Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ПОТОКОВОГО СЕРВІСУ ДЛЯ ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ З ПЕРСОНАЛЬНИМИ РЕКОМЕНДАЦІЯМИ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-3
Васильєв Р.Р.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Кіношенко Д.К.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Васильєву Руслану Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка потокового сервісу для перегляду фільмів та серіалів з персональними рекомендаціями

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 24 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, платформа .Net, мова програмування C#, бібліотека React, бібліотека TensorFlow, брокер повідомлень RabbitMQ, мова програмування TypeScript, середовище розробки Rider, середовище розробки WebStorm, середовище розробки PyCharm, хмарне сховище AWS S3.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд існуючих потокових сервісів для перегляду медіа-контенту.
2. Види фільтрацій та поєднання їх у гібридну фільтрацію для рекомендацій контенту.
3. Проектування архітектури та модулів потокового сервісу.
4. Створення нейронної мережі для рекомендацій за допомогою бібліотеки TensorFlow.
5. Розробка потокового сервісу для перегляду фільмів та серіалів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність потокових сервісів для перегляду фільмів та серіалів, постановка задачі, проектування модулів та архітектури, проектування системи рекомендацій, демонстрація та тестування роботи потокового сервісу.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	09.04.24-11.04.24	
3	Аналіз літератури з досліджуваної проблеми	12.04.24-18.04.24	
4	Аналіз технічних засобів	19.04.24-25.04.24	
5	Розробка системи рекомендацій	26.04.24-03.05.24	
6	Програмна реалізація	04.05.24-23.05.24	
7	Оформлення пояснювальної записки	24.05.24-26.05.24	
8	Перевірка на плагіат	25.05.24	
9	Рецензування	26.05.24	
10	Підготовка презентації та доповіді	27.05.24-02.06.24	
11	Занесення роботи в електронний архів	05.06.24	
12	Попередній захист кваліфікаційної роботи	05.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Кіношенко Д.К.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 66 с., 29 рис., 32 джерела.

ASP NET CORE, C#, REACT, AWS S3, RABBIT MQ, GRAPH QL, ПЕРСОНАЛЬНІ РЕКОМЕНДАЦІЇ, КОЛЛАБОРАТИВНА ФІЛЬТРАЦІЯ, TENSORFLOW.

Об’єктом роботи є перегляд фільмів та серіалів з персональними рекомендаціями.

Метою роботи є розробка застосунку з авторизацією, можливістю перегляду фільмів та серіалів, системами вподобань, рекомендацій, що базуються на фільтрації та роботі нейронної мережі, та повідомлень.

Було створено застосунок ASP NET Core для сервісної частини, Python застосунок для роботи нейронної мережі, для створення якої було використано бібліотеку TensorFlow, клієнтський застосунок з використанням бібліотеки React. Було налаштовано комунікацію між застосунками за допомогою REST та AMQP та створено засоби роботи з AWS S3.

Результатом роботи є створений комплексний онлайн застосунок для перегляду фільмів та серіалів з персональними рекомендаціями.

ASP NET CORE, C#, REACT, AWS S3, RABBIT MQ, GRAPH QL, PERSONALIZED RECOMMENDATIONS, COLLABORATIVE FILTERING, TENSORFLOW.

The object of work is watching movies and TV shows with personalized recommendations.

The aim of the work is to develop an application with authorization, the ability to watch movies and TV shows, preference systems, recommendations based on filtering and neural network operation, and notifications.

Was created an ASP NET Core application for the service part, a Python application for the neural network, which was created using the TensorFlow library, and a client application using the React library. Communication between the applications was set up using REST and AMQP, and tools for working with AWS S3 were created.

The result is a comprehensive online application for watching movies and TV shows with personalized recommendations.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 6
Вступ 7
1 Аналіз сучасного стану потокових сервісів для перегляду контенту 8
1.1 Огляд потокових сервісів для перегляду фільмів та серіалів 8
1.2 Аналіз підходів для проєктування та реалізації системи рекомендацій контенту 10
1.3 Сучасний стан платформи .NET та методи розробки онлайн застосунків, які використовують разом з нею 12
1.4 Вибір хмарного сховища для зберігання контенту 14
1.5 Огляд фреймворків та бібліотек для реалізації клієнтської частини 15
1.6 Постановка задачі 17
2 Проєктування потокового сервісу для перегляду фільмів та серіалів. Вибір інструментів для розробки. 19
2.1 Вибір backend архітектури застосунку та розбиття на модулі 19
2.2 Проєктування бази даних 21
2.3 Розробка системи безпечного зберігання контенту для потокової передачі 23
2.4 Проєктування системи рекомендацій контенту 25
2.5 Проєктування спілкування застосунку з клієнтом та між мікросервісами 30
2.6 Створення авторизації та аутентифікації за допомогою JWT 33
3 Розробка потокового сервісу для перегляду фільмів та серіалів 38
3.1 Вибір середовища для розробки 38
3.2 Реалізація серверної та клієнтської частини 39
3.3 Огляд функціоналу сервісу та його тестування 41
Висновки 62
Перелік джерел посилання 64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Token – унікальний ідентифікатор для доступу до ресурсів

API – Application Programming Interface (інтерфейс програмного застосунку)

AWS – Amazon Web Services (хмарні сервіси Amazon)

Virtual DOM – Virtual Document Object Model (віртуальна модель об’єктів документа)

SQL – Structured Query Language (мова структурованих запитів)

JS – JavaScript (мова програмування JavaScript)

ВСТУП

У наш час передові технології у дуже значній мірі використовуються у галузі розваг. Найбільш масштабними у сфері медіа-контенту залишаються фільми та серіали, тому що вони є найцікавішими для великої кількості людей будь якого віку. Перегляд фільмів та серіалів усе частіше відбувається у режимі онлайн на спеціальних вебсайтах, які здійснюють потокове завантаження контенту, завдяки чому його перегляд є дуже оптимізованим і доступним навіть при поганому з'єднанні. Виходить так, що під час перегляду вже завантаженої частини контенту, відбувається завантаження наступної частини. Завдяки цьому нам не потрібно завантажувати великі файли з контентом повністю і витрачати на це багато часу.

Метою кваліфікаційної роботи є створення такого потокового сервісу для перегляду фільмів та серіалів. Цей сервіс буде об'єднувати у собі зручний та стильний дизайн, перегляд контенту у плеєрі, можливість швидкого ознайомлення з загальною оцінкою користувачів сервісу, описом фільму чи серіалу та коротким фрагментом з нього.

Дуже важливою частиною роботи є система персональних рекомендацій контенту для користувача, що заснована одразу на колаборативній фільтрації, контент-фільтрації та роботі нейронної мережі, що натренована на великій виборці оцінок фільмів інших користувачів. Основною ідеєю є те, щоб користувач заходив на вебсайт і одразу бачив той контент, який йому сподобається і він не буде витрачати зайвий час на пошук і взагалі на розуміння того, що саме він зараз хотів би подивитися – усе це сервіс зробить за нього і запропонує вибірку найкращого контенту на основі його дій на вебсайті. Щоб найбільш чітко визначати вподобання користувача також присутня система позитивних і негативних відгуків на контент та відстеження того, які фільми повністю переглянув користувач. Щоб користувач не пропустив усі новинки фільмів та серіалів, які можуть йому сподобатися, також була створена система персональних повідомлень.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПОТОКОВИХ СЕРВІСІВ ДЛЯ ПЕРЕГЛЯДУ КОНТЕНТУ

1.1 Огляд потокових сервісів для перегляду фільмів та серіалів

У сучасному світі, де цифрові технології проникають у кожну сферу нашого життя, фільми та серіали стали не лише основним джерелом розваг, але й значною частиною культурного обміну і важливим елементом соціальної взаємодії. Потокові медіа-сервіси, які дозволяють користувачам дивитися відеоконтент через інтернет без необхідності завантаження файлів, стали ключовими гравцями в цій трансформації. З появою широкосмугового інтернету та розвитком мобільних технологій споживання відеоконтенту зазнало революційних змін. Люди тепер можуть переглядати фільми та серіали де завгодно і коли завгодно, обираючи з величезної кількості жанрів та програм, що відповідають їхнім індивідуальним уподобанням. Це стало можливим завдяки інноваційним алгоритмам, які аналізують переглядові звички та рекомендують контент, що найбільш відповідає особистим перевагам користувача. Потокові сервіси суттєво змінили пейзаж медіа-індустрії, спонукаючи традиційні телеканали та кінотеатри до адаптації їхніх стратегій під нові реалії [1]. Вони не тільки надають платформу для старих та нових творів, але й створюють власний оригінальний контент, що часто здобуває міжнародне визнання та нагороди. Результатом цих змін стала не лише більша доступність контенту, але й зростання його різноманітності, що сприяє культурному розмаїттю та інноваціям. Проте, цей прогрес не приходить без викликів. Питання конфіденційності, управління даними, авторські права та насиченість ринку є серед ключових аспектів, що вимагають уваги. В цьому контексті, глибокий аналіз потокових сервісів та їх впливу на споживчі звички стає необхідністю для розуміння сучасних тенденцій і підходів у цій галузі. Розглянемо успішні приклади таких потокових сервісів, які поєднують у усі інноваційні рішення.

Amazon Prime Video, наприклад, став важливим гравцем на ринку завдяки своїй інтеграції з ширшими послугами Amazon, включаючи Amazon Prime. Один з вражаючих аспектів Prime Video — це їхня функція «X-Ray», розроблена у співпраці з IMDb. Ця функція дозволяє глядачам отримувати інформацію про акторів та музичні треки у сцені без потреби виходити з перегляду, пропонуючи глибокий іммерсивний досвід [2].

Hulu, інший ключовий сервіс, виділяється своєю спроможністю пропонувати контент одразу після трансляції на телебаченні, роблячи платформу привабливою для любителів актуальних телешоу. Крім того, Hulu впровадив технологію навчання з підкріпленням для оптимізації своїх рекомендацій, забезпечуючи користувачам більш персоналізований вибір контенту [3].

Disney+ відрізняється своєю унікальною стратегією контенту, що включає величезну бібліотеку від Disney, Pixar, Marvel, Star Wars і National Geographic. Вони також запровадили технологію GroupWatch, яка дозволяє друзям та сім’ї дистанційно дивитися фільми та шоу разом, створюючи відчуття спільного перегляду, незалежно від географічного розташування учасників [4].

HBO Max, як платформа, зосереджена на високоякісному оригінальному контенті, включаючи велику колекцію фільмів і серіалів Warner Bros. Однією з особливостей HBO Max є їхня здатність випускати нові кінофільми одночасно в театрах і на платформі, що дало користувачам доступ до новітніх фільмів в комфорті їхнього дому під час пандемії [5].

Ці сервіси не тільки перетворили спосіб, яким ми споживаємо розважальний контент, але й підняли планку технологічної інтеграції та інновацій, завдяки чому користувачі мають змогу отримувати не тільки широкий доступ до контенту, але й високоякісний персоналізований досвід перегляду. Крім того, інтеграція зі смарт-телевізорами, мобільними пристроями та іншими потоковими сервісами робить доступ до контенту ще більш зручним.

1.2 Аналіз підходів для проєктування та реалізації системи рекомендацій контенту

Системи рекомендацій контенту є важливим компонентом багатьох сучасних онлайн-сервісів, особливо тих, які пропонують медіа-контент, таких як фільми та серіали. Ці системи допомагають користувачам знаходити контент, який їм найбільше подобається, та сприяють збільшенню їхньої взаємодії з сервісом. В цьому розділі ми розглянемо основні підходи до розробки та реалізації систем рекомендацій, такі як контентна фільтрація, колаборативна фільтрація, гібридні моделі, та використання нейронних мереж.

Контентна фільтрація базується на аналізі властивостей об’єктів, які рекомендуються. Наприклад, у контексті фільмів, такий підхід може аналізувати жанр, режисера, акторів, чи навіть сюжетні ключові слова кожного фільму. Цей метод використовує профіль користувача, що створюється на основі їхніх попередніх переглядів або вказаних уподобань, для знаходження нового контенту зі схожими характеристиками. Основною перевагою цього підходу є його простота в реалізації та здатність працювати без даних про інших користувачів, але він часто може пропонувати обмежений або монотонний вибір [6].

Колаборативна фільтрація – це техніка, яка використовує дані про взаємодію користувачів з контентом для генерування рекомендацій. Цей підхід може бути поділений на два основних типи: основана на користувачах та основана на предметах. Основана на користувачах – вона порівнює профілі користувачів, знаходячи схожості у виборах і рекомендуючи контент, який сподобався схожим користувачам. Основана на предметах – визначає схожість між предметами на основі інтеракції користувачів із ними і рекомендує ті, що мають високі оцінки від користувачів, які також високо оцінили предмет, що переглядається.

Гібридні системи рекомендацій поєднують елементи обох попередніх підходів – контентної фільтрації та колаборативної фільтрації, а іноді й інші методи, для покращення якості рекомендацій. Це може включати змішування рекомендацій, що генеруються різними моделями, або створення єдиної моделі, яка враховує як атрибути контенту, так і інформацію про поведінку користувачів. Такий підхід дозволяє уникнути обмежень, властивих кожному методу окремо, забезпечуючи більш диверсифікований і точний вибір контенту [7].

Використання нейронних мереж в системах рекомендацій є порівняно новим, але дуже перспективним напрямком. Нейронні мережі можуть автоматично виявляти складні залежності в даних та ефективно використовувати великі обсяги інформації для генерації рекомендацій [8].

Глибоке навчання, зокрема, дозволяє витягувати корисні признаки з неструктурованих даних, таких як текст опису фільму або зображення, що може значно підвищити релевантність та особистісну спрямованість рекомендацій. Типовий приклад використання нейронних мереж у рекомендаційних системах – це алгоритми, що засновані на архітектурі autoencoders, які здатні вчитися представляти дані в стислому вигляді, а потім реконструювати вхідні дані, мінімізуючи втрату інформації. Інший приклад – використання конволюційних нейронних мереж для аналізу візуального контенту фільмів і серіалів, що дозволяє краще зрозуміти їхній контент і контекст [9].

Важливим аспектом систем рекомендацій є також алгоритми ранжування, які визначають, які саме рекомендації показувати користувачеві і в якому порядку. Завдання цих алгоритмів – максимізувати корисність кожної рекомендації для користувача. Тут часто використовуються методи машинного навчання, що включають регресійні моделі, дерева рішень та ансамблеві методи, які можуть адаптуватися до змінних уподобань користувачів і змін у доступному контенті.

1.3 Сучасний стан платформи .NET та методи розробки онлайн застосунків, які використовують разом з нею

Платформа .NET від Microsoft продовжує залишатися однією з найпопулярніших у світі розробки програмного забезпечення, особливо для створення вебзастосунків та служб. Ця платформа пропонує розробникам потужний інструментарій, який підтримує модульний підхід до архітектури застосунків, включаючи моноліти та мікросервіси. У цьому розділі розглядається сучасний стан платформи .NET, інтеграція з SQL Server, можливості ASP.NET Core, та загальні методи розробки вебзастосунків на цій платформі.

ASP.NET Core є переробленою, крос-платформною, високопродуктивною версією ASP.NET, яка дозволяє розробникам побудувати вебзастосунки та сервіси, IoT-застосунки. ASP.NET Core було спроєктовано з урахуванням модульності та легкості налаштування, що дозволяє розробникам легко інтегрувати тільки ті компоненти, які необхідні для їх конкретної аплікації, знижуючи тим самим навантаження та підвищуючи продуктивність. Все починається, коли HTTP-запит досягає вебсервера, який у випадку ASP.NET Core зазвичай є Kestrel, високопродуктивний вебсервер, вбудований безпосередньо у фреймворк. Kestrel обробляє вхідні з’єднання, перетворюючи їх на об’єкти запитів, які потім можуть бути оброблені у застосунку. Після того як Kestrel приймає запит, він передає його через конвеєр middleware.

Middleware – це компоненти програмного забезпечення, які можуть обробляти запити та відповіді перед та після того, як вони досягають основної логіки застосунку [10]. Вони виконують різноманітні функції, такі як аутентифікація, логування, управління сесіями, управління кешуванням тощо. Один з ключових компонентів middleware – маршрутизатор, який визначає, до якого контролера та методу має бути направлений запит, виходячи з URL-адреси та HTTP-методу (рис. 1.1).

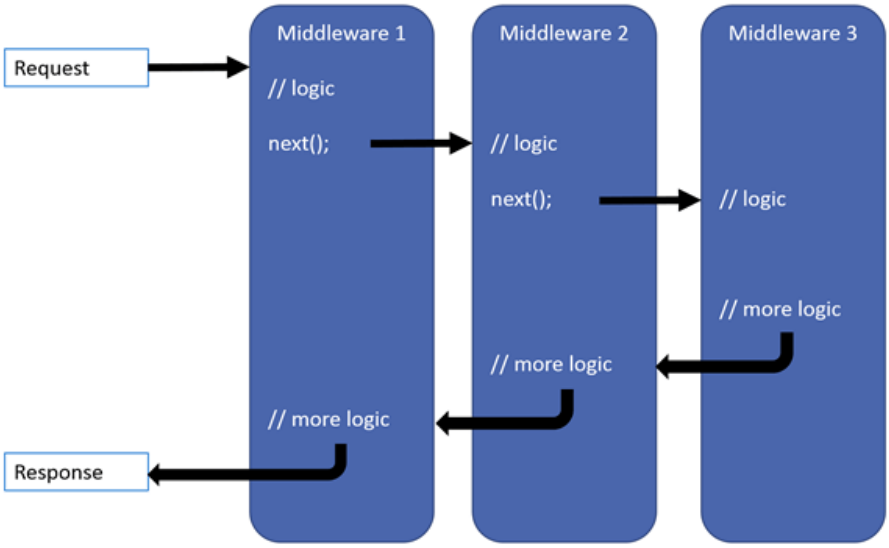

Рисунок 1.1 – Шлях запиту та відповіді у ланцюзі middleware застосунку ASP.NET Core

Маршрутизація в ASP.NET Core може бути налаштована дуже гнучко завдяки використанню атрибутів на методах контролерів або через конвенційне визначення маршрутів. Після маршрутизації запит потрапляє до відповідного контролера. Контролери у ASP.NET Core – це класи, які містять логіку обробки різних запитів. Вони взаємодіють з моделями для виконання операцій із даними і формують відповіді, використовуючи представлення або форматуючи дані як JSON або XML для API відповідей. Останнім кроком у життєвому циклі запиту є відправлення відповіді клієнту. Після обробки в контролері та, можливо, у представленні, відповідь проходить назад через конвеєр middleware, де кожен компонент може виконати додаткові операції перед тим, як відповідь буде відправлена клієнту [11].

SQL Server продовжує бути однією з найпопулярніших систем управління базами даних, що інтегрується з .NET. Ця РСУБД володіє високою продуктивністю, розширеними можливостями аналітики та надійною безпекою, що робить її ідеальним вибором для корпоративних застосунків. Інтеграція SQL Server з .NET легко виконується завдяки таким інструментам,

як Entity Framework, який дозволяє розробникам керувати базами даних через об’єктно-орієнтований інтерфейс.

В архітектурі сучасних вебзастосунків, платформа .NET підтримує як традиційну монолітну архітектуру, так і сучасні підходи, такі як мікросервіси. Монолітні архітектури вважаються відносно простими для розробки та розгортання, але можуть бути важкими для масштабування, оскільки всі компоненти застосунку тісно інтегровані. Навпаки, мікросервісна архітектура дозволяє застосунку бути розбитим на менші, незалежно розгортані сервіси, що можуть бути легко масштабовані та управляються незалежно.

Завдяки гнучкості ASP.NET Core і його інтеграції з різноманітними інструментами та сервісами, платформа .NET забезпечує великі можливості для розробки сучасних вебзастосунків. Вона підтримує як традиційні застосунки, так і новітні підходи, такі як мікросервіси, дозволяючи розробникам створювати масштабовані, ефективні та легко підтримувані рішення.

1.4 Вибір хмарного сховища для зберігання контенту

Вибір хмарного сховища для зберігання контенту є критичним рішенням для будь-якого потокового сервісу, що пропонує фільми та серіали. Розглядаючи основні платформи, як Microsoft Azure, AWS S3, та Firebase, кожна з них має свої унікальні переваги та недоліки, які важливо враховувати при виборі оптимального рішення.

Microsoft Azure Storage пропонує тісну інтеграцію з іншими продуктами Microsoft, що є великою перевагою для компаній, які вже використовують ці продукти. Azure також відомий своїми розширеними можливостями управління даними, включаючи автоматичне шифрування і різноманітні опції резервного копіювання. Проте, вартість і складність конфігурації можуть бути бар’єрами, особливо для нових користувачів.

AWS S3 відрізняється відмінною масштабованістю і надійністю, роблячи його ідеальним для зберігання великих обсягів даних. Завдяки гнучкості та інтеграції з іншими сервісами AWS, S3 може підтримувати складні багатосервісні архітектури. Це дозволяє розробникам використовувати S3 в широкому спектрі застосунків, хоча вартість за запити та передачу даних може бути значною [12].

Firebase Storage, з іншого боку, є особливо зручним для розробників завдяки простим API та інтеграції з Google Cloud. Це робить його привабливим для проєктів, які потребують швидкої розробки та простоти використання. Проте, Firebase може мати обмеження при обробці великих обсягів даних і підходить для великомасштабних застосунків [13].

Після детального аналізу цих платформ, для потокового сервісу, який може включати декілька застосунків, написаних на різних мовах програмування та використовуючих різні технології, найкращим рішенням є використання AWS S3. Висока масштабованість, гнучкість та можливості управління доступом, які надає S3, є критично важливими для забезпечення ефективності та надійності сервісу, що забезпечує великі обсяги медіа-контенту і вимагає різноманітності в інтеграції сервісів.

1.5 Огляд фреймворків та бібліотек для реалізації клієнтської частини

При розробці клієнтської частини вебзастосунків, вибір правильного фреймворку або бібліотеки є вирішальним для досягнення оптимальної продуктивності, швидкості розробки та масштабування. Серед найпопулярніших інструментів для розробки клієнтських застосунків виступають Angular, Vue та React, кожен з яких пропонує унікальні переваги і особливості.

Angular є повнофункціональним фреймворком, який був розроблений командою Google і спрямований на розробку великих, складних застосунків.

Він використовує TypeScript, який додає строгу типізацію і об’єктно-орієнтовані можливості до JavaScript, що сприяє більшій чистоті коду і зменшенню помилок у великих проєктах. Angular також включає вбудований Dependency Injection, що є потужним інструментом для управління залежностями і реалізації модульної архітектури. Втім, деякі розробники вважають Angular складним у навчанні та інтеграції через його обширність та високий поріг входження.

Vue, з іншого боку, вирізняється своєю простотою та гнучкістю. Цей прогресивний фреймворк забезпечує легкість інтеграції в існуючі проєкти, що робить його популярним вибором для додавання інтерактивних компонентів до сторінок без переписування існуючого коду. Vue також пропонує детальну документацію та широкий набір інструментів, що сприяє швидкій розробці. Однак, через свою відносну новизну порівняно з Angular чи React, Vue може мати менш розгалужене співтовариство та меншу кількість доступних ресурсів для навчання.

React, розроблений Facebook, є бібліотекою для побудови користувацьких інтерфейсів і відомий своєю продуктивністю та гнучкістю. Він дозволяє розробникам створювати великі вебзастосунки, які можуть оновлювати дані без перезавантаження сторінки. Основною особливістю React є використання віртуального DOM, що дозволяє оптимізувати оновлення вебсторінки та забезпечити високу швидкість рендеринга. React також підтримує компонентний підхід до розробки, що спрощує управління кодом та перевикористання компонентів.

Обираючи між цими технологіями для нашого проєкту, я вирішив вибрати React. Цей вибір був зумовлений декількома ключовими перевагами. По-перше, React пропонує велику гнучкість і масштабованість, що є важливим для проєкту з потенційно великою аудиторією. По-друге, віртуальний DOM і компонентна архітектура забезпечують високу продуктивність і швидкість роботи застосунку, що є критично важливим для користувацького досвіду в середовищі потокових медіа. Нарешті, велика спільнота та багатство наявних

ресурсів роблять React особливо привабливим для швидкої розробки та підтримки застосунку на різних етапах його життєвого циклу.

1.6 Постановка задачі

Таким чином, розробка потокового сервісу для перегляду фільмів та серіалів з персональними рекомендаціями є актуальним та складним завданням, яке включає у себе використання багатьох найпередовіших технологій у галузі розробки вебзастосунків.

Об’єктом роботи є перегляд фільмів та серіалів з персональними рекомендаціями.

Метою роботи є розробка застосунку з авторизацією, можливістю перегляду фільмів та серіалів, системами вподобань, рекомендацій, що базуються на фільтрації та роботі нейронної мережі, та повідомлень.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати існуючі потокові сервіси для перегляду фільмів та серіалів;

– дослідити функціональні можливості основних конкурентів;

– оцінити переваги та недоліки популярних потокових сервісів;

– розглянути функціонал який надає платформа .NET для створення вебзастосунків;

– проаналізувати функціонал для роботи з базами даних, такий як Entity Framework Core;

– розглянути підходи для створення системи рекомендацій;

– вивчити сучасні бібліотеки та методи для створення рекомендаційних нейронних мереж;

– дослідити бібліотеки TensorFlow та PyTorch;

– вивчити методи обробки та підготовки даних для навчання нейронних мереж;

– ознайомитися з архітектурами нейронних мереж, такими як CNN та RNN, та їх застосуванням у рекомендаційних системах;

– спроєктувати схему бази даних для потокового сервісу;

– створити ER-діаграму бази даних;

– спроєктувати та створити клієнтську та сервісну частину застосунку для адміністрування та наповнення контентом потокового сервісу;

– розробити інтерфейс адміністратора для завантаження контенту;

– реалізувати функціонал для перегляду та редагування контенту;

– створити функціонал для операцій з хмарним сховищем AWS S3;

– спроєктувати та створити сервісну частину потокового сервісу;

– реалізувати логіку обробки запитів користувачів;

– визначити ключові характеристики контенту для фільтрації;

– розробити алгоритм колаборативної фільтрації;

– розробити алгоритм контентної філтрації;

– розробити нейронну мережу для рекомендацій фільмів та серіалів та обучити її на великій виборці даних;

– підготувати вибірку даних для навчання нейронної мережі;

– провести навчання та оцінку ефективності моделі;

– створити систему персональних рекомендацій із багатьох компонентів;

– реалізувати систему повідомлень на потоковому сервісі;

– створити дизайн та втілити його у клієнтській частині застосунку.

2 ПРОЄКТУВАННЯ ПОТОКОВОГО СЕРВІСУ ДЛЯ ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ. ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ.

2.1 Вибір backend архітектури застосунку та розбиття його на модулі

При виборі архітектури для застосунку у першу чергу потрібно розуміти те, наскільки масштабним буде проєкт і як багато часу піде на налаштування правильної архітектури. Часом буває так, що уся користь від гнучкості та гарного масштабування системи не дуже потрібна для конкретного бізнесу і краще б було витрачати усі ресурси налаштування на написання логіки самого застосунку.

У випадку потокового сервісу для перегляду фільмів і серіалів найкращим рішенням буде чиста архітектура, оскільки таким чином дуже складна і гілляста система набуває зрозумілого вигляду. Потоковий сервіс можна розвивати у дуже різним напрямках, тому кожен блок архітектури повинен бути гнучким та незалежним від інших. Саме тому є гарним рішенням реалізація саме чистої архітектури [14].

Чиста архітектура – це концепція у сфері програмного забезпечення, яка має на меті зробити системи більш зрозумілими, гнучкими та підтримуваними. Вона була популяризована Робертом Мартіном (Uncle Bob) і є частиною більш широкого спектру архітектурних підходів, які включають принципи, такі як Onion Architecture та Hexagonal Architecture (Ports and Adapters). Основна ідея Чистої архітектури полягає в тому, що залежності в програмному коді повинні бути спрямовані від зовнішніх шарів до внутрішніх, а не навпаки (рис. 2.1). Це дозволяє зменшити взаємозв’язок компонентів і полегшити модифікацію та розширення системи. Чиста архітектура також передбачає поділ системи на окремі шари, кожен з яких має свої чітко визначені обов’язки і взаємодіє з іншими шарами через добре визначені інтерфейси [15].

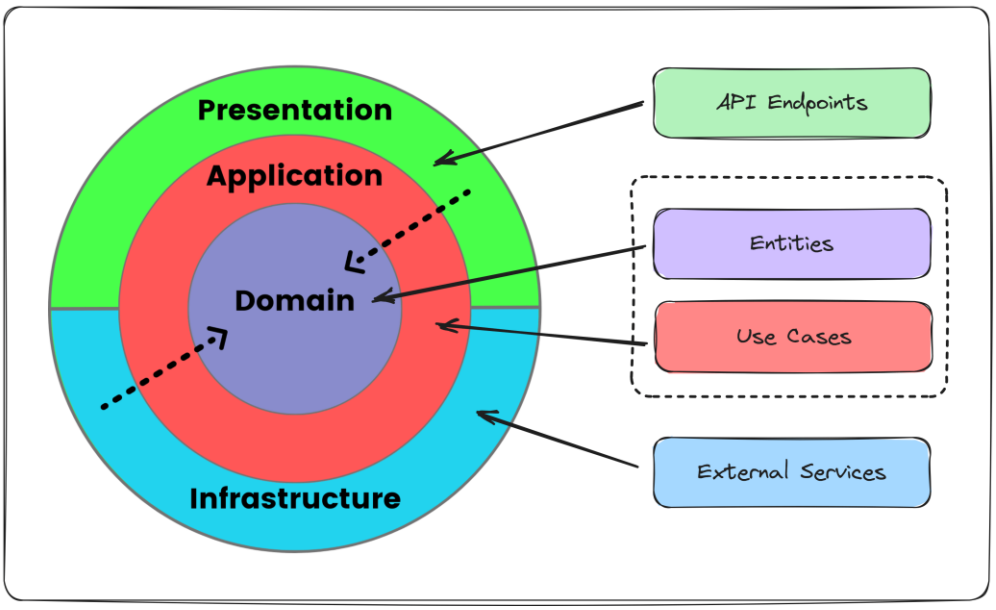

Рисунок 2.1 – Діаграма шарів чистої архітектури

Domain – це внутрішній шар, який включає бізнес-логіку і бізнес-правила вашої системи. Тут знаходяться сутності (entities), які є високорівневими моделями предметної області, і використовуються у всій системі. Вони не залежать від зовнішніх аспектів, таких як бази даних чи вебінтерфейси. Також тут можуть бути сервіси, які виконують основні операції бізнес-логіки.

Application – шар, що виконує координацію дій між зовнішніми шарами і доменним шаром. Він містить логіку застосування, яка оркеструє спосіб взаємодії користувача з доменом, керуючи даними між UI/интерфейсом користувача, зовнішніми агентами і доменним шаром. Цей шар відповідає за реалізацію випадків використання (use cases), які керують даними і логікою домену [16].

Infrastructure – шар, який надає реалізації технічних інтерфейсів, визначених в застосуванні або домені. Це можуть бути засоби для доступу до бази даних, зовнішніх сервісів, файлових систем та інших технологій, що використовуються для взаємодії із зовнішнім світом. Цей шар також відповідає за реалізацію всіх адаптерів або портів, визначених в шарі застосування.

Persistence – цей шар відповідає за механізми зберігання і відновлення даних, що використовуються застосуванням. Він може включати реалізацію конкретних технологій баз даних (як SQL, NoSQL), а також специфікації для взаємодії з файловими системами або іншими формами постійного зберігання даних. Цей шар часто інтегрується з інфраструктурним шаром для виконання своїх функцій [17].

За шар Presentation буде відповідати застосунок ASP NET CORE який буде використовуватися як API для отримання запитів від клієнтів. Також буде створений проєкт Contracts у якому будуть збережені усі запити та відповіді за допомогою яких буде відбуватися спілкування між застосунком та клієнтами. На цей проєкт будуть йти посилання від усіх інших проєктів.

2.2 Проєктування бази даних

В структурі бази даних для застосунку, що каталогізує фільми та телевізійні шоу, основними сутностями є Movie та TV Show. Кожен з цих елементів має свої жанри, режисерів та акторів, що потребують виокремлення в окремі таблиці. Таке розподілення дозволяє забезпечити більш гнучке та ефективне управління даними.

Для зв’язку між фільмами чи шоу та їх жанрами створюється допоміжна таблиця, яка зазвичай містить ідентифікатори обох сутностей. Це дозволяє кожному фільму або шоу мати багато жанрів, а кожен жанр може бути прив’язаний до багатьох фільмів або шоу.

Країна випуску є важливим атрибутом для кожного фільму або серіалу, оскільки ця інформація може використовуватись для регіональних рекомендацій в системах, що надають контент відповідно до геолокації користувача.

Сутність серіалу (TV Show) має більш складну структуру порівняно з фільмом через наявність сезонів та епізодів. Для кожного сезону створюється

окрема таблиця, а для епізодів – ще одна, де кожен епізод безпосередньо залежить від сезону, що вказується через поле у таблиці епізодів. Такий підхід дозволяє зберігати строгу ієрархічну взаємозв’язок між епізодами та сезонами, забезпечуючи правильне відображення структури серіалу [18]. Ці сутності та їх взаємозв’язки є ключовими для створення ефективної бізнес-логіки застосунку та забезпечення легкості управління і аналізу даних в базі.

Не менш важливим є проєктування користувачів системи. У нашому випадку потрібно створити дві сутності – адміністратора і користувача. Адміністратор матиме змогу авторизуватися у адміністративній частині застосунку, завантажувати контент, слідкувати за станом і статистикою. Основна роль адміністратора полягає в управлінні контентом та моніторингу системи.

Сутність користувача значно цікавіша для розгляду, оскільки вона пов’язана з багатьма таблицями, такими як лайки та дизлайки фільмів і серіалів, переглянуті епізоди, підписки та повідомлення. Загалом структура бази даних вийшла дуже об’ємною, але прозорою для розробки. Завдяки технологіям ASP.NET Core, можна з легкістю керувати допоміжними таблицями як полями класів та об’єктами. Продумана структура відкриває великий простір для реалізації складного функціоналу [19].

Як можна побачити з наведеної структури, велика увага приділяється саме контенту і тому, як користувач взаємодіє з ним. Це створює гарний фундамент для якісної побудови персональної системи рекомендацій контенту (рис. 2.2). Такий підхід дозволяє ефективно керувати користувацькими даними та забезпечує високу якість сервісу для кінцевих користувачів, що є вирішальним у створені усіх пов’язаних з даними модулів застосунку та отримання важливих для формування рекомендацій метрик з дій користувача. У такому підході кожна дія буде впливати на відображення контенту, що дозволить змінювати його при кожному перезавантаженні сторінки [20].

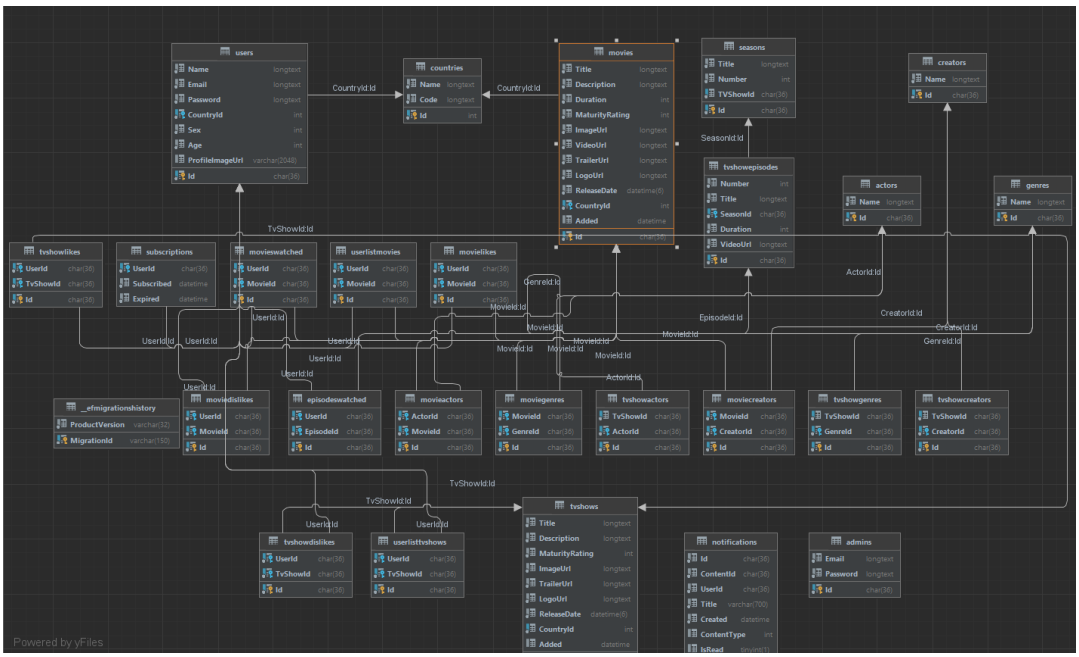

Рисунок 2.2 – Діаграма бази даних потокового сервісу для перегляду фільмів та серіалів

2.3 Розробка системи безпечного зберігання контенту для потокової передачі

У програмуванні застосунків, які містять медіа контент, гарним тоном вважається використання хмарного сховища для його збереження. У потоковому сервісі для перегляду фільмів та серіалів ми будемо використовувати хмарне сховище від компанії Amazon під назвою S3. Продумаємо таку систему, завдяки якої можна буде знайти фільм чи серіал за допомогою його унікального ідентифікатору. Для цього зробимо ієрархію папок у хмарному сховищі – створимо папки Movies та TV Shows.

Далі для кажного фільму та серіалу будемо створювати папку, у назві якої буде ідетифікатор відповідного контенту. Наступні ресурси, які відносяться до медіа будемо іменувати як «назва-ідентифікатор» [21]. Сам фільм чи епізоди є дуже важливим ресурсом який треба захистити. Для цього будемо встановлювати кожному такому ресурсу особливий доступ, завдяки якому ніхто без авторизації чи AWS токену не зможе отримати до нього

доступ. Щоб отримати доступ з нашого клієнтського застосунку, потрібно реалізувати точку доступу у нашому API, виклик якої буде завантажувати захищений фільм у систему, класти його у кеш та віддавати медіа контент потоково, таким чином наш фронтенд застосунок зможе не скачувати увесь фільм чи епізод, а завантажувати малу його частину кожен раз, коли буде надсилати запит на відповідний ендпоінт.

Інший контент, такий як трейлери або зображення, будуть у відкритому доступі, щоб не витрачати час та дорогоцінні ресурси на завантажування їх у застосунок і передачу для клієнту. Відкритий доступ цих не захищених ресурсів дасть змогу швидко відмалювати потрібні компоненти, які потребують завантаження контенту.

Для серіалів потрібно буде створювати папку, наповнювати її контентом і вже у цій папці створювати папки з найменуванням ідентифікатору сезону. У цих папках також будуть особливі права, щоб після завантаження епізодів у сезони вони були захищені. Таким чином ми будемо мати гнучку систему хмарного зберігання усіх даних, що потрібні нам для роботи потокового сервісу [22].

Ця реалізація дозволяє без проблем вносити зміни у все завантажені структури фільмів та серіалів. Саме оновлення даних представляє собою дві операції – видалення старого контенту та завантаження нового, з подальшим оновленням усіх посилань на старий контент. Варто зазначити, що у базі даних будуть збережені саме шляхи до файлів у хмарному сховищі, завдяки яким можна буде завантажувати або видаляти потрібний контент. Важливим плюсом AWS S3 є те, що видалення папки з файлами є можливим і більш того, разом з нею будуть видалені усі дочірні файли та папки. Завдяки цьому ми зможемо легко видаляти фільм чи серіал, просто видаливши корінну папку. Це значно спрощує процес управління контентом і забезпечує ефективну організацію файлів у хмарному сховищі. Використання AWS S3 також гарантує високу надійність і доступність збережених даних, що є критично важливим для потокового сервісу [23].

2.4 Проєктування системи рекомендацій контенту

У наш сучасний цифровий вік, системи рекомендацій контенту стали невід’ємною частиною багатьох онлайн-платформ, що надають користувачам персоналізований досвід. Ця технологія дозволяє вебсайтам, медіа-сервісам і електронним торговим площадкам забезпечувати релевантний і цікавий контент для кожного відвідувача, підвищуючи тим самим задоволеність користувачів і збільшуючи їхнє залучення.

Завдяки прогресу в областях штучного інтелекту і машинного навчання, сучасні системи рекомендацій здатні аналізувати величезні обсяги даних про поведінку користувачів, їхні переваги та інтеракції з контентом. Це дозволяє не просто адаптувати контент під запити конкретного користувача, а й передбачити його потреби та інтереси, навіть перш ніж він сам усвідомить їх.

Технології, що лежать в основі систем рекомендацій, постійно розвиваються. Вони включають алгоритми колаборативної фільтрації, які використовують інформацію про взаємодію між користувачами та контентом для формування предложень; контент-базовані методи, що аналізують характеристики об’єктів для рекомендацій; і гібридні системи, що поєднують обидва ці підходи для покращення точності рекомендацій [24].

Наш потоковий сервіс буде використовувати одразу декілька підходів, формуючи собою гібридну фільтрацію. Коли людина надсилає запит на отримання фільмів та серіалів, відкривши сторінку сайті, для неї буде формуватися відповідь з моделі штучного інтелекту, матеріалами для навчання якої слугували дані багатьох людей які оцінювали фільми, колаборативна фільтрація у межах потокового сервісу, яка дивиться на вподобання користувачів та рекомендує контент виключно на цьому – що подобається схожим на вас користувачам те, скоріше за все, сподобається і вам. Під схожими користувачами маються на увазі ті, що лайкають те ж саме, що і ви та дізлайкають те ж саме, що і ви. Звісно, стовідсоткове співпадіння бути не може, тому рекомендації будуть будуватися на тих користувачах,

коефіцієнт інтересів яких буде максимально наближений до вашого. Також буде використана контентна фільтрація, яка буде дивитися на те, які фільми лайкнув користувач і рекомендувати інші фільми з акторами, що грали у лайкнутому фільмі, з такими ж жанрами як у лайкнутого фільму та фільми створені тими ж режисерами, що створили лайкнуті фільми. Також будуть братися до уваги дізлайки. Важливим аспектом є таблиця того, які фільми та серіали вже переглянув користувач. Вона буде використана у всіх видах рекомендацій, щоб не рекомендувати вже переглянутий контент і розміти який контент є цікавим для користувача. Також треба вважати контент переглянутим, якщо користувач переглянув його хоча б на 80 відсотків, тому що користувач може відкрити контент просто щоб ознайомитись – не обов’язково він йому сподобається.

Створення контентної фільтрації не є складною задачею. Набагато цікавіше і складніше спроєктувати рекомендації за допомогою моделі штучного інтелекту та колаборативної фільрації. Почнемо з другого. Найкращій вибір для вирішення такого роду завдань є створення мікросервісу на мові програмування Python, тому що на ній написані найкращі бібліотеки для математичних розрахунків. Основною ідеєю є аналіз взаємодій між користувачами та об’єктами, що вони оцінюють (наприклад, фільми), для виявлення схожості між користувачами [25].

Спочатку вводяться дані користувача у вигляді оцінок, які вони виставили різним фільмам. Ці дані трансформуються у таблицю (DataFrame), де рядки представляють користувачів, а стовпці – фільми. Кожне значення в таблиці відображає оцінку, яку користувач виставив фільму.

Якщо таблиця не має достатньо даних, система повертає пустий список рекомендацій, оскільки аналіз схожості не можливий. При достатньому обсязі даних створюється матриця схожості, що вимірює косинусну схожість між користувачами. Косинусна схожість оцінює, наскільки два користувачі подібні один одному на основі їх оцінок фільмів.

Косинусна схожість між двома користувачами обчислюється шляхом визначення косинуса кута між їхніми оціночними векторами в багатовимірному просторі оцінок. Кожен користувач представлений вектором, де кожна позиція у векторі відповідає оцінці певного фільму. Якщо користувач не оцінив певний фільм, ця позиція у векторі може бути заповнена нулем або іншим значенням, що позначає відсутність оцінки [26].

Скалярний добуток цих векторів дає суму добутків відповідних оцінок, що обоє користувачі виставили. Наприклад, якщо один користувач оцінив фільм на 4, а інший на 5, то внесок цього фільму в скалярний добуток буде 20. Цей процес повторюється для всіх фільмів, що обидва користувачі оцінили. Косинус кута між векторами розраховується як скалярний добуток векторів, поділений на добуток їх норм (довжин). Норми векторів визначаються як квадратний корінь із суми квадратів оцінок, що користувач виставив. Це означає, що косинусна схожість враховує не лише взаємний напрям векторів, а й їхню величину. Косинусна схожість варіюється від -1 до 1, де значення 1 вказує на повну схожість (вектори спрямовані однаково), 0 вказує на відсутність схожості (вектори ортогональні), а -1 свідчить про повну протилежність (вектори спрямовані в протилежні сторони) [27].

Далі, для конкретного користувача, що запитує рекомендації, обчислюються схожі користувачі. Система враховує лише тих користувачів, чия схожість перевищує певний поріг, ігноруючи користувачів без схожості (нульова схожість). На основі оцінок схожих користувачів система рахує кількість позитивних та негативних відгуків для кожного фільму. Це дозволяє виділити фільми, які мають більшу кількість позитивних оцінок у порівнянні з негативними.

На завершальному етапі фільтруються фільми, які вже були оцінені запитувачем, і віддаються рекомендації тих, що мають високий рейтинг серед схожих користувачів. Таким чином, користувач отримує персоналізований список фільмів, які ймовірно сподобаються на основі інтересів схожих користувачів.

Далі треба спроєктувати мікросервіс який буде отримувати запит та повертати рекомендації фільмів для конкретного користувача. У цьому випадку ми маємо справу з системою рекомендацій фільмів, що базується на машинному навчанні за допомогою TensorFlow і TensorFlow Recommenders (TFRS). Логіка та алгоритми, які використовуються в коді, охоплюють кілька ключових аспектів роботи з даними і моделювання. Спочатку здійснюється імпорт та об’єднання декількох наборів даних, які містять інформацію про фільми, їх ключові слова, учасників знімальної групи та інше.

Дані очищаються та трансформуються для використання – наприклад, перетворення текстових списків на зрозумілі рядки та видалення подальших дублікатів [28]. Модель створюється один раз із використанням даних про рейтинги фільмів інших користувачів. Вона не навчається заново при кожному запиті, а замість цього використовує навчені параметри для роботи з новими даними. Модель включає ембедінги для користувачів та фільмів, що дозволяють ефективно обчислювати прогнози та схожості. Коли функція рекомендації викликається, система спочатку відловлює ідентифікатор користувача з повідомлення від брокера повідомлень. За цим ідентифікатором визначається користувач і його поточні оцінки фільмів. На основі цих даних модель вираховує рекомендації. Модель використовує TensorFlow для створення та тренування ембедінгів, які репрезентують користувачів і фільми у векторному просторі. За допомогою цих ембедінгів модель може оцінити схожість між користувачами та фільмами та робити відповідні рекомендації, використовуючи задачі ранжування і вибору. Модель оцінює, які фільми найбільш відповідають перевагам користувача, засновуючись на раніше виставлених оцінках та інформації, отриманій з ембедінгів. Вона враховує не тільки оцінки, але й метадані фільмів, такі як жанри та актори. Основою моделі є ембедінги користувачів та фільмів. Ембедінги – це вектори у багатовимірному просторі, які представляють абстрактні характеристики користувачів та фільмів. Наприклад, у векторі користувача можуть бути

«вбудовані» його переваги щодо жанрів, режисерів або акторів, тоді як вектор фільму містить інформацію про його жанри, популярність чи загальні оцінки. Ключовим елементом рекомендацій є обчислення схожості між ембедінгами. Модель використовує косинусну схожість або інші метрики для визначення ступеня подібності між вектором користувача та векторами різних фільмів.

У задачі ранжування модель намагається передбачити реальну оцінку, яку користувач виставить фільму. Вона використовує архітектуру нейронної мережі для обчислення передбачуваної оцінки на основі векторів користувача та фільму. Функція втрат, така як середньоквадратичне відхилення, використовується для оптимізації ваг моделі [29].

А задача вибірки фокусується на визначенні найбільш релевантних фільмів для користувача. Вона використовує алгоритми, що знаходять кращі збіги між вектором користувача та векторами всіх доступних фільмів, зазвичай використовуючи техніки, як-от «k-nearest neighbors». Модель періодично оновлюється з новими даними про користувачів та їх оцінки, що дозволяє їй адаптуватися до змін у перевагах користувачів та нових трендах у фільмах. Це означає, що система рекомендацій стає лише кращою з часом. TensorFlow і TFRS дозволяють ефективно імплементувати ці процеси завдяки своїм бібліотекам для глибокого навчання та оптимізації роботи з великими наборами даних. Вони надають інструменти для створення, тренування, оцінки та розгортання складних моделей машинного навчання, оптимізуючи обчислення та забезпечуючи високу продуктивність [30].

Таким чином ми маємо гібридну систему рекомендацій, яка включає у себе колаборативну фільтрацію, роботу нейронної мережі та контентну фільтрацію. Поєднання усіх цих елементів повинно зробити вибірку фільмів та серіалів максимально підходящу під вимоги користувача, судячи з його дій у застосунку.

2.5 Проєктування спілкування застосунку з клієнтом та між мікросервісами

Клієнтський застосунок повинен надсилати запити на сервісний застосунок і отримувати відповіді – це базова логіка роботи майже усіх вебзастосунків, які існують на даний момент. Найкраще для цього підходить технологія REST.

REST, або Representational State Transfer, є архітектурним стилем, що широко використовується для проєктування мережевих застосунків, зокрема вебсервісів. Цей стиль був розроблений Роєм Філдінгом, одним із головних авторів специфікації HTTP. Основна ідея REST полягає в тому, що вебзастосунки повинні бути структуровані як набір ресурсів, до яких можна доступатися і керувати через стандартні HTTP методи.

У REST кожен ресурс ідентифікується за допомогою URI (Uniform Resource Identifier). Клієнти взаємодіють з цими ресурсами, використовуючи стандартні методи HTTP, такі як GET для отримання даних від сервера, POST для створення нового ресурсу на сервері, PUT для оновлення існуючого ресурсу і DELETE для видалення ресурсу.

Комунікація в стилі REST базується на безстаних запитах і відповідях. Це означає, що кожен запит від клієнта до сервера містить усю необхідну інформацію для обробки запиту. Сервер не зберігає ніякого стану клієнта між запитами. Це сприяє простоті і видимості взаємодій, а також полегшує масштабування системи, оскільки не потрібно синхронізувати стан між різними серверами.

Для формату обміну даними в REST часто використовуються формати, такі як JSON або XML, які дозволяють легко серіалізувати і десеріалізувати дані. JSON зокрема став дуже популярним, бо він легко інтегрується з JavaScript, що робить його особливо підходящим для сучасних вебзастосунків. Ще однією ключовою особливістю REST є використання гіпермедіа (HATEOAS – Hypermedia as the Engine of Application State). Це принцип, за

яким клієнтські застосунки керуються динамічною навігацією, заснованою на гіперпосиланнях, що отримуються з відповідей сервера. Це забезпечує більшу гнучкість, оскільки клієнти не потребують знання про те, як конструювати URI для взаємодій з сервером на основі жорстко закодованих шляхів.

Якщо розглядати адміністративний застосунок, у якому для створення і керування фільмами та серіалами, і деякі функції потокового сервісу, потрібні будуть такі ресурси як актори, режисери, фільми, серіали та жанри, при тому їх отримання може здійснюватися зовсім по різним параметрам, з різними фільтраціями і у різній кількості. Для того, щоб не реалізовувати багато ендпоінтів для отримання значень і не дублювати більшість коду, потрібно використати мову запитів GraphQL.

GraphQL – це мова запитів для API, розроблена Facebook, яка дозволяє клієнтам точно визначати, які дані вони хочуть отримати або змінити. Відмінною рисою GraphQL є те, що вона дозволяє зробити запит на складні, вкладені дані за допомогою одного запиту, що знижує кількість необхідних запитів до сервера та усуває проблему надлишковості інформації, що отримується. Наприклад, якщо адміністративний застосунок потребує отримання інформації про фільм, включаючи дані про акторів, режисера та жанри, все це можна зробити за допомогою одного запиту в GraphQL, вказавши лише потрібні поля. Це зменшує витрати на трафік і оптимізує процес взаємодії з сервером. Крім того, GraphQL має потужну систему типів, яка допомагає в автоматичному документуванні API і забезпечує валідацію запитів на етапі їх формування. Це значно полегшує розробку як на стороні сервера, так і на стороні клієнта, оскільки розробники точно знають, які дані вони можуть відправляти і отримувати.

Щодо Hot Chocolate, це одна з реалізацій GraphQL для .NET Core, яка дозволяє легко інтегрувати GraphQL у вебзастосунку, розроблені на ASP.NET Core. Hot Chocolate дозволяє створювати схеми GraphQL, обробляти запити і пов’язувати типи .NET з типами GraphQL. Він також забезпечує підтримку передових функцій, таких як підписки (subscriptions), що дозволяють клієнтам

отримувати оновлення в реальному часі, використовуючи вебсокети. Інтеграція Hot Chocolate в ASP.NET Core загалом вимагає додавання бібліотеки через NuGet, налаштування мідлваре у файлі Startup, що дозволяє обробляти запити GraphQL, та розробку схеми, яка визначає, як дані пов’язані з типами GraphQL. Ця платформа допомагає створити ефективний, легко масштабований API, який може обслуговувати різні запити клієнтів із мінімальними затримками.

Оскільки застосунок потокового сервісу для перегляду фільмів та серіалів складається ще й з двох мікросервісів на мові Python, які було розглянути раніше, нам потрібно використовувати брокер повідомлень, щоб сповіщати ці мікросервіси про різні дії у головному застосунку ASP NET CORE. Для цієї задачі чудово підійде брокер повідомлень RabbitMQ, про який йшлося у першому розділі.

Таким чином ми маємо налагоджену систему комунікації між усіма компонентами потокового сервісу. ASP NET CORE застосунок отримує запит від клієнту, за потреби звертається до AWS S3, надсилає запит потрібним мікросервісам, звертається до бази і надає відповідь клієнту за допомогою REST (рис. 2.3).

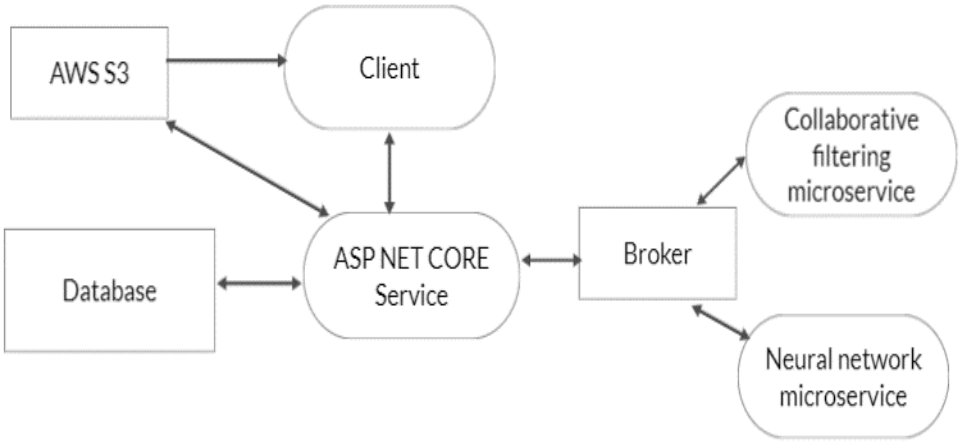

Рисунок 2.3 – Діаграма комунікації застосунків та мікросервісів у потоковому сервісі та перегляду фільмів та серіалів

2.6 Створення авторизації та аутентифікації за допомогою JWT

Авторизація та аутентифікація у застосунку здійснюються за допомогою JWT (JSON Web Tokens), які є стандартним засобом безпечного обміну заявками між двома сторонами. Цей процес починається зі створення та верифікації токенів, що дозволяє системі ідентифікувати користувачів та забезпечувати їм доступ до ресурсів згідно з їх ролями та дозволами. У ядрі системи лежить клас JwtProvider, що відповідає за генерацію JWT. Початковим етапом є отримання налаштувань JWT через залежності, які ін’єктуються в конструктор. Ці налаштування містять ключ безпеки, емітента, аудиторію та час життя токена.

Далі процес створення токена розгортається наступним чином. Спочатку формується масив заявок (claims), які включають унікальний ідентифікатор користувача, його електронну пошту та роль. Ці дані інкапсульовані в структуру Claim, яка є стандартною для .NET. Різні методи класу Claims відповідають за створення різних наборів заявок залежно від типу користувача: звичайний користувач, адміністратор, користувач без підписки. Після формування необхідних заявок, виконується їх серіалізація у вигляді JWT. Для цього використовується симетричний ключ безпеки, який генерується з рядка конфігурації. Симетричний ключ застосовується разом з алгоритмом HMAC SHA256 для підписання токена, що гарантує його непідробленість та цілісність.

Алгоритм HMAC SHA256 є одним із методів забезпечення цілісності та автентичності даних за допомогою криптографічного хешування. Цей алгоритм широко використовується для підпису даних, зокрема у створенні безпечних токенів, як JWT. Він є частиною сімейства криптографічних хеш-функцій SHA-2, розроблених Національним інститутом стандартів і технологій (NIST). Ця функція приймає вхідні дані будь-якої довжини та перетворює їх у вихідний рядок фіксованої довжини 256 бітів (32 байти).

Основна мета SHA256 – забезпечити унікальний і непередбачуваний хеш, що мінімізує ймовірність отримання однакових хешів з різних вхідних даних (колізії). HMAC посилює криптографічну стійкість хеш-функції, інтегруючи секретний ключ у процес хешування. Цей ключ є таємною послідовністю, яка відома лише джерелу та отримувачу повідомлення, що значно ускладнює спроби злому.

Якщо довжина ключа більша за блок хеш-функції (у випадку SHA256 – 512 бітів або 64 байти), ключ спочатку хешується за допомогою SHA256, а потім доповнюється нулями до довжини блоку. Якщо ключ коротший, він доповнюється нулями до 64 байтів. До ключа додаються два типи заповнення: внутрішнє заповнення (ipad), де ключ перетворюється з послідовністю байтів, що складається з повторюваних значень 0x36 та зовнішнє заповнення (opad), де той самий ключ перетворюється з послідовністю байтів, що складається з повторюваних значень 0x5C. У внутрішньому хеші результат XOR ключа з ipad конкатенується з вхідними даними та хешується. У зовнішньому хеші результат XOR ключа з opad конкатенується з внутрішнім хешем і знову хешується. Результатом цього дворівневого процесу хешування є остаточний HMAC, який може бути використаний для перевірки цілісності та автентичності даних. Цей підхід гарантує, що зміна навіть одного біта в оригінальних даних або у ключі призведе до зміни вихідного HMAC, що робить його важливим інструментом у боротьбі з підробками та несанкціонованим доступом.

JWT формується з декількох частин: заголовка, який визначає тип токена та алгоритм його підпису, вантажу, який містить заявки та інформацію про токен, та підпису, який є результатом шифрування попередніх двох частин за допомогою ключа безпеки. Цей токен має обмежений час дії, який визначається та контролюється за допомогою налаштувань системи. Коли користувач намагається отримати доступ до ресурсу, сервер перевіряє пред’явлений JWT, розшифровуючи його за допомогою того ж ключа, який був використаний для його створення. Сервер перевіряє справжність підпису

та актуальність токена, а також зіставляє заявки в токені з дозволами, необхідними для доступу до запитуваних ресурсів. Це забезпечує, що користувач має відповідні права для виконання певних дій.

Цей механізм не тільки забезпечує безпеку обміну даними та контроль доступу до ресурсів, але й дозволяє ефективно масштабувати систему, оскільки токени JWT можуть оброблятися незалежно без потреби постійно звертатися до бази даних для перевірки прав користувачів. Такий підхід ідеально підходить для розподілених систем та застосунків з великою кількістю користувачів.

Реєстрація залежностей та налаштування автентифікації та авторизації за допомогою JWT в середовищі Dependency Injection (DI) вашого застосунку відбувається через серію конфігураційних кроків, які інтегруються у систему під час запуску застосунку. Цей процес забезпечує, що всі компоненти застосунка мають доступ до потрібних налаштувань та служб для обробки аутентифікації та авторизації користувачів.

Dependency Injection – це техніка програмування, яка використовується для забезпечення більшої гнучкості та зменшення залежностей між компонентами програми. Основна ідея полягає у внесенні залежностей об’єктів ззовні, а не їх створенні всередині об’єктів. Це дозволяє кожному компоненту залишатися незалежним від інших компонентів програми, спрощуючи управління залежностями та тестування програм.

У класичному програмуванні, компоненти часто створюють свої залежності власноруч, що робить їх тісно пов’язаними та важкими для тестування або перевикористання. За допомогою DI, ці залежності вводяться в компоненти зовні, зазвичай через конструктор, методи установки або властивості. Це забезпечує більшу модульність та гнучкість програмного забезпечення, оскільки дозволяє змінювати поведінку компонентів без зміни їхнього коду. DI забезпечується за допомогою так званого «контейнера DI», який управляє створенням об’єктів та розподілом їх залежностей. Контейнер зберігає «рецепти» для створення кожного типу об’єкта, які називаються

реєстраціями або прив’язками. Коли програма потребує певного об’єкта, вона звертається до контейнера, який перевіряє наявні реєстрації та створює об’єкт, ін’єктуючи усі необхідні залежності.

Цей підхід також підтримує принцип «інверсії контролю», де програмні компоненти не контролюють своє життя і залежності, натомість цим керує зовнішнє середовище або фреймворк. В результаті, компоненти стають більш абстрактними та легшими у управлінні, що веде до кращої підтримки та можливості повторного використання коду. Таким чином, Dependency Injection допомагає зменшити прямі залежності між класами, підвищує гнучкість програми, полегшує тестування та сприяє кращому дизайну програмної архітектури, роблячи програмні компоненти легшими для розуміння та змін.

Застосунок спочатку реєструє метод автентифікації за замовчуванням, у цьому випадку, через механізм JwtBearer, який є стандартом для роботи з API, що використовують JWT. Ця схема визначається як основний спосіб розпізнавання користувачів. Під час реєстрації схеми JwtBearer, система конфігурується за допомогою параметрів, що визначають правила перевірки токена. Забезпечує, що токен був випущений довіреною стороною і призначений для використання у вашому застосунку. Гарантує, що токен ще дійсний і не прострочений. Запобігає використанню підроблених токенів, переконуючись, що підпис був створений за допомогою відомого і секретного ключа. становлення параметра на нуль забезпечує строге дотримання часу життя токена без будь-якого додаткового часового проміжку [31].

Після налаштування автентифікації система реєструє різні політики авторизації, кожна з яких вимагає певні заявки (claims) від користувачів, що намагаються отримати доступ до ресурсів. Політики можуть вимагати, щоб користувач мав певну роль, таку як «Admin», «User» чи «NotSubscribedUser», що вказує на різні рівні доступу до ресурсів застосунку.

Ролі визначають загальний рівень доступу користувача. Наприклад, роль «Admin» може мати доступ до всіх ресурсів, включаючи адміністративну

панель і функції керування користувачами. Роль «User» може мати доступ лише до основного функціоналу застосунку, тоді як роль «NotSubscribedUser» може мати обмежений доступ, наприклад, тільки до базового контенту без преміум-функцій. Заявки – це конкретні атрибути, що асоціюються з користувачем. Вони можуть включати інформацію про електронну пошту користувача, підтверджений статус, дати реєстрації, і інші деталі, які можуть бути використані для додаткової авторизації.

Ключі та параметри для JWT зберігаються у конфігураційних файлах або змінних середовища, звідки вони завантажуються за допомогою інтерфейсу. Це дозволяє легко змінювати налаштування без переписування коду. Використання інтерфейсу забезпечує централізоване управління конфігурацією, що полегшує підтримку і налаштування безпеки застосунку.

Цей підхід до реєстрації та налаштування залежностей забезпечує, що застосунок має гнучкі та безпечні механізми для управління аутентифікацією та авторизацією користувачів. Завдяки централізованому управлінню конфігурацією та використанню зовнішніх налаштувань, ви можете адаптувати поведінку безпеки свого застосунку до змінюваних вимог без необхідності глибоких втручань у код.

Політики авторизації можуть бути налаштовані так, щоб перевіряти наявність конкретних заявок. Наприклад, політика може вимагати, щоб користувач мав заявку з певним значенням, що дозволяє контролювати доступ до конкретних функцій або ресурсів. Це дозволяє створювати гнучкі правила доступу, що відповідають потребам застосунку. Таким чином, налаштування і управління політиками авторизації, ключами та параметрами JWT забезпечує високий рівень безпеки і гнучкість застосунку, дозволяючи легко адаптуватися до нових вимог і підтримувати стабільну роботу системи. Крім того, використання централізованих політик авторизації сприяє підтримці коду, роблячи його більш організованим і легким для розуміння. Завдяки такому підходу розробники можуть швидко вносити зміни в політики безпеки, що дозволяє оперативно реагувати на нові загрози та вимоги бізнесу.

3 РОЗРОБКА ПОТОКОВОГО СЕРВІСУ ДЛЯ ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ

3.1 Вибір середовища для розробки

Для розробки застосунку були обрані такі засоби: Rider, Postgres, WebStorm, Docker Desktop для роботи у контейнерах Radis та RabbitMQ. Rider є дуже зручним та сучасним інструментом для роботи з платформою .NET, який включає у себе весь необхідний функціонал для веброзробки. Його конкурентом є відома Visual Studio, але у випадку обраного стеку для цього застосунку її додаткові функції, що впроваджені у неї одразу після завантаження, є надлишковими і не мають сенсу. Саме тому для написання ASP NET CORE застосунку з усіма додатковими до нього проєктами була обрана IDE Rider.

Postgres є універсальним і продуктивним рішенням, коли мова іде про вибір реляційної бази даних. Ця СУБД не прив’язана для конкретної платформи і може бути запущена на будь якій системі. Усі застосунки для графічного менеджменту бази є легкими та сучасними, а самі запити до бази проходять швидко і оптимально, нічим не уступаючи іншим аналогам. У .NET є готове рішення по впровадженню бази даних Postgres у застосунок за допомогою npm пакету.

WebStorm – це IDE, яке в основному використовують для написання застосунків, що слугують фронтендом. У нашому випадку ми повинні створити два застосунки з бібліотекою React на мові Typescript. Оглянувши потреби, можна прийти до висновку що WebStorm є гарним вибором який нічім не поступається конкурентом та ідеально підходить для реалізації поставленої задачі.

Docker Desktop – це застосунок, який забезпечує інтегроване середовище для розробки, використання та управління Docker контейнерами на робочих станціях з операційними системами Windows або macOS. Він інтегрований з

операційною системою користувача, що дозволяє використовувати Docker безпосередньо на робочому столі без необхідності налаштовувати додаткові віртуальні машини або інше проміжне ПЗ. На Windows він може використовувати WSL 2 (Windows Subsystem for Linux) або Hyper-V для емуляції Linux середовища, де Docker Engine запущений. Включає в себе Docker Engine, який є основним компонентом, що керує створенням, запуском і розподілом Docker контейнерів. Docker Engine взаємодіє з ОС через серію драйверів та API, забезпечуючи ізоляцію та управління ресурсами. Він надає графічний інтерфейс користувача (GUI), що дозволяє користувачам легко керувати контейнерами, образами та вольюмами. Користувачі можуть візуально стежити за станом своїх контейнерів, використовувати drag-and-drop для управління образами, а також моніторити використання ресурсів.

3.2 Реалізація серверної та клієнтської частини

Серверний застосунок має структуру, яка складається з 7-ми різних проєктів які були описані раніше (рис. 3.1). У папці API міститься проєкт StreamingService.Api. Це ASP NET CORE застосунок у якому реалізовані ендопінти і який являє собою точку входу у застосунок. У цьому проєкті реєструються усі залежності та налаштовується основна конфігурація. Інші проєкти містяться у папці Core. Проєкт StreamingService.Test знаходиться окремо від двох папок і містить тести для сервісної частини застосунку. Усі проєкти написані на 7 версії .NET. У цій сучасній версії додано багато зручних технологій для написання коду та покращена роботи із загальними та базовими технологіями, які використовуються у багатьох проєктах протягом значного проміжку часу існування платформи. Ці покращення значно підвищують продуктивність розробників, дозволяючи їм зосередитися на бізнес-логіці замість боротьби з технічними проблемами.

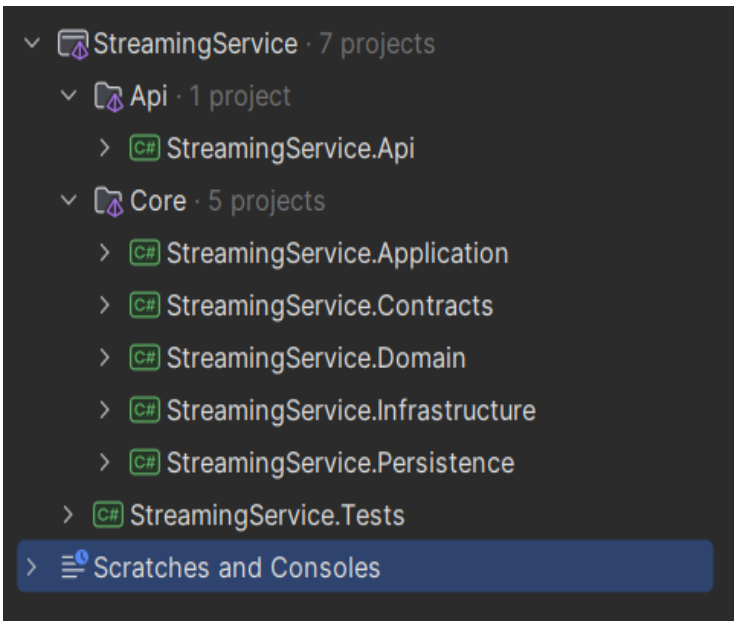

Рисунок 3.1 – Структура сервісної частини застосунку

Клієнтські застосунки для користувацької та адміністративної частини мають однакову структуру. Самі проєкти мають поділяються на папки public та src (рис. 3.2). У папці public зберігаються усі шрифти та зображення, що використовуються у системі. У папці src розміщені усі основні елементи застосунку, а саме: папка api, у якій створені основні налаштування для надіслання запитів та первинної обробки відповідей з серверу; папка app, у якій містяться усі стандартні налаштування застосунку і головний файл який є стартовою точкою; папка components у якій містяться усі основні сторінки с застосунку і допоміжний функціонал для них; папка features містить усі функції застосунку, по суті у ній усі логічні частини, у яких свій сервіс для запитів, свої моделі даних та модулі; папка components, у якій реалізовані усі компоненти візуальної частини для повторного використання та папка store, у якій збережені налаштування для системи зберігання даних у клієнтському застосунку у рамках сеансу. Також, для забезпечення зручності розробки та підтримки коду, всі папки у структурі організовані за принципом модульності, що дозволяє легко додавати нові функції та компоненти без ризику порушення існуючої логіки.

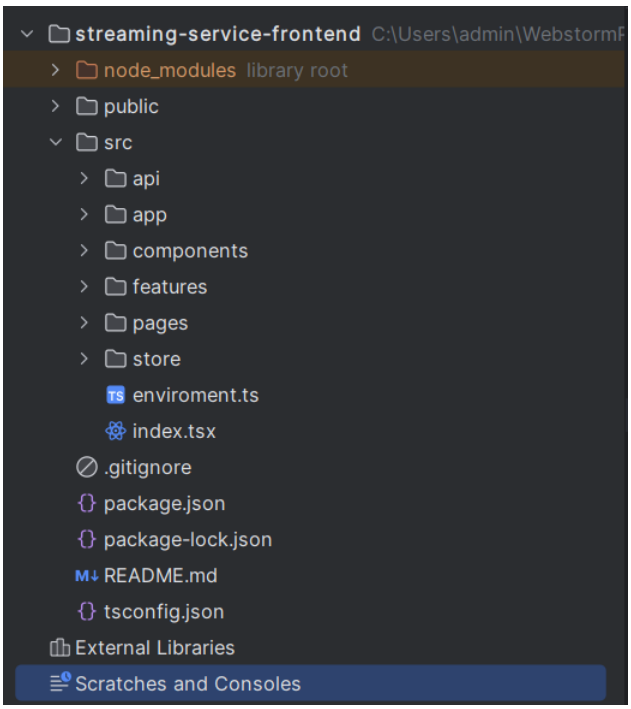

Рисунок 3.2 – Структура клієнтських частин застосунку

3.3 Огляд функціоналу сервісу та його тестування

Почнемо з початкової сторінки застосунку (рис. 3.3). Вона буде показана користувачу у тому разі, якщо він на даний момент не авторизований у застосунку і переходить на головну сторінку.

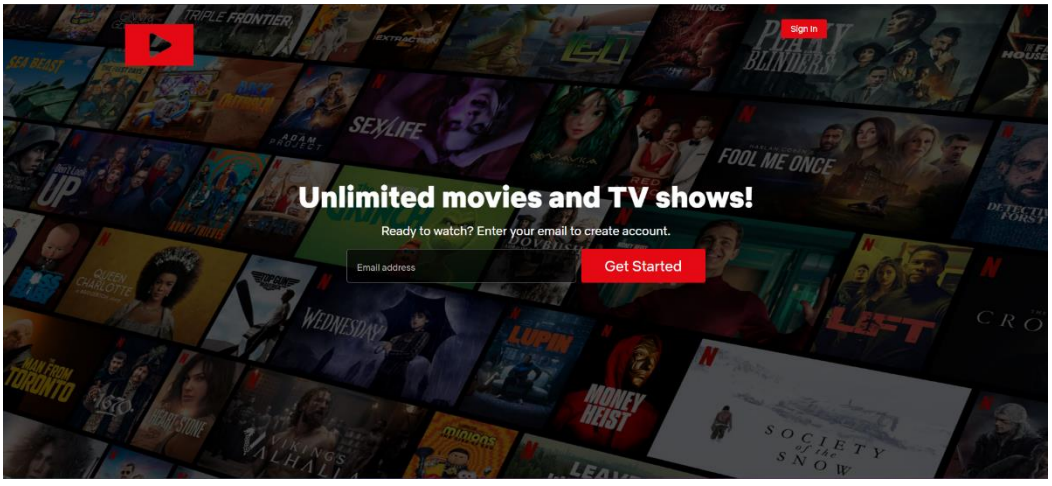

Рисунок 3.3 – Початкова сторінка застосунку для не авторизованого користувача

На сторінці можна побачити поле для вводу email адреси для початку чи продовження реєстрації та кнопку переходу на сторінку входу. Якщо ввести email у неправильному форматі, від полем вводу виникне помилка з відповідним текстом (рис. 3.4).

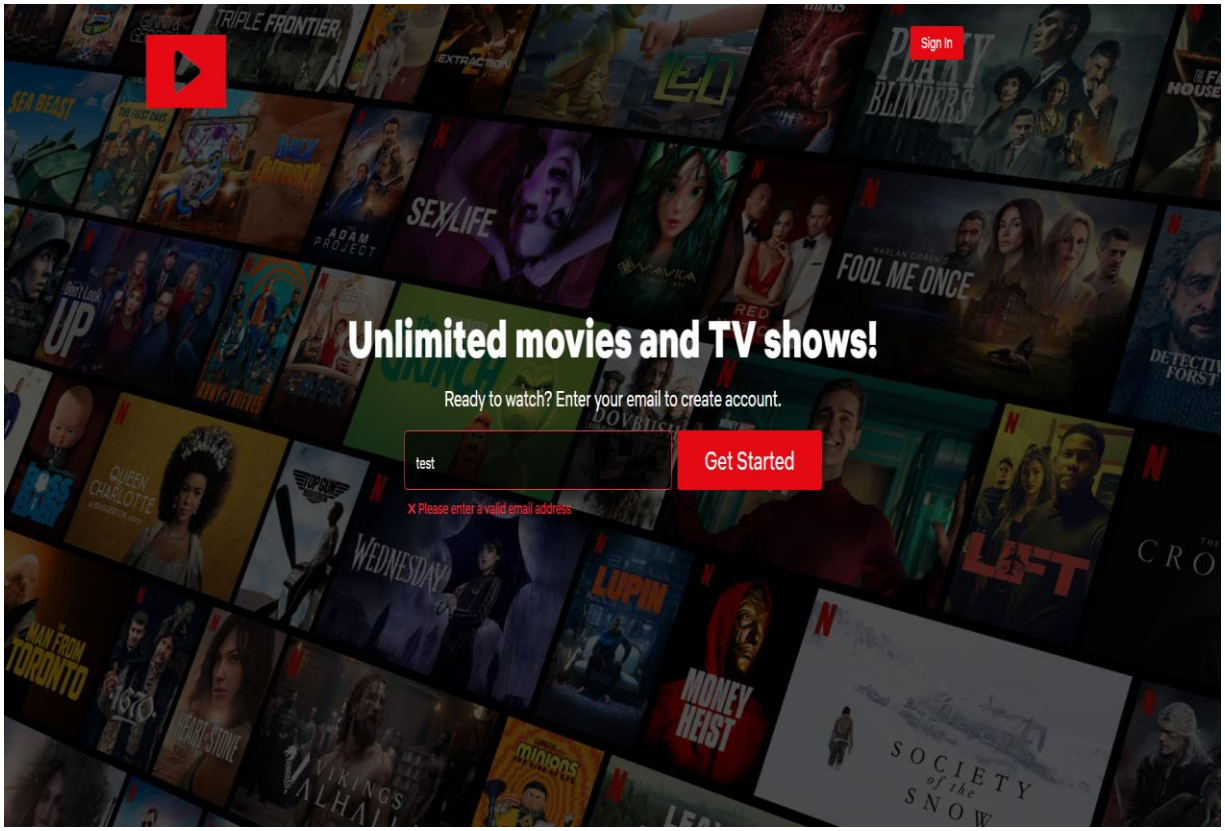

Рисунок 3.4 – Відображення помилки, якщо користувач ввів email у невірному форматі

Після введення коректної електронної адреси користувача буде перенаправлено на сторінку реєстрації, яка складається з трьох кроків. У першій частині потрібно буде підтвердити введену раніше адресу та ввести пароль. Введена раніше адреса буде автоматично підтягнута при завантаженні сторінки. Якщо ввести дані у невірному форматі, під відповідними полями будуть показані помилки (рис. 3.5). Після успішного введення і підтвердження даних користувач зможе перейти до наступних кроків реєстрації, що включають додаткову інформацію і налаштування облікового запису.

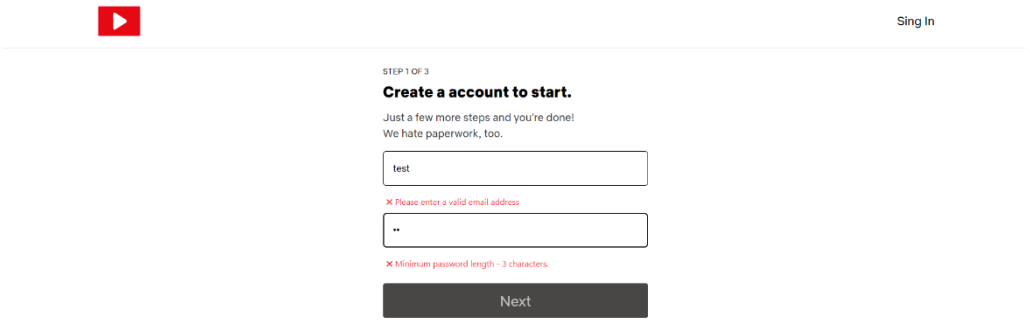

Рисунок 3.5 – Перший крок реєстрації з введеними у невірному форматі даними

Якщо користувач буде намагатися створити акаунт на email, на який вже було зареєстровано акаунт, він отримає відповідну помилку від сервера, яка сповістить його про це (рис. 3.6).

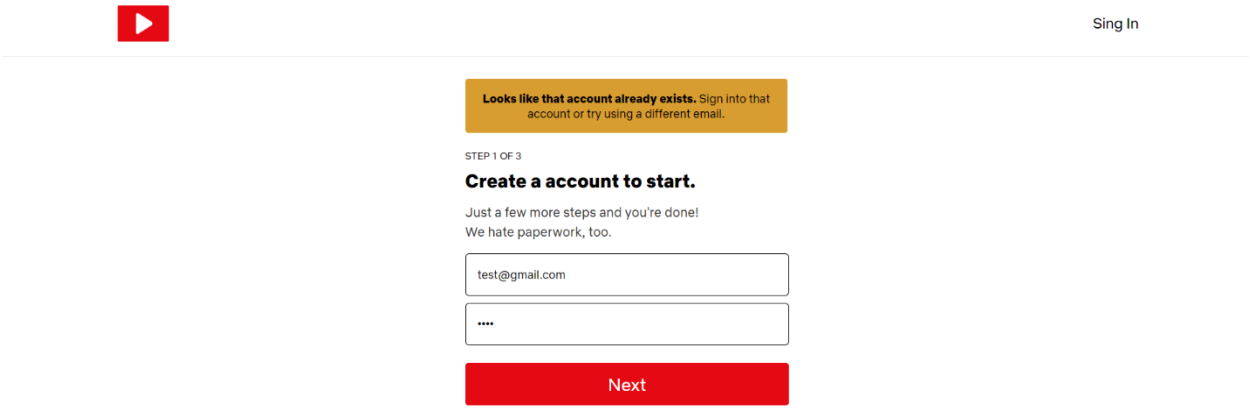

Рисунок 3.6 – Спроба перейти на наступний крок реєстрації з email, на який раніше вже було створено акаунт

Наступний крок реєстрації це дані про користувача. Якщо користувач введе занадто коротке ім’я або занадто малий вік, він отримає відповідні помилки та не зможе продовжити реєстрацію (рис. 3.7).

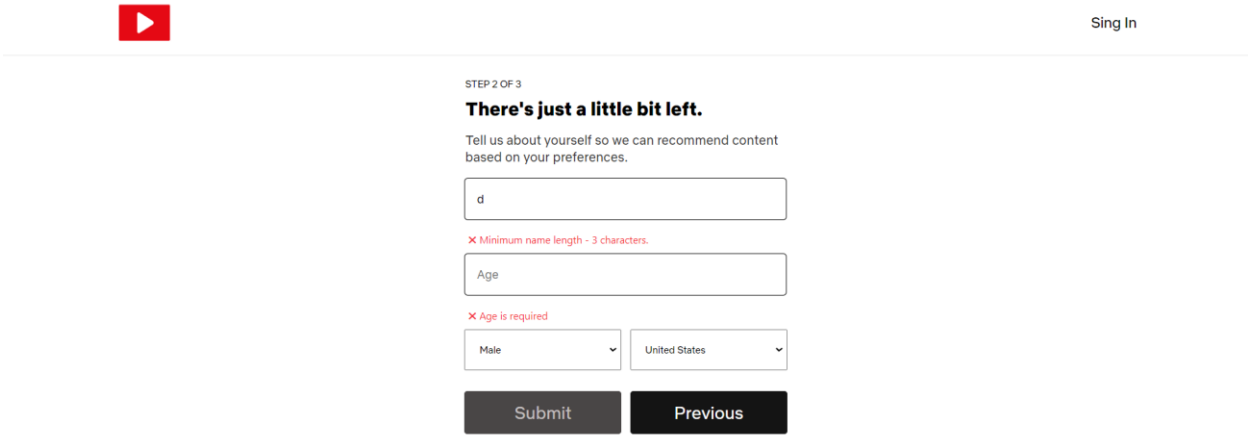

Рисунок 3.7 – Другий крок реєстрації з неправильно введеними даними

Останнім кроком реєстрації є введення своїх платіжних даних для початку підписки на застосунок. Потрібно ввести номер картки, дату її вичерпання та секретний код, а також встановити флаг на згоду з умовами користування сервісом (рис. 3.8).

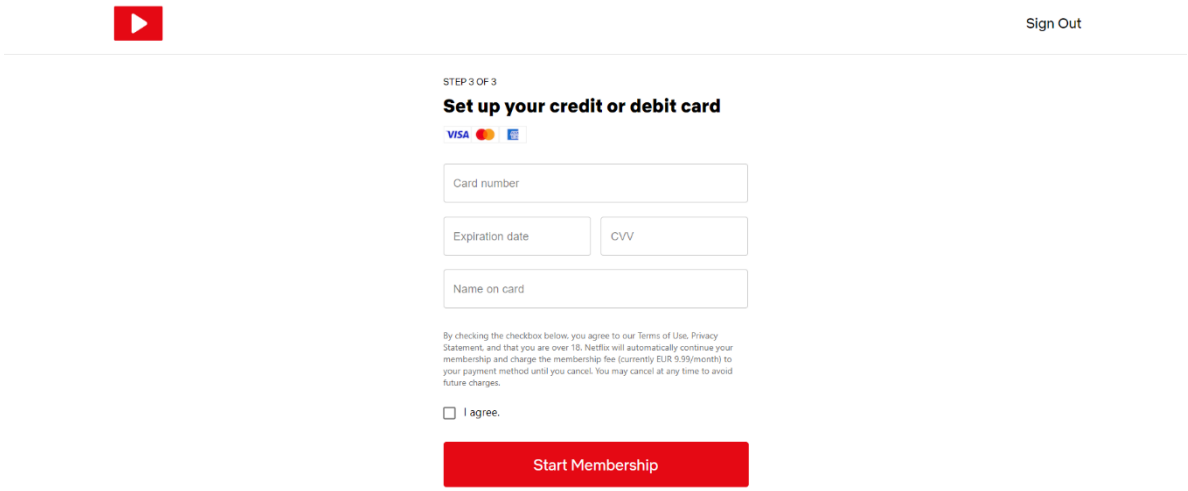

Рисунок 3.8 – Останній крок реєстрації акаунту з введенням платіжних даних

На цій сторінці також є валідація даних і якщо ввести дані у невірному форматі, користувач отримає відповідні помилки. Також надіслати запит на створення підписки неможливо, поки користувач не виправить усі помилки

вводу платіжних даних та не прийме умови договору та політики застосунку. Після успішної підписки користувач зможе почати користуватися сервісом для перегляду фільмів та серіалів. Його одразу буде направлено на головну сторінку з фільмами для авторизованих користувачів, де він одразу побачить трейлер одного з закріплених фільмів, інтерфейс користувача та список фільмів, який був йому рекомендований (рис. 3.9).

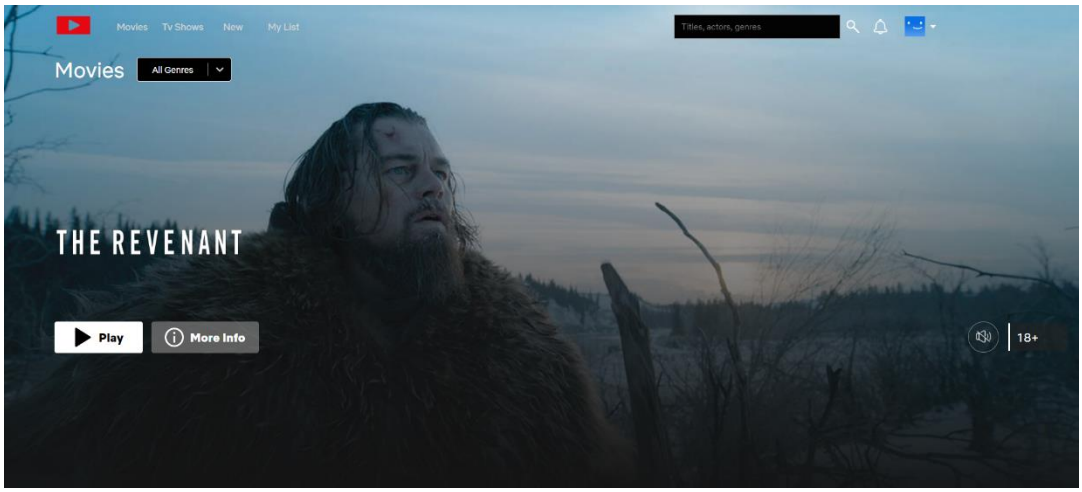

Рисунок 3.9 – Головна сторінка застосунку з фільмами

Якщо навести курсор на один з фільмів чи серіалів, відкриється вікно короткої інформації про цей контент. У цьому вікні буде вказана коротка інформація про фільм або серіал, така як назва, жанр, тривалість, рік випуску та короткий опис сюжету (рис. 3.10).

Вікно короткої інформації також міститиме кілька інтерактивних кнопок, які дозволяють користувачу здійснювати різні дії з обраним контентом. Серед доступних кнопок будуть кнопка для додавання фільму або серіалу до персонального списку користувача, кнопка лайку для позначення контенту як улюбленого, кнопка дизлайку для позначення контенту як непопулярного, і кнопка для перегляду детальної інформації про фільм або серіал. Натиснувши на відповідні кнопки, користувач зможе зберігати вподобані фільми та серіали для подальшого перегляду, виражати свою симпатію до контенту, виключати небажаний контент зі своїх рекомендацій, а

також отримувати доступ до повної інформації про фільм або серіал, включаючи акторський склад, режисерів, відгуки та інші важливі деталі.

Ця функціональність забезпечує зручний і швидкий доступ до основної інформації про фільми та серіали, дозволяючи користувачам ефективно взаємодіяти з контентом і здійснювати необхідні дії без переходу на окремі сторінки. Інтерактивні елементи у вікні короткої інформації підвищують зручність користування сайтом і сприяють кращій організації перегляду контенту.

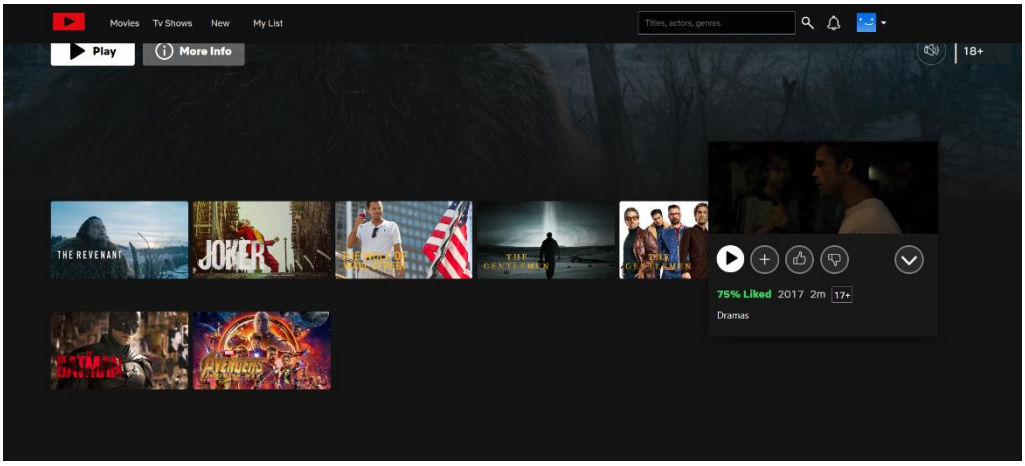

Рисунок 3.10 – Модальне вікно короткої інформації про фільм при наведенні на нього у списку

При кліку на кнопку додаткової інформації відкриється відповідне вікно, де буде представлена уся детальна інформація про фільм, дії до фільму, такі як додавання до списку та фільми, що схожі на відкритий фільм (рис. 3.11). Якщо лайкнути фільм кнопка лайку стане зеленого кольору, а якщо дізлайкнути, кнопка дізлайкну стане червоного кольору. При додаванні фільму чи серіалу у свій список іконка кнопки додавання зміниться з плюсу на галочку. При кліку на схожий фільм відкриється інформація про нього, замінивши інформацію про фільм, що раніше був обраний. Оновлення вікна інформації при кліку на схожий фільм забезпечує безперервний і зручний перегляд контенту, дозволяючи легко перемикатися між різними фільмами.

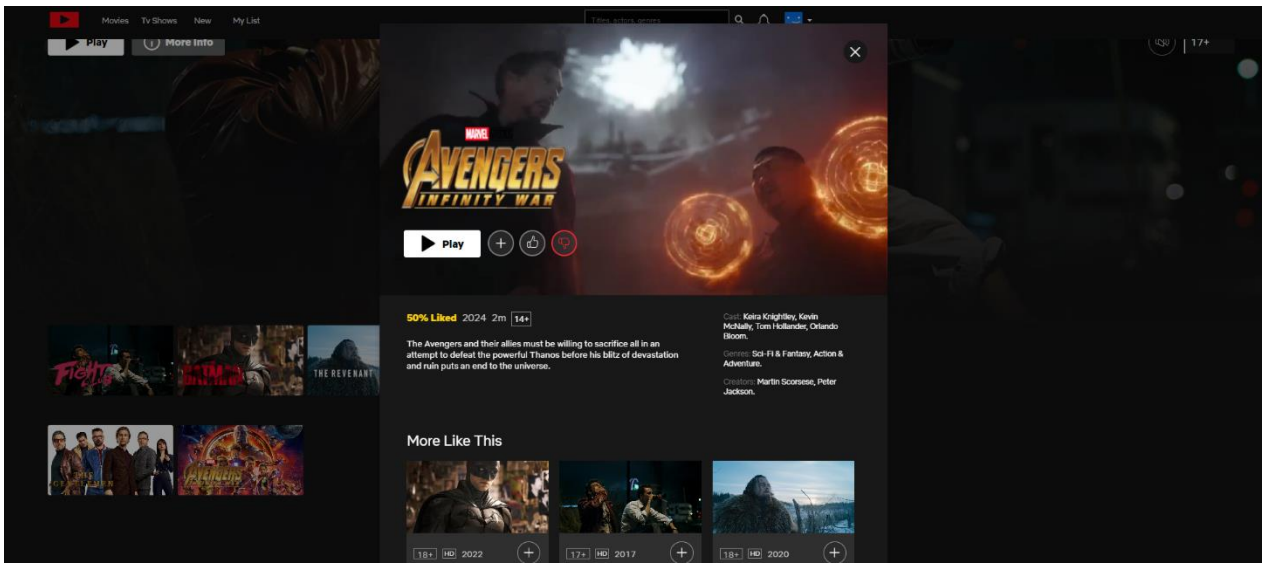

Рисунок 3.11 – Модальне вікно детальної інформації про фільм

При натисканні на кнопку «Play» відкривається програвач, який дозволяє переглядати фільм або серіал (рис. 3.12). Однією з цікавих особливостей цього плеєра є його здатність запам’ятовувати точне місце, де користувач завершив перегляд. Завдяки цьому, при наступному відкритті фільму чи серіалу програвач автоматично відтворює контент саме з того моменту, де користувач зупинився. Це значно підвищує зручність користування і дозволяє продовжити перегляд без необхідності вручну шукати потрібний епізод.

Плеєр оснащений функціоналом для паузи, що дозволяє користувачеві зупинити відтворення у будь-який момент. Крім того, є можливість перемотування фільму чи серіалу на 10 секунд вперед або назад, що дозволяє швидко переглянути вже побачені сцени або пропустити нецікаві моменти. Регулювання звуку забезпечує комфортний рівень гучності під час перегляду, що дозволяє налаштувати звук відповідно до особистих вподобань. Повноекранний режим надає можливість насолоджуватися відео на весь екран, що покращує занурення у контент і робить перегляд більш приємним. Крім цього, плеєр підтримує субтитри, що робить контент доступнішим для широкої аудиторії, включаючи людей з вадами слуху та тих, хто хоче переглядати відео на іншій мові.

Рисунок 3.12 – Плеєр для перегляду фільмів та серіалів

Фільми та серіали можна фільтрувати за жанрами, і для цього на сторінках сайту передбачено статичний випадаючий список, у якому користувач може обрати потрібний жанр. Випадаючий список розташований у зручному місці на сторінці, забезпечуючи легкий доступ до функції фільтрації. Після того, як користувач обрав жанр з випадаючого списку, система автоматично відфільтрує фільми або серіали, залишивши лише ті, які відповідають обраному жанру (рис. 3.13). Це дозволяє користувачу швидко знаходити контент, який відповідає його вподобанням, знижуючи час пошуку і підвищуючи зручність використання сайту.

Функція фільтрації за жанрами є важливою складовою інтерфейсу, оскільки допомагає користувачам орієнтуватися в широкому асортименті контенту і знаходити саме те, що вони хочуть переглянути. Відображення результатів фільтрації є інтуїтивно зрозумілим і відбувається без необхідності додаткових дій з боку користувача, забезпечуючи плавний і приємний досвід користування сайтом.

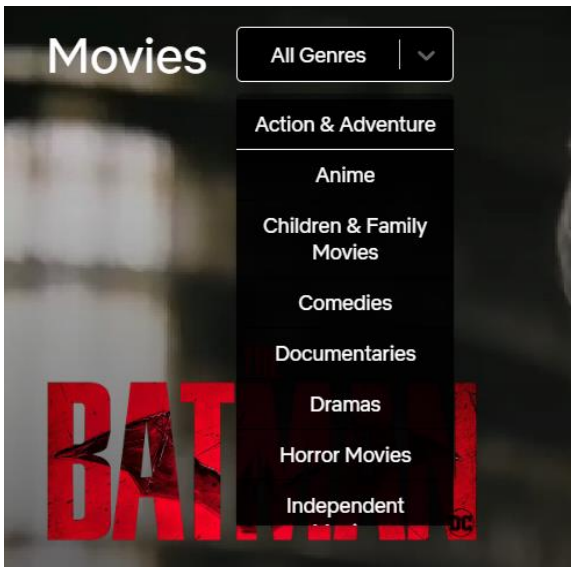

Рисунок 3.13 – Випадаючий список для фільтрації фільмів та серіалів по жанру

Також у застосунку реалізований повний пошук, завдяки якому користувач може шукати фільми та серіали одразу по жанрам, акторам, назві, режисерам та опису. Для цього у шапці застосунку є відповідне поле вводу та кнопка пошуку з іконкою лупи. Ввівши потрібний запит та натиснувши на кнопку користувача направить на сторінку знайдених даних де він зможе побачити свій запит і результат пошуку (рис. 3.14).

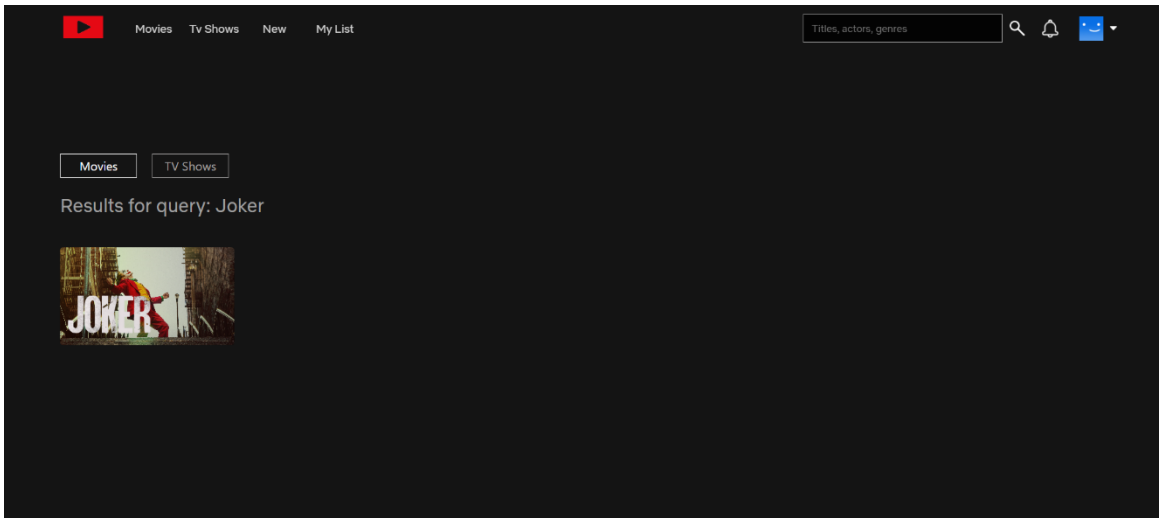

Рисунок 3.14 – Сторінка знайденого контенту по запиту користувача

Цей код реалізує асинхронний метод, який обробляє запит на пошук фільмів, використовуючи параметри запиту. Перш за все, метод приймає об’єкт, який містить інформацію про пошуковий запит користувача, і токен скасування, який дозволяє припинити виконання завдання за певних умов.

Спершу метод звертається до репозиторію фільмів за допомогою асинхронного виклику, щоб отримати список усіх фільмів. Цей виклик блокує подальше виконання методу до тих пір, поки не буде отримано всі фільми з бази даних або іншого джерела. Після отримання списку фільмів, метод отримує рядок пошукового запиту і приводить його до нижнього регістру за допомогою методу, щоб забезпечити нечутливий до регістру. пошук. Далі, метод використовує LINQ-запит, щоб відфільтрувати список фільмів відповідно до пошукового запиту. Він перевіряє, чи містить назва фільму (в нижньому регістрі) підрядок, що відповідає пошуковому запиту, використовуючи метод Contains. Якщо назва фільму містить цей підрядок, фільм додається до результуючого списку. Аналогічно, метод перевіряє, чи містить жанр фільму, ім’я актора або ім’я творця підрядок, що відповідає пошуковому запиту. Для цього використовуються вкладені запити, що перевіряють відповідні колекції (MovieGenres, MovieActors, MovieCreators) за допомогою методу, щоб з’ясувати, чи відповідає хоча б один елемент цих колекцій пошуковому запиту.

Після того, як всі відповідні фільми відфільтровані, метод проєктує результуючий набір фільмів у новий формат, зокрема в об’єкти MovieForUser. Кожен об’єкт створюється на основі відфільтрованого фільму та ідентифікатора користувача, що міститься в запиті. Це забезпечує створення спеціальних представлень фільмів для конкретного користувача. Нарешті, метод повертає результат у вигляді списку, загорнутого в об’єкт, що може включати додаткову інформацію про статус виконання запиту або помилки, якщо вони виникли. Таким чином, код реалізує повний цикл пошуку фільмів за різними критеріями, включаючи назву, жанри, акторів і творців, з

урахуванням нечутливості до регістру, і формує результуючий набір даних, що специфічний для користувача.

Потоковий сервіс включає адміністративну частину, яка надає можливість адміністрування контенту і керування системою. Почнемо з розгляду сторінки логіну адміністративної частини. Сторінка логіну адміністративної частини призначена для автентифікації адміністраторів, забезпечуючи безпечний доступ до функціональності адміністрування. На цій сторінці користувач повинен ввести свою електронну пошту та пароль у відповідні поля. Поле для введення електронної пошти вимагає вказати адресу у вірному форматі, що гарантує правильність введених даних і уникнення помилок, пов’язаних із некоректним введенням. Поле для пароля повинно містити надійний пароль, який відповідає встановленим вимогам безпеки, що забезпечує захист облікових записів від несанкціонованого доступу.

Після введення даних користувач натискає кнопку для входу, щоб здійснити спробу автентифікації. Після цього система перевіряє введені дані на відповідність вимогам формату. Якщо введені дані не відповідають цим вимогам або якщо під час спроби входу сервер повертає помилку, система відобразить відповідні повідомлення-тости (рис. 3.15). Ці повідомлення можуть містити інформацію про те, що введена електронна пошта або пароль є некоректними, або про те, що виникла серверна помилка. Наприклад, якщо користувач ввів неправильний пароль, повідомлення-тост інформує його про це, вказуючи на необхідність повторної спроби входу з правильними даними.

Повідомлення-тости з’являються у верхньому правому куті екрану і автоматично зникають через кілька секунд. Вони допомагають користувачу швидко зрозуміти, що стало причиною невдалої спроби входу, і виправити помилки у введених даних. Це дозволяє зменшити час, витрачений на повторні спроби входу, і підвищити загальну ефективність взаємодії з системою. Такий підхід покращує користувацький досвід, роблячи процес автентифікації більш інтуїтивним і зручним.

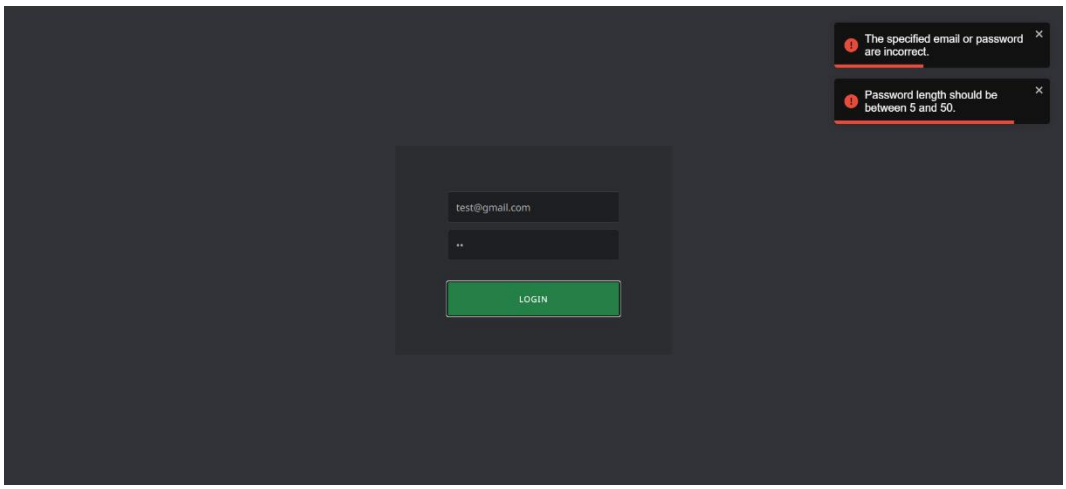

Рисунок 3.15 – Сторінка логіну адміністративної частини застосунку

Після успішного входу в систему адміністратор отримає відповідне повідомлення про успішну автентифікацію і буде автоматично перенаправлений на головну сторінку адміністративної частини застосунку. Це повідомлення слугуватиме підтвердженням того, що введені облікові дані були правильними і доступ до адміністративної частини надано.

На головній сторінці адміністративної частини адміністратор зможе обрати перегляд існуючих фільмів чи серіалів. Відображення контенту відбуватиметься у вигляді таблиць, які містять коротку інформацію про кожен завантажений раніше фільм або серіал (рис. 3.16). У цих таблицях можна буде побачити назву контенту, його жанр, дату завантаження, тривалість, а також інші важливі параметри. Крім того, на цій сторінці можуть бути доступні функції для пошуку та фільтрації контенту, що дозволяє адміністратору швидко знаходити необхідні фільми або серіали за різними критеріями. Наприклад, можна буде здійснювати пошук за назвою, жанром або датою додавання.

Інтерфейс головної сторінки адміністративної частини розроблений таким чином, щоб забезпечити максимальну зручність і ефективність роботи адміністратора. Кожен елемент таблиці може мати інтерактивні кнопки для редагування або видалення відповідного контенту, що дозволяє легко керувати наявним медіаконтентом.

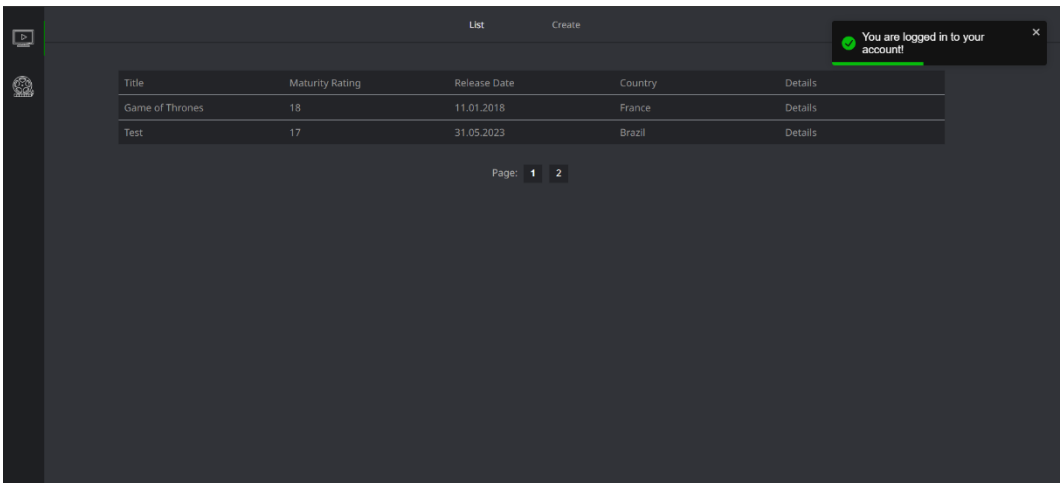

Рисунок 3.16 – Головна сторінка адміністративної частини застосунку

Натиснувши на деталі фільму чи серіалу користувач потрапить на сторінку з їх редагуванням де зможе змінити чи просто переглянути інформацію (рис. 3.17). У нижній частині сторінки можна замінити медіа контент, такий як логотип, трейлер та інше. Також є кнопка «Delete», натиснувши на яку фільм буде видалено, а уся інформація у всіх сховищах буде очищена.

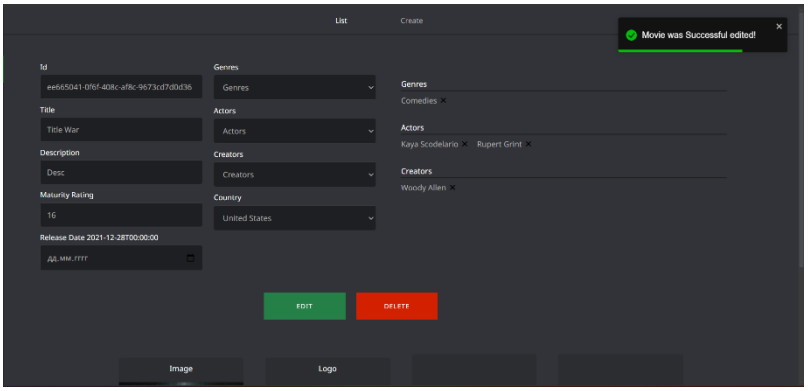

Рисунок 3.17 – Сторінка редагування фільму

У верхній частині цього розділу розташована опція, яка дозволяє переключитися на сторінку створення фільму. На сторінці створення фільму користувачу необхідно буде ввести всю необхідну інформацію про фільм, включаючи вибір акторів, жанрів та режисерів, а також завантаження необхідних медіа даних та іншої інформації. Якщо користувач спробує

створити фільм без заповнення всіх необхідних полів або з некоректними даними, система проведе валідацію введених даних. У випадку виявлення помилок користувач отримає спливаючі вікна з повідомленнями про конкретні недоліки в заповненій інформації (рис. 3.18). Це дозволить йому оперативно виявити та виправити помилки, що забезпечить правильне та повне введення даних. Таким чином, функціонал створення фільму включає в себе зручний та інтуїтивно зрозумілий інтерфейс для введення всієї необхідної інформації, а також ефективну систему валідації даних, що гарантує точність і повноту введених даних.

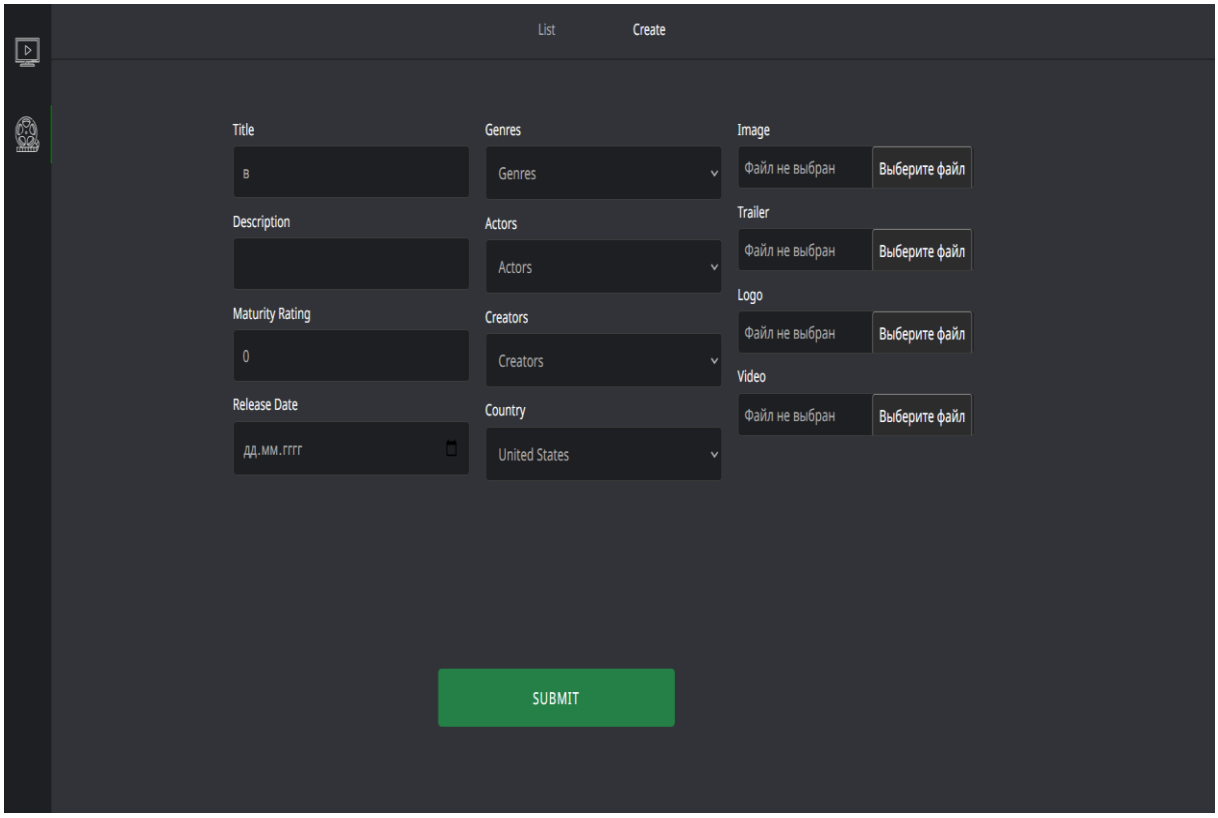

Рисунок 3.18 – Сторінка створення фільму

Сторінка редагування серіалу відрізняється тим, що знизу у неї є список усіх сезонів відповідного серіалу та кнопка для їх створення (рис. 3.19). У списку надається коротка інформація про сезон та його номер. При наведенні курсору на відповідний сезон він буде

підсвічуватися, що демонструє користувачу те, що на нього можна натиснути для перегляду інформації.

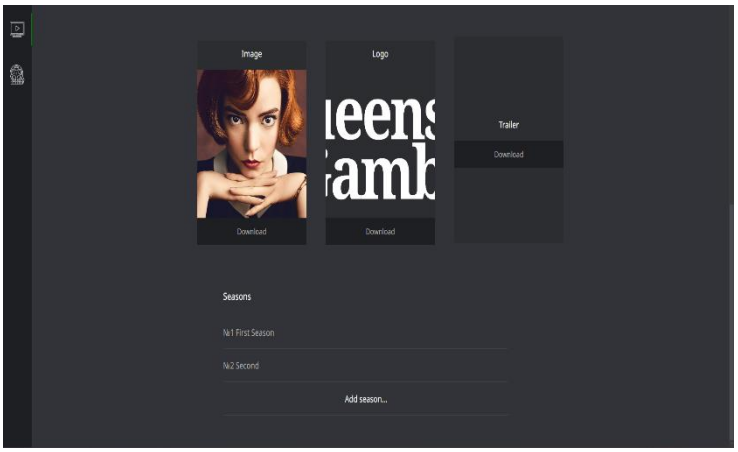

Рисунок 3.19 – Сторінка редагування серіалу

Якщо натиснути на кнопку додавання сезону, відкриється модальне вікно з необхідними полями (рис. 3.20). Додавання сезону не є складною операцією і має перевірку лише на розмір назви сезону.

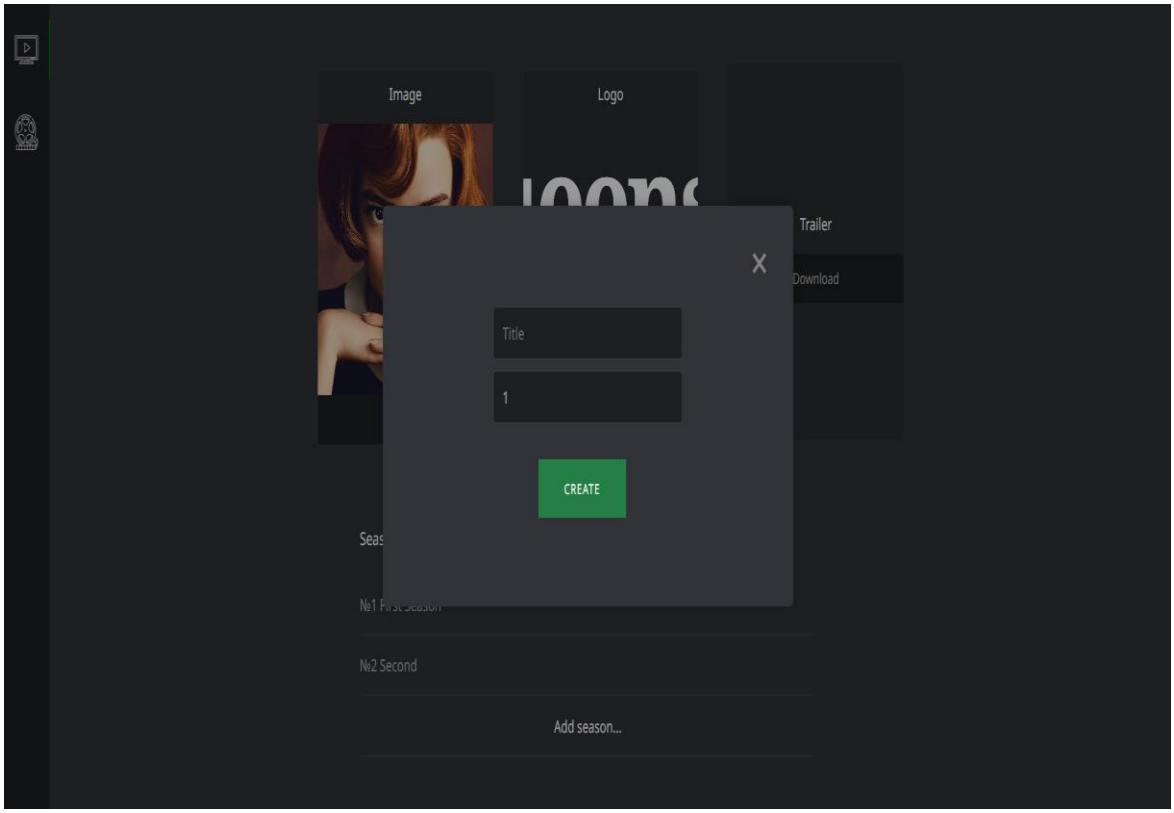

Рисунок 3.20 – Модальне вікно для додавання сезону у серіал

При створенні нового сезону користувач отримає спливаюче повідомлення, яке сповістить про успішне виконання операції. Це повідомлення буде відображати інформацію про успіх створення сезону, надаючи користувачу впевненість у тому, що його дія була успішною. Натиснувши на один із уже створених сезонів, користувач буде перенаправлений на сторінку редагування сезону. На цій сторінці він зможе змінити назву сезону, порядковий номер, а також редагувати епізоди, які входять до цього сезону (рис. 3.21). Це надає користувачу гнучкість у керуванні вмістом сезону та можливість швидко вносити зміни. Ідентифікатор сезону є статичним, що означає, що користувач не має можливості змінювати його. Цей ідентифікатор призначений лише для перегляду, що забезпечує стабільність і унікальність кожного сезону у системі. Загалом, функціонал створення та редагування сезонів надає користувачам інтуїтивно зрозумілий та зручний інтерфейс для управління контентом, забезпечуючи при цьому стабільність і надійність даних.

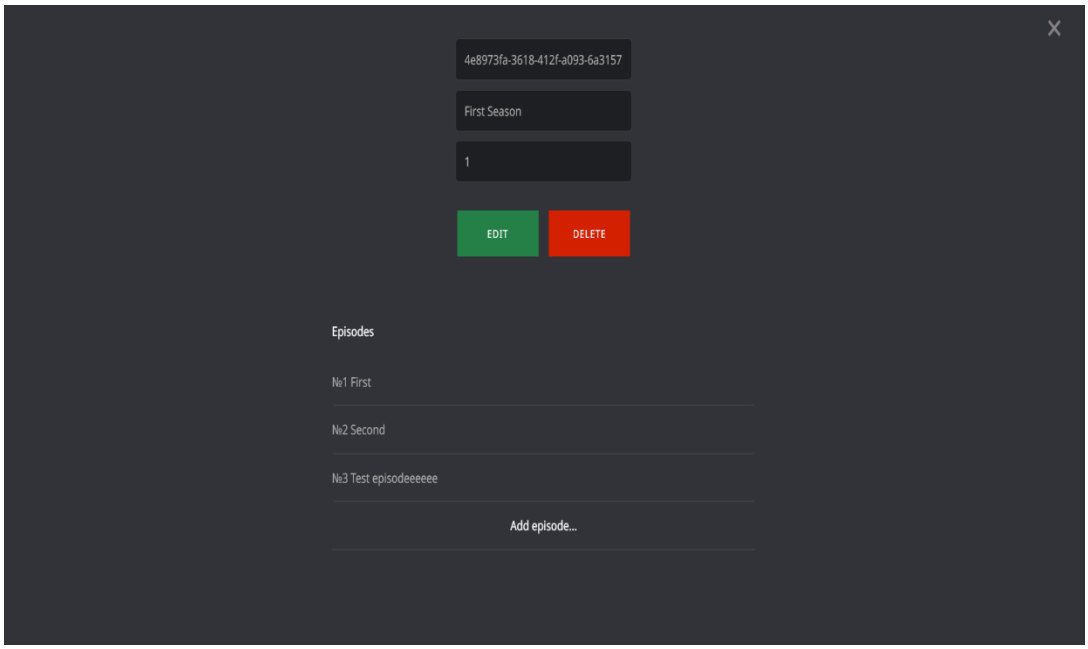

Рисунок 3.21 – Сторінка редагування сезону

При натисканні на кнопку «Створити епізод» відкривається модальне вікно, яке містить декілька важливих елементів. Вікно включає поле для

введення назви епізоду, де користувач може вказати унікальну назву для нового епізоду. Також є поле для введення порядкового номера епізоду, що дозволяє точно вказати місце епізоду у серіалі. Крім того, в модальному вікні є кнопка для завантаження відповідного відео епізоду, що дозволяє користувачам прикріпити файл відео, який буде асоційований з новим епізодом.

На вікні також розташована кнопка «Створити». Натиснувши цю кнопку, користувач ініціює процес валідації введених даних. Під час валідації система перевіряє, чи всі дані введені правильно і чи відповідають вони необхідним форматам. Валідація включає перевірку коректності назви, порядкового номера, а також перевірку формату і розміру завантаженого відео файлу.

Якщо всі дані коректні, епізод успішно створюється, і користувач побачить спливаюче вікно з повідомленням про успішне створення епізоду. Це повідомлення підтверджує, що введені дані відповідають вимогам і новий епізод успішно доданий до системи.

У разі, якщо під час валідації буде виявлено помилки або невідповідності у введених даних, користувач побачить спливаюче вікно з текстом відповідної помилки. Це повідомлення вказує на конкретні проблеми, які потрібно виправити, наприклад, неправильний формат назви або недопустимий файл відео. Такий підхід дозволяє користувачу швидко внести необхідні корективи і повторно спробувати створити епізод.

Завантажене відео епізоду автоматично розміщується у хмарному сховищі, яке відповідає встановленим нормам зберігання даних у застосунку. Використання хмарного сховища забезпечує надійне зберігання відеоматеріалів і доступність даних у будь-який час (рис. 3.22). Це дозволяє забезпечити гнучкість архітектури роботи з даними, оскільки відеоматеріали можна легко зберігати, переглядати і керувати ними без необхідності займатися фізичним зберіганням.

Таким чином, функціонал створення епізодів у системі забезпечує зручність і простоту у використанні. Система гарантує високу якість введених даних завдяки процесу валідації і надійність їх зберігання завдяки використанню хмарного сховища. Користувачі можуть легко створювати нові епізоди, знаючи, що їхні дані будуть збережені в безпечному і доступному місці.

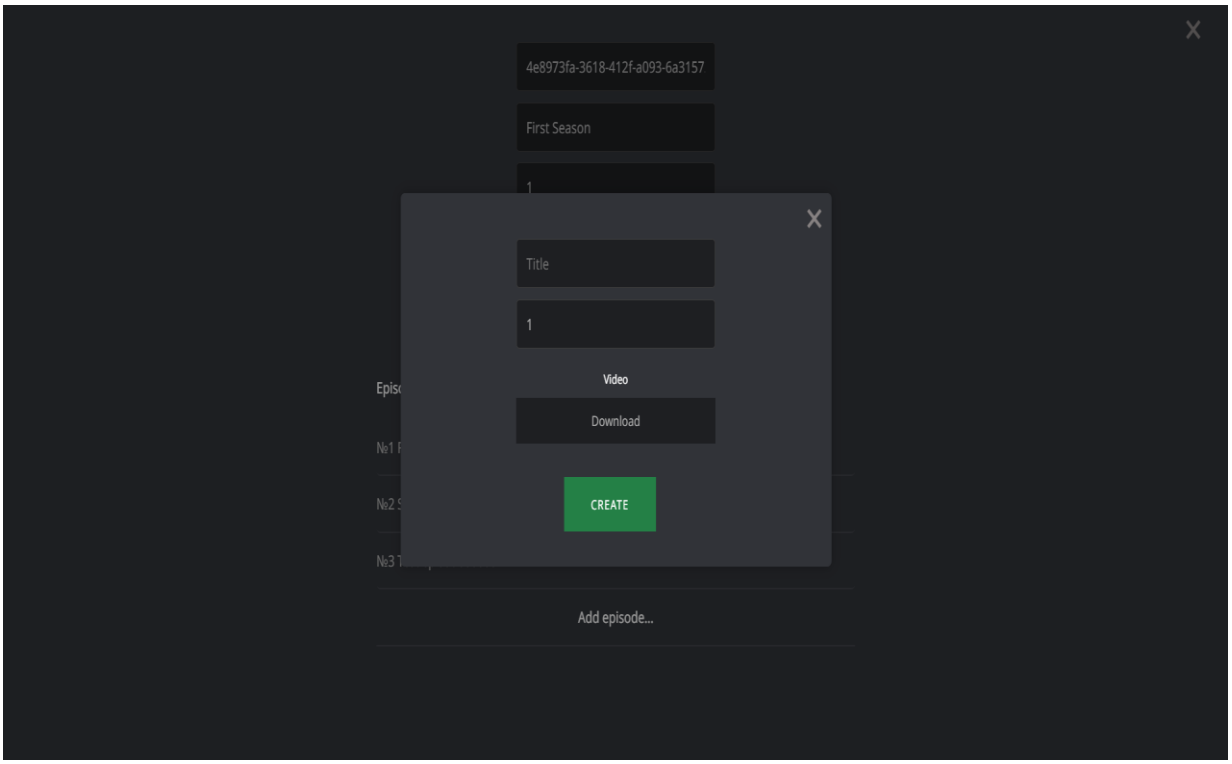

Рисунок 3.22 – Модальне вікно для створення епізоду

Повертаючись до основного застосунку, можна помітити, що час сесії користувача було вичерпано, в результаті чого він опинився на сторінці входу в систему. Для перевірки валідації даних, користувач може спробувати увійти, використовуючи неіснуючий обліковий запис або залишивши поля для вводу даних порожніми (рис. 3.23). У випадку, якщо користувач спробує увійти з неіснуючим обліковим записом, система проведе валідацію введених даних.

Якщо виявиться, що введені дані некоректні або обліковий запис не існує, користувач отримає спливаючі повідомлення з відповідними помилками. Подібним чином, якщо поля для вводу даних залишаються

порожніми, система також видасть спливаючі повідомлення про необхідність заповнення обов’язкових полів.

Цей функціонал забезпечує належний контроль за вірністю введених даних, що сприяє безпеці і правильності процесу авторизації у системі. Спливаючі повідомлення про помилки допомагають користувачам швидко зрозуміти, які саме дані потрібно виправити або заповнити для успішного входу в систему.

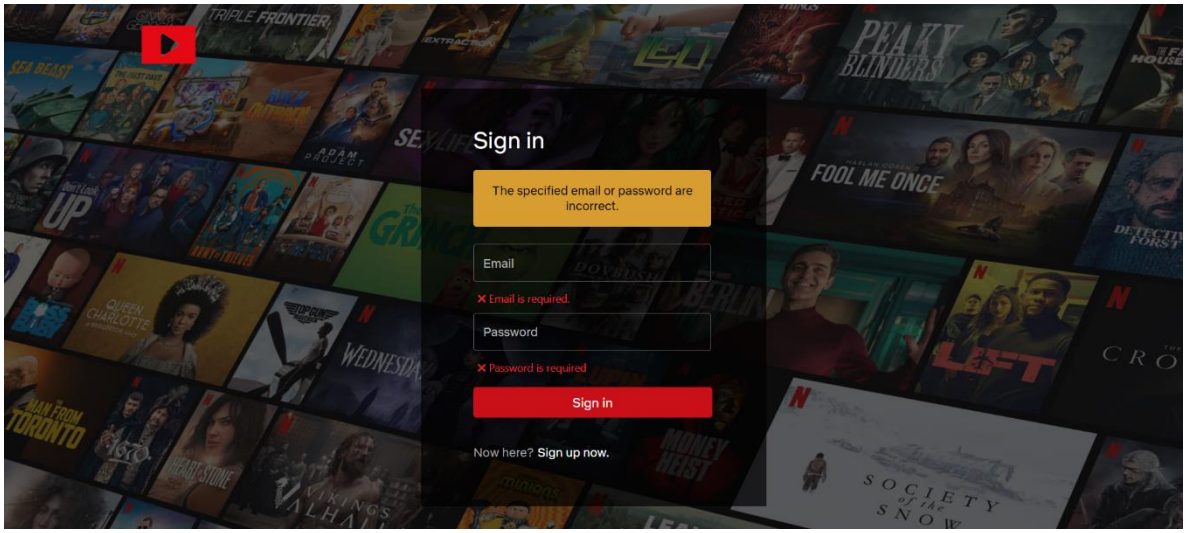

Рисунок 3.23 – Модальне вікно для створення епізоду

Фільми та серіали, які були додані на сервіс нещодавно, будуть позначені спеціальним блоком, який повідомляє користувача про новинку (рис. 3.24). Це відображення буде присутнє на всіх сторінках і елементах, де розміщено цей фільм чи серіал. Блок повідомлення про новинку з’являтиметься автоматично і з часом зникатиме, оскільки фільм чи серіал перестане бути новим на платформі. Такий підхід забезпечує користувачів актуальною інформацією про нові додані матеріали, сприяючи кращій орієнтації серед контенту і підвищуючи зручність користування сервісом. Відображення новинок допомагає користувачам швидко знаходити найновіші фільми та серіали, а автоматичне зникнення повідомлень зберігає інтерфейс чистим і ненавантаженим зайвою інформацією, що покращує загальний досвід користувача.

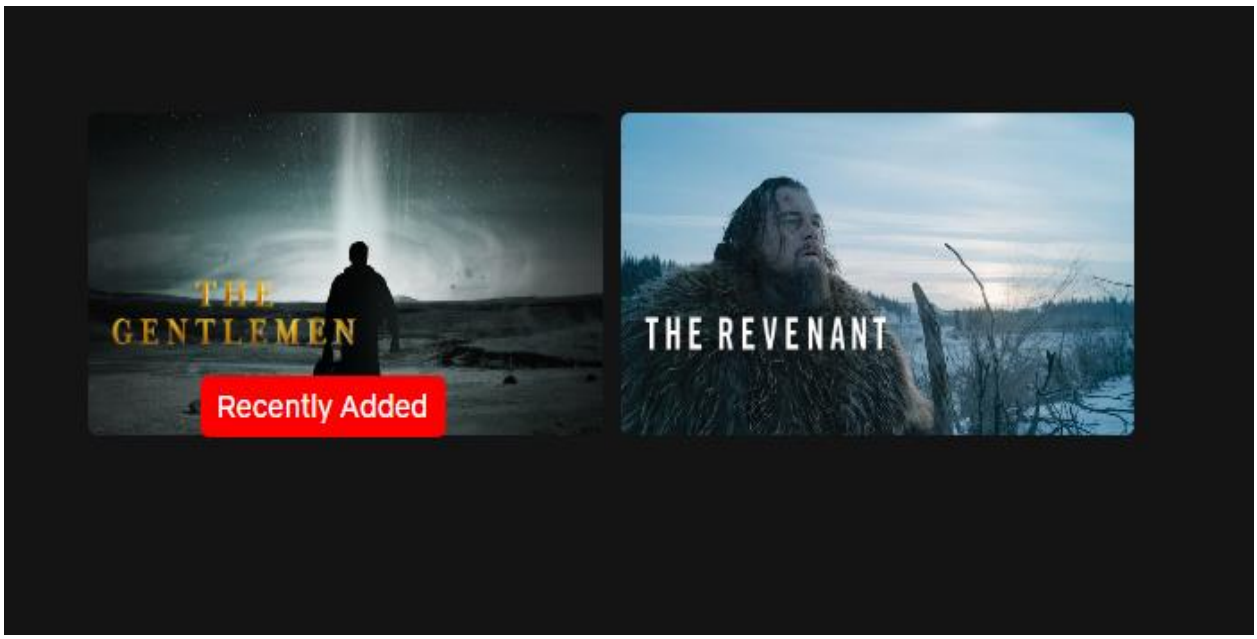

Рисунок 3.24 – Блок повідомлення про те, що фільм чи серіал було нещодавно додано на платформу

При створенні нового фільму чи серіалу з відповідним жанром, актором або режисером, усім користувачам, які вподобали інші фільми з подібним контентом раніше, будуть надіслані повідомлення про вихід нового фільму (рис. 3.25). Система автоматично визначить користувачів, які виявили інтерес до схожого контенту, і надішле їм сповіщення.

Процес створення фільму чи серіалу включає вибір жанру, акторів та режисерів. Коли адміністратор додає новий фільм або серіал, система аналізує ці параметри і порівнює їх з вподобаннями користувачів, що зберігаються в базі даних.

У випадку з серіалом, подібне повідомлення буде надіслане при додаванні нового сезону або епізоду. Це гарантує, що користувачі завжди будуть в курсі нових випусків своїх улюблених серіалів. Наприклад, якщо додано новий сезон популярного серіалу, всі користувачі, які вподобали попередні сезони, отримають сповіщення про його вихід. Аналогічно, при додаванні нового епізоду, користувачі, які слідкують за серіалом, будуть проінформовані.

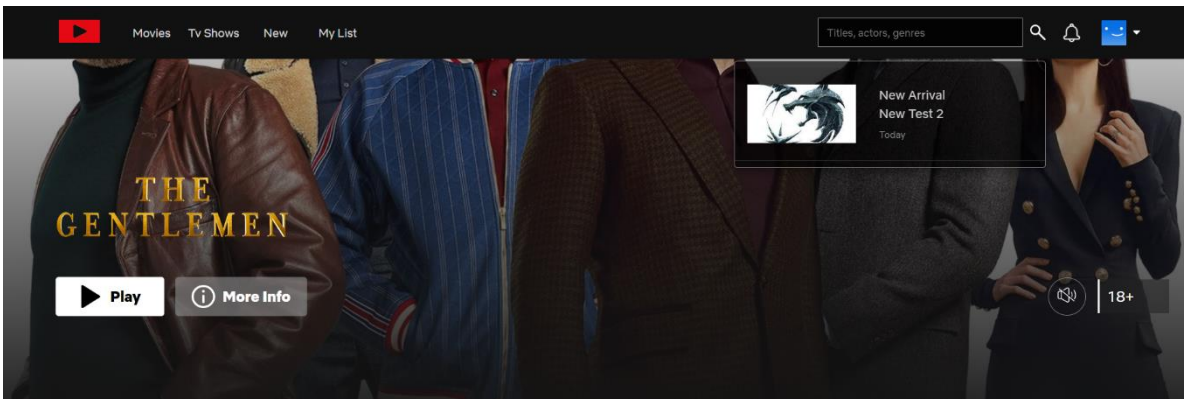

Рисунок 3.25 – Повідомлення про вихід нового фільму який може бути цікавим для користувача

Користувач отримує повідомлення шляхом того, що при створенні фільму чи серіалу надсилається відповідна команда яку перехоплює відповідний клас. Цей клас починає свою роботу із запиту інформації про конкретний фільм за ідентифікатором та списку всіх фільмів із репозиторію. Далі він перевіряє, чи існує фільм із зазначеним ідентифікатором. Якщо фільм не знайдено, метод повертає помилку з повідомленням про те, що фільм не знайдено. У разі успішного знаходження фільму з нього витягуються списки жанрів, акторів та творців, пов’язаних із цим фільмом. Потім метод фільтрує список усіх фільмів, щоб знайти ті, які мають хоча б один збігаючий жанр, актора або творця із заданим фільмом. Цей процес дозволяє визначити фільми, які можуть бути схожими на вихідний фільм. Після цього метод витягує користувачів, яким сподобалися знайдені схожі фільми, зі списку всіх фільмів. Для кожного знайденого користувача створюється нове сповіщення. Такі сповіщення дозволяють користувачам отримувати інформацію про нові релізи, які можуть бути їм цікавими на основі їхніх попередніх вподобань. Це значно підвищує залученість користувачів і сприяє збільшенню часу, проведеного на платформі. Такий підхід забезпечує персоналізований користувацький досвід, підвищуючи задоволеність від використання платформи.

ВИСНОВКИ

У рамках кваліфікаційної роботи був спроєктований та розроблений потоковий сервіс для перегляду фільмів та серіалів.

Проведено аналіз сучасного стану потокових сервісів, що включав огляд найбільш популярних платформ та їх функціональних можливостей. Особливу увагу було приділено методам проєктування та реалізації систем рекомендацій контенту, що є важливим аспектом для підвищення залученості користувачів. Також було досліджено сучасний стан платформи .NET, методи розробки онлайн-застосунків на її основі та вибрано відповідне хмарне сховище для зберігання контенту. Огляд фреймворків та бібліотек для реалізації клієнтської частини дозволив вибрати оптимальні інструменти для розробки, що забезпечують високу продуктивність та зручність користувачів. На етапі проєктування сервісу було обрано архітектуру backend та проведено розбиття його на модулі для забезпечення масштабованості та легкості в обслуговуванні.

Проєктування бази даних та розробка системи безпечного зберігання контенту для потокової передачі дозволили забезпечити надійність і захищеність даних. Окрему увагу приділено проєктуванню системи рекомендацій контенту та спілкуванню застосунку з клієнтом та між мікросервісами. Це забезпечило високий рівень персоналізації контенту та ефективну взаємодію між компонентами системи. Створення авторизації та аутентифікації за допомогою JWT дозволило гарантувати безпеку користувацьких даних та зручність доступу до сервісу. На етапі розробки було вибрано відповідне середовище для розробки, що забезпечило швидкість та ефективність роботи. Реалізація серверної та клієнтської частини включала всі необхідні функціональні можливості для забезпечення якісного перегляду контенту. Проведено тестування сервісу, яке підтвердило його працездатність та відповідність встановленим вимогам. Загалом, розроблений потоковий сервіс для перегляду фільмів та серіалів відповідає сучасним стандартам

якості, забезпечує високий рівень безпеки та зручності користувачів. Проведені дослідження та використані технології дозволили створити надійний і функціональний продукт, який має потенціал для подальшого розвитку та вдосконалення.

Результати роботи апробовано у вигляді тез доповідей під час 28-го Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ» [32].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Brown, T. (2021). Streaming platforms and the future of entertainment. Media Innovations Journal, 10(2), 34-56.

2. Amazon Prime Video – video streaming service by Amazon. URL: https://www.primevideo.com/ (дата звернення 12.04.2024).

3. Hulu – video streaming service. URL: https://www.hulu.com/ (дата звернення 12.04.2024).

4. Disney+ – video streaming service by The Walt Disney Company. URL: https://www.disneyplus.com/ (дата звернення 12.04.2024).

5. HBO Max – video streaming service by WarnerMedia. URL: https://www.hbomax.com/ (дата звернення 12.04.2024).

6. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент.

7. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

8. Tvoroshenko, I., Pomazan, V., Gorokhovatskyi, V., & Kobylin, O. (2023). Application of video data classification models using convolutional neural networks.

9. Tvoroshenko, I., Gorokhovatskyi, V., Kobylin, O., & Tvoroshenko, A. (2023). Application of deep learning methods for recognizing and classifying culinary dishes in images.

10. Smith, J. R. (2022). Middleware components in ASP.NET Core: A comprehensive guide. Journal of Software Engineering Practices, 14(3), 89-112.

11. Doe, A. L. (2023). The role of middleware in enhancing ASP.NET Core applications. International Journal of Web Development, 21(4), 56-78.

12. AWS S3 – cloud storage service by Amazon Web Services. URL: https://aws.amazon.com/s3/ (дата звернення 16.04.2024).

13. Firebase Storage – cloud storage service by Google. URL: https://firebase.google.com/products/storage/ (дата звернення 16.04.2024).

14. Williams, P. D. (2023). The evolution of software architecture: From monoliths to clean architecture. International Journal of Advanced Software Development, 22(4), 123-150.

15. Johnson, M. K. (2021). Clean architecture: Implementing robust software design. Journal of Software Architecture and Design, 19(2), 45-68.

16. Davis, L. M. (2022). Domain-driven design and its application in modern software architecture. Journal of Applied Software Engineering, 15(1), 34-58.

17. Brown, K. E. (2023). Implementing layered architecture with ASP.NET Core. Software Development Insights, 20(2), 77-99.

18. Cherednichenko, O., Vovk, M., Yanholenko, O., & Yakovleva, O. (2020). Towards the technology of employers' requirements collection development. In Integrated Computer Technologies in Mechanical Engineering: Synergetic Engineering (pp. 228-239). Cham: Springer International Publishing.

19. Kuzomin, O., & Lyashenko, V. (2022). Key Elements of a Specialized Complex for Solving Modeling Problems.

20. Kuzomin, O., & Lyashenko, V. (2022). Situational-Linguistic Modeling in Diagnostic Decision-Making Systems.

21. Mashtalir, S. V., Stolbovyi, M. I., & Yakovlev, S. V. (2019). Clustering video sequences by the method of harmonic k-means. Cybernetics and Systems Analysis, 55, 200-206.

22. Gorokhovatskyi, V., Gorokhovatskyi, O., Yevgenyi, P., & Olena, P. (2018). Quantization of the Space of Structural Image Features as a Way to.

23. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks.

24. Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. *International scientific symposium «INTELLIGENT SOLUTIONS-S». Computational intelligence (results, problems and perspectives). Decision making theory: proceedings of the international symposium, September 28, 2023, Kyiv-Uzhorod, Ukraine*, pp. 25-27.

25. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for fast metric data search in structural methods for image classification. *IEEE Access*, *10*, 124738-124746.

26. Gorokhovatskyi, V., Tvoroshenko, I., & Chmutov, Y. (2022). Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень. *Advanced Information Systems*, *6*(3), 5-12.

27. Gadetska, S., Gorokhovatskyi, V., Stiahlyk, N., & Vlasenko, N. (2022). Aggregate parametric representation of image structural description in statistical classification methods.

28. Tvoroshenko, I., & Gorokhovatskyi, V. (2022). The Application of Hybrid Intelligence Systems for Dynamic Data Analysis.

29. Dyadun, S., Yakovlev, S., & Kobylin, O. (2022). Mathematical Modeling of Steady Flow Distribution in Water Supply Networks with Pumping Stations and Regulating Capacitances. In *ProfIT AI* (pp. 78-83).

30. Kobylin, O., & Lyashenko, V. (2020). Time series clustering based on the k-means algorithm.

31. Yakovleva, O., Kovtunenko, A., Liubchenko, V., Honcharenko, V., & Kobylin, O. (2023). Face Detection for Video Surveillance-based Security System. In *COLINS (3)* (pp. 69-86).

32. Васильєв Р.Р., Кіношенко Д.К. (2024). МАТЕМАТИЧНА БАЗА ДЛЯ СТВОРЕННЯ ГІБРИДНОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РЕКОМЕНДАЦІЙ З ГЛИБОКИМ НАВЧАННЯМ. 28-ий міжнародний молодіжний форум «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ».