Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти − магістр
за освітньо-професійною програмою
«Комп'ютерні науки»

зі спеціальності
122 – Комп'ютерні науки

тема роботи:
«Веб-сервіс для пошуку та збереження у бібліотеку колекції кінострічок на основі глядацьких вподобань**»**

Виконав студент гр. КН-23-м _____ Приймак В.Р.

Керівник _____ Маринич І.А.

Нормоконтроль _____ Маринич І.А.

Завідувач кафедри _____ Рубан С.А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедри: *д.т.н. Рубан С.А.*

_____

« *5* » *липня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-23-м *Приймаку Владиславу Романовичу*

1. Тема кваліфікаційної роботи: *«Веб-сервіс для пошуку та збереження у бібліотеку колекції кінострічок на основі глядацьких вподобань»*

затверджено наказом по університету № *594с* від *04.07.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 77с., додатки, презентація у Microsoft PowerPoint (21 слайд) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Маринич І. А.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*05.09.24*
2	*Розділ 1*	*29.09.24*
3	*Розділ 2*	*15.10.24*
4	*Розділ 3*	*15.11.24*
4	*Висновки*	*20.11.24*
5	*Оформлення кваліфікаційної роботи*	*24.11.24*
6	*Підготовка презентації та графічного матеріалу*	*28.11.24*
7	*Підготовка доповіді до захисту*	*10.12.24*

6. Дата видачі завдання: *04.07.2024р.*

Керівник _____/Маринич І.А./

7. Запевнення: *Я, Приймак Владислав Романович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Приймак В.Р./

АНОТАЦІЯ

Приймак В.Р. «Веб-сервіс для пошуку та збереження у бібліотеку колекції кінострічок на основі глядацьких вподобань».

Кваліфікаційна робота на здобуття ступеню вищої освіти магістр за освітньо-професійною програмою «Автоматизація та комп'ютерно-інтегровані технології» зі спеціальності 122 – Комп'ютерні науки. – Криворізький національний університет, Кривий Ріг, 2024.

Об'єктом дослідження є аналіз популярності веб-сервісів для потокового відтворення відео та розробка програмного забезпечення для збереження персоналізованих колекцій кінострічок.

У першому розділі проведено ґрунтовний теоретичний аналіз популярності сучасних веб-сервісів, зокрема стрімінгової платформи Netflix. Особливу увагу приділено дослідженню моделей формування глядацьких уподобань, їхніх ключових чинників та впливу на вибір контенту користувачами. Розглянуто, як алгоритми рекомендацій і маркетингові стратегії впливають на зростання аудиторії та закріплення лідерських позицій у галузі розваг.

У другому розділі представлено практичну частину дослідження: проведено аналіз популярності Netflix за допомогою мови програмування Python, побудовано графіки, що ілюструють динаміку переглядів, рейтингів та тенденцій.

У третьому розділі детально описано процес створення веб-додатку, розробленого із застосуванням сучасного технологічного стеку. Представлено архітектурну структуру додатку, його ключові функціональні можливості, а також покроково розглянуто інтеграцію між клієнтською та серверною частинами. Особливу увагу приділено оптимізації роботи з даними та реалізації ефективної взаємодії компонентів.

Ключові слова: NETFLIX, FULL-STACK WEB APPLICATION, ВЕБ-САЙТ, REACT, TYPESCRIPT, EXPRESSJS, MONGODB, ZUSTAND.

ANNOTATION

Pryimak V.R. «A web service for searching and saving a collection of films to the library based on viewer preferences».

Graduation master's work for obtaining an education degree «Master» for the educational and professional program «Automation and computer-integrated technologies» in specialty 122 – «Computer science». – Kryvyi Rih National University, Kryvyi Rih, 2024.

The purpose of the work is to analyze the popularity of web services for streaming video playback and develop software for storing personalized collections of films.

The first section provides a thorough theoretical analysis of the popularity of modern web services, in particular the Netflix streaming platform. Particular attention is paid to the study of models of audience preferences, their key factors, and the impact on the choice of content by users. The author examines how recommendation algorithms and marketing strategies affect audience growth and consolidation of leadership positions in the entertainment industry.

The second section presents the practical part of the study: an analysis of the popularity of Netflix using the Python programming language is carried out, graphs are built that illustrate the dynamics of views, ratings and trends.

The third section describes in detail the process of creating a web application developed using a modern technology stack that includes React, TypeScript, Zustand, ExpressJS, and MongoDB. The architectural structure of the application, its key functionalities are presented, and the integration between the client and server parts is discussed step by step. Particular attention is paid to optimizing data processing, ensuring user-friendliness, and implementing effective component interaction.

Keywords: Netflix, Full-stack web application, web-site, React, Typescript, ExpressJS, MongoDB, Zustand.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ..... 7
ВСТУП 8
РОЗДІЛ 1 ОСНОВНІ КОНЦЕПЦІЇ ТА МЕТОДИ ДАТА-АНАЛІЗУ ДЛЯ ПРОВЕДЕННЯ ДОСЛІДЖЕНЬ…….....10
1.1 Інтелектуальний аналіз даних. Методи та алгоритми 10
1.2 Стадії інтелектуального аналізу даних 13
1.3 Фактори впливу на популярність фільмів на стрімінгових платформах……...15
1.4 Аналіз росту стрімінгових платформ 18
Висновок до першого розділу..... 21
РОЗДІЛ 2 ДАТА-АНАЛІЗ НАЙПОПУЛЯРНІШОЇ СТРІМІНГОВОЇ ПЛАТФОРМИ NETFLIX……..23
2.1 Аналіз даних стрімінгової платформи Netflix за допомогою Python.. 23
2.2 Огляд технологій та їх можливостей..... 33
Висновок до другого розділу 37
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПОРТАЛУ……..39
3.1 Розробка backend частини 39
3.2 Розробка frontend частини 57
Висновок до третього розділу..... 73
ВИСНОВКИ..... 74
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface. Набір методів, які надають розробнику засоби для швидкої розробки програмного забезпечення.

IDE – Integrated Development Environment. Інтегроване середовище розробки, яке використовується розробниками для написання програмного коду.

SPA – Single Page Application. Модель розробки сайтів в якому є тільки одна сторінка і контент динамічно оновлюється на ній при взаємодії з користувачем.

DOM – Document Object Model.

Компонент – структурна одиниця бібліотеки React. Весь додаток складається з компонентів, де у кожного є своя логіка.

ІС – інформаційна система.

БД – база даних проекту, місце збереження інформації БД.

Фронтенд – візуальна частина з якою взаємодіє користувач.

Бекенд – серверна частина, доступ до якої є тільки у розробників, а користувач може тільки відправляти запити з фронтенду та отримувати відповідь.

ВСТУП

У сучасному світі інформаційні технології стрімко розвиваються, формуючи нові підходи до задоволення повсякденних потреб користувачів. Однією з найбільш динамічних галузей є ринок стрімінгових сервісів, що надають можливість доступу до широкого спектра контенту, зокрема кінофільмів, серіалів, документальних програм та інших форматів відео. Висока конкуренція на цьому ринку змушує провайдерів впроваджувати інноваційні методи для залучення користувачів, серед яких персоналізація контенту відіграє ключову роль.

Стрімінгові сервіси, такі як Netflix, Amazon Prime, Disney+ та інші, не лише забезпечують доступ до медіа, а й використовують алгоритми штучного інтелекту та машинного навчання для формування персоналізованих рекомендацій. Однак навіть за умов досконалих алгоритмів користувачі нерідко стикаються з проблемами організації власного контенту, зокрема збереження улюблених кінострічок у доступній формі чи управління своєю бібліотекою фільмів.

Одним із важливих напрямів сучасних досліджень є вивчення закономірностей популярності відеоконтенту. Аналіз тенденцій у переглядах дозволяє не лише краще зрозуміти вподобання глядачів, але й створювати нові ефективні моделі для вдосконалення рекомендаційних систем. При цьому постає необхідність поєднання аналітичних методів із розробкою програмних рішень, які забезпечать зручний доступ до відповідного контенту.

Актуальність цієї роботи обумовлена зростанням попиту на індивідуалізований контент та потребою в інструментах для систематизації даних. Веб-сервіси, здатні надавати такі можливості, мають великий потенціал для широкого застосування, як у комерційній, так і в освітній чи культурній сферах.

Метою цієї роботи є створення повноцінного веб-сервісу для пошуку, аналізу та організації колекцій кінострічок на основі глядацьких уподобань.

Для досягнення цієї мети необхідно вирішити кілька ключових завдань. По-перше, дослідити популярність стрімінгових платформ, зокрема Netflix, шляхом вивчення їхніх підходів до формування рекомендацій і динаміки розвитку. По-друге, провести аналітичну обробку даних із використанням інструментів програмування, що дозволить виявити закономірності у вподобаннях користувачів. І, по-третє, розробити сучасний веб-додаток, який поєднує функціональність динамічного пошуку, збереження контенту та персоналізації інтерфейсу.

Методи дослідження, застосовані в роботі, включають аналіз літературних джерел, статистичну обробку даних, використання алгоритмів машинного навчання для моделювання популярності контенту, а також реалізацію програмних рішень за допомогою сучасних технологій, таких як React, TypeScript, Express та MongoDB.

Результати цього дослідження мають практичну цінність, оскільки дозволяють створити універсальний інструмент для роботи з відеоконтентом. Такий підхід сприятиме підвищенню задоволеності користувачів, а також відкриває нові можливості для дослідників у галузі аналізу даних та розробників у сфері створення веб-додатків.

Таким чином, проведене дослідження поєднує теоретичні та практичні аспекти, забезпечуючи комплексний підхід до вирішення поставлених завдань. Робота спрямована на вдосконалення існуючих методів аналізу популярності контенту та реалізацію інноваційного інструменту, що підвищує комфорт і зручність взаємодії користувача з веб-сервісами.

РОЗДІЛ 1
ОСНОВНІ КОНЦЕПЦІЇ ТА МЕТОДИ ДАТА-АНАЛІЗУ ДЛЯ ПРОВЕДЕННЯ ДОСЛІДЖЕНЬ

1.1 Інтелектуальний аналіз даних. Методи та алгоритми

Розвиток способів запису й зберігання інформації призвів до експоненційного збільшення обсягів зібраної та аналізованої інформації. Ці об’єми даних настільки великі, що самостійний аналіз людиною стає неможливим, хоча потреба у такому аналізі є очевидною. У цих «сирих даних» містяться знання, які можна використати для прийняття рішень. Традиційна математична статистика, хоч і використовувалась як основний інструмент аналізу даних, протягом тривалого часу, проте не вирішувала всі проблеми, які виникали. Це призвело до необхідності розвитку нових сучасних підходів та методологій для обробки та аналізу даних. Інтелектуальний аналіз даних (ІАД) виступив однією з таких нових методологій. Серед причин популярності ІАД можна виділити наступні:

- стрімке накопичення даних (рахунок вже йде на екзабайти);
- загальна комп’ютеризація бізнес-процесів;
- проникнення Інтернету в усі сфери діяльності;
- прогрес в області інформаційних технологій: вдосконалення СУБД і сховищ даних;
- прогрес в області виробничих технологій: стрімке зростання продуктивності комп’ютерів, обсягів накопичувачів, впровадження GRID систем. [1]

Алгоритми, що використовуються у Інтелектуальному Аналізі Даних (ІАД), потребують значної обчислювальної потужності. Раніше це становило складність для широкого практичного використання ІАД. Проте зростання продуктивності сучасних процесорів усунуло цю перешкоду. Тепер у прийнятний період часу можна провести якісний аналіз сотень тисяч та навіть

мільйонів записів. ІАД є міждисциплінарною областю, що виникла та розвивалася на основі таких наук, як прикладна статистика, розпізнавання образів, штучний інтелект, теорія баз даних тощо.

До набору методів та алгоритмів Інтелектуального Аналізу Даних (ІАД) відносяться різноманітні інструменти: штучні нейронні мережі, дерева рішень, символьні правила, методи найближчих та к-найближчих сусідів, метод опорних векторів, байєсові мережі, лінійна регресія, кореляційно-регресійний аналіз. Також в переліку зустрічаються ієрархічні та неієрархічні методи кластеризації, зокрема алгоритми к-середніх і к-медіан, методи пошуку асоціативних правил, такі як алгоритм Apriori, метод обмеженого перебору, еволюційне програмування, генетичні алгоритми та різноманітні методи візуалізації даних.[2] Варто відзначити, що більшість з цих методів ІАД розроблені у рамках теорії штучного інтелекту. Згадати також, що не існує єдиної точки зору щодо включення конкретних завдань до ІАД. Більшість авторитетних джерел відносять до ІАД такі задачі, як класифікація, кластеризація, прогнозування, асоціація, візуалізація, аналіз виявлення відхилень, оцінювання, аналіз зв’язків та підведення підсумків. Розглянемо деякі з них.

Класифікація - це одне з найпоширеніших та фундаментальних завдань Інтелектуального Аналізу Даних (ІАД). У процесі вирішення цієї задачі визначаються специфічні ознаки, що описують різні групи об'єктів у досліджуваному наборі даних - класи. Ці ознаки дозволяють віднести новий об'єкт до певного класу. Для вирішення задачі класифікації використовуються різні методи, такі як метод найближчого сусіда (Nearest Neighbor), к-найближчого сусіда (k-Nearest Neighbor), байєсові мережі (Bayesian Networks), дерева рішень та нейронні мережі (neural networks). [3]

Кластеризація, також відома як процес створення кластерів, представляє собою логічний наступ класифікації. Це більш складна задача, оскільки на відміну від класифікації, класи об'єктів не передбачені заздалегідь. У результаті кластеризації об'єкти групуються за спільними ознаками чи

властивостями. Один із прикладів методу кластеризації - це спеціальний тип нейронних мереж, які відомі як карті Кохонена. Ці мережі здатні самоорганізовуватись без заздалегідь заданої структури чи вчителя. [4]

Асоціація (Associations). Асоціація, як завдання Інтелектуального Аналізу Даних (ІАД), спрямована на виявлення закономірностей та взаємозв'язків між подіями у наборі даних. Особливість асоціації полягає у тому, що пошук зв’язків проводиться не на основі властивостей окремих об’єктів, а між різними подіями, що відбуваються паралельно. Найвідомішим алгоритмом для розв'язання задачі пошуку асоціативних правил є алгоритм Apriori.

Послідовна асоціація (Sequence), дозволяє виявити часові закономірності між різними транзакціями. Підхід до послідовності схожий на асоціацію, проте його спрямованість полягає у встановленні відносин не між одночасно відбуваючимися подіями, а між подіями, що сталися у певний часовий інтервал. Це завдання також відоме як виявлення послідовних шаблонів (sequential pattern) в рамках аналізу даних. Правило послідовності формулюється так: після події X через певний час настає подія Y. [5]

Прогнозування - це процес оцінки пропущених або майбутніх значень цільових числових показників за допомогою аналізу існуючих даних. Для цього використовуються різноманітні методи, такі як математична статистика, нейронні мережі та інші, які дозволяють враховувати особливості набору даних. [6]

Графічна репрезентація, також відома як візуалізація або графовий аналіз, використовується для створення образу аналізованих даних у вигляді графіків або діаграм. Ці графічні методи відображають можливі закономірності або взаємозв'язки в наборі даних. Наприклад, це може бути представлення даних у двовимірному або тривимірному вигляді. [7]

1.2 Стадії інтелектуального аналізу даних

ІАД може складатися з двох або трьох стадій:

- Стадія 1. Виявлення закономірностей (вільний пошук).
- Стадія 2. Використовування виявлених закономірностей для прогнозу невідомих значень (прогностичне моделювання).

Додатково до цих етапів іноді додають етап оцінки (валідації), який слідує за етапом вільного пошуку [8]. Мета валідації полягає в перевірці правильності виявлених закономірностей. Проте часто вважається, що валідація є складовою частиною першого етапу, оскільки в багатьох методах, зокрема при використанні нейронних мереж і дерев рішень, передбачено розділення загального набору даних на тренувальний та валідаційний, що дає можливість перевірити правильність отриманих результатів.

- Стадія 3. Аналіз виключень – стадія, призначена для виявлення і пояснення аномалій, знайдених у закономірностях.

На етапі вільного пошуку (Discovery) проводиться аналіз набору даних з метою виявлення прихованих закономірностей, не попередньо обмежуючи себе певними гіпотезами щодо цих закономірностей. Закономірність (law) визначає суттєвий та сталий взаємозв'язок, який постійно повторюється і визначає етапи та форми розвитку різних явищ або процесів. [9]

На даній стадії система Інтелектуального Аналізу Даних (ІАД) встановлює шаблони, які, наприклад у системах OLAP, потребують аналітика розглядати та створювати велику кількість запитів для їх отримання. Однак у вільному пошуку саме система виявляє ці шаблони, звільняючи аналітика від цієї рутинної роботи. Цей підхід особливо корисний у великих базах даних, де складно виявити закономірності шляхом створення запитів через потребу випробувати безліч різноманітних варіантів.

Вільний пошук включає в себе наступні дії:

- виявлення закономірностей умовної логіки (conditional logic);

- виявлення закономірностей асоціативної логіки (associations and affinities);

- виявлення трендів і коливань (trends and variations). [10]

Описані дії у межах стадії вільного пошуку виконуються за допомогою:

- індукції правил умовної логіки (задачі класифікації і кластеризації, опис в компактній формі близьких або подібних груп об’єктів);

- індукції правил асоціативної логіки (задачі асоціації і послідовності і витягування за їх допомогою інформації)

- визначення трендів і коливань (початковий етап задачі прогнозування). [11]

На етапі вільного пошуку також необхідно проводити перевірку достовірності виявлених закономірностей. Це означає, що потрібно перевіряти ці закономірності на частинах даних, які не використовувалися для формування цих закономірностей.

Прогностичне моделювання (Predictive modeling). Друга стадія ІАД – прогностичне моделювання – використовує результати роботи першої стадії. Тут знайдені закономірності використовуються безпосередньо для прогнозування. Прогностичне моделювання охоплює такі дії:

- прогнозування невідомих значень (outcome prediction);

- прогнозування розвитку процесів (forecasting);

У процесі прогностичного моделювання розв’язуються задачі кваліфікації і прогнозування.

Під час вирішення завдання класифікації, результати першої фази (створення правил на основі індукції) використовуються для віднесення нового об'єкта до відомого, передбаченого класу з певною впевненістю, враховуючи відомі параметри.

Під час вирішення завдання прогнозування, результати першого етапу (визначення тенденцій або коливань) використовуються для передбачення невідомих (пропущених або майбутніх) значень цільової змінної (змінних).[12]

Закономірності, виявлені на цьому етапі, формуються від часткового до загального. Цей процес дозволяє отримати загальні знання про певний клас об'єктів на основі вивчення окремих представників цього класу.

Прогностичне моделювання, навпаки, дедуктивне. Закономірності, отримані на цій стадії, формуються від загального до часткового. Тут ми одержуємо нове знання про деякий об’єкт або ж групу об’єктів на підставі:

- знання класу, до якого належать досліджувані об’єкти;
- знання загального правила, що діє в межах цього класу об’єктів. [13]

На третьому етапі Інтелектуального Аналізу Даних (ІАД) проводиться аналіз винятків або аномалій, що були виявлені у встановлених закономірностях. Дія, яка виконується на цьому етапі, полягає у виявленні відхилень (deviation detection). Для виявлення цих відхилень потрібно визначити норму, яка розраховується на етапі вільного пошуку. Стадія аналізу виключень може бути використана як процес очищення даних. [14]

1.3 Фактори впливу на популярність фільмів на стрімінгових платформах

Стрімінгові медіа набувають все більшої популярності серед споживачів на сучасному медіа-ринку. Цей попит пояснюється зручністю та доступністю стрімінгових платформ для звичайних користувачів. За наявності спеціальних пристроїв (таких як планшети, комп'ютери, смартфони і т.д.), які підтримують стрімінгове відтворення, і підключення до Інтернету, користувач має можливість отримувати доступ до відео- та аудіоконтенту, який можна переглядати або слухати онлайн. Це означає, що немає потреби повністю завантажувати відео або аудіофайл перед його відтворенням. Згідно з даними звіту Global Digital на 2018 рік, кількість користувачів Інтернету перевищила позначку у 4 мільярди, що свідчить про надзвичайну важливість стрімінгового мовлення, що продовжує активно зростати.

Netflix визнаний одним з провідних сервісів відеострімінгу, який забезпечує можливість онлайн-перегляду фільмів і серіалів. На сьогоднішній день компанія активно розвивається не лише як платформа для потокового відтворення контенту, а й як продюсер оригінальних телесеріалів і шоу. Інформація вказує на те, що у світі налічується 117,58 мільйонів підписників цієї платформи. Саме на його прикладі проведемо аналіз популярності всіх платформ для стрімінгу.

Netflix був створений у 1997 році і на початку своєї діяльності зосереджувався на прокаті фільмів на DVD. Проте з 2007 року компанія також запустила потоковий сервіс мовлення. На сьогоднішній день Netflix є найбільшим постачальником фільмів та серіалів завдяки угодам з популярними кіностудіями і виробництву власного оригінального контенту з 2011 року. Із понад 190 країн, учасники отримують швидкий доступ до контенту, що робить Netflix одним з найдоступніших сервісів. Однак вміст бібліотеки відрізняється в залежності від регіону через різні умови придбання ліцензій на трансляцію. Це може бути зумовлено вартістю ліцензій у різних країнах або ексклюзивними угодами з іншими стрімінговими сервісами. Наприклад, різниця у законодавстві про авторське право призводить до різниці у доступі до контенту. Аналіз бібліотек Netflix у різних країнах, проведений виданням Finder, показує різний обсяг пропозиції контенту. Зараз основну аудиторію Netflix складають мешканці США і Великобританії, які мають вищу кількість передплатників порівняно з іншими країнами. Хоча вартість передплати практично однакова для всіх країн, доступний контент може суттєво відрізнятися в залежності від регіону.

На сьогоднішній день більшу частину аудиторії Netflix складають мешканці США і Великобританії. Вартість передплати на місяць становить 10,99 доларів США для американців і 7,99 фунтів стерлінгів для британців – стандартний пакет, який надає одночасний доступ до сервісу з двох пристроїв, і 13,99 доларів і 9,99 фунтів стерлінгів відповідно – преміальний пакет, який включає в себе одночасний доступ з чотирьох пристроїв і якість ULTRA HD.

Рисунок 1.1 – Візуалізація сервісу Netflix

У грудні 2017 року IHS Markit опублікувала звіт, де надана статистика використання сервісу Netflix у країнах Центральної та Східної Європи. Україна посідає передостаннє місце із показником 2 % від загальної кількості домогосподарств, які мають доступ до широкосмугового Інтернету та передплачують за Netflix. За даними Liga.net, ця цифра еквівалентна понад 100 тис. абонентам. Росія займає останнє місце з відсотковим показником, що не перевищує навіть 1 %. Такі відмінності можна пояснити різними підходами країн до покарання за порушення авторських прав. Наприклад, у Великобританії передбачена кримінальна відповідальність, включаючи позбавлення волі, за завантаження та використання матеріалів, захищених авторським правом, без відповідного дозволу власника. Закон "Про цифрову економіку", що регулює відносини у сфері цифрового медіамовлення, визначає відповідальність за будь-яке порушення при завантаженні контенту з онлайн-мережі. Подібна ситуація спостерігається й у США. Ймовірно, сервіс Netflix користується такою популярністю серед американських і британських користувачів через легальний доступ до бібліотеки фільмів, серіалів і програм. Щодо України та Росії, ці країни відомі своєю популярністю торрент-сайтів та

скачуванням піратських версій серед користувачів. За словами глави Української антипіратської асоціації Володимира Ілінга, законодавство про авторське право забороняє розміщення будь-яких матеріалів у мережі без згоди власника прав, однак провайдери сьогодні не несуть відповідальності за зміст розміщених на їхній технічній базі матеріалів. Тому отримання інформації про користувачів, які завантажують або розповсюджують фільми, є неможливим. Ця форма отримання медійного контенту в цих країнах залишається безкарною, що призводить до ситуації, коли користувачі не мають потреби платити за контент, який надає Netflix, і можуть отримати його абсолютно безкоштовно.

Ще одним фактором, який значно обмежує можливості потенційних споживачів контенту Netflix у країнах, які знаходяться на етапі розвитку, є відсутність доступу до високошвидкісного Інтернету. [15]

1.4 Аналіз росту стрімінгових платформ

Феномен потокового мультимедіа є новим і стрімко розвивається, залишаючись недостатньо вивченим на даний момент. Спроби відображення мультимедійної інформації на комп'ютерах розпочалися у середині XX століття, але через високу вартість комп'ютерів і їхню обмежену функціональність прогрес у цій сфері був обмеженим. Завершення 90-х років XX століття відзначилося стрімким розвитком Інтернету, який мав достатню пропускну здатність мереж, використовував стандартизовані протоколи й формати, що зробило потокове мультимедіа доступним для широкого кола користувачів. Проте цей вид мультимедійного вмісту продовжує удосконалюватися через розвиток технічних засобів і мультимедійних платформ. Один з таких потокових сервісів - Netflix, який запровадив стрімінгову технологію у 2007 році. Поступово цей сервіс поширив свою діяльність на міжнародному рівні, і зараз його можна отримати в більш ніж 190 країнах. Netflix пропонує не лише придбані фільми та серіали, а й власний

контент, такий як Netflix Original. Робота з контентом та індивідуальний підхід до кожного абонента стали ключовими компонентами успіху компанії. Аналітики Netflix використовують контентні алгоритми, які враховують уподобання користувачів на платформі, їх рейтинги та часті запити, а також популярність фільмів і серіалів на інших сайтах. Персоналізовані алгоритми навіть враховують те, коли користувач зупиняється або повертається до перегляду, щоб створити актуальний список рекомендованих фільмів. Вони аналізують жанри, акторів, теми, які цікавлять глядача тощо. За даними Netflix, 75 % переглядів відбувається на основі рекомендацій. Ще однією особливістю компанії є випуск всього сезону одразу, що є важливою перевагою порівняно з традиційним телебаченням. Це дає можливість глядачам дивитися серіали без рекламних перерв і тижневих затримок перед новими епізодами, а також бути впевненими, що серіал не буде скасовано посеред сезону.

Таблиця 1.1 – Ріст активних користувачів Netflix

	2012 р.	2013 р.	2014 р.	2015 р.	2016 р.	2017 р.	2018 р.
Користувачі США, млн	27,15	33,42	39,11	44,74	49,93	54,75	56,71
Міжнародні користувачі, млн	6,12	10,93	18,28	30,02	44,37	62,83	68,29

(За даними ресурсу recode.net)

За прогнозами Digital TV Research до 2023 р. Північна Америка та Західна Європа, які є лідерами за кількістю передплатників, разом забезпечать 62 % загальної абонентської бази Netflix (див. табл. 1.2). Серед країн за кількістю користувачів Netflix лідирують США, Великобританія, Бразилія, Канада, Німеччина

Таблиця 1.2 – Кількість передплатників контенту Netflix

	2017 р.	2018.р	2023.р
Інші, млн	1	1	2
Північна Америка, млн	59	66	77
Латинська Америка, млн	14	17	25
Африка, млн	1	2	4
Країни MENA, млн	2	3	8
Західна Європа, млн	26	32	48
Східна Європа, млн	3	5	9
Азіатсько-Тихоокеанський регіон, млн	7	12	28

(За даними ресурсу recode.net)

Що стосується доступу контенту Netflix, то лідером є США. Двома гіршими країнами у цьому плані в глобальному масштабі є Зімбабве та Марокко. (див. табл. 1.3)

Таблиця 1.3 – Доступ до контенту Netflix

Країна	ТБ-шоу	К-ть фільмів	Процент американської телевізійної бібліотеки	Процентна доля бібліотеки фільмів США
США	1157	4593	100%	100%
Американська Самоа	985	4538	85,13%	98,9%
Канада	623	2562	53,85%	55,78%

Продовження табл. 1.3

Великобританія	442	1586	38,20%	34,53%
Німеччина	328	1440	28,35%	31,35%
Україна	203	562	17,55%	12,24%
Росія	113	509	9,77%	11,08%
Судан	164	359	14,17%	7,82%
Зімбабве	466	199	40,28%	4,33%
Марокко	39	118	3,37%	2,57%

(За даними ресурсу recode.net)

Більшість абонентів Netflix залишається в Північній Америці (зокрема США, Канада) та Західній Європі (Велика Британія, Німеччина), що пояснюється тим, що ці країни належать до групи високорозвинених економічно і технологічно. Крім того, вони зосереджують основну частку населення цих регіонів. Це дає користувачам можливість передплатити сервіс і отримати доступ до онлайн-бібліотеки для перегляду фільмів та серіалів легально, без порушення законодавства про авторські права та піратство. Найменша кількість абонентів зосереджена в Африці та країнах MENA, які характеризуються низьким рівнем економічного розвитку і загальним рівнем життя.

Ситуація також не краща в країнах Східної Європи. Сервіс Netflix виявився не дуже популярним у цьому регіоні з кількох причин: по-перше, мала кількість доступного контенту; по-друге, різниця у ціновій політиці.

Висновок до першого розділу

У розділі розглянуто ключові концепції та методи дата-аналізу, а також їхнє застосування для вивчення факторів впливу на популярність фільмів на стрімінгових платформах. Аналіз даних дозволяє глибше зрозуміти взаємозв’язки між такими показниками, як жанр, акторський склад, рейтинг, рецензії глядачів та рекламні кампанії, що формують популярність контенту. Використання методів статистичного аналізу, кластеризації та машинного

навчання допомагає ідентифікувати закономірності в поведінці аудиторії та прогнозувати попит на певні типи контенту.

Поєднання теоретичних знань із практичними методами аналізу даних дає змогу оцінити вагомість окремих чинників і їхній внесок у формування успішності фільмів. Наприклад, аналіз рекомендаційних систем дозволяє з'ясувати, як персоналізовані пропозиції впливають на залучення аудиторії, а використання візуалізації даних забезпечує зручне представлення складних аналітичних результатів для прийняття обґрунтованих рішень.

Таким чином, інтеграція сучасних методів дата-аналізу в дослідження популярності фільмів на стрімінгових платформах відкриває нові перспективи для оптимізації контенту, маркетингових стратегій і розробки бізнес-моделей. Це підтверджує, що дата-аналіз є потужним інструментом для підвищення ефективності досліджень і досягнення практичних результатів у сфері медіа та розваг.

РОЗДІЛ 2
ДАТА-АНАЛІЗ НАЙПОПУЛЯРНІШОЇ СТРІМІНГОВОЇ ПЛАТФОРМИ NETFLIX

2.1 Аналіз даних стрімінгової платформи Netflix за допомогою Python

Netflix розпочав свою діяльність у 1997 році як сервіс прокату DVD-дисків поштою, який використовував модель "оплата за оренду". Користувачі переглядали та замовляли фільми, які хотіли, на їхньому сайті, робили замовлення, і Netflix доставляв їх до ваших дверей. Після того, як орендарі закінчували перегляд DVD-дисків, вони просто відправляли їх назад. 10 липня 2020 року Netflix став найбільшою розважальною/медійною компанією за ринковою капіталізацією. За схожою траєкторією, як і Amazon, компанія перетворилася з формату DVD-дисків, що надсилаються поштою, на короля оригінального контенту. Зважаючи на вплив Covid-19 на роботу кінотеатрів, а також на те, що все більше людей змушені залишатися вдома, Netflix може незабаром стати життєздатною альтернативою. Аналіз буде проводитись на даних, які складаються з телевізійних шоу та фільмів, доступних на Netflix станом на 2019 рік. [16]

Дані завантажені з бібліотеки Kaggle https://www.kaggle.com/shivamb/netflix-shows. Kaggle - це ключове сховище наборів даних, які використовують фахівці з даних для співпраці або участі у вирішенні проблем, пов'язаних з даними. Набір даних зібрано з Flixable, який є сторонньою пошуковою системою Netflix. 2018 року вони опублікували цікавий звіт, який показує, що кількість телевізійних шоу на Netflix майже потроїлася з 2010 року. З 2010 року кількість фільмів на стрімінговому сервісі зменшилася більш ніж на 2 000 найменувань, тоді як кількість телешоу зросла майже втричі. Буде цікаво дослідити, які ще інсайти можна отримати з цього ж набору даних. [17]

Підготовка даних. Дані були підготовлені та очищені за допомогою Pandas за допомогою наступних кроків:

import pandas as pd

df = pd.read_csv(data_dir + '/netflix_titles.csv')

df.read()

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
0	81145628	Movie	Norm of the North: King Sized Adventure	Richard Finn, Tim Maltby	Alan Marriott, Andrew Toth, Brian Dobson, Cole...	United States, India, South Korea, China	September 9, 2019	2019	TV-PG	90 min	Children & Family Movies, Comedies	Before planning an awesome wedding for his gra...
1	80117401	Movie	Jandino: Whatever it Takes	NaN	Jandino Asporaat	United Kingdom	September 9, 2016	2016	TV-MA	94 min	Stand-Up Comedy	Jandino Asporaat riffs on the challenges of ra...
2	70234439	TV Show	Transformers Prime	NaN	Peter Cullen, Sumalee Montano, Frank Welker, J...	United States	September 8, 2018	2013	TV-Y7-FV	1 Season	Kids' TV	With the help of three human allies, the Autob...
3	80058654	TV Show	Transformers: Robots in Disguise	NaN	Will Friedle, Darren Criss, Constance Zimmer, ...	United States	September 8, 2018	2016	TV-Y7	1 Season	Kids' TV	When a prison ship crash unleashes hundreds of...
4	80125979	Movie	#realityhigh	Fernando Lebrija	Nesta Cooper, Kate Walsh, John Michael Higgins...	United States	September 8, 2017	2017	TV-14	99 min	Comedies	When nerdy high schooler Dani finally attracts...
...	...	...	...	...	...	...	...	...	...	...	...	...
6229	80000063	TV Show	Red vs. Blue	NaN	Burnie Burns, Jason Saldaña, Gustavo Sorola, G...	United States	NaN	2015	NR	13 Seasons	TV Action & Adventure, TV Comedies, TV Sci-Fi ...	This parody of first-person shooter games, mil...
6230	70286564	TV Show	Maron	NaN	Marc Maron, Judd Hirsch, Josh Brener, Nora Zeh...	United States	NaN	2016	TV-MA	4 Seasons	TV Comedies	Marc Maron stars as Marc Maron, who interviews...
6231	80116008	Movie	Little Baby Bum: Nursery Rhyme Friends	NaN	NaN	NaN	NaN	2016	NaN	60 min	Movies	Nursery rhymes and original music for children...
6232	70281022	TV Show	A Young Doctor's Notebook and Other Stories	NaN	Daniel Radcliffe, Jon Hamm, Adam Godley, Chris...	United Kingdom	NaN	2013	TV-MA	2 Seasons	British TV Shows, TV Comedies, TV Dramas	Set during the Russian Revolution, this comic ...
6233	70153404	TV Show	Friends	NaN	Jennifer Aniston, Courteney Cox, Lisa Kudrow, ...	United States	NaN	2003	TV-14	10 Seasons	Classic & Cult TV, TV Comedies	This hit sitcom follows the merry misadventure...

Рисунок 2.1 – Початкові дані для аналізу

Набір даних було завантажено у фрейм даних Panda, щоб дослідити кількість рядків і стовпців, діапазони значень тощо. Відсутні та некоректні дані були відфільтровані для покращення аналізу даних.

Це все добре, але людина любить все яскраве та інтерактивне, а не сухі таблиці та цифри, тому перейдемо до дослідницького аналізу та візуалізації даних. Дані було проаналізовано за допомогою візуалізації даних, щоб визначити інформацію про різні типи шоу, доступні на Netflix. Використаємо для цього бібліотеки seaborn та matplotlib.

import seaborn as sns

import matplotlib

import matplotlib.pyplot as plt

%matplotlib inline

sns.set_style('darkgrid')

matplotlib.rcParams['font.size'] = 14

matplotlib.rcParams['figure.figsize'] = (9, 5)

matplotlib.rcParams['figure.facecolor'] = '#00000000'

По-перше, було проаналізовано різницю в типах доступних шоу.

sns.countplot(x='type', data=df)

plt.title("Type of the Show on Netflix");

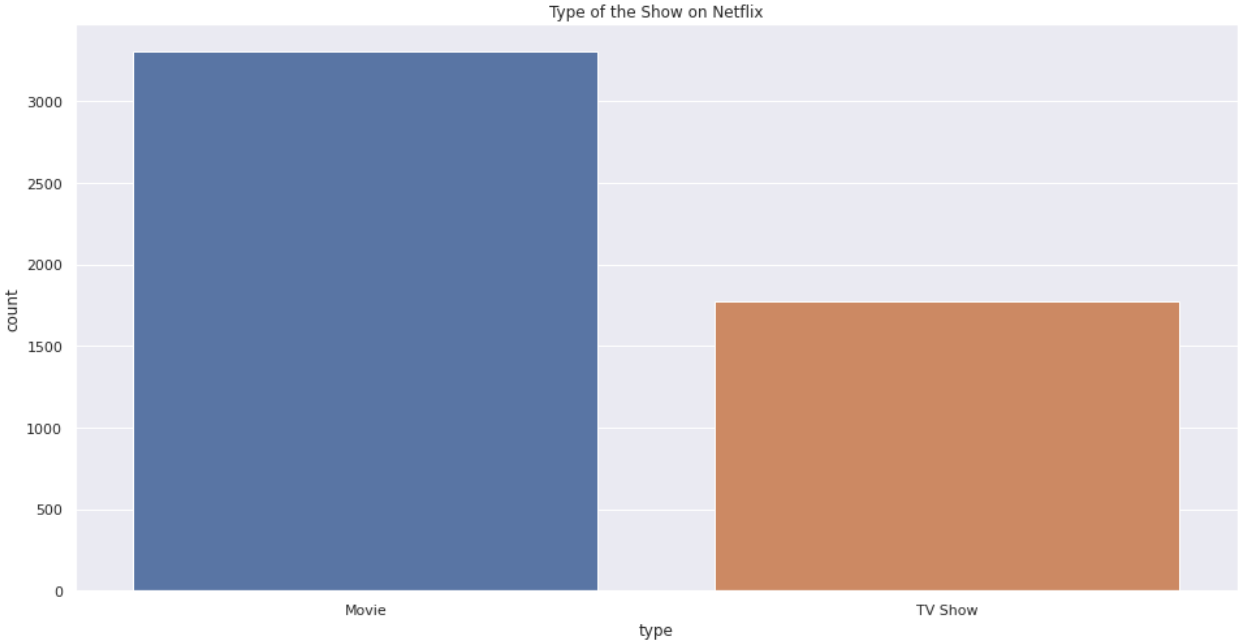

Рисунок 2.2 – Типи шоу на Netflix

Як бачимо, кількість фільмів, доступних на Netflix, перевищує кількість телевізійних шоу у співвідношенні 2:1. Це може бути пов'язано з тим, що Netflix не виробляє більшість власного контенту, а покладається на продукти конкурентів, які розробляють власні стрімінгові платформи. На платформі Netflix є безліч різних жанрів фільмів і серіалів. Ми розглянемо, які жанри є найпопулярнішими.

plt.figure(figsize=(12,6))

df[df["type"]=="Movie"]["listed_in"].value_counts()[:10].plot(kind="barh",color="blue")

plt.title("Top 10 Genres of Movies",size=18);

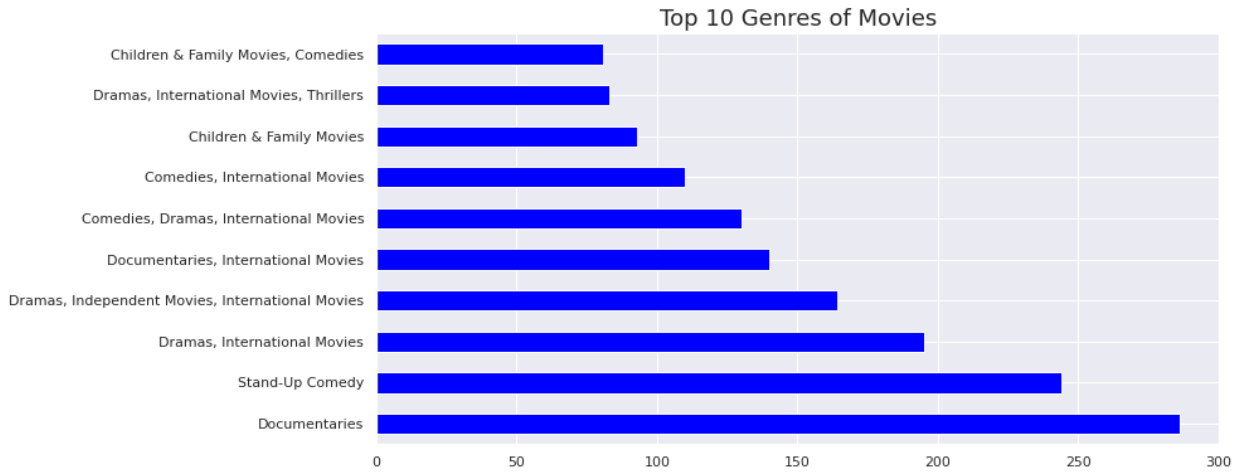

Рисунок 2.3 – Топ-10 жанрів фільмів

Дані показують, що документальні фільми є найбільшою категорією фільмів. Це може бути пов'язано з тим, що документальні фільми можуть мати нижчу ліцензійну плату від кіностудії і є дешевшим джерелом контенту. [18]

Тепер ми розглянемо, на які жанри телешоу припадає найбільше переглядів на Netflix, використовуючи гістограму.

plt.figure(figsize=(12,6))

df[df["type"]=="TV Show"]["listed_in"].value_counts()[:10].plot(kind="barh",color="green")

plt.title("Top 10 Genres of TV Shows",size=18);

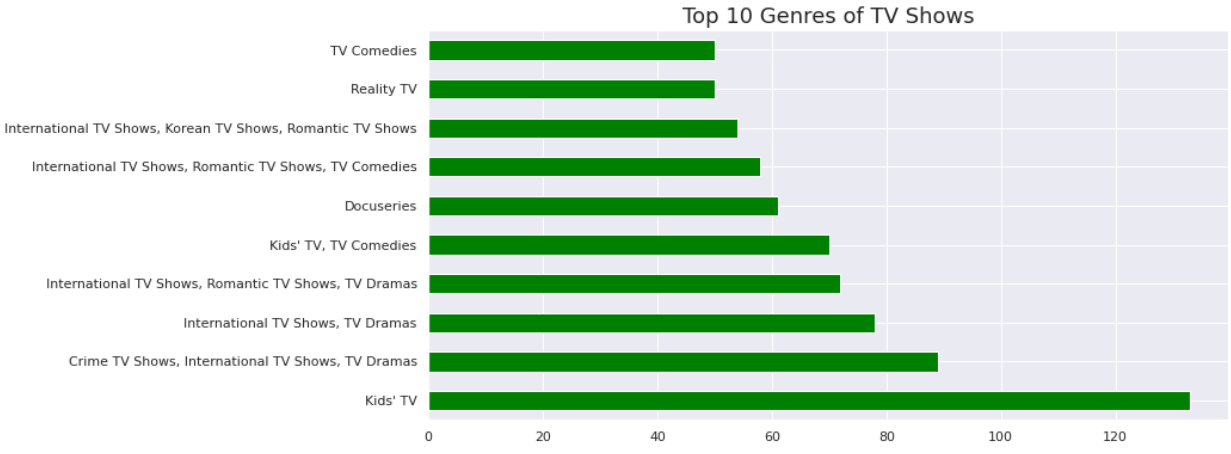

Рисунок 2.4 – Топ-10 жанрів ТВ-шоу

Висока кількість дитячих телепередач також може бути пов'язана з тим, що цей тип контенту дешевший у ліцензуванні, ніж комедії, кількість яких є

найнижчою. Більшість телевізійних комедій виробляються великими мережами та кіностудіями, які вимагають преміум-ліцензії або зберігають їх для власних платформ. Тепер ми розглянемо частоту релізів за останні десять років.

```
df = df[df['release_year']>2010]
sns.kdeplot(data=df['release_year'], label='release_year', shade=True)
plt.title('Number of shows released per year')
plt.show()
```

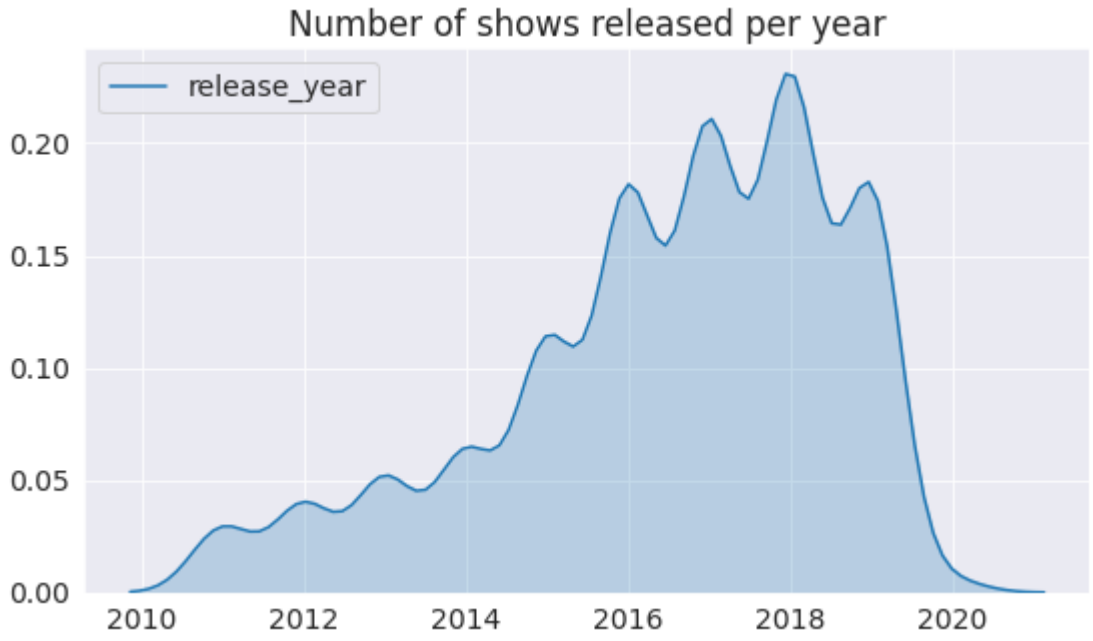

Рисунок 2.5 – Частота релізів нового контенту

Як бачимо, 2018 рік був піковим - тоді було випущено найбільшу кількість серіалів. Різкий спад у 2019 році може бути пов'язаний з тим, що конкуренти почали впроваджувати власні стрімінгові сервіси. Наприклад, усі чотири серіали Marvel не були продовжені на Netflix, а права на них повернулися до Marvel/Disney.

```
plt.figure(figsize=(11,11))
sns.countplot(x='rating',data=df)
plt.title("Ratings");
```

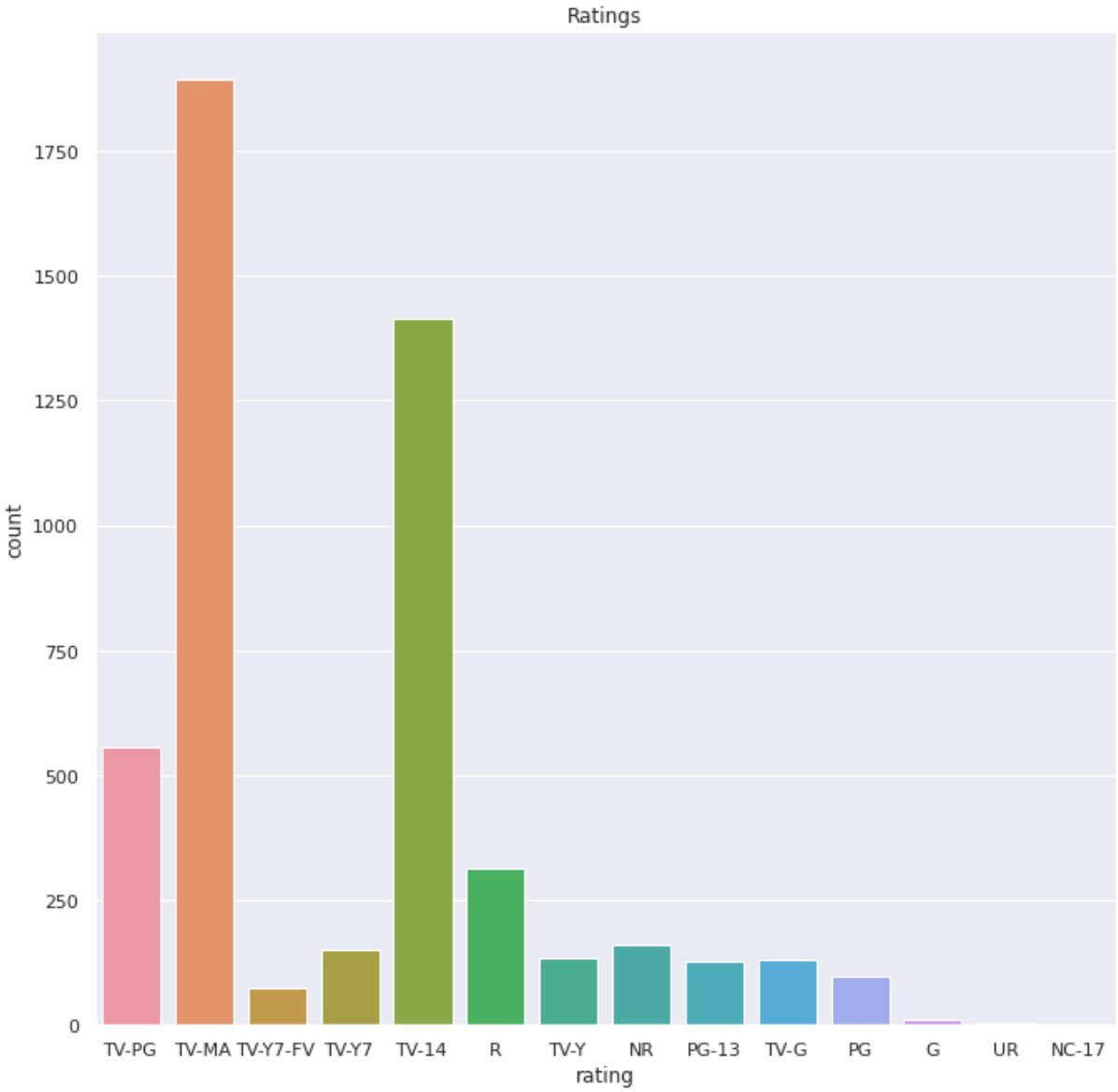

Рисунок 2.6 – Типи показів за рейтингом

Використовуючи seaborn, ми розглянемо деякі інші цікаві аспекти даних, такі як :

- Випуск шоу
- Країни з найбільшим випуском контенту
- Тривалість шоу

sns.countplot(data=df,x='release_year',hue='type');

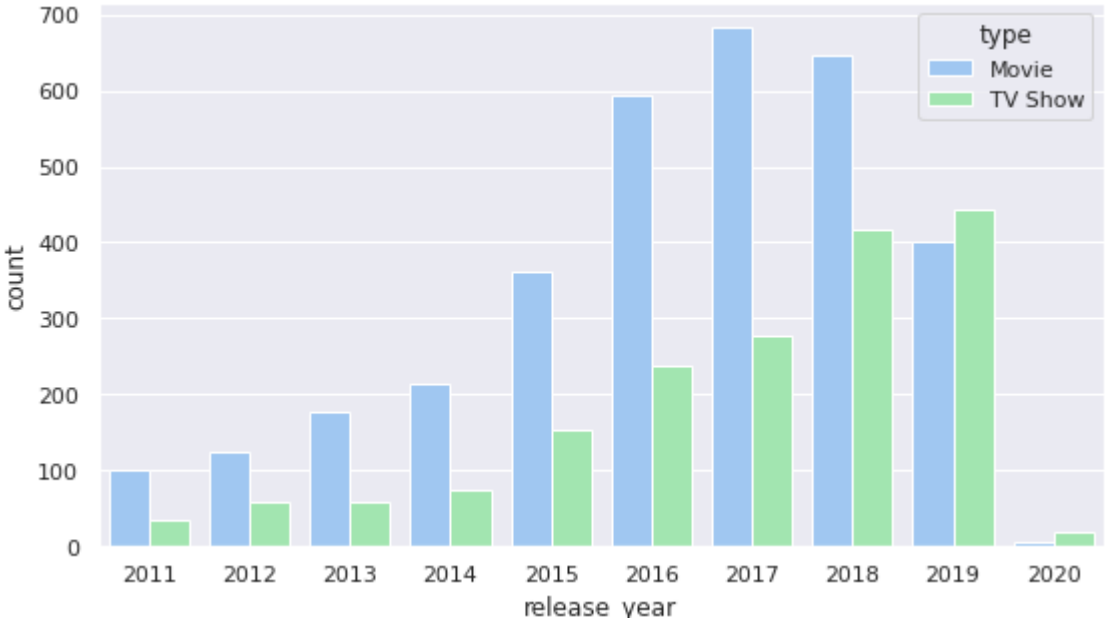

Рисунок 2.7 – Частота випуску протягом останніх десяти років

У 2011-2018 роках відбулося значне збільшення контенту, більшість якого складають фільми. Netflix також почав виробляти власні оригінальні фільми, деякі з яких також змагаються з великими студіями за нагороди Американської кіноакадемії. [19]

За допомогою методу value_counts ми можемо визначити країни з найбільшою кількістю створеного контенту.

plt.figure(1, figsize=(15, 7))

plt.title("Country with maximum content creation")

sns.countplot(x = "country", order=df['country'].value_counts().index[0:10] ,data=df,palette='Accent');

На Сполучені Штати припадає більшість контенту, створеного на Netflix, - майже 1600 найменувань. Індія посідає друге місце, можливо, через популярність боллівудських фільмів.

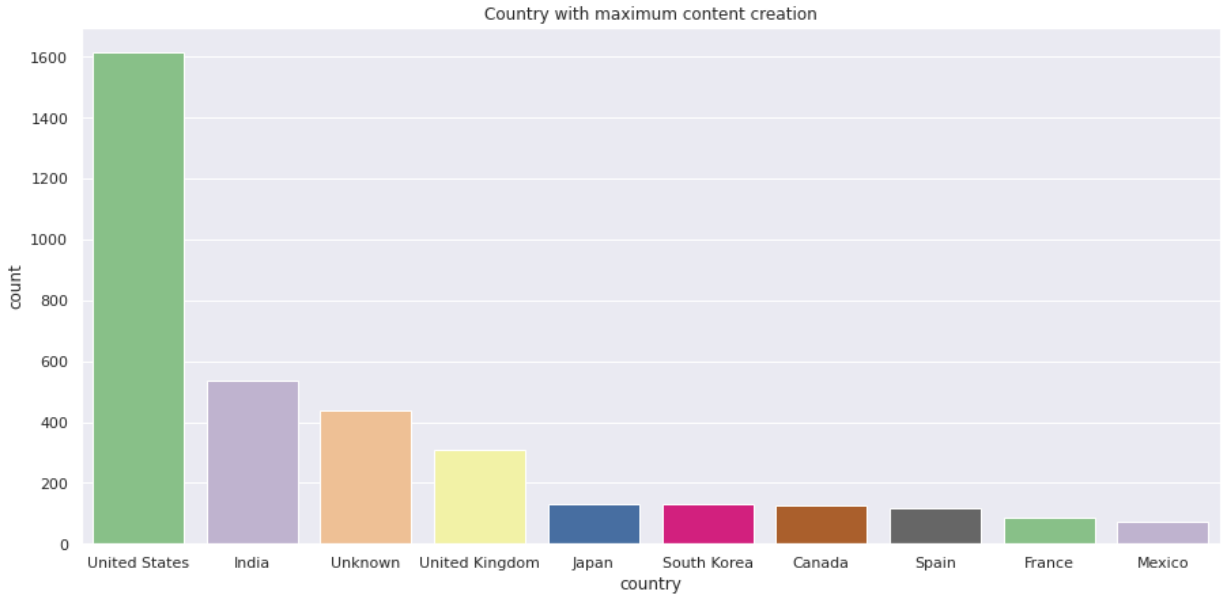

Рисунок 2.8 – Рейтинг країн за кількість зробленого контенту

Який тип контенту виробляється в кожній країні? Щоб відповісти на це питання, нам потрібно використати метод згрупування за рамками даних, щоб об'єднати рядки для кожної країни. Потім буде застосовано стовпчикову діаграму з використанням seaborn.

```
data_country = df.groupby(['country']).count()['show_id'].to_frame().reset_index()
data_country = data_country.sort_values('show_id',ascending=False)
data_country = data_country.head(5).reset_index(drop=True)
list_c = data_country['country']
top_8_data = df[df['country'].isin(list_c)]
year_with_show = top_8_data.groupby(['country','type']).count()['show_id'].to_frame().reset_index()
year_with_show = year_with_show.sort_values(by = 'show_id', ascending= False)
plt.style.use('seaborn-pastel')
plt.figure(figsize=(20,10))
sns.barplot(x="country", y="show_id", hue="type",data= year_with_show)
plt.xlabel("")
```

plt.title('Shows by Country')

plt.ylabel("")

plt.show()

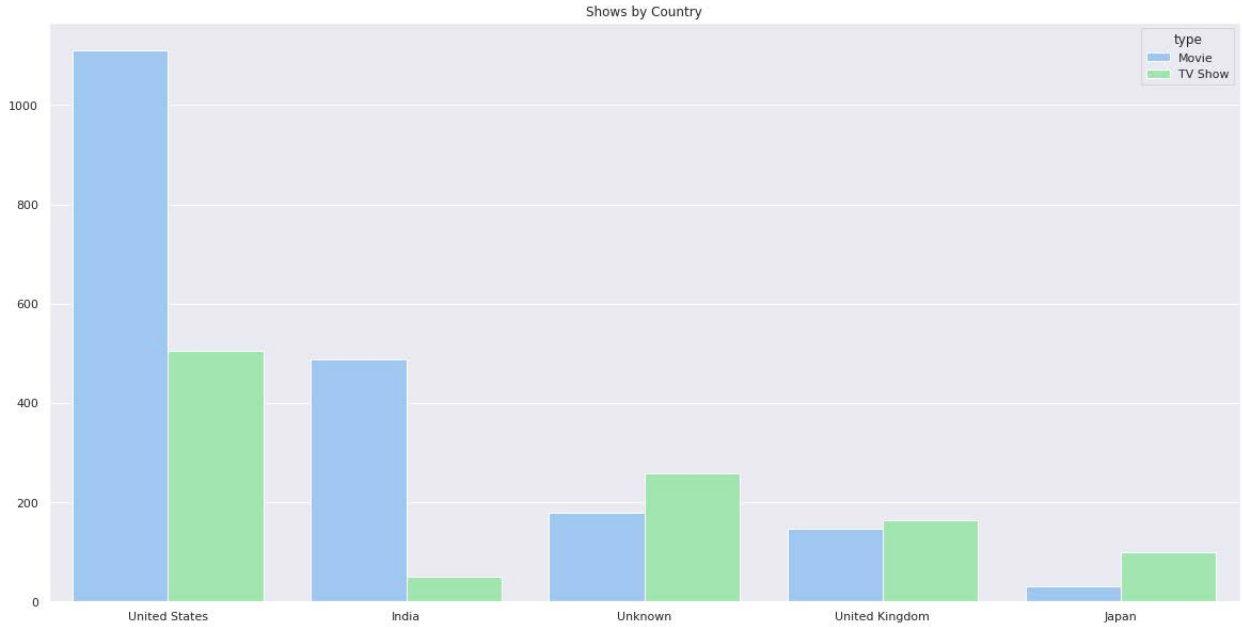

Рисунок 2.9 – Рейтинг країн за типом контенту

Фільми, вироблені в Сполучених Штатах, становлять більшість контенту загалом і з цієї країни зокрема. Слід зазначити, що фільми з Індії також становлять значну частину контенту, знову ж таки через популярність і розмір ринку Боллівуду.

plt.figure(figsize=(12,10))

sns.set(style="darkgrid")

ax = sns.countplot(y="listed_in", data=df, palette="Set1", order=df['listed_in'].value_counts().index[0:15])

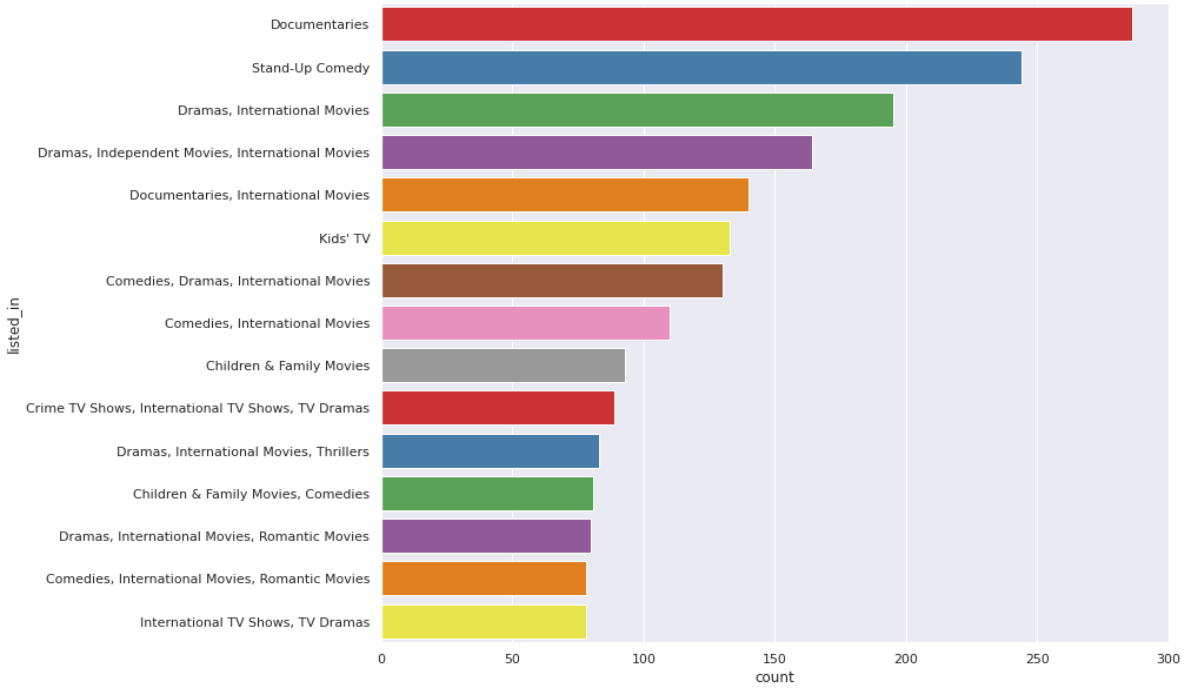

Рисунок 2.10 – Загальний жанровий склад контенту

Загалом, документальні фільми та стендап-комедійні шоу складають більшість контенту на Netfix.

plt.rcParams['figure.figsize'] = (16,8)

sns.countplot(y = 'duration', data = df, order = df['duration'].value_counts()[:10].index)

plt.title('duration analysis',fontsize = 20);

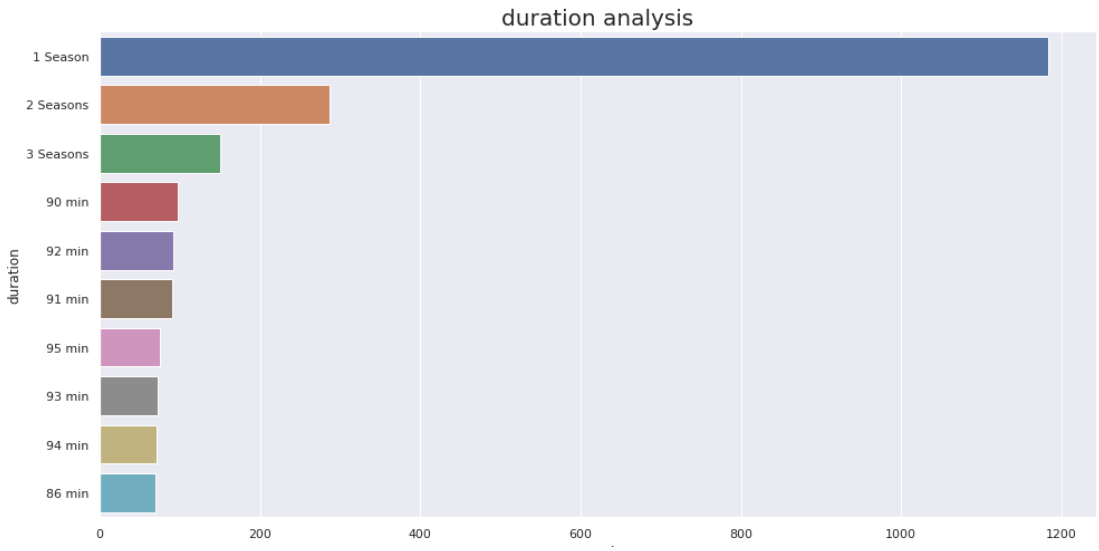

Рисунок 2.11 – Тривалість серіалів та фільмів

Більшість телевізійних шоу триває лише один сезон. Це може бути пов'язано з різними факторами, такими як вартість ліцензії, недостатня популярність та вартість виробництва.

Висновки щодо проведеної роботи з аналізу стрімінгової платформи Netflix наступні - з моменту заснування Netflix як каталогу поштових замовлень до сьогоднішнього дня він є найбільшим провайдером стрімінгових послуг у світі, його зростання було феноменальним. Зараз він конкурує з більшістю постачальників оригінального контенту, які створюють власні потокові сервіси.

Після зростання у 2011-2018 роках спостерігається уповільнення темпів створення нового контенту через конкуренцію, а також зупинку виробництва через Covid-19. Хоча це призвело до зменшення кількості контенту, кількість підписників Netflix зросла разом із вартістю акцій через те, що люди були змушені перебувати на карантині по всьому світу. [20]

2.2 Огляд технологій та їх можливостей

React – відкрита Javascript бібліотека, яка покликана вирішувати проблеми часткового оновлення змісту вебсторінки, з якими стикаються в розробці односторінкових застосунків (SPA). Розробляється Meta (раніше Facebook) і спільнотою індивідуальних розробників.

Бібліотеку React вже визнано потужним інструментом для створення користувацьких інтерфейсів. Інтерфейс користувача – одна з найважливіших частин веб-додатку, адже це те з чим користувач стикається протягом всієї взаємодії з програмою.

Браузер регулярно перевіряє будь-які зміни в DOM і оновлює його відповідно. Багато факторів впливають на зміну об’єктної моделі документа. Щоразу, коли сторінка змінюється, браузер оновлює DOM. Сучасні сайти мають великий DOM, тому оновлення займає багато часу, що уповільнює загальну продуктивність веб-додатку. React покликаний вирішити цю

проблему за допомогою віртуального DOM. Віртуальний DOM – це копія справжнього DOM і спершу зміни вносяться в нього, а вже потім оновлюється в справжньому і тим самим оптимізує роботу застосунку.

Замість того, щоб здійснювати повільні та малоефективні операції з безпосередньо реальною об’єктною моделлю документа, React використовує її спрощенну версію – віртуальну DOM. Завдяки цьому оновлення відбуваються в реальній DOM, що значно підвищує ефективність. Таким чином, віртуальна DOM надає розробникам дві ключові переваги, які ми зараз детально розглянемо.

Оновлення всієї DOM для того, щоб зробити веб-сторінку «реактивною», є надзвичайно неефективним процесом, оскільки вимагає значного обсягу обчислювальних ресурсів. Саме тому React використовує іншу стратегію: при будь-якій зміні веб-сторінки, відбувається оновлення спеціальної віртуальної DOM. React підтримує в пам’яті дві версії віртуальної DOM: одну, що відображає оновлені зміни, і резервну копію, які існувала до змін. Після модифікації React порівнює ці дві версії, виявляючи елементи, що зазнали змін, і лише тоді оновлює відповідні частини реальної DOM, мінімізуючи кількість операцій над нею.

Одним із найважливіших завдань будь-якого стартапу є створення вебдодатка, який буде водночас швидким та чуйним, а також забезпечення високого рівня обслуговування клієнтів. Віртуальна DOM, порівняно з реальною DOM, споживає значно менше ресурсів та здійснює оновлення з більшою оперативністю, що суттєво підвищує загальну продуктивність додатка.

Віртуальна DOM забезпечує миттєвий відгук сторінки на запити до сервера та дозволяє відображати зміни в реальному часі. Як приклад, Facebook використовує віртуальну DOM для динамічного оновлення чатів і стрічок новин користувачів без необхідності перезавантажувати сторінку, що значно покращує зручність взаємодії та швидкість роботи платформи.

У ReactJS розробляються багаторазові компоненти, які значно підвищують ефективність розробки. Компоненти інтерфейсу користувача, як правило, можна легко використовувати повторно в інших частинах проєкту або навіть у різних проєктах без суттєвих змін, що сприяє кращій модульності та спрощує підтримку коду.

Крім того, розробники React-додатків мають в своєму арсеналі численні бібліотеки готових компонентів з відкритим вихідним кодом. Ці бібліотеки значно скорочують час на створення додатка, що є критичним фактором для стартапів, які прагнуть оптимізувати не лише фінансові витрати, але й час, необхідний для виходу на ринок. [21]

Typescript – це мова програмування, створена та підтримувана Microsoft, яка виступає суворою синтаксичною надмножиною над Javascript і вводить можливість опціональної статичної типізації. Основною метою Typescript є полегшення розробки масштабних додатків, завдяки чому його код компілюється в Javascript. Оскільки Typescript є надмножиною Javascript, будь-який існуючий Javascript-код також вважається валідним у контексті Typescript, що дозволяє безболісно інтегрувати його в уже наявні проєкти.

Typescript може застосовуватися для розробки Javascript-додатків як на стороні клієнта, так і на серверній стороні, як, наприклад, у випадку з Node.js. Існують різні методи транскомпіляції Typescript у Javascript. Ви можете скористатися вбудованим компілятором Typescript, або використовувати компілятор Babel, що додає гнучкості у виборі інструментів під час розробки.

Typescript підтримує файли визначень, які містять інформацію про типи для вже існуючих бібліотек Javascript, подібно до того, як заголовкові файли в C++ описують структуру наявних об’єктних файлів. Це дає змогу іншим програмам використовувати значення з таких файлів так, наче вони є статично типізованими об’єктами в Typescript. Існують заголовкові файли від сторонніх розробників для широковживаних бібліотек, таких як jQuery, MongoDB, D3.js. Крім того, доступні заголовки Typescript для основних модулів Node.js, що дозволяє розробляти Node.js-додатки з використанням Typescript.

Компілятор Typescript написаний безпосередньо на самій мові Typescript і компілюється в Javascript. Він ліцензується відповідно до умов ліцензії Apache License 2.0. Починаючи з Microsoft Visual Studio 2012 Update 2, Typescript став повноцінною мовою програмування в середовищі розробки, нарівні з C# та іншими мовами Microsoft. Офіційне розширення також додає підтримку Typescript у Visual Studio 2012. Андерс Хайльсберг, відомий як головний архітектор C# та автор мов Delphi і Turbo Pascal, брав активну участь у створенні Typescript.

Анотації для примітивних типів у Typescript охоплюють числові (number), булеві (boolean) і рядкові (string) значення. Для структур, що мають слабку або динамічну типізацію, використовується тип any, який дозволяє зберігати значення будь-якого типу без обмежень, надаючи більшу гнучкість у типізації при збереженні можливості статичної перевірки. Але на практиці ліпше використовувати тип any лише в тих випадках, де не можна типізувати об’єкт.

Typescript також зберігає поведінкові характеристики Javascript під час виконання програм. Наприклад, у Javascript ділення на нуль призводить до отримання значення Infinity, а не до винятку Divided by zero, як це відбувається в багатьох інших мовах. У цьому контексті Typescript принципово не змінює логіку виконання Javascript-коду, залишаючи вихідні механізми роботи без змін, додаючи лише інструменти для статичної перевірки та типізації.

Це означає, що при перенесенні коду його виконання залишиться ідентичним, навіть якщо Typescript виявить помилки типів. Незважаючи на те, що Typescript здатний виявляти й повідомляти про такі помилки на етапі компіляції, вони не впливають на фактичну поведінку коду під час виконання. Це дозволяє зберегти повну сумісність з Javascript, навіть у випадках, коли типізація не є строгою або коректною. Підсумувавши все вище сказане, можна дійти висновку, що Typescript покликаний для розробників, щоб попередити про можливі помилки до запуску коду. [23]

Для збірки нашого проєкту ми будемо використовувати Vite. Чому саме його і які проблеми він вирішує? До того, як модулі ES стали доступними в браузерах, розробники не мали власного механізму для створення Javascript за модульним принципом. Ось чому всі знайомі з поняттям «бандлінг»: використання інструментів, які сканують, обробляють і об’єднують наші вихідні модулі у файли, які можна запускати в браузері.

З часом ми побачили такі інструмнети, як Webpack, Rollup, Parcel, які значно покращили досвід розробки для фронтенд-розробників.

Однак, оскільки ми створюємо все більш і більш амбітні додатки, кількість JavaScript, з яким ми маємо справу, також різко зростає. Нерідко великомасштабні проекти містять тисячі модулів. Ми починаємо стикатися з вузьким місцем у продуктивності інструментів на JavaScript: часто може знадобитися невиправдано довге очікування (іноді до декількох хвилин!), щоб запустити сервер розробки, і навіть з гарячою заміною модулів (HMR) редагування файлів може зайняти пару секунд, щоб відобразитися в браузері. Повільний цикл зворотного зв'язку може суттєво вплинути на продуктивність та задоволення розробників.

Vite має на меті вирішити ці проблеми, використовуючи нові досягнення в екосистемі: наявність нативних модулів ES у браузері та поява інструментів JavaScript, написаних на мовах, що компілюються в нативні.

Під час холодного запуску сервера розробників, збірка на основі пакунків повинна нетерпляче повзати і збирати весь ваш додаток, перш ніж його можна буде обслуговувати.

Vite покращує час запуску сервера розробників, спочатку розділяючи модулі в додатку на дві категорії: залежності та вихідний код. [22]

Висновок до другого розділу

Розділ, присвячений аналізу даних стрімінгової платформи Netflix за допомогою Python, продемонстрував ефективність інструментів обробки та візуалізації даних для дослідження динаміки популярності контенту. За

допомогою бібліотек Pandas, Matplotlib та Seaborn вдалося виявити ключові тренди у переглядах, вплив жанрів і оцінок глядачів на рейтинг фільмів, а також взаємозв’язок між датами релізу та популярністю. Аналіз підтвердив, що методи Python забезпечують високу точність і гнучкість у роботі з великими обсягами даних, створюючи основу для прийняття стратегічних рішень.

У частині, присвяченій розгляду технологій для веброзробки, було оцінено можливості використання React, TypeScript та Express для створення сучасних, інтерактивних вебдодатків. React забезпечує компонентний підхід, що сприяє масштабованості проєкту, TypeScript гарантує типобезпеку й зручність у підтримці коду, а Express слугує надійною платформою для розробки серверної частини. Така комбінація технологій дозволяє реалізувати ефективні клієнт-серверні додатки з високим рівнем продуктивності та зручністю для користувачів.

Загалом, поєднання аналізу даних із Python та сучасних вебтехнологій створює потужний інструментарій для розробки продуктів, орієнтованих на реальні потреби користувачів, а також для проведення глибоких досліджень у сфері стрімінгових платформ.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПОРТАЛУ

3.1 Розробка backend частини

Створення будь-якого проєкту починається зі створення репозиторію та його клонування на локальну машину для подальшої розробки.

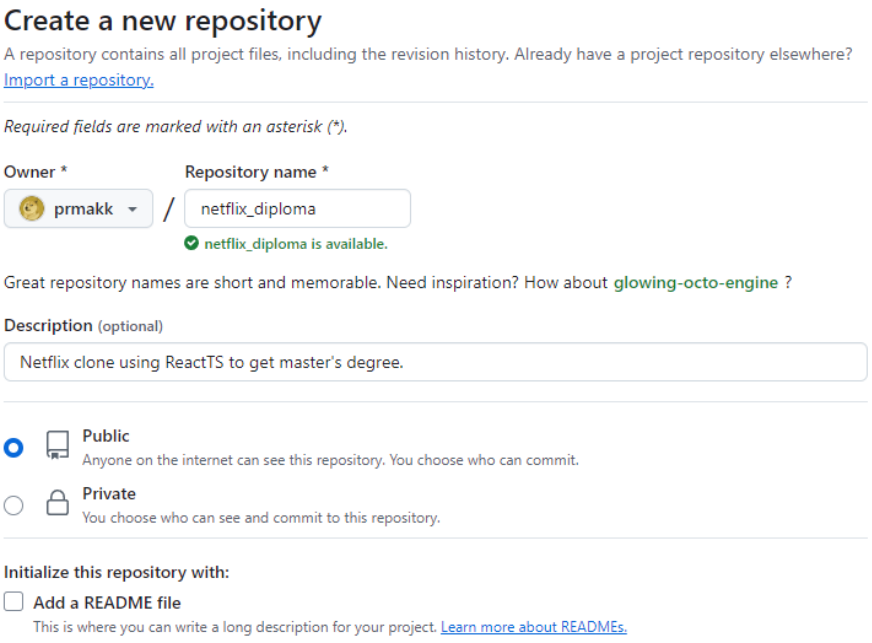

Рисунок 3.1 – Створення Github репозиторію

Тепер створенний репозиторій потрібно склонувати до локальної машини. Для цього потрібно скопіювати посилання на нього та виконати команду *git clone* в середовищі розробки.

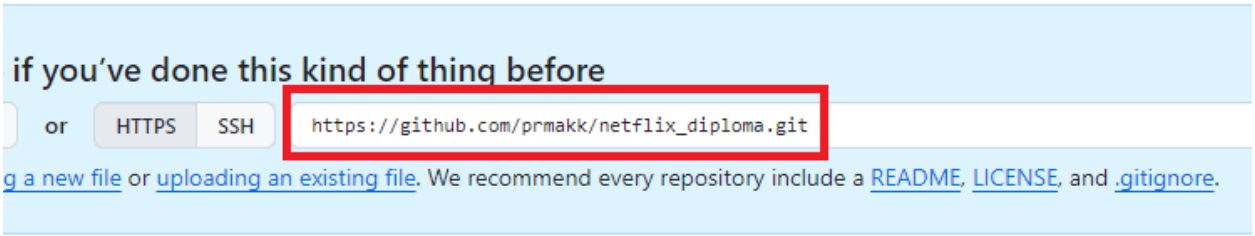

Рисунок 3.2 – Посилання на репозиторій

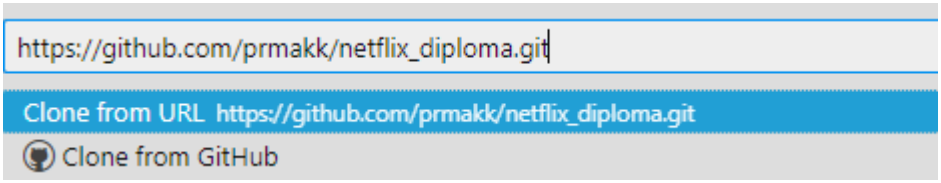

Рисунок 3.3 – Клонування репозиторію в IDE

Після цих дій в нас є ініціалізований репозиторій та його копія на локальній машині. Розділимо в нашій директорії логіку на окремі папки backend та frontend. Зазвичай першим розробляється бекенд продукту, або це відбувається одночасно з верстkою візуальної частини. Перейдемо в директорію backend командою *cd backend* і проініціалуємось командою *npm init*.

Для встановлення будь-якої бібліотеки в терміналі потрібно виконати команду *npm install *назва бібліотеки**. Також можна встановити декілька бібліотек одночасно просто перерахувавши їх через пробіли. Встановимо перші бібліотеки, які нам знадобляться для розробки бекенду командою *npm install express jsonwebtoken mongoose cookie-parser dotenv axios bcryptjs*.

- Express – бібліотека для написання серверної частини додатку.
- Jsonwebtoken – бібліотека для роботи з JWT токенами та авторизацією.
- Mongoose – бібліотека для роботи з базою даних Mongo.
- Cookie-parser – бібліотека для роботи з cookie-файлами.
- Dotenv – бібліотека для роботи зі змінними оточення.
- Axios – бібліотека для роботи з HTTP-запитами.
- Bcryptjs – бібліотека для шифрування паролів, оскільки зберігати паролі у відкритом вигляді в базі даних є поганою практикою.

Після первісного налаштування проєкту ми можемо зробити перший комміт та зберегти всі наші зміни. Для цього в терміналі послідовно потрібно виконати наступні команди: *git init, git add ., git commit –m “chore: init”, git push origin main*.

Пройдемося по кожній команді окремо:

- git init: використовується один раз під час створення проєкту для відстеження змін в директорії

- git add . : додає всі змінені файли для нового комміту

- git commit –m “chore: init”: робить комміт, а флаг m відповідає за повідомлення в комміті. Тобто розробник описує, які зміни відбулися в цьому комміті.

- git push origin main: відправляємо всі зміни в головну гілку віддаленого репозиторію.

Ці команди будуть використовуватися впродовж розробки всього проєкту. Як тільки реалізуємо новий функціонал, закоммітимо цей прогрес і в разі потреби зможемо до нього повернутися.

Внесемо деякі зміни до файлу *package.json* в головній директорії. Додамо поле *“type”* зі значенням *“module”*, щоб використовувати сучасний код для імпортів бібліотек. В поле scripts додамо *“dev”: “node --watch backend/server.js”*, щоб запускати одразу головний файл серверу та відслідковувати зміни в ньому. В полі “main” замінимо *index.js* на *“backend/server.js”*, оскільки наш головний файл називається *server.js*.

Створимо наш базовий сервер і перевіримо його роботу. Запустимо його на порті 5000 і виведемо в консоль, що сервер успішно розпочав свою роботу.

backend > JS server.js > ...

```
1   import express from "express";
2
3   const app = express();
4
5   app.get("/", (req, res) => {
6       res.json("Server is ready.");
7   });
8
9   app.listen(5000, () => console.log("Server is working :)"));
10
```

Рисунок 3.4 – Код серверу

Якщо тепер в терміналі виконаємо команду *npm run dev*, яку ми раніше описали, то наш сервер успішно запуститься по адресу https://localhost:5000 і

віддасть нам повідомлення “Server is ready”, тому що саме таку відповідь ми прописали для get-запиту на головний ендпоінт.

Створимо директорію routes для всіх майбутніх файлів з ендпоінтами. В цій директорії створимо перший файл *auth.routes.js* для роботи з авторизацією користувача.

server.js U

backend > server.js > ...

```
import express from "express";

import authRoutes from "./routes/auth.route.js";

const app = express();

app.use("/api/v1/auth", authRoutes);

app.listen(5000, () => console.log("Server is working :)"));

```

auth.route.js U

backend > routes > auth.route.js > ...

```
import express from "express";

const router = express.Router();

router.get("/signup", (req, res) => {
    res.json("Signup route");
});

router.get("/login", (req, res) => {
    res.json("Login route");
});

router.get("/logout", (req, res) => {
    res.json("Logout route");
});

export default router;

```

Рисунок 3.5 – Сервер та ендпоінти авторизації

Але ці функції, які викликаються при запиті на ендпоінт також краще перенести в окрему директорію controllers. Це найкраща практика розробки ПЗ з розділенням логіки. В папці controllers створимо файл auth.controller.js.

auth.route.js U

backend > routes > auth.route.js > default

```
import express from "express";

import { signup, login, logout } from "../controllers/auth.controller.js";

const router = express.Router();

router.get("/signup", signup);
router.get("/login", login);
router.get("/logout", logout);

export default router;

```

auth.controller.js U

backend > controllers > auth.controller.js > logout

```
export async function signup(req, res) {
    res.json("Signup route");
}

export async function login(req, res) {
    res.json("Login route");
}

export async function logout(req, res) {
    res.json("Logout route");
}

```

Рисунок 3.6 – Перенесена логіка в контроллер

Тобто логіка програми на даний час ніяк не змінилася, ми просто розподілили код в структурі проєкту. Закоммітимо даний результат, щоб

зберегти наші зміни. Виконаємо в терміналі команди *git add .*, *git commit –m “feat: added server routes and auth controllers”*, *git push origin main*.

Перейдемо до створення бази даних, щоб можна вже було працювати над реєстрацією користувачів. Для нашого проєкту ми будемо використовувати MongoDB, оскільки вона проста у використанні.

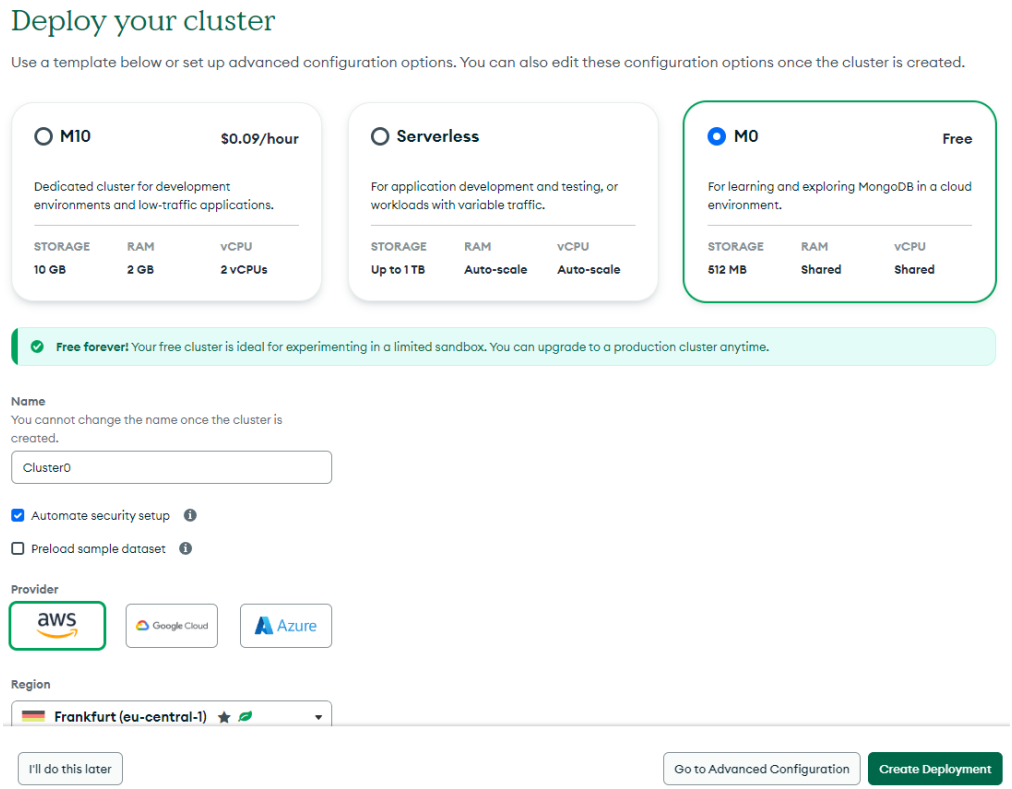

Рисунок 3.7 – Створення кластеру в MongoDB

Після створення кластеру нам видадуть рядок підключення, який нам потрібно скопіювати і вставити у створенний файл *.env.local*. Це файл для змінних оточення, тобто ми можемо їх використовувати при розробці не у відкритому вигляді. Обов’язково в файл *.gitignore* потрібно додати рядок **.local*, щоб наші змінні не потрапили у відкритий доступ на GitHub.

ackend > JS server.js > ...

```
1  import express from "express";
2  import dotenv from "dotenv";
3  import mongoose from "mongoose";
4
5  import authRoutes from "./routes/auth.route.js";
6
7  dotenv.config({ path: ".env.local" });
8
9  const app = express();
10
11 mongoose
12     .connect(process.env.MONGO_URI)
13     .then(() => console.log("DB OK"))
14     .catch((err) => console.log(err));
15
16 app.use("/api/v1/auth", authRoutes);
17
18 app.listen(5000, () => console.log("Server is working :)"));
```

Рисунок 3.8 – Підключення до бази даних

Для роботи з даними в базі даних потрібні моделі з якими ми будемо працювати в коді і відправляти в базу. Для цього створимо нову папку models і в ній файл *user.model.js*, де будуть описані всі поля, які потрібні для моделі користувача.

backend > models > JS user.model.js > ...

```
import { Schema, model } from "mongoose";

const userSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
    image: {
        type: String,
        default: "",
    },
    favorites: {
        type: Array,
        default: [],
    },
});

export const User = model("User", userSchema);
```

Рисунок 3.9 – Модель користувача

Згідно рисунку 3.9 ми можемо бачити, які поля будуть у користувача в базі даних. Це ім’я з типом рядок, рядок цей повинен бути обов’язково та повинен бути унікальним. Те саме і для пошти користувача, тому що для одного аккаунта – одна пошта. Пароль з типом рядок та обов’язковий. Аватар може бути не обов’язковий і буде масив з фільмами та серіалами, які користувач додасть в обране.

Потрібно зберегти прогрес коммітом – *git commit –m “feat: database connected”*.

Оскільки база даних в нас підключена, можемо перейти до реалізації авторизації, а саме до функції реєстрації. Як взагалі вона буде працювати? Користувач вводить дані в форму на клієнті, натискає кнопку і дані, які він

передав відсилаються на наш ендпоінт на сервері. Відбувається перевірка логіна і якщо його не існує в базі даних, то ми успішно реєструємо нового користувача і відправляємо JWT токен на клієнт. Якщо ж логін вже існує в базі, то сервер відправляє помилку і ми на клієнті її відобразимо повідомленням «Користувач з таким логіном вже існує».

Щоб ми змогли парсити дані з тіла запиту, потрібно додати рядок *app.use(express.json())* перед запуском сервера.

Для тестування нашого API (див. рис. 3.10), ми будемо використовувати програму Postman. Вона безкоштовна і дозволяє робити HTTP-запити до нашого сервера.

Рисунок 3.10 – Програма для HTTP-запитів

```
    try {
        const { email, password, username } = req.body;

        if (!email || !password || !username) {
            return res
                .status(400)
                .json({ success: false, message: "All fields are required" });
        }

        const emailRegex =
            /^((?!\.)[\w-_.]*[^.])(@\w+)(\.\w+(\.\w+)?[^.\W])$/gim;

        if (!emailRegex.test(email)) {
            return res
                .status(400)
                .json({ success: false, message: "Invalid email" });
        }

        if (password.length < 6) {
            return res.status(400).json({
                success: false,
                message: "Password must be at least 6 characters",
            });
        }

        const existingUserByEmail = await User.findOne({ email: email });

        if (existingUserByEmail) {
            return res.status(400).json({
                success: false,
                message: "User with this email already exists",
            });
        }

        const existingUserByUsername = await User.findOne({
            username: username,
        });

        if (existingUserByUsername) {
            return res.status(400).json({
                success: false,
                message: "User with this username already exists",
            });
        }

        const newUser = new User({
            email,
            password,
            username,
        });

        res.status(201).json({
            success: true,
            user: { ...newUser._doc, password: "" },
        });

        await newUser.save();
    } catch (error) {
        console.log("Error in signup controller: ", error.message);
        res.status(500).json({
```

Рисунок 3.11 – Код для реєстрації користувача

Пройдімось згори донизу і розберемося, що тут відбувається. Ми дістаємо деструктуризацією пошту, нікнейм та пароль, які нам прийшли з

клієнту. Далі ми це перевіряємо чи всі взагалі поля нам відправили. Якщо ні – повертаємо помилку, з повідомленням, що всі поля обов’язково повинні бути заповнені.

Після цього ми за допомогою regex-рівняння перевіряємо чи коректна пошта нам прийшла, а не просто набір символів. Якщо ні – повертаємо помилку, з повідомленням, що пошта некоректна.

Переходимо до пароля і перевіряємо його довжину. Якщо менша ніж 6 символів – повертаємо помилку.

Якщо ж ці всі перевірки пройшли успішно, спробуємо знайти такого користувача в нашій базі даних за поштою. Якщо буде співпадіння, то повертаємо помилку, що користувач з такою поштою вже існує. Те саме зробимо і для нікнейму.

І якщо тут немає помилок, то ми знаємо, що всі дані прийшли коректні і такого користувача не існує в нашій базі, тому ми успішно його генеруємо і зберігаємо в БД.

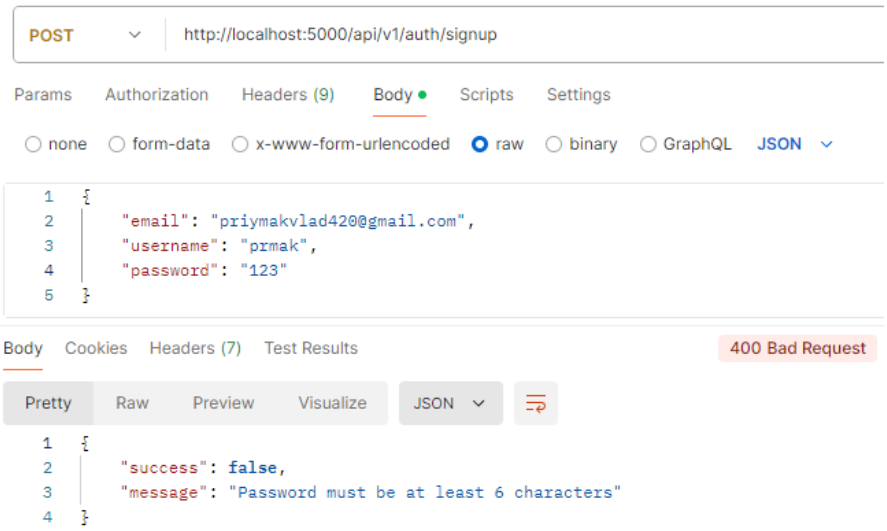

Рисунок 3.12 – Запит на реєстрацію з помилкою

Ми відправили тестовий запит, але з занадто коротким паролем. Сервер успішно це перевірив і повернув нам помилку. Відправимо тепер коректний запит.

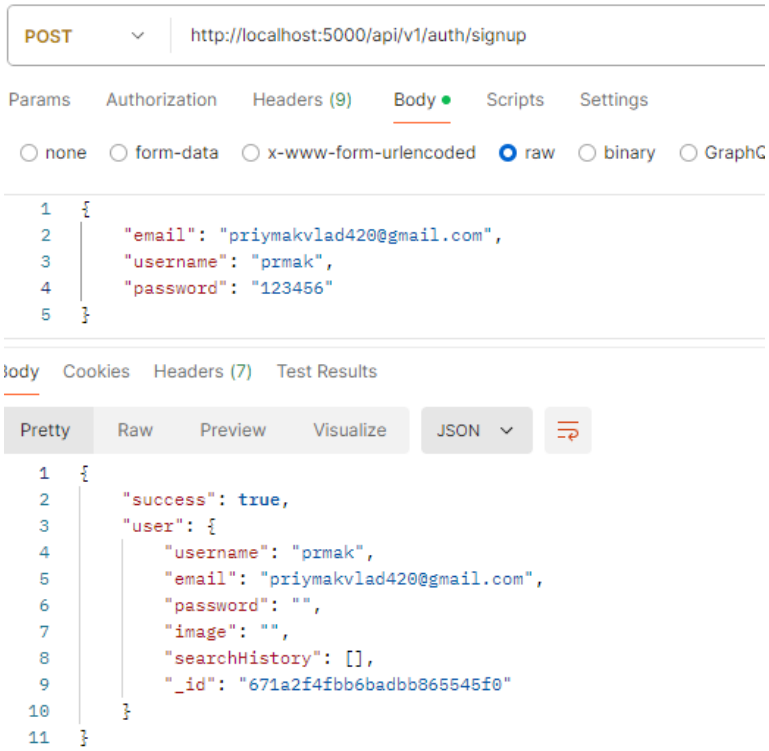

Рисунок 3.13 – Коректний запит на реєстрацію

Як ми бачимо реєстрація нового користувача пройшла успішно і сервер повернув нам дані, але без паролю, що важливо.

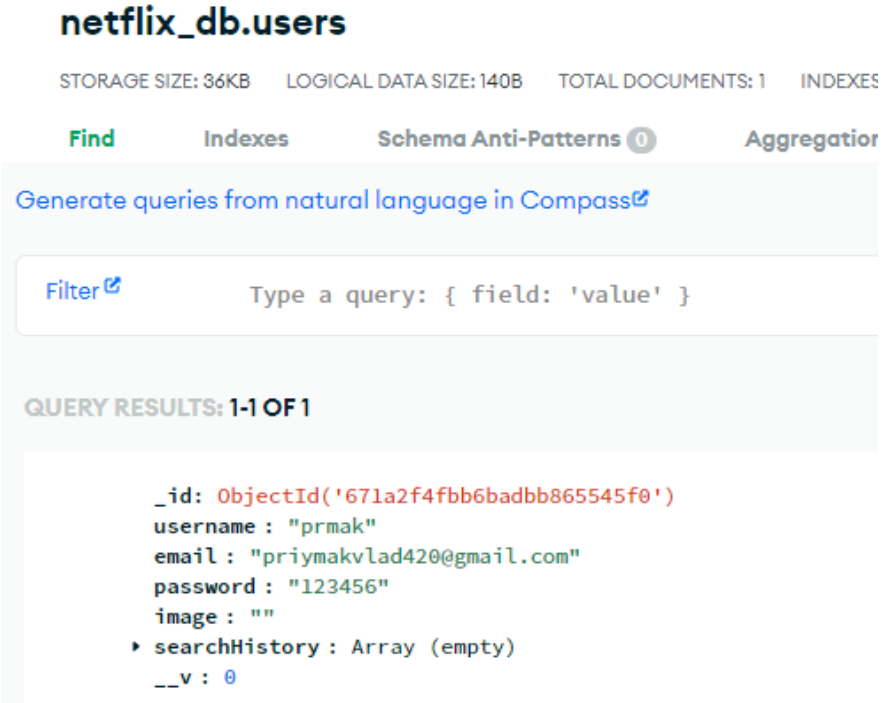

Рисунок 3.14 – Новий користувач в БД

Новий користувач успішно зберігся у базі даних, але досвідчений розробник одразу побачить тут помилку. Пароль зберігається у відкритом вигляді, що є грубою помилкою, адже якщо станеться витік даних, то всі паролі одразу буде видно хакеру.

Для цього потрібно шифрувати паролі на стороні серверу за домогою дуже складних алгоритмів, які майже нереально зламати.

```
const salt = await bcryptjs.genSalt(10);
const hashedPassword = await bcryptjs.hash(password, salt);

const newUser = new User({
    email,
    password: hashedPassword,
    username,
});
```

Рисунок 3.15 – Хешування паролю

Для хешування використаємо бібліотеку bcryptjs та декілька її методів. Потрібно згенерувати так звану сіль, яка буде змішувати наш пароль до нечитаємого вигляду. Після цього до паролю, який прийшов з клієнту у чистому вигляді, ми додаємо сіль і хешуємо і вже після цих дій ми зберігаємо пароль у хешованому вигляді до бази даних.

Рисунок 3.16 – Користувач в БД з хешованим паролем

Як бачимо на рис. 3.16 в базі даних з’явився користувач з хешованим паролем. Це абсолютно той самий пароль «123456», але вже в хешованому вигляді, який неможливо прочитати.

Для створення JWT токену та cookie створимо нову папку utils з файлом *generateToken.js* і напишемо наступний код.

backend > utils > JS generateToken.js > generateTokenAndSetCookie

```
1   import jwt from "jsonwebtoken";
2
3   export const generateTokenAndSetCookie = (userId, res) => {
4       const token = jwt.sign({ userId }, process.env.JWT_SECRET, {
5           expiresIn: "15d",
6       });
7
8       res.cookie("jwt-netflix", token, {
9           maxAge: 15 * 24 * 60 * 60 * 1000, //15 days in ms
10          httpOnly: true, //prevent XSS attack, make it not be accessed by JS
11          sameSite: "strict",
12          secure: process.env.NODE_ENV !== "development",
13      });
14
15      return token;
16  };
17
```

Рисунок 3.17 – Генерація JWT та відправка cookie

Функція приймає айді користувача і генерується токен на його основі та секретної фрази, яка зберігається в .env.local файлі. Це може бути абсолютно будь-яка фраза і вона буде використовуватись для дешифрування токену. Зробимо наступну змінну JWT_SECRET = jwt_token_kryvyi_rih_national_university.

Токен буде діяти 15 діб і у відповідь ми відправляємо cookie з цим токеном на клієнт. Цю функцію ми будемо викликати після створення користувача і перед його збереженням до БД в файлі *auth.controller.js*.

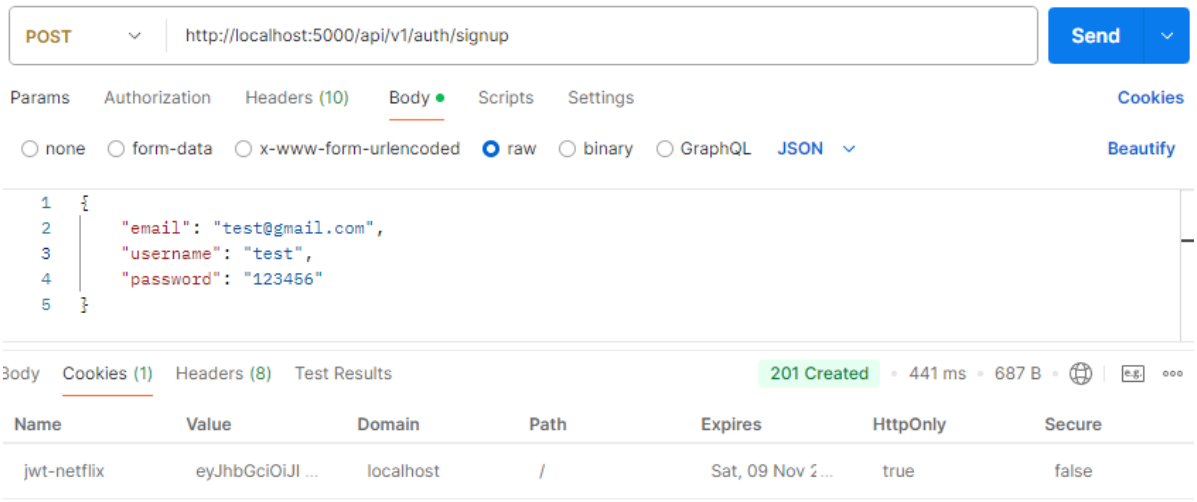

Рисунок 3.18 – JWT токен успішно повернувся у відповідь

Як можемо побачити на рис. 3.18 у відповідь на реєстрацію нового користувача, нам повертається cookie з jwt-токеном. Зробимо новий комміт, тому що ми реалізували повністю функціонал реєстрації користувача – *git commit –m “feat: implemented signup logic”*.

Реалізуємо функціонал виходу з аккаунту (logout). Для цього нам просто потрібно видалити ті cookie, які ми відправляли в реєстрації. Не буде cookie з jwt-токеном – не буде доступу до аккаунту, логіка тут проста.

```
export async function logout(req, res) {
    try {
        res.clearCookie("jwt-netflix");
        res.status(200).json({
            success: true,
            message: "Logged out successfully",
        });
    } catch (error) {
        console.log("Error in logout controller", error.message);
        res.status(500).json({
            success: false,
            message: "Internal server error",
        });
    }
}
```

Рисунок 3.19 – Реалізація виходу з аккаунту

Протестувавши цей запит в Postman, cookie успішно видаляються і логіка виходу реалізована. Закоммітимо новий функціонал – *git commit –m “feat: implemented logout logic”*.

```
76  export async function login(req, res) {
77      try {
78          const { email, password } = req.body;
79
80          if (!email || !password) {
81              return res
82                  .status(400)
83                  .json({ success: false, message: "All fields are required" });
84          }
85
86          const user = await User.findOne({ email: email });
87          if (!user) {
88              return res
89                  .status(404)
90                  .json({ success: false, message: "Invalid credentials" });
91          }
92
93          const isPasswordCorrect = await bcryptjs.compare(
94              password,
95              user.password
96          );
97
98          if (!isPasswordCorrect) {
99              return res
100                 .status(404)
101                 .json({ success: false, message: "Invalid credentials" });
102         }
103
104         generateTokenAndSetCookie(user._id, res);
105
106         res.status(201).json({
107             success: true,
108             user: { ...user._doc, password: "" },
109         });
110     } catch (error) {
111         res.status(500).json({
112             success: false,
113             message: "Internal server error",
114         });
115     }
116 }
```

Рисунок 3.20 – Реалізація входу в аккаунт

Як ми можемо бачити на рисунку 3.20, ми дістаємо з тіла запиту пароль та пошту, перевіряємо їх наявність і вже після цього ми намагаємось знайти такого користувача в базі даних. Якщо ж такий користувач існує, то ми порівнюємо пароль з тим хешем паролю, який в нас зберігається в базі даних. Після перевірки ми відправляємо новий токен, який будет діяти 15 діб.

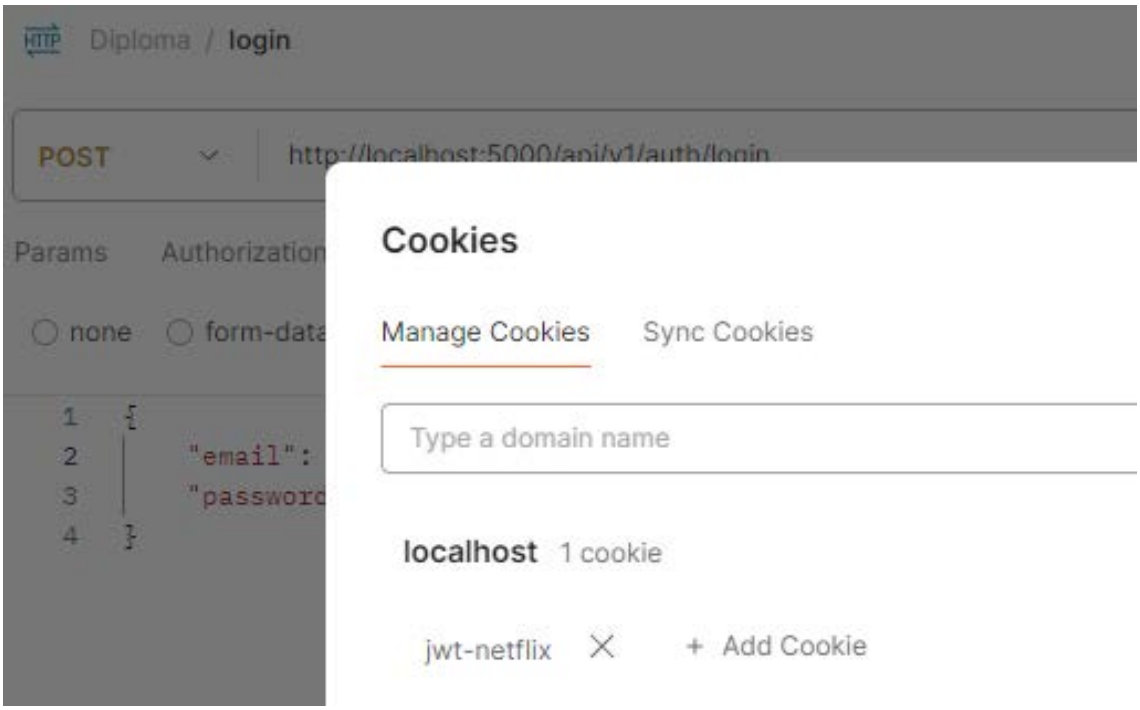

Рисунок 3.21 – Отриманий JWT-токен авторизації

Підсумуємо, що ми вже реалізували. Реєстрацію, генерацію токену авторизації, фукнціонал логіну та виходу з аккаунту. Яка ж логіка буде на клієнті? Ми будемо відправляти пароль та пошту і отримувати (або не отримувати) токен авторизації у відповідь. Далі буде відбуватися перевірка, якщо токен існує, то будемо відмальовувати один екран, якщо немає, то інший.

Закоммітимо даний результат в терміналі – *git commit –m “feat: implemented login logic”*.

Розробимо функціонал для перевірки чи авторизований користувач. Для цього в *auth.route.js* додамо новий ендпоінт і напишемо для нього свою функцію. Але це буде не дуже правильно оскільки це повинен бути захищенний роут. Тобто треба написати мідлвейр, який буде перевіряти чи є у користувача взагалі доступ до цієї сторінки.

backend > middleware > JS protectRoute.js > ...

```
import jwt from "jsonwebtoken";
import { User } from "../models/user.model.js";

export const protectRoute = async (req, res, next) => {
    try {
        const token = req.cookies["jwt-netflix"];

        if (!token) {
            return res.status(400).json({
                success: false,
                message: "Unauthtorized - No Token Provided",
            });
        }

        const decoded = jwt.verify(token, process.env.JWT_SECRET);

        if (!decoded) {
            return res.status(401).json({
                success: false,
                message: "Unauthtorized - Invalid token",
            });
        }

        const user = await User.findById(decoded.userId).select("-password");

        if (!user) {
            return res.status(404).json({
                success: false,
                message: "User not found",
            });
        }

        req.user = user;

        next(); //if we pass all checks, we allow to call next function
    } catch (error) {
        console.log("Error in protect route middleware");
        res.status(500).json({
            success: false,
            message: "Internal server error",
        });
    }
};
```

Рисунок 3.22 – Реалізація міддлвару для захищенних роутів

Міддлвар (від англ. middleware) – це проміжне програмне забезпечення або функція, яка обробляє запити між різними частинами програми. У контексті веб-розробки вони використовуються для обробки HTTP-запитів до того, як вони дійдуть до основної логіки програми або після її обробки. Міддлвари часто використовуються для:

1. Аутентифікації та авторизації – перевірка прав доступу користувача.
2. Логування – запис подій і дій користувача.
3. Маніпуляції з даними – зміна або валідація даних перед передачею далі.

Нам міддлвар потрібен якраз для реалізації першого пункту із цього списку. Ми дістаємо jwt-токен, який ми відправляємо при логіні. Якщо він є і він коректний, то у користувача є права доступу зайти на цю сторінку. Закоммітимо нову фічу – *git commit –m “feat: added protectRoute middleware”*.

Повернемося до файлу auth.route.js і створимо ендпоінт /authCheck.

```
134  export async function authCheck(req, res) {
135      try {
136          res.status(200).json({ success: true, user: req.user });
137      } catch (error) {
138          console.log("Error in authCheck controller", error.message);
139          res.status(500).json({
140              success: false,
141              message: "Internal server error",
142          });
143      }
144  }
145
```

Рисунок 3.23 – Ендпоінт перевірки авторизації

Одна з головних функцій подібних проєктів, це додавання фільмів або інших картин в обране, щоб не загубити.

```
export async function addFavorite(req, res) {
    try {
        const { userId, movieId } = req.body;

        await User.findOne({ _id: userId }).then((user) => {
            if (user && !user.favorites.includes(movieId)) {
                user.favorites.push(movieId);
                user.save();
                return res.status(200).json({
                    success: true,
                    message: "Content successfully added",
                });
            } else {
                return res.status(400).json({
                    success: false,
                    message: "Content already added",
                });
            }
        });
    } catch (error) {
        console.log("Error in addFavorite controller", error.message);
        res.status(500).json({
            success: false,
            message: "Internal server error",
        });
    }
}
```

Рисунок 3.24 – Ендпоінт для додавання в обране

Ми будемо діставати айді фільму з рядка запиту в браузері або будемо передавати в запиті і айді користувача. Потім ми знаходимо користувача з цим

айді, перевіряємо чи немає фільма з цим айді в його обраних і вже потім додаємо.

Подібно до цього реалізуємо ендпоінт для видалення картини з обраного.

```
export async function removeFavorite(req, res) {
    try {
        const { userId, movieId } = req.body;

        if (!userId || !movieId) {
            return res.status(400).json({
                success: false,
                message: "User ID and Movie ID are required",
            });
        }

        const user = await User.findOne({ _id: userId });

        if (!user) {
            return res.status(404).json({
                success: false,
                message: "User not found",
            });
        }

        if (!user.favorites.includes(movieId)) {
            return res.status(400).json({
                success: false,
                message: "Content not found in favorites",
            });
        }

        user.favorites = user.favorites.filter((id) => id !== movieId);

        await user.save();

        return res.status(200).json({
            success: true,
            message: "Content successfully removed",
        });
    } catch (error) {
        console.log("Error in removeFavorite controller", error.message);
        res.status(500).json({
            success: false,
            message: "Internal server error",
```

Рисунок 3.25 – Ендпоінт для видалення з обраного

Подібно до додавання функція також буде приймати айді користувача та фільму, але тепер редагувати масив з обраними і оновлювати в БД.

На цьому моменті ми описали всю потрібну нам логіку та можемо перейти до створення клієнтської частини додатку. Там ми вже побачимо не тільки рядки коду, а й візуальний результат нашої праці.

3.2 Розробка frontend частини

Тепер за допомогою пакувальника Vite створимо React проєкт та розпочнемо його налаштування. Перейдемо в директорію frontend і в терміналі виконаємо команду *npm create vite@latest*.

```
PS C:\Vlad\frontend\netflix_diploma> npm create vite@latest
Need to install the following packages:
create-vite@5.5.3
Ok to proceed? (y)
√ Project name: ... .
√ Select a framework: » React
√ Select a variant: » TypeScript + SWC

Scaffolding project in C:\Vlad\frontend\netflix_diploma...

Done. Now run:

  npm install
  npm run dev

PS C:\Vlad\frontend\netflix_diploma>
```

Рисунок 3.26 – Установка проєкту

Як можна побачити на рис. 3.26 ми виконали цю команду і встановили проєкт з наступними налаштуваннями: фреймворк – React, мова програмування – Typescript. Тепер потрібно встановити залежності командою *npm install*.

Встановимо перші бібліотеки, які нам знадобляться для розробки бекенду командою *npm install sass axios lucide-react react-player react-hot-toast react-router-dom zustand*.

- SASS - це метамова на основі CSS, призначена для збільшення рівня абстракції і спрощення файлів каскадних таблиць стилів. Також нам стануть доступні міксіни, зручне створення змінних та інші функціональні можливості.
- Axios – бібліотека для HTTP запитів.
- Lucide-react – бібліотека з великою кількістю іконок.

- React-player – для зручного ютуб плеєру, в якому повинні бути фільми.
- React-hot-toast – красиві анімовані сповіщення.
- React-router-dom – базова бібліотека для роботи з SPA.
- Zustand – глобальний стейт менеджер.

Розпочнемо з налаштування роутингу нашого додатку. В файлі *main.tsx* треба обернути *<App />* в *<BrowserRouter>*. Після в *<App />* ми можемо описати роути, але для цього нам потрібно відповідні сторінки.

Створимо папку *pages* в головній директорії *src* і в ній окремі папки під кожну сторінку. В проєкті буде використовуватися модульна структура для опису стилів, тому в кожній папці буде знаходитись сам компонент і файл з розширенням *module.scss*.

Також створимо папку components, де будуть зберігатися наші компоненти. Звичайно це все можна не сортувати по папкам, але це погана практика і треба дотримуватися найпростішої архітектури.

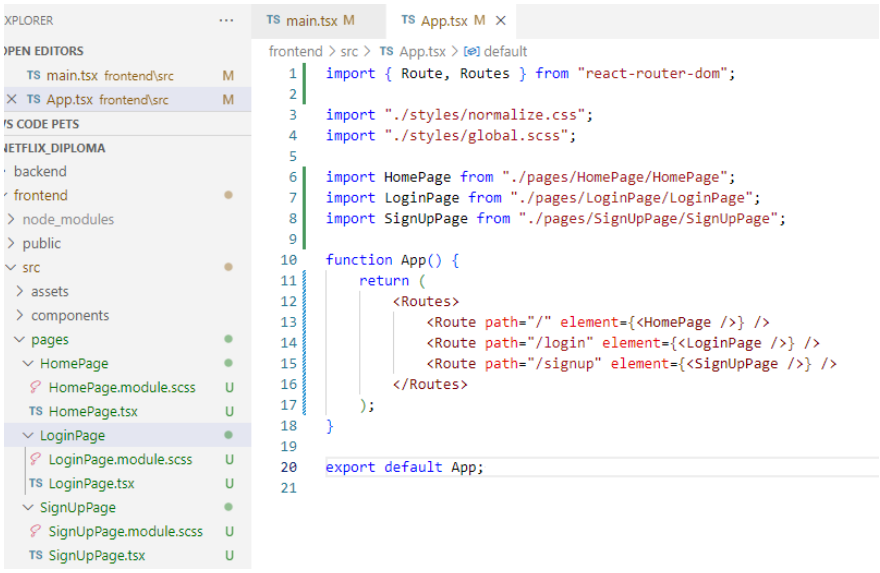

Рисунок 3.27 – Роутинг та дерево директорій

Як ми можем бачити на рис. 3.27, під кожний роут є своя папка зі своїм компонентом. Тепер якщо в браузері перейти по цим ендпоінтам, то будуть відображатися відповідні сторінки.

В папці *public* будуть зберігатися ассети, які знадобляться при розробці. Вони будуть в форматі webp для кращої оптимізації.

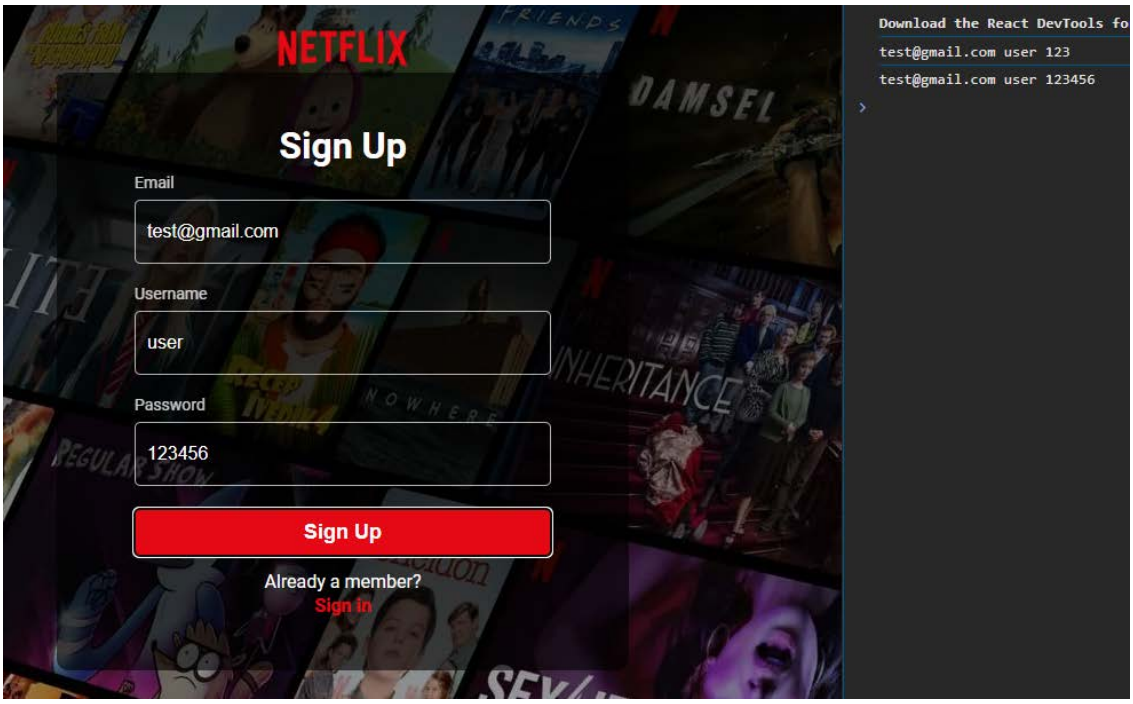

Рисунок 3.28 – Сторінка для реєстрації

На рис. 3.28 реалізовано форму реєстрації з трьома полями для введення інформації. При відправці форми ми отримуємо дані та виводимо в консоль на даний момент.

Сторінка логіну буде виглядати майже так само. Буде тільки 2 поля, це пошта та пароль та трішки інший текст і посилання. Зафіксуємо результат коммітом *git commit –m “feat: added auth pages”*.

Важливо відзначити, що в папці HomePage треба створити ще два екрани, вони будуть показуватися в залежності від того чи авторизований користувач.

```tsx
import { FC, useState } from "react";

import HomeScreen from "./HomeScreen";
import AuthScreen from "./AuthScreen";

const HomePage: FC = () => {
    const [isAuth, setIsAuth] = useState<boolean>(false);
    return isAuth ? <HomeScreen /> : <AuthScreen />;
};

export default HomePage;
```

Рисунок 3.29 – Умовний рендерінг для екранів

Тобто є булева змінна, якщо вона дорівнює істині, то буде відображатися один екран, якщо ні – інший.

<*AuthScreen* /> це просто лендінг, який буде показуватися користувачам, котрі ще не зайшли в свій аккаунт. Він доволі простий, декілька відео, картинок та текст. Для нього був виконаний адаптивний дизайн і він буде відображатися коректно на всіх девайсах.

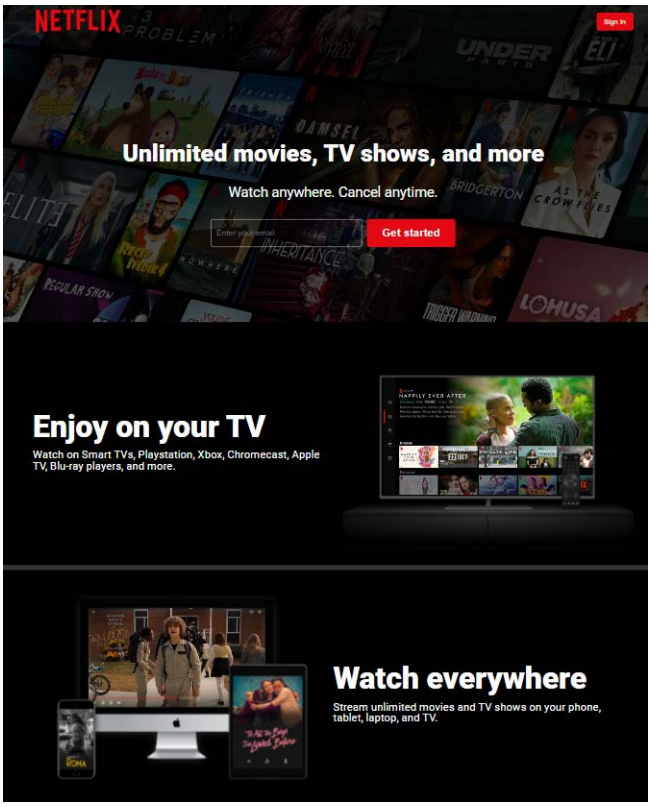

Рисунок 3.30 – Головний екран для неавторизованих користувачів

На рис. 3.30 можна побачити поле для вводу пошти. Зробимо, щоб коли користувач вводить пошту та хоче розпочате реєстрацію, його буде перекидувати на сторінку реєстрації. [24]

```
const [email, setEmail] = useState<string>("");
const navigate = useNavigate();

const handleFormSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    navigate(`/signup?email=${email}`);
};
```

Рисунок 3.31 – Редірект на сторінку реєстрації

Редірект відбувається за допомогою хука useNavigate() і також як параметр ми передаємо пошту користувача, щоб одразу її підставити у відповідне поле на сторінці реєстрації.

Оскільки зараз все готово для тестування реєстрації зі сторони клієнту, то створимо загальний стор. Нагадаю, що це файл з усіма станами додатку, щоб ми могли в будь-якій точці проєкту звератится до них. [25]

```
1  import axios from "axios";
2  import { toast } from "react-hot-toast";
3  import { create } from "zustand";
4
5  interface ICredentials {
6      email: string;
7      username: string;
8      password: string;
9  }
10
11 interface IStore {
12     user: [] | null;
13     isSigningUp: boolean;
14     signup: (credentials: ICredentials) => Promise<void>;
15     login: () => Promise<void>;
16     logout: () => Promise<void>;
17     authCheck: () => Promise<void>;
18 }
19
20 export const useAuthStore = create<IStore>((set) => ({
21     user: null,
22     isSigningUp: false,
23     signup: async (credentials) => {
24         set({ isSigningUp: true });
25         try {
26             const response = await axios.post(
27                 "/api/v1/auth/signup",
28                 credentials
29             );
30             set({ user: response.data.user, isSigningUp: false });
31             toast.success("Account created successfully");
32         } catch (error: any) {
33             toast.error(error.response.data.message || "An error occurred");
34             set({ isSigningUp: false, user: null });
35         }
36     },
```

Рисунок 3.32 – Загальний store додатку

Що ми тут бачимо? Функцією create ми створили стор та описали деякі його поля. Оскільки ми використовуємо Typescript, то потрібно ще додатково описати типи до цих полів. Першою функцією ми реалізували реєстрацію. Для цього в <*SignUpPage* /> викличемо функцію *signup* з нашого стору і передамо туди дані, які вводить користувач. «Зловимо» помилки, якщо вони є і за допомогою бібліотеки toast зробимо гарні сповіщення.

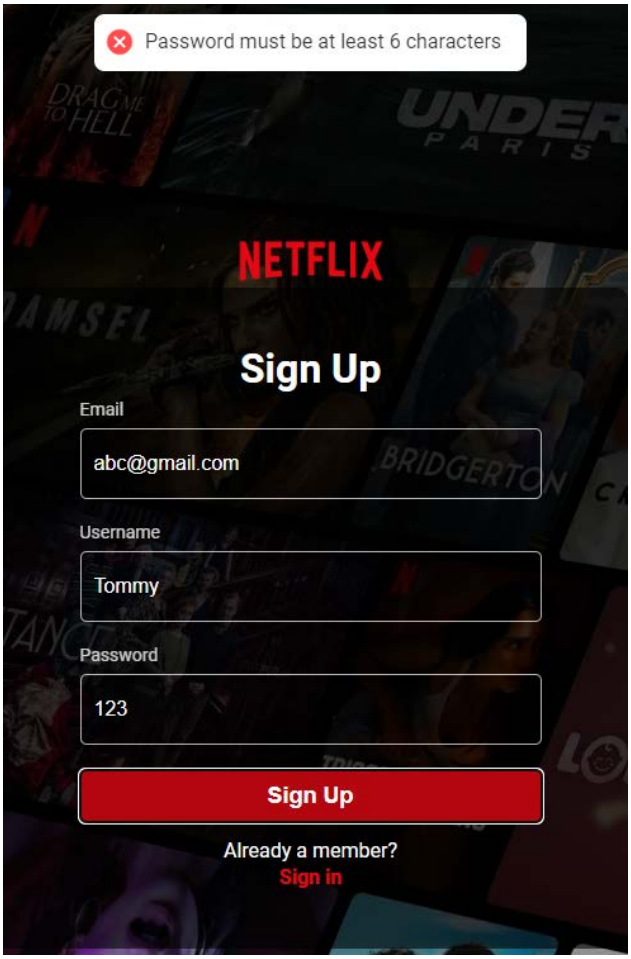

Рисунок 3.33 – Спроба невдалої реєстрації

Ми бачимо помилку, оскільки ми описали на бекенді, що пароль повинен бути мінімум 6 символів. Це бекенд нам повернув і ми гарно сповістили користувача.

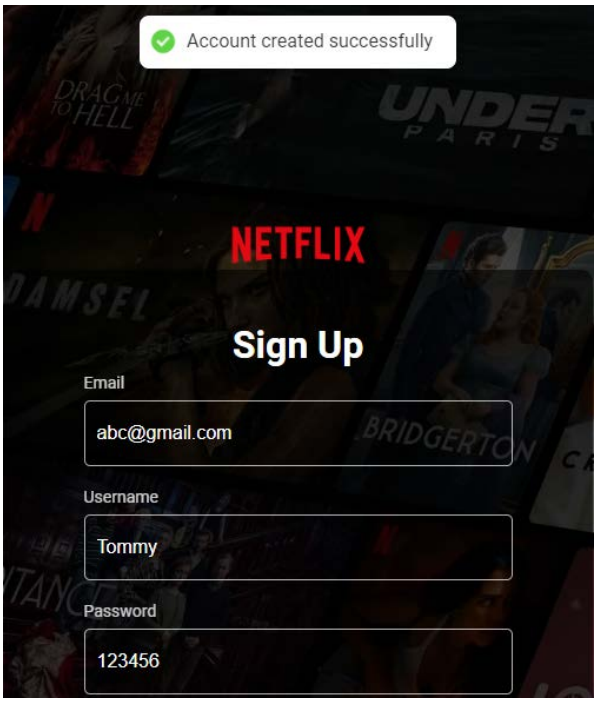

Рисунок 3.34 – Спроба вдалої реєстрації

Сервер повернув нам повідомлення, що аккаунт успішно створено і ми сповістили про це користувача.

Тепер на фронтенді напишемо функції для запиту на ендпоінт authCheck і будемо її викликати при завантаженні додатку.

```
40      authCheck: async () => {
41          set({ isCheckingAuth: true });
42
43          try {
44              const response = await axios.get("/api/v1/auth/authCheck");
45              set({ user: response.data.user, isCheckingAuth: false });
46          } catch (error) {
47              set({ isCheckingAuth: false, user: null });
48          }
49      },
50  }));
```

Рисунок 3.35 – Функція запиту до ендпоінту

Тепер кожний раз коли користувач заходить на наш сайт буде відбуватися перевірка чи є користувач та чи є в нього права доступу до певних сторінок нашого сайту.

```
import { useAuthStore } from "../../store/authUser";

const HomePage: FC = () => {
    const { user } = useAuthStore();
    return user ? <HomeScreen /> : <AuthScreen />;
};
```

Рисунок 3.36 – Перевірка логіну

Тепер ми можемо показувати користувачу відповідний екран в залежності від того авторизований він чи ні.

```
        <Route path="/" element={<HomePage />} />
        <Route
            path="/login"
            element={!user ? <LoginPage /> : <Navigate to={"/"} />}
        />
        <Route
            path="/signup"
            element={!user ? <SignUpPage /> : <Navigate to={"/"} />}
        />
    </Routes>
```

Рисунок 3.37 – Рефактор роутів

На рис. 3.37 зміни, які додали логіку, якщо користувач вже авторизований, то йому не треба доступи на сторінки реєстрації та авторизації. Ми будемо його посилати на головну сторінку. Якщо ж він захоче вийти з аккаунту, то ці сторінки стануть йому доступними.

```
41    logout: async () => {
42        set({ isLoggingOut: true });
43        try {
44            await axios.post("/api/v1/auth/logout");
45            set({ user: null, isLoggingOut: false });
46            toast.success("Logged out successfully");
47        } catch (error: any) {
48            set({ isLoggingOut: false });
49            toast.error(error.response.data.message || "Logout failed");
50        }
51    },
```

Рисунок 3.38 – Функція виходу з аккаунту

Напишемо функцію логауту, щоб на сторінці *<HomeScreen />* (яка ще не побудована) ми повісили її на кнопку і змогли використовувати її логіку. Закоммітимо – *git commit –m “feat: added logout logic”*.

Реєстрація є, вихід з аккаунту є, залишилось реалізувати функціонал логіну в свій аккаунт. Напишемо просту функцію для відправки пошти та паролю та «повісимо» її на кнопку логіну на відповідній сторінці.

```
login: async (credentials) => {
    set({ isLoggingIn: true });
    try {
        const response = await axios.post(
            "/api/v1/auth/login",
            credentials
        );
        set({ user: response.data.user, isLoggingIn: false });
        toast.success("Logged in successfully");
    } catch (error: any) {
        toast.error(error.response.data.message || "An error occurred");
        set({ isLoggingIn: false, user: null });
    }
},
```

Рисунок 3.39 – Фунція логіну в аккаунт

Весь функціонал авторизації та реєстрації ми успішно реалізували, тому тепер можемо перейти до створення різних сторінок для авторизованих користувачів. Не забудьмо закоммітити результат – *git commit –m “feat: added login logic”*.

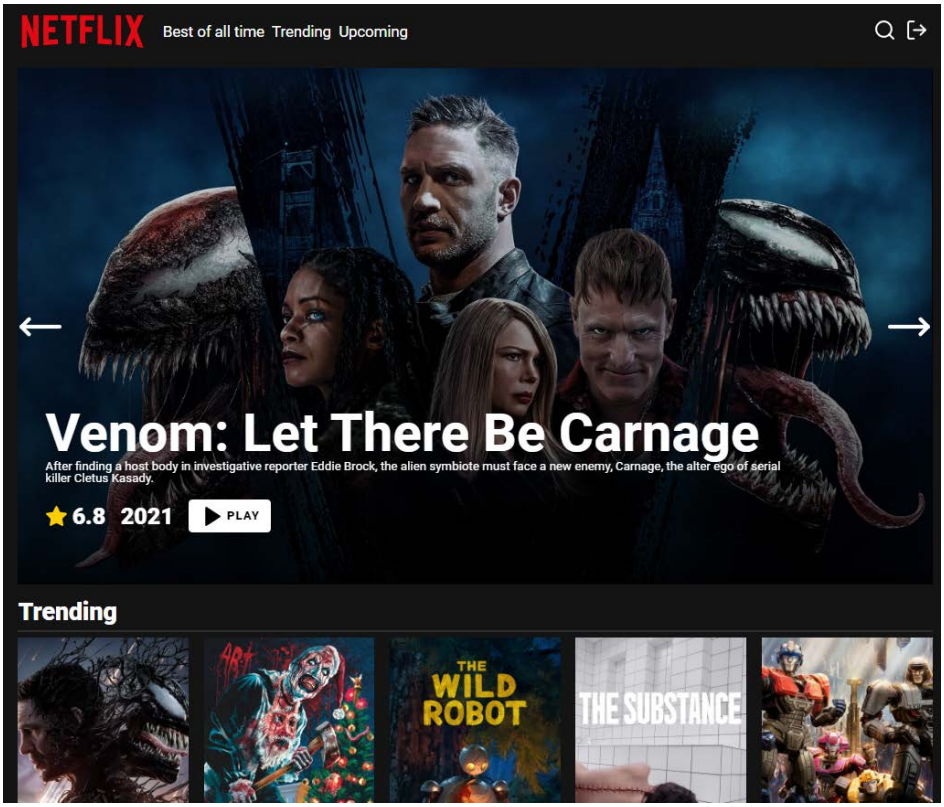

Рисунок 3.40 – Головна сторінка для авторизованого користувача

Такий вигляд має сторінка для авторизованого користувача. Одразу його зустрічає слайдер з фільмами, які нещодавно вийшли у прокат. Після нього йде три секції з трендовими фільмами, кращими за весь час та які скоро повинні вийти.

Всі фільми в цих секціях є посиланнями на окрему сторінку з цим фільмом. Ми передаємо в пошукову стрічку айді фільму і на сторінці фільму виконуємо інший запит на деталі фільму по айді, який дістаємо хуком *useParams()* зі стрічки.

Рисунок 3.41 – Hover ефект

При наведенні на фільм, постер та інформація буде трішки збільшуватись і буде з’являтися іконка плеєру.

Рисунок 3.42 – Сторінка з фільмом

На сторінці з фільмом ми будемо бачити більше інформації про нього (рік випуску, жанри, дату релізу, який хронометраж фільму та короткий опис). Під основною інформацією вбудований плеєр, на якому можна подивитися трейлер фільму.

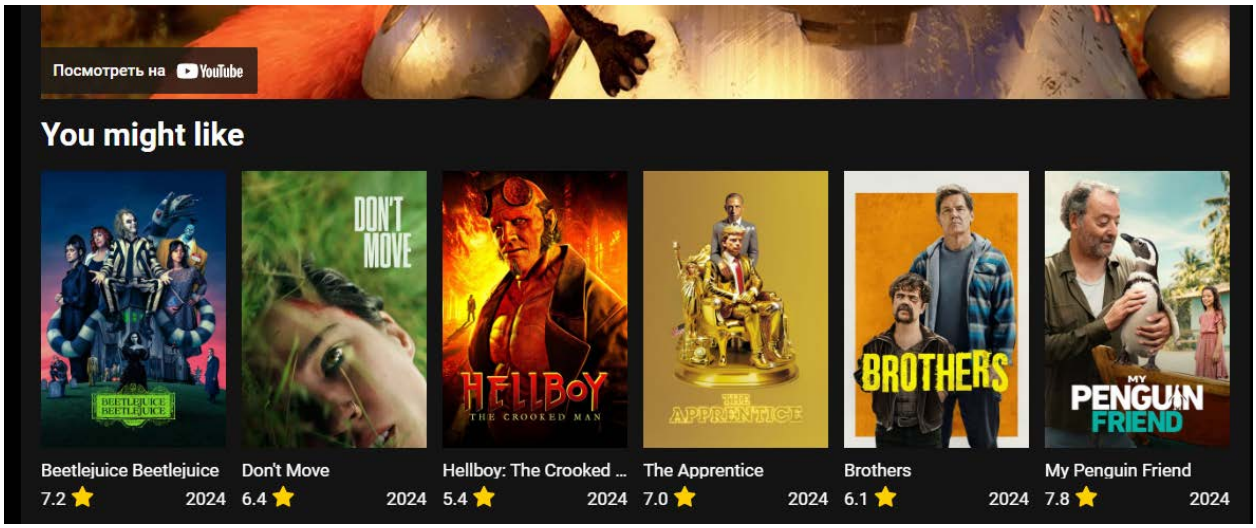

Рисунок 3.43 – Слайдер зі схожими фільмами

Під кожним фільмом або серіалом є рекомендації для користувача зі схожими картинами.

```
getRecommendedMovies: async (id) => {
    try {
        const response = await axios.get(
            `https://api.themoviedb.org/3/movie/${id}/recommendat
        );
        set({ recommendedMovies: response.data.results });
    } catch (error: any) {
        toast.error("An error occured, try again later");
    }
},
```

Рисунок 3.44 – Запит на рекомендовані картини

Фунція приймає айді, який ми дістаємо з параметрів сторінки і виконуємо запит на сервери TMDB. Нам він відповідає масивом фільмів, який ми вже відмальовуємо на сайті. В цілому всі картини ми отримуємо подібними запитами і вже далі маніпулюємо даними.

Але поки запит обробляється і нам повертаються дані, проходять секунди при повільному інтернеті. Аби користувач знав що відбувається процес завантаження даних додамо скелетон. Це така заглушка, яка буде відображатися замість картинки.

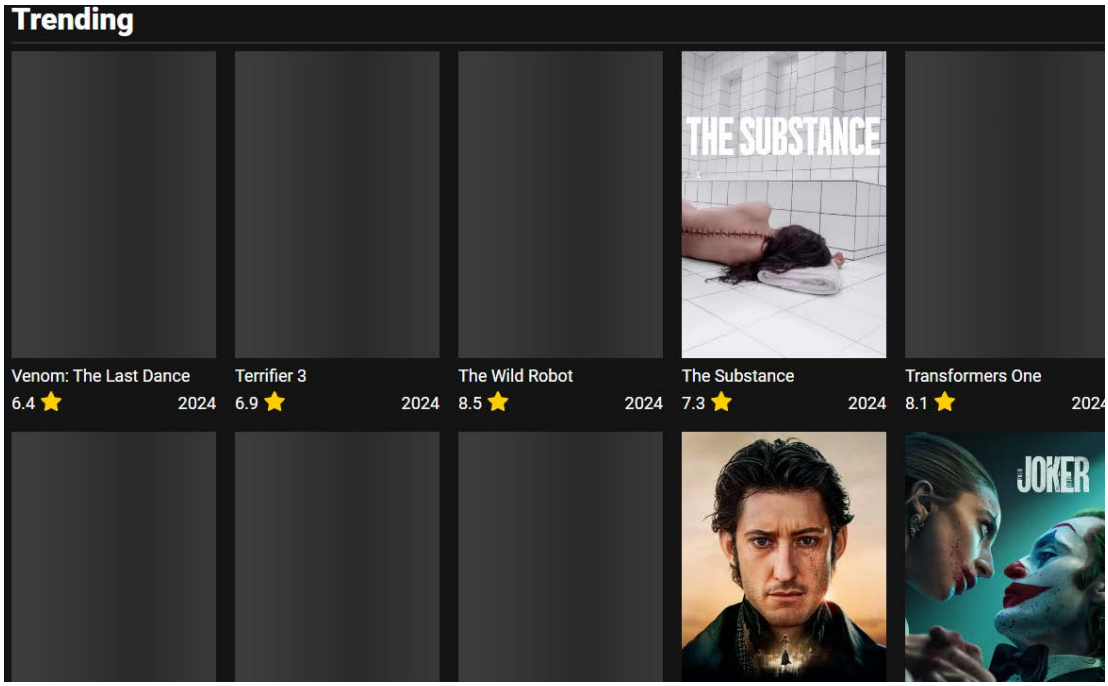

Рисунок 3.45 – Скелетон замість незавантажених постерів

Тепер користувач розуміє, що відбувається підвантаження даних, а не сайт зламався.

frontend > src > components > Skeleton > Skeleton.tsx > default

```tsx
import { FC } from "react";

import styles from "./Skeleton.module.scss";

interface ISkeleton {
    maxWidth: string;
    ratio: string;
}

const Skeleton: FC<ISkeleton> = ({ maxWidth, ratio }) => {
    return (
        <div
            className={styles.skeleton}
            style={{ maxWidth: `${maxWidth}px`, aspectRatio: `${ratio}` }}
        ></div>
    );
};

export default Skeleton;

```

Рисунок 3.46 – Код скелетону

Скелетон приймає декілька вхідних даних(пропсів) і за рахунок цього його можна підлаштувати під будь-який контент. Тобто цей компонент став універсальним, що і є основною ідеєю бібліотеки React.

Рисунок 3.47 – Додавання в обране(клієнтська частина)

На сторінці з фільмом додамо блок з кнопкою на яку повісимо логіку додавання фільму в обране. На відповідний ендпоінт відправляємо айді фільму

і в базі даних айді додається до масиву. Вже потім ці айді нам знадобляться для того, щоб відмалювати потрібні нам фільмі в особистому кабінеті.

Повернемося до головної сторінки, на даний момент ми отримуємо за замовчуванням тільки першу сторінку в якій є 20 фільмів. Але ж ми можемо робити запити на різні сторінки. Для цього існує пагінація. Тобто ми обираємо сторінку і отримуємо якісь нові дані.

Напишемо компонент для цього, будемо приймати сторінку, яку натиснув користувач і виконувати новий запит.

```
const totalPages = 50;

const getDisplayedPages = () => {
    const visiblePages = 5;
    const half = Math.floor(visiblePages / 2);
    let start = Math.max(1, currentPage - half);
    let end = Math.min(totalPages, currentPage + half);

    if (currentPage <= half) {
        end = Math.min(totalPages, visiblePages);
    } else if (currentPage + half >= totalPages) {
        start = Math.max(1, totalPages - visiblePages + 1);
    }

    return Array.from({ length: end - start + 1 }, (_, i) => start + i);
};

const displayedPages = getDisplayedPages();

const handleClick = (e: React.MouseEvent<HTMLButtonElement>) => {
    onPageChange(+e.currentTarget.value);
    scroller.scrollTo(scrollTo, {
        smooth: true,
        duration: 1000,
    });
};

return (
    <div className={styles.pagination}>
        {displayedPages.map((page) => (
            <button
                key={page}
                value={page}
                onClick={handleClick}
                className={page === currentPage ? styles.active : undefined}
            >
                {page}
            </button>
        ))}
    </div>
);
```

Рисунок 3.48 – Компонент для пагінації

А на головній сторінці в нас є залежність, що якщо змінюється сторінка, то виконуємо новий запит. Сторінку ми змінюємо відповідно в цьому компоненті. Також можна побачити, що номера сторінок динамічно змінюються в залежності від обраної сторінки.

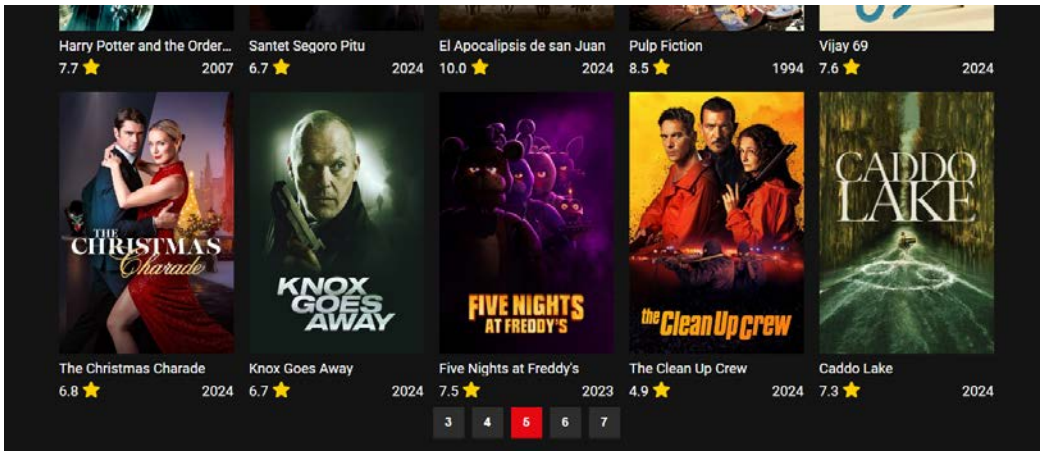

Рисунок 3.49 – Пагінація на головній сторінці

Але як же нам шукати фільми, якщо ми знаємо назву і не хочемо гортати нескінечно всі сторінки? Треба розробити сторінку пошуку, де користувач будет вводити частину або всю назву фільма та буде отримувати результат пошуку.

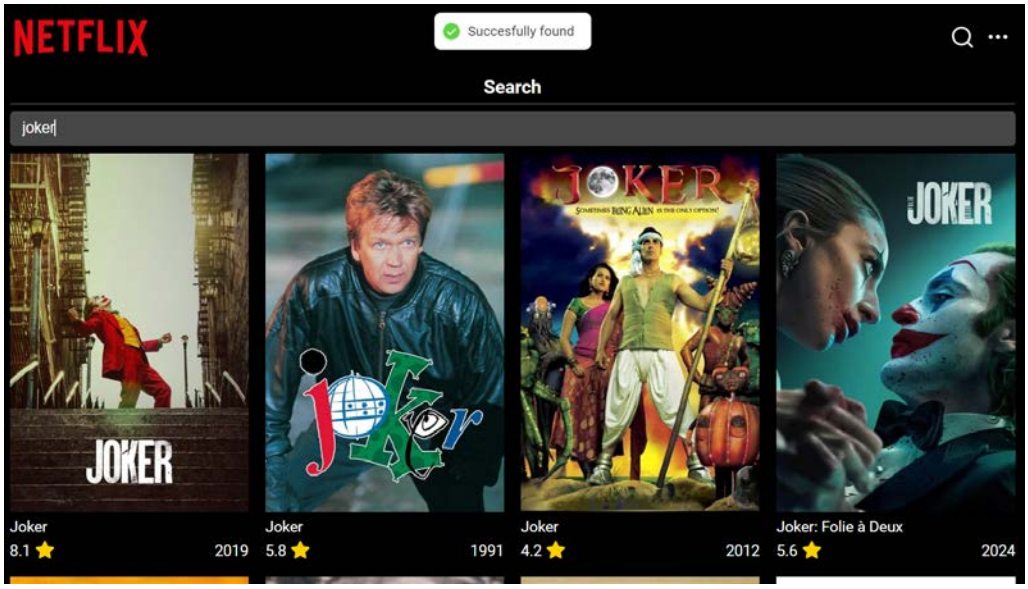

Рисунок 3.50 – Результат пошуку

Для реалізації пошуку нам потрібна допоміжна функція-хук *useDebounce*. Вона буде відкладати запит до API на секунду, щоб не робити запит на кожну букву, яку користувач вводить в поле пошуку. Тобто він введе якісь дані і тільки після цього виконається запит. Це зроблено з метою кращої оптимізації та користувацького досвіду.

```
frontend > src > hooks > useDebounce.tsx > ...
import { useEffect, useState } from "react";

function useDebounce<T>(value: T, delay: number): { debouncedValue: T } {
    const [debouncedValue, setDebouncedValue] = useState(value);

    useEffect(() => {
        if (typeof value === "string" && value.trim().length === 0) {
            setDebouncedValue(value);
            return;
        }

        const handler = setTimeout(() => {
            setDebouncedValue(value);
        }, delay);

        return () => {
            clearTimeout(handler);
        };
    }, [value, delay]);

    return { debouncedValue };
```

Рисунок 3.51 – Функція useDebounce

Після того, як знайшли потрібний нам фільм і додали в обране, можемо перейти до особистого кабінету і побачити його там.

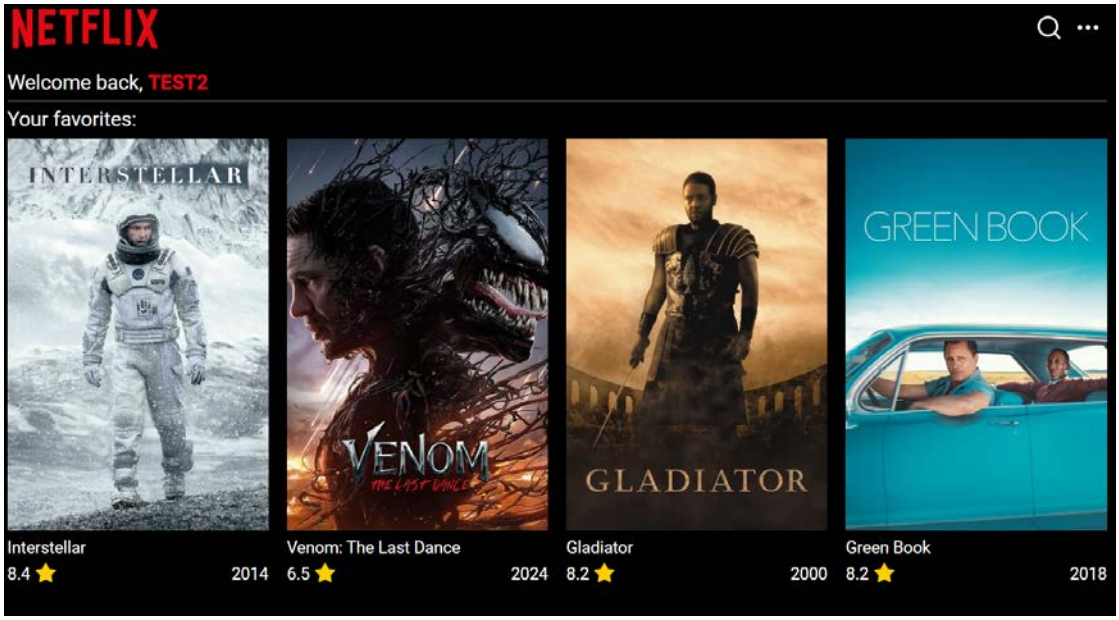

Рисунок 3.52 – Особистий кабінет

В ньому ми бачимо напис «Welcome back» та юзернейм, який користувач вказав при реєстрації. Ну і відповідно всі його фільми, які він додав в обране.

Напишемо функціонал для видалення з обраного. Створимо в сторі нову функцію для POST-запиту на видалення фільму.

```
    removeFromFavorite: async (userId, movieId) => {
        try {
            const response = await axios.post("/api/v1/auth/removefavorite", {
                userId: userId,
                movieId: movieId,
            });
            toast.success(response.data.message);
        } catch (error: any) {
            toast.error(error.response.data.message);
        }
    },
}));
```

Рисунок 3.53 – Функція видалення з обраного

В компоненті зі сторінкою <*MoviePage* /> у нас є 2 обробника кліку. Якщо фільм вже доданий, то буде кнопка видалення з логікою видалення *handleRemoveFavorite*, якщо ж не доданий, то кнопка додавання з *handleAddFavorite*.

```
const handleAddMovie = () => {
    if (user?._id && movieId) {
        addToFavorite(user._id, movieId);
        setIsMovieAdded(true);
    }
};

const handleRemoveMovie = () => {
    if (user?._id && movieId) {
        removeFromFavorite(user._id, movieId);
        setIsMovieAdded(false);
    }
};
```

Рисунок 3.54 – Обробники кліків на кнопки

Висновок до третього розділу

На цьому етапі можна сказати, що весь потрібний функціонал на клієнтській частині реалізований і ми обробили всі ендпоінти, які описували на бекенді. Як результат, ми маємо повноцінний full-stack додаток з авторизацією, різними маніпуляціями з БД і не тільки. Фронтенд частина вийшла гарною з приємним дизайном та користувацьким досвідом. Були використані сучасні технології та практики роботи з ними.

ВИСНОВКИ

У ході виконання магістерської роботи було досліджено основні аспекти аналізу популярності стрімінгових сервісів, зокрема Netflix, що дозволило визначити ключові фактори, які впливають на формування глядацьких уподобань. На основі отриманих даних було розроблено практичний підхід до аналізу популярності кінострічок, реалізований за допомогою мови програмування Python. Результати аналізу були візуалізовані у вигляді графіків, що сприяло кращому розумінню тенденцій переглядів та їхньої динаміки.

У рамках практичної частини також створено повнофукнціональний веб-сервіс для пошуку та збереження кінострічок у персоналізовану бібліотеку, орієнтований на вподобання користувача. Для розробки було використано сучані технології, такі як React і Typescript для клієнтської частини, Express для сервеної логіки та MongoDB як базу даних. У процесі розробки реалізовано функції аутентифікації користувачів, динамічного пошуку, збереження та видалення з обраного, перегляду обраних кінострічок.

Досягнуті результати підтверджують, що запропонований підхід забезпечує користувачам можливість ефективно взаємодіяти з великою кількістю фільмів, створюючи персоналізовані добірки на основі інтересів. Розроблений сервіс є гнучким і масштабованим, що дозволяє легко адаптувати його для подальшого розширення функціоналу або інтеграції з іншими платформами.

Усі поставлені цілі для досягнення результату в магістерській роботі були успішно виконані.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Чубукова І.А. Data Mining: навч. посіб. М.: Інтернет-університет информаційних технологий БІНОМ: Лабораторія знань, 2006. 382 с.

2. Дюк В. Data Mining: навч. курс (+CD)/.Дюк В., Самойленко А. Вид-во Київ, 2001. 368 с.

3. Knowledge Discovery Through Data Mining: What Is Knowledge Discovery, Tandem Computers Inc., 1996, 253 p. --

4. Кречетов Н. Продукти для інтелектуального аналізу даних // *Ринок програмних засобів*, N14-15_97. 1997. с. 32–39.

5. Кісєльов М., Є, Соломатин: Засоби видобутку знань у бізнесі та фінансах // *Відкриті системи*. 1997. № 4. с. 41–44.

6. Data Mining and Image Processing Toolkits. URL http://datamining.itsc.uah.edu/adam/. (Дата звернення 15.09.2024)

7. Ф. Барсегян, М. Купріянов, В. Степаненко, І. Холод.: Методи і моделі аналізу даних OLAP и DataMining / 2008. 354 с.

8. Kitchin, Rob; McArdle, Gavin «What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets». URL: https://journals.sagepub.com/doi/10.1177/2053951716631130. (Дата звернення 18.09.2024)

9. Big Data’s Fourth V. URL: https://spotlessdata.com/blog/big-data-fourth-v. (Дата звернення 19.09.2024)

10. Wes McKinney (2011). «Pandas: a foundation python library for Data Analysis and Statistics».

11. Data Mining Curriculum: A Proposal (Version 1.0). URL: https://www.kdd.org/exploration_files/CURMay06.pdf. (Дата звернення 05.10.2024)

12. В. Є. Бахрушин / Методи аналізу даних: навчальний посібник для студентів – Запоріжжя: КПУ, 2011. – 268 с. ISBN 978-966-414-103-8.

13. The Future of Big Data? Three Use Cases of Prescriptive Analytics. URL: https://vanrijmenam.nl/future-big-data-cases-prescriptive-analytics/. (Дата звернення 11.10.2024)

14. Diagnostic. URL: https://www.gartner.com/en/information-technology/glossary/diagnostic-analytics.

15. Netflx and Big Data How big data became important to Netflix. URL: https://www.academia.edu/35644610/Netflix_and_Big_Data. (Дата звернення 12.10.2024)

16. Data preprocessing in detail. URL: https://developer.ibm.com/articles/data-preprocessing-in-detail/. (Дата звернення 15.10.2024)

17. Ситник В.Ф., Краснюк М. Т. Інтелектуальний аналіз даних (дейтамайнінг): Навч. посібник: КНЕУ, 2007. 376 с.

18. Hilary L. Seal (1967). «The historical development of the Gauss linear model».

19. Optimally splitting cases for training and testing high dimensional classifiers. URL: https://brb.nci.nih.gov/techreport/Dobbin-SampleSplitting.pdf. (Дата звернення 23.10.2024)

20. Data analysis for Omie Sciences: Methods and Applications, Volume 82 1st Edition Joaquim Jaumot Carmen Bedia Roma Tauler, USA, 2018, ISBN: 9780444640451, 740 pages.

21. Reactjs ReactDom. URL: https://legacy.reactjs.org/docs/react-dom.html. (Дата звернення 04.11.2024)

22. Vite Official Documentation. URL: https://vite.dev/guide. (Дата звернення 05.11.2024)

23. The starting point for learning Typescript. URL: https://www.typescriptlang.org/. (Дата звернення 12.11.2024)

24. Wayback Machine – Internet Archive. URL: https://wayback.archive.org/. (Дата звернення 13.11.2024)

25. Zustand Documentation. URL: https://zustand.docs.pmnd.rs/. (Дата звернення 13.11.2024)

26. Тронь В. В., Маринич І. А. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 174 - «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». Кривий Ріг : Видавничий центр КНУ, 2022, 50 с.

27. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

28. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

29. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

30. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).