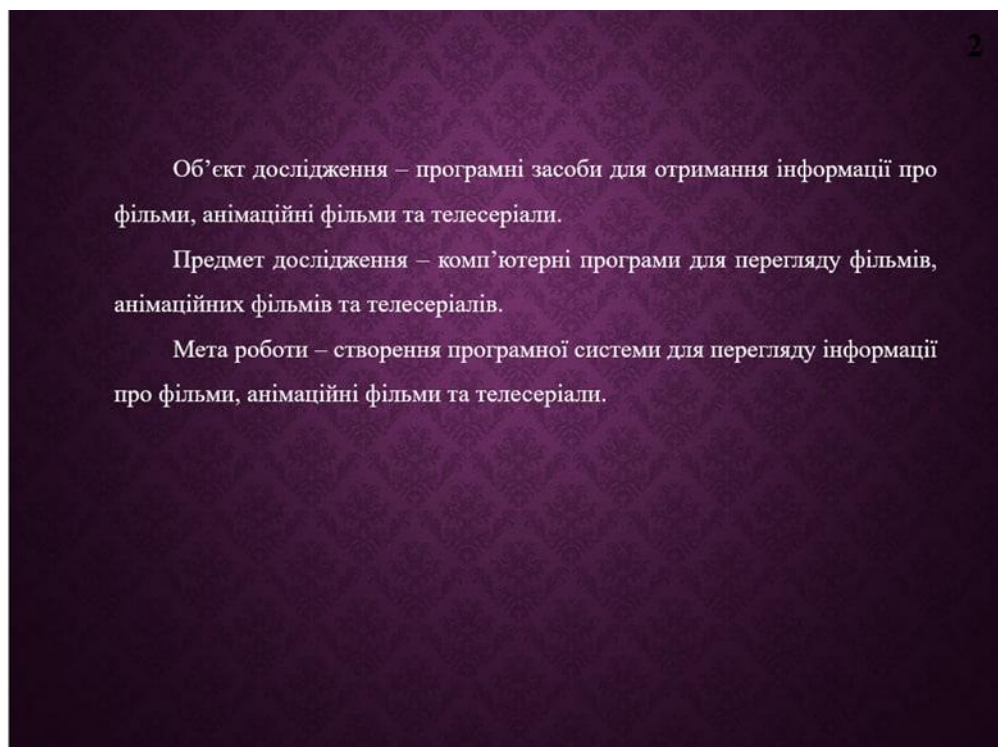


Рисунок Б.2 – Слайд № 2

Постановка завдань

3

- визначення засобів розробки;
- визначення вимог до програми;
- проектування програми перегляду інформації про телеіндустрію;
- програмна реалізація.

Рисунок Б.3 – Слайд № 3

Вибір веб-фреймворку розробки мовою TypeScript

4

Показник	Angular	React
Високорівневий API	+	–
Висока оптимізація	+	–
Кросплатформеність	–	+
Формування динамічних вебсторінок	+	+
Використовувана мова програмування	TypeScript	TypeScript

Рисунок Б.4 – Слайд № 4

Схема структури бази даних

5

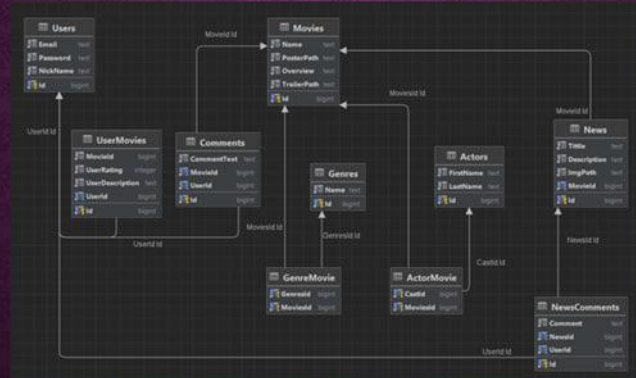


Рисунок Б.5 – Слайд № 5

Схема функціонування програми

6

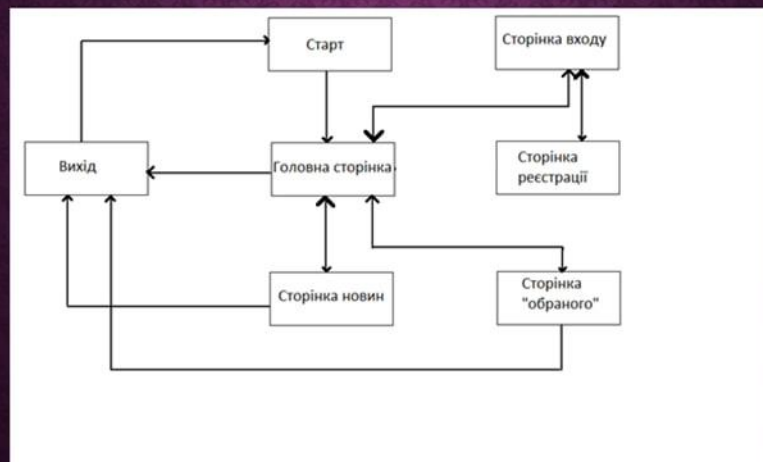


Рисунок Б.6 – Слайд № 6



Рисунок Б.7 – Слайд № 7

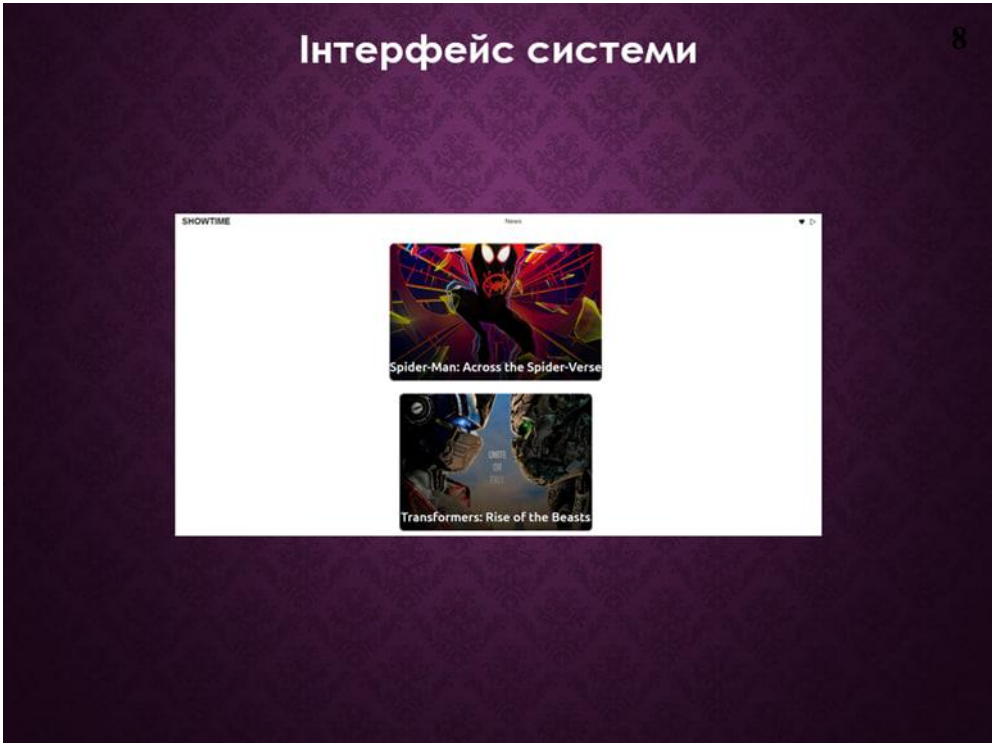


Рисунок Б.8 – Слайд № 8

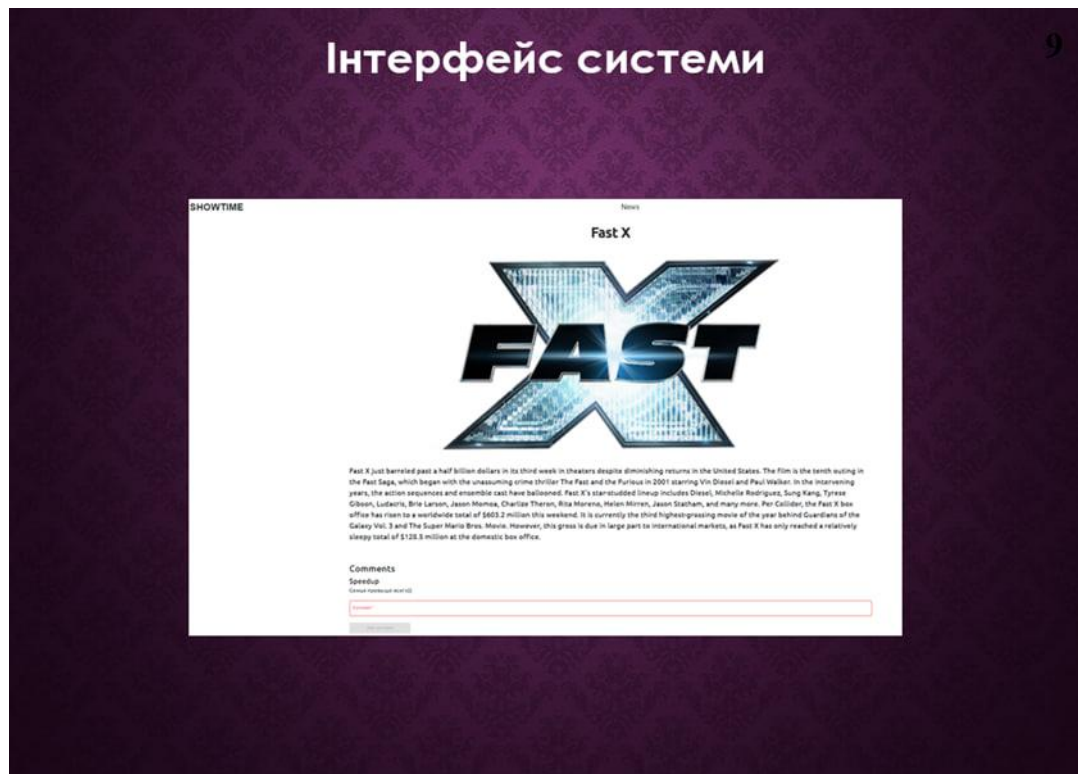


Рисунок Б.9 – Слайд № 9

Висновки

В результаті було розробки було створено програмне забезпечення веб-сайту з інформацією про фільми, анімаційні фільми та телесеріали, які допомагають глядачам отримати доступ до важливої інформації, такої як огляди, рецензії, новини та поради щодо фільмів. За результатами аналізу проблеми виконано аналіз аналогів і їх порівняння з розробкою, визначено її особливості.

За результатами проектування представлено результати порівняння фреймворків для розробки вебзастосунків мовою TypeScript та C#, обрано фреймворк Angular у цій якості, що визначило структуру програми на основі використання шаблону проектування MVC, представлено функціональні вимоги до програми, діаграму варіантів використання, схему бази даних і описано її відповідні таблиці і зв'язки між ними.

За результатами реалізації програми подано схему її функціонування, детально описано створені одиниці програми, включаючи класи моделей і уявлень, та загальну логіку роботи користувача. Під час виконання розробки та за результатами її виконання реалізовано тестування програми, що дозволило отримати результат, який відповідає запланованим для реалізації у технічному завданні вимогам.

Рисунок Б.10 – Слайд № 10