МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія проєктування системи відео на замовлення»
здобувача групи ІН.м - 23 Голуба Івана Олеговича

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

Іван ГОЛУБ
(підпис)

Керівник,
старший викладач,
кандидат фізико-математичних наук
Оксана ШОВКОПЛЯС
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-23 Голуба Івана Олеговича

1. Тема роботи: «Інформаційна технологія проєктування системи відео на замовлення»
затверджую наказом по СумДУ від *«06» грудня 2023 року № 1412-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 18 друдня 2023 року*
3. Вхідні дані до кваліфікаційної роботи_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для проектування системи відео на замовлення. 3) Розроблення системи відео на замовлення. 4) Аналіз результатів. 5) Оформлення пояснювальної записки до кваліфікаційної роботи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.11-12.11.23	
2	*Огляд технологій, що використовуються для проектування системи відео на замовлення*	13.11-19.11.23	
3	*Розроблення системи відео на замовлення*	20.11-03.12.23	
4	*Аналіз отриманих результатів*	04.11-08.12.23	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	09.11-17.12.23	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 90 стр., 41 рис., 1 таб., 1 додаток, 27 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи на сьогоднішній день є дуже актуальною, оскільки потокове відео в прямому ефірі та відео на вимогу наразі знаходиться на вершині пошукових запитів в мережі Інтернет, та наразі відео трафік становить більше ніж 80% від усього споживаного Інтернет трафіку в усьому світі. Проведений аналіз шляхом розробки відповідної інформаційної системи відповідає на питання правильного вибору протоколів потокового зв'язку з адаптивним бітрейтом та використання розробленої стримінгової платформи для пошуку та доступу до інформації стосовно популярних фільмів та сералів.

Об'єкт дослідження — процес проєктування системи відео на замовлення з аналізом протоколів потокового зв'язку.

Мета роботи — створення інформаційної технології проектування системи відео на замовлення та проведення аналізу стосовно правильного вибору протоколів потокового зв'язку з адаптивним бітрейтом. Платформа запропонує велику бібліотеку фільмів і серіалів, що дозволяє користувачам отримувати доступ до широкого спектру розважального вмісту за запитом, та використовувати пошуковий механізм для конкретних видів запитів.

Методи дослідження — аналіз на порівняння динамічне адаптивних потокових протоколів, прийняття рішень по їх відповідному використанню.

Результати — створено інформаційну технологію проектування системи відео на замовлення. Виконана робота включала пошук необхідної інформації відповідно до теми роботи, використання та аналіз роботи світових

стрімінгових платформ, проведення аналізу протоколів потокового зв’язку з адаптивним бітрейтом, обрані інфраструктурні програмні рішення, плагіни, та інтерфейси програмування для написання додатку, використання новітніх технологій Front-End розробки, такі як JavaScript, React.JS, React Router, Styled Components, SASS/CSS, HTML, Rest API, розгортання додатку та його тестування, аналіз додатку за допомогою сторонніх сервісів.

ІНФОРМАЦІЙНА СИСТЕМА, СТРІМІНГ, HLS, JAVASCRIPT, MPEG-DASH, REACT, REACT ROUTER, STYLED COMPONENTS, VOD, VIDEO ON DEMAND

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи. На сьогоднішній день світові сервіси потокового передавання відео, які використовують передплати, або ж безкоштовні сервіси з використанням реклами, значно розвиваються. Також, на сьогодні технологічно доставка контенту дуже вдосконалилась. До останніх досягнень можна віднести покращену якість відео - HDR та 4K, набагато кращі алгоритми потокового передавання відео та вдосконалені інтерфейси користувача, так званий ‘user experience’, сприяють покращенню передавання потокового зв’язку з адаптивним бітрейтом. Використання функцій онлайн відео-трансляцій та відео по запиту (VOD), набули популярності не лише в телевізорах, але й стали інтегрованими в різноманітні смарт-пристрої, включаючи ігрові консолі, смарт-телевізори (STB), потокові медіа плеєри, та мобільні телефони, що робить доступ користувачів до їх вмісту на різних діагоналях екранів та різних за розмірами пристроях більш доступнішими та зручнішим. Перераховані позитивні аспекти спонукали обрати дану тему для розробки стримінгової платформи.

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки на сьогодні відео трафік в мережі Інтернет становить більше ніж 80% від усього споживаного Інтернет трафіку в усьому світі. Прогрес у технології потокового передавання відео контенту та вдосконалення алгоритмів стиснення відео сприяє зростанню попиту на послуги стримінгових платформ.

Об’єкт дослідження - процес проєктування системи відео на замовлення з аналізом протоколів потокового зв’язку.

Предмет дослідження – методи і моделі автоматизованої інформаційної системи відео на замовлення.

Гіпотеза. Стримінговий додаток з сучасним дизайном інтерфесу, автоматизованим користувацьким досвідом та відповідним протоколом передачі відео контенту, дозволить реалізувати максимально зручний концепт платформи з великою базою останніх новинок кінематографу, для отримання

глядачами максимального задоволення від користування додатком та перегляду відео контенту.

Новизна. Описане у даній роботі програмне рішення дозволить створити простий та доступний стримінговий додаток для користування будь-яких верств населення з максимальним комфортом користування та спростити пошук та перегляд відео контенту.

Структура. Дана робота складається з аналітичного огляду стримінгових платформ та конкурентів, аналізу протоколів потокової передачі даних з адаптивним бітрейтом, обрання найбільш відповідного протоколу для розв’язання задачі. Також була обрана мова програмування, фреймворк, середовище розробки та плагіни для написання платформи, виконана практична реалізація додатка та проведений аналіз отриманих даних.

Зв'язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп’ютерних наук та пов’язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Огляд і аналіз предметної області проектування

Потокове відео в прямому ефірі та відео на вимогу швидко піднялося в на вершину найпопулярніших пошукових запитів Інтернету в останні роки. До 2022 року – ‘відео’ вже становило 82% від усього споживаного Інтернет-трафіку в усьому світі[1]. Можна з впевненістю сказати, що причини цього зрозумілі – проаналізувавши пристрої на яких відтворювалось та наразі відтворюється відео контент, ми побачимо, що громадяни використовують не тільки смарт-телевізори, а також мобільні пристрої, планшети, персональні комп’ютери та ігрові консолі, які наразі широко доступні та дуже потужні, а відео контент є надзвичайно привабливим засобом, який може передавати інформацію швидше та ефективніше, ніж інші формати вмісту.

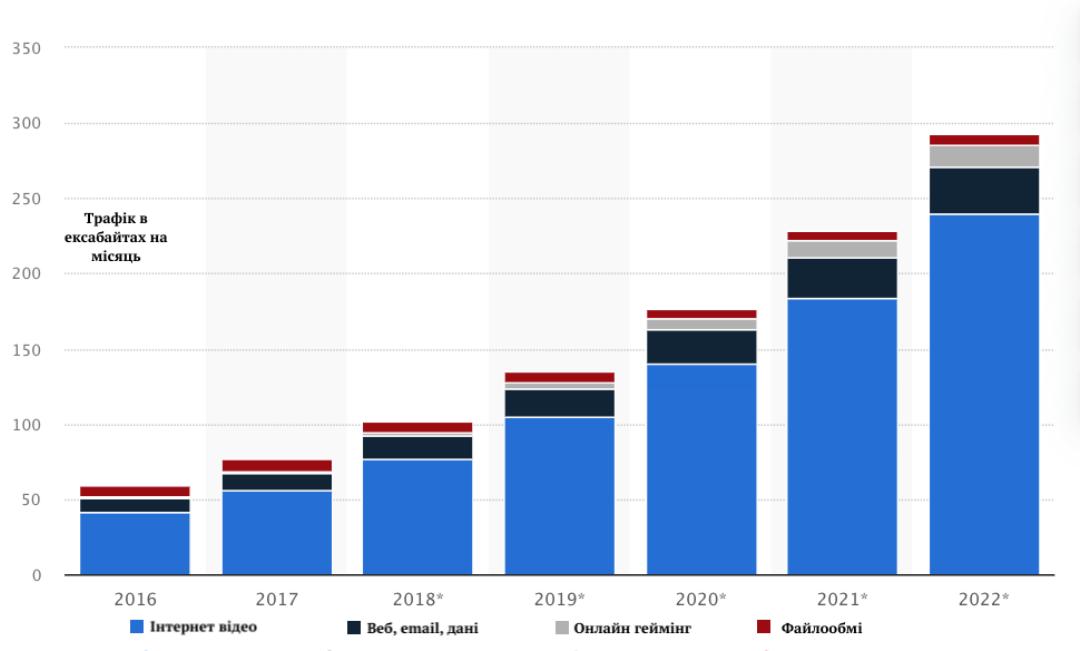

Рисунок 1.1– Зростання ринку стрімінгових послуг 2016 – 2022 роки[1].

Можна з впевненістю сказати, що глядачі тільки виграють від використання послуг «Відео на вимогу» (VOD) у порівнянні з аналоговим телебачення, це сприяє поширенню популярності та більш широкому застосуванню цієї платформи.

Ми можемо побачити певні переваги використання VOD:

- Зручність перегляду та адаптивність відео потоку – ‘відео на вимогу’ дозволяє глядачам дивитися програму, де та коли завгодно, усуваючи необхідність дотримуватися розкладу трансляції.
- Клієнти платформ мають можливість переглядати свої улюблені серіали та фільми у будь який час та де завгодно.
- Платформи з використанням систем ‘відео на вимогу’ загалом надають величезний і різноманітний вибір контенту, наприклад останні новинки кінематографу та бестселери, популярні серіали, документальні або фантастичні фільми, шоу програми, тощо.

Однак, незважаючи на свою доступність та кількість плюсів, потокове відео по запиту не позбавлене мінусів та невеликих труднощів в питаннях розробки. До найбільш поширених можна віднести буферизацію – для відтворення відео з гарною якістю та без затримок потрібна гарна швидкість Інтернету, то ж за її відсутності відтворення відео може відбуватися з затримками або зупинятися. Також до найрозповсюдженіших проблем відноситься перевантаження мережі Інтернет, або збої на стрімінгових сервера потокової передачі, що можуть призвести до перегляду відео в поганій якості, або навіть повній зупинці відтворення контенту. Останній показник, але не менш важливий - це затримка потокового відео в реальному часі. Ця затримка може бути спричинена різноманітними факторами, зокрема перевантаженням Інтернет мережі, вибором неправильного протоколу та процесом кодування або декодування відео.

Не зважаючи на всі вищеперераховані труднощі, можна з впевненістю сказати, що при стабільному Інтернет з'єднанні - швидкість, зручність, доступність та якість відтворення відео контенту являє собою те, що на сьогодні є беззаперечним лідером сучасного індивідуального доставлення абонентові телевізійних програм і фільмів.

Можна також зазначити величезну конкуренцію серед стрімінгових платформ, кожна з яких має величезну кількість переваг порівняно з іншими. До найбільш популярних світових платформ можна віднести Netflix – перевагами якого є величезна доступна бібліотека фільмів, телешоу, та унікального контенту. Створення власноруч різноманітного контенту, відсутність реклами та чудовий UI/UX.

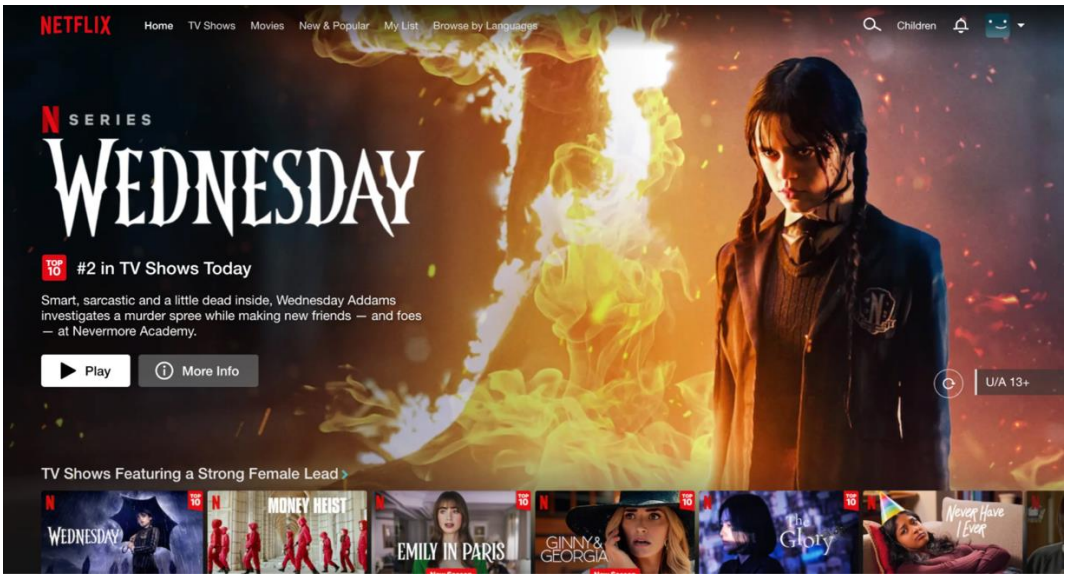

Рисунок 1.2– Головний екран Netflix

Стримінгова платформа YouTube Premium – користувачі даного додатку можуть дивитися відео контент без жодної реклами, також наявність фонового відтворення дозволяє користувачам переглядати відео під час використання інших програм у фоновому режимі, наявність чудової служби підтримки. Також одним із лідерів ринку є Apple TV Plus – вироблення оригінального матеріалу, інтеграція між усіма наявними платформами Apple, потокове передавання відео контенту без перерви на рекламу для своїх підписників. Також Apple створила власну технологію потокової передачі з регульованим бітрейтом, відому як HTTP Live Streaming (HLS). Відтворення відео контенту з даним протоколом тепер не залежить від певної платформи, однак спочатку його використовували для трансляції відеоконтенту в прямому ефірі та у відео на вимогу через HTTP на пристрої iOS, Apple TV і ПК з macOS.

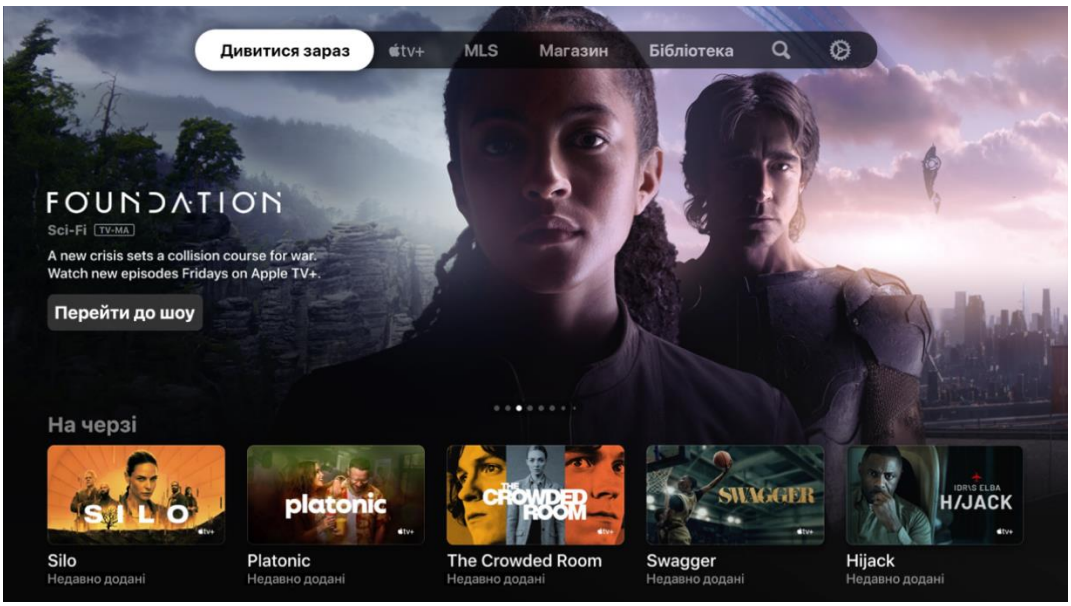

Рисунок 1.3– Головний екран Apple TV

Як не згадати відчизняні стрімінгові платформи такі як MEGOGO, Sweet TV, Київстар ТБ – лідери послуг з ‘відео на вимогу’ та ‘прямих трансляцій’ в Україні. Дані стрімінговою платформо також передають відео контент, такий як відео на вимогу та відео у реальному часі через інтернет використовуючи протоколи потокового зв’язку з адаптивним бітрейтом.

1.2 Огляд стримінгових протоколів потокового зв’язку та технології адаптивної трансляції потокового відео

Провівши аналіз стримінгових сервісів, концептів розробки та архітектурних підходів, обрання конкретних стримінгових протоколів серед різних платформ - можна стверджувати, що саме відео в наш буремний час змушує світ обертатися. Однак, якщо ігнорувати труднощі пов’язані з відтворенням потокового відео, такі як затримка (Latency), буферизація (Buffering), погана якість відео (Poor video quality) – даний бізнес з використання концепту відео на вимогу (VOD) може ніколи не стати прибутковим. Технологія Adaptive Bitrate Streaming (ABR) та доступні протоколи – HLS та MPEG-DASH змогли набагато полегшити роботу компаніям і стримінговим платформам подоланням можливих перешкод пов’язані з наданням потокового відео контенту.

Дуже важливо розуміти різниці у потокових протоколах, основні закони та правила розробки, інструкції та технологічні рамки, які контролюють адаптивну передачу відео контенту через мережу Інтернет, у режимі реального часу або відео на вимогу. При обранні стримінгового протоколу для трансляції відео дуже важливо розуміти ключові відмінності між MPEG-DASH і HLS, який з цих протоколів найкраще підійде для конкретної платформи, розуміти ключову концепцію Adaptive Bitrate Streaming (ABR) - технологію, яку самі ці протоколи намагаються реалізувати ‘під капотом’ та стандартизувати своїми унікальними способами.

1.2.1 Технологія адаптивної трансляції потокового відео (Adaptive bitrate streaming)

Важливо розуміти, що адаптивний бітрейт потокового передавання (ABR) автоматично змінює бітрейт потокового передавання, щоб забезпечити найкращий можливий досвід перегляду шляхом виявлення в режимі реального часу стану мережі глядача та конкретного девайса на якому відтворюється відео контент. Трансляції потокового передавання набирають популярності порівняно з прогресивним потоковим відео, оскільки воно може змінювати якість поточного відео в режимі реального часу в залежності від швидкості, стану мережі Інтернет та пропускної здатності. Завдяки цій особливості під час перегляду відео контенту якість може автоматично підстраюватись, та ймовірність зупинки чи буферизації відео зменшується.

Маючи різну швидкістю Інтернету метод адаптивної трансляції потокового відео поділяє стрім на маленькі частинки, кодує кожну з них з різним бітрейтом, а програвач, тобто відео плеєр на стороні клієнта перевіряє якість мережі та запитує оптимальні варіанти якості від сервера, що забезпечує безперебійне відтворення відео контенту без жодної буферизації та з мінімальною кількістю затримок, або навіть взагалі без них.

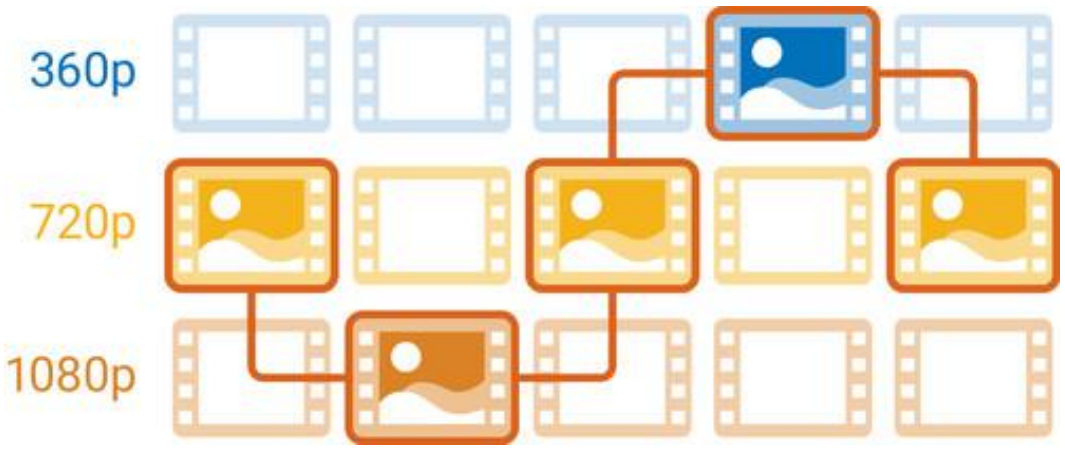

Рисунок 1.4 – Зміна якості відео в залежності від особливостей стану мережі Інтернет та пропускної здатності

Також варто розуміти концепцію алгоритмів для вибору бітрейту в адаптивному потоковому відео [20]. Існує 2 основних алгоритми – це пропускна здатність (Throughput) та алгоритм на основі буфера (Buffer-based). Для того, щоб вибрати правильний бітрейт для відео стрімінгу, алгоритм на основі пропускної здатності використовує швидкість мережевого з'єднання. Юзер отримає набагато якісніший відео контент, якщо швидкість мережі Інтернет буде достатньо швидкою. Для того, щоб була можливість гарантувати безперебійне відтворення відео контенту без затримок в разі погіршення швидкості Інтернету, або повільного з'єднання з мережею з самого початку відтворення, буде подано відео контент (стрім) нижчої якості. В свою чергу, алгоритм на основі буфера вираховує найбільш правильний бітрейт для відео стрімінга, використовуючи конкретний рівень буфера у відео плеєра юзера. Якщо буфер може бути заповненим без наявних пауз у буферизації, то реально отримати відео стрім набагато вищої якості. Відео контент нижчої якості може бути використаний, якщо буфер відео стрімінгу низький для того щоб уникнути затримок під час відтворення.

1.3 Адаптивний протокол потокової передачі даних HLS – HTTP Live Streaming

Щоб краще розуміти різницю між адаптивними протоколами передачі даних, розглянемо кожен з них окремо та проведемо аналіз відмінностей та переваг.

Компанією Apple було створено технологію потокової передачі з регульованим (адаптивним) бітрейтом, відому як HTTP Live Streaming (HLS). Використання даного протоколу наразі взагалі не залежить від конкретної платформи на якій відтворюється контент, але треба розуміти, що на самому початку імплементації цей протокол використовувався для потокової передачі відео контенту - прямих відео трансляцій (Live TV) та відео на вимогу (VOD) через HTTP на всі наявні пристрої в лінійці Apple, такі як комп'ютери з операційною системою macOS, девайси з IOS (iPad, iPhone) та Apple TV. Концепція роботи даного протоколу – це розподілення відео стрімінгу на керовані фрагменти за допомогою HLS. Наступним кроком створюються велика кількість версій кожного маленького сегмента. Надалі кожен маленький сегмент стрімінгу кодується з окремим наявним бітрейтом для певної роздільної здатності [2].

Завдяки цьому дана технологія повністю підтверджує свою документацію, що реальний юзер який дивиться відео контент даного протоколу отримує найкращу можливу якість відео контенту. Та головною перевагою даної технології є те, що на льоту (on the fly) відео плеєр на стороні реального юзера в залежності від швидкості мережі Інтернет, наявного девайса та пропускної здатності може підстроюватись до обставин та обирати найкращу якість з можливих, або підтримувати наявну та не змінюючи її.

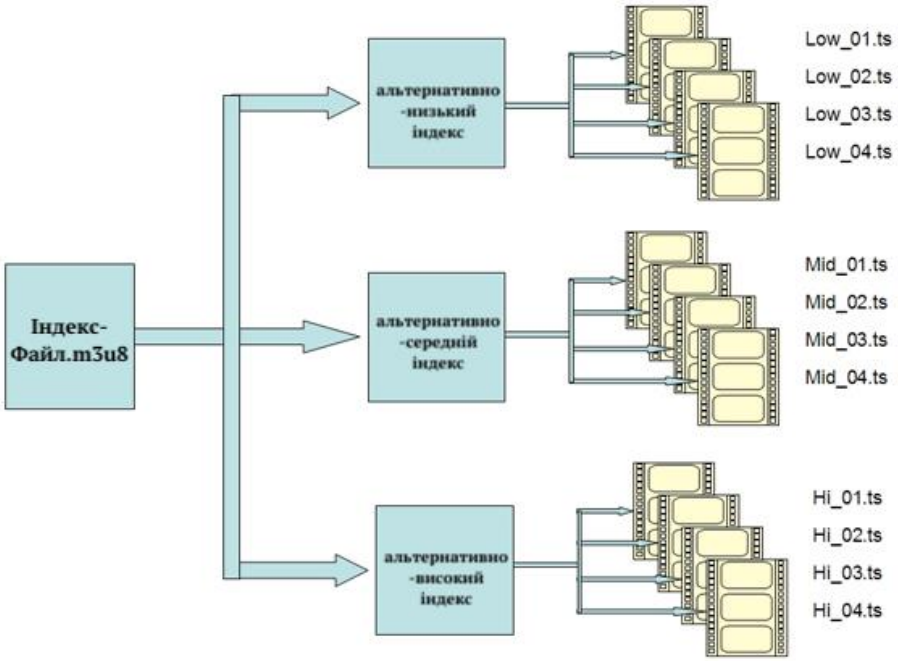

Рисунок 1.5 – Використання HLS кількох закодованих файлів, індексні файли використовуються для вказівки програвача на різні потоки та окремі сегменти аудіо/відео даних у цих потоках

Розглядаючи переваги даного протоколу, треба виділити найбільші з них:

- Підтримка даного протоколу майже усіма наявними девайсами у світі. Дана технологія HTTP Live Streaming підтримується на нативному рівні майже в усіх сучасних девайсах. Ми можем з впевненістю сказати що кожен веб-браузерер з Media Source Extensions API підтримує дану концепцію. Також наразі кожен медіа плеер базований на HTML5 (Castlabs playeer, Shaka player, та інші), будь якій девайс Apple (ПК з macOS, Apple TV, iPod, iPhone, iPad), будь який смарт телевізор або мобільний браузер має підтримку даного стримінгового протоколу.
- За допомогою Content delivery network (CDN) протокол HTTP Live Streaming може бути масштабований та кешований. Це можливо завдяки тому, що дани протокол бере за використання сервери з HTTP, завдяки чому можливе спрощення інтеграції з веб-інфраструктурою яка використовується. До значної переваги можна віднести також

можливість стримінгових платформ та клієнтів-виробників доставляти відео контент до своїх глядачів у форматі якої відбувався запис, або ж у форматі прямої трансляції підтримуючи якість відео контенту на найвищому рівні.

- Також величезною перевагою використання протоколу HTTP Live Streaming є величезна кількість ‘фічей’ (Features) які дозволяються масштабувати дану технологію. До особливостей масштабування можна віднести наявність кількох доріжок субтитрів та аудіо, можливість шифрування, наявність аутентифікації – що дає змогу легкого користування зі сторони кінцевих юзерів та полегшенох розробки зі сторони девелоперів. Завдяким цим особоливостям навіть вразливі категорії населення можуть з легкістю користуватися відео контентом використовуючи або ж субтитри, або ж вибір мови яка їм більше підходить.

При значних перевагах користування даним стрімінговим протоколом він має також і невеликі недоліки, які треба розглянути:

- Як ми знаємо для кодування відео контенту використовуються відео кодеки, і нажаль в даному протоколі є обмеження по їх використанню. Є величезна кількість кодеків які можуть бути використані для кодування або декодування відео контенту – але HLS протокол дещо обмежений в їх кількості. З документації можна отримати інформацію, що Apple HLS має стандарти які є дуже строгими та мають бути використані тільки деякі з кодеків та форматів [2]:
  1) Для кодування відео - H.264 та H.265
  2) Для кодування аудіо - AC3, E-AC3 MP3 або HE-AAC, AAC.

Ми можемо побачити зі статей та документації, що компанія Apple вже намагається і буде розвивати оптимізацію даного протоколу, щоб в майбутньому уникнути будь яких обмежень та мати підтримку усіх можливих девайсів, операційних систем, кодеків, форматів, тощо.

На основі вищезазначеної інформації використання даного протокол потокової передачі може бути використано в таких випадках, як:

- Розробка програми для багатьох платформ, особливо девайсів Apple, в яких на нативном рівні йде підтримка протоколу HTTP Live Streaming. Тобто використання платформи для відео контенту робить використання цього протоколу найбільш оптимізованим рішенням при розробці веб сайтів з VOD, включаючи додатки для девайсів Apple з комерційною ціллю.
- Розробка та використання додатків з концептом відео на вимогу (VOD). Компанією Apple була розроблена ідея не звертати увагу на затримку відео контенту і намагатися вплинути на неї, а навпаки, була розробка, яка використовує даний протокол для визначення приорітетності численних потоків із багатьма параметрами якості відео контенту. Даний концепт ідеально призначений для відтворення відео на вимогу, тому що відео плеєр може адаптуватися для швидкості Інтернету, девайсу та пропускній здатності і віддавати найкращу якість глядачу.

1.4 Адаптивний протокол потокової передачі даних MPEG – DASH

Розглядаючи адаптивний протокол потокової передачі даних MPEG-DASH (Dynamic Adaptive Streaming over HTTP) – який був розроблений групою експертів з рухомих зображень (MPEG - Moving Picture Experts Group), ми можемо з впевненістю сказати, що він розроблений подібно до інших концептів адаптивної потокової передачі. MPEG-DASH також ділить відео стрім на керовані фрагменти для передачі по комп'ютерних мережах [1]. Спільно з цим, MPEG-DASH все ж таки відрізняється за своєю реалізацією:

- Даний протокол здатен працювати з будь яким кодеком. При використанні розширених відкритих кодеків, можна отримати набагато кращу якість відео контенту (приклади кодеків: WMV, VP9). Маючи таку гарну особливість, вона призводить і до зворотнього ефекту – в цьому випадку MPEG-DASH дозволяє мати лише один потік із певним

бітрейтом, хоча в свою чергу Http Live Sreaming дозволяє мати кілька потоків із різними бітрейтами для певної роздільної здатності.

- Кодування передачі відео фрагментів, яке використовує даний адаптивний протокол, дає змогу доставляти сегменти юзеру з надзвичайною швидкістю, завдяки тому що як тільки вони стають доступними йде їх передача, замість того, щоб чекати, поки весь сегмент буде закодований та упакований перед його доставкою. Крім того, даний протокол взагалі не вказує, яка кількість сегментів повинна знаходитись в черзі, щоб розпочалося відтворення. У результаті відбувається набагато менша затримка відтворення відео контенту [25].

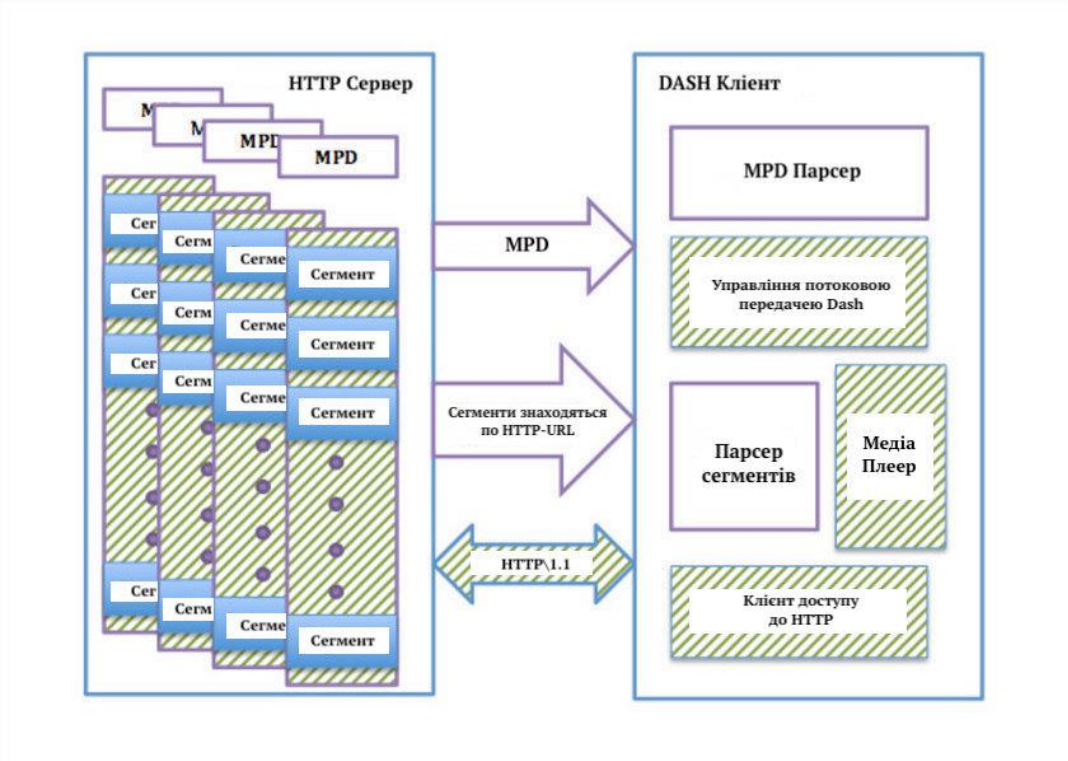

Рисунок 1.6 – Високорівнева клієнт-серверна архітектура MPEG-DASH з наглядним розподілення сегментів

В порівнянні з адаптивним протоколом HTTP Live Streaming, даний протокол має як свої переваги, так і свої недоліки, тому повинен бути використаний з розумінням всіх своїх особливостей і під свої конкретні задачі та цілі.

На основі вищезазначеної інформації ми розуміємо що використання динамічного адаптивного стримінгу дозволяє полегшити розробку та використовувати будь які кодеки для кодування та декодування аудіо та відео доріжок – так як цей протокол є ‘агностіком’ до кодеків. Це дозволяє ‘бізнесу’ доставляти відео контент в вищій якості і з більшою швидкістю. Звісно ми відносимо це до плюсів використання даного протоколу, як і те що даний протокол ідеально підходить для стримінгу з низькою затримкою (low latency). В порівнянні з HTTP Live Streaming, даний протокол використовує кодування передачі фрагментів не накладаючи будь яких суворих вимог до мінімального розміру сегмента який повинен бути, то ж він значно більше підходить для стримінгу з низькою затримкою. Також варто відзначити що динамічний адаптивний стримінг є повністю безкоштовним, то ж це може зменшити витрати при веденні комерційного бізнесу [4].

Але як і з HTTP Live Streaming - не уникнути недоліків при користуванні DASH стрімінгом, то ж розглянемо деякі з них. Незважаючи на те, що динамічний адаптивний стрімінг був розроблений як нейтральний щодо платформи на якій він використовується , все ж таки він досі є несумісним із усіма пристроями компанії Apple. Також варто згадати, що для відтворення відео стрімінгу використовуючи даний концепт будуть потрібні сторонні відео програвачі на основі HTML5, а також підтримка специфікація W3C під назвою Media Source Extensions, яка дозволяє JavaScript надсилати потоки байтів до медіа-кодеків у відео плеєрах [20]. Варто звернути увагу і на відсутність стандартів - відкритий динамічний адаптивний протокол є як і знахідкою та і причиною ‘болю’ з обох боків. Даний протокол не передбачає жодних витрат на ліцензування так як є ‘open-source’ протоколом, але також не містить чітких вказівок щодо створення сегментів, яка повинна бути довжина цих сегментів та це ж стосується будь-яких інших вимог. Свобода вибору кожного постачальника контенту створювати власний досвід користування даним протоколом дозволяє налаштовувати різноманітні рішення, але також створює

сумнівний досвід для нових користувачів та розробників, які починають користуватися даним концептом. Додатково до перерахованих мінусів користування даним протоколом можна згадати швидкість передачі данних. Динамічний адаптивний стрімінг створює значну затримку, коли використовується як протокол прийому – цей цикл передбачає збір і надсилання потоку даних на сервер. Можливо б було удосконалити даний підхід використовуючи WebRTC або ж RTMP для кодування прямого відео стрімінгу, потім надсилати його на сервер, та останнім пунктом використовувати його для доставки глядачам відео контенту [1].

То ж на основі вищезазначеної інформації використання даного динамічного протоколу потокової передачі може бути використано в таких випадках, як:

- Для відтворення прямих відео трансляції буде ідеальним кейсом використання даного протоколу.

- Користування даним протоколом компаніями, які надають послуги потокового відео. Адаптивне потокове передавання — це доволі поширений концепт, який використовують найбільші стримінгові компанії світу, такі як Netflix та Amazon Prime Video, YouTube та Hulu. То ж десятки мільйонів людей у світі можуть підтвердити якість наданого відео контенту.

- Також ідеальним кейсом буде використання даної технології у відеоспостереженні. Даний стримінговий протокол MPEG-DASH дає змогу в режимі реального часу, з мінімальною затримкою та адаптивною якістю передавати пряму відеотрансляцію з камер відео нагляду через мережу Інтернет. Тому ця технологія буде чудовим вибором для компаній які займаються спостереженням за безпекою.

1.5 Аналіз та порівняння протоколів потокового зв’язку HLS та MPEG-DASH

Розглянувши обидва адаптивні стримінгові протоколи, можна з впевненістю сказати що вони досягають однієї мети – доставка медіаконтенту

до кінцевого користувача через мережу Інтернет, але є досить помітні відмінності в їхніх методах і техніках, та важливо обрати правильний протокол при розробці додатку. То ж розглянемо всі основні відмінності між MPEG-DASH та HLS [10]:

Мною була створена дана порівняльна таблиця з характеристикою та аналізом відповідних протоколів потокового зв'язку (табл. 1.1).

Таблиця 1.1 Характеристика, аналіз та порівняння протоколів потокового зв'язку HLS та MPEG-DASH

Критерії	HLS (HTTP Live Streaming)	MPEG-DASH (Dynamic Adaptive Streaming over HTTP)
Підтримка кодеків	HLS суворо відповідає специфікаціям підтримки кодеків, для аудіо підтримуючи лише AC3, HE-AAC, AAC, MP3 та E-AC3, та для відео лише H.264 і H.265.	Немає жодних обмежень. Наразі є підтримка усіх можливих аудіо та відео кодеків, такі як E-AC3, AAC, MP3, AC3, Opus, Vorbis, , H.264, H.265, AV1, VP9 та інших.
Сумісність з девайсами	Підтримка майже на усіх мобільних пристроях(не тільки Apple). Також підтримка майже на кожній платформі та браузерах (з підтримкою Media Source Extensions API)	Підтримка на кожній платформі та браузерах (з підтримкою Media Source Extensions API) за винятком пристроїв на базі iOS, браузері Safari, та Apple TV
Ліцензія	Власний протокол потокової передачі який був створений Apple (не був офіційно	Відкритий стандарт розроблений групою MPEG і який було стандартизовано

	стандартизований за межами даної специфікації)	відповідно до ISO/IEC 23009-1:2012
Затримка відео потоку	Регульована та чітко встановлена довжина сегмента для HLS спричиняє збільшення затримки. Помітні повільніші оновлення сегментів, що збільшує час до першого кадру та може спричиняти затримку прямих трансляцій від 16 до 20 секунд. Кожен сегмент повинен завантажуватися щонайменше кожні 6 секунд.	Використовуються менші сегменти за довжиною: від 1 до 5 секунд, це зменшує затримки під час відтворення та показу першого кадру. Також DASH використовує фрагментоване кодування передачі відео потоку. Це дає змогу передавати частини сегментів без мінімальних обмежень очікування та як тільки вони стають доступними.
Сегментація для ABR (Adaptive bitrate streaming)	З визначеним часом сегмента 6 секунд для відео стримінгу та буфером із 3 сегментів - HLS використовує для своїх сегментів стандарт MPEG-2 Transport Stream або MPEG-4 Part 14.	DASH забезпечує швидшу та плавну потокову передачу з меншою затримкою та більш частими оновленнями сегментів. DASH використовує MPEG-4 Part 30 для своїх сегментів зі змінною тривалістю сегмента (2-10 секунд, без буфера)
DRM (Digital Rights Management)	Використовується технологія FairPlay, яка є	MPEG-DASH підтримує велику кількість DRM - таких як Google Widevine,

	ексклюзивною для продуктів Apple.	Adobe Primetime і Microsoft PlayReady, але за виключенням FairPlay.

Незважаючи на те, що HLS та MPEG-DASH є широко використовуваними адаптивними протоколами потокового відео, вони відрізняються між собою функціями адаптивної потокової передачі відео контенту, сумісністю з браузерами, розміром сегмента та джерелами [3]. Цільова аудиторія, тобто глядачі, види девайсів на яких відтворюється відео стрім, та бажаний рівень контролю над потоковим передаванням відео стрімів – це ті параметри, які впливають на рішення який протокол обрати для розробки.

1.5.1 Аналітика відео потоку та порівняння метрик відеопрогравача з використанням протоколів HLS та MPEG-DASH

Спочатку був проведений аналіз протоколу потокового зв’язку MPEG - DASH. Використовуючи офіційний довідковий клієнт (платформу) DASH Industry Forum – Reference Client версії 4.7.3, було завантажено та відтворено відео стрім протоколу MPEG-DASH.

Було проаналізовано п’ять основних характеристик після відтворення перших 45 секунд відео:

- Довжина буфера (Buffer Length) - 65.143
- Завантаження бітрейту (Bitrate Downloading) – 14932 kbps
- Затримка (Latency) (мін|сер|макс) : 0.00 | 0.00 | 0.00
- Завантаження (Download) (мін|сер|макс): 0.1 | 0.1 | 0.1
- Коефіцієнт (Ratio) (мін|сер|макс): 571.43 | 666.67 | 800.00

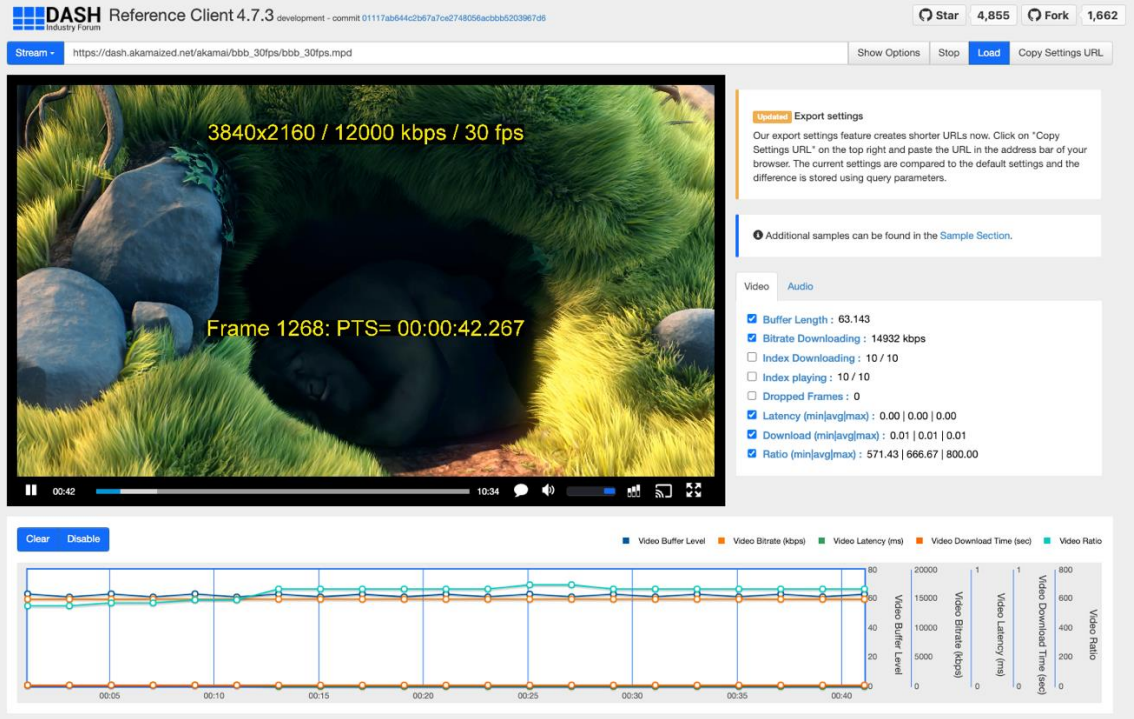

Рисунок 1.7 – Аналітичні метрики відтворення MPEG-DASH протоколу.

Варто зазначити, що стрім був статичний (тобто такий же як при використання відео на замовлення). Перше що ми зразу можемо помітити з аналізу, що затримка відео (Latency) – взагалі відсутня та дорівнює 0.00. Також неймовірний результат завантаження бітрейту (Bitrate Downloading) який дорівнює 14932 kbps. Якщо перевести це числа в більш зрозумілі мегабіти в секунду то отримаємо 14.58 Mbps – що являє собою величезною швидкістю передачі даних. Довжина буфера (Buffer Length) - 65.143 та Завантаження (Download) - 0.1 | 0.1 | 0.1 є надзвичайно гарними показниками відео відтворення. Ці чудові показники були досягнуті використанням менших сегменти за довжиною: від 1 до 5 секунд, що зменшило затримку (в нашому випадку затримка взагалі відсутня) під час відтворення та показу першого кадру відео стріма. Використання DASH фрагментованого кодування передачі відео потоку дало змогу передавати частинки сегментів без будь яких обмежень очікування. Як тільки сегменти стали доступними вони зразу почали відтворювались, що також ми можемо бачити в результатах отриманих вище.

Як висновок можна з впевненістю сказати, що протокол MPEG-DASH як і заявлено в документації - ідеально підходить як для статичних так і для прямих відео трансляцій, відео відтворюється з мінімальними затримками та в чудовій якості.

Аналізуючи протокол потокового зв'язку HLS – було використано платформу Bitmovin, в якій було завантажено та відтворено статичний відео стрім протоколу HLS. Проаналізувавши також перші 45 секунд відео, маємо відповідні проаналізовані характеристики:

- Довжина буфера (Buffer Length) - 46.5 (в середньому)
- Завантаження бітрейту (Bitrate Downloading) – 2519 kbps
- Затримка (Latency) (мін|сер|макс) : 0.01 | 0.015 | 0.02

Аналізуючи дані показники можемо відзначити що швидкість завантаження бітрейту (Bitrate Downloading) – 2519 kbps, перевівши дане число в мегабіти в секунду отримаємо 2.45 Mbps, що менше ніж на 10 Mbps від 15.58 Mbps, які ми отримали у MPEG-DASH. Також довжина буфера (Buffer Length) становила 46.5, що доволі гарний результат, але порівнюючи з довжиною буфера MPEG-DASH на рівні 65.14, маємо на 19 одиниць менше значення. Затримка була майже відсутня, в середньому становила 0.015. Але також, порівнюючи з протоколом DASH – в вищезазначених показниках вона взагалі була відсутня.

Як висновок можна зазначити, що у конкретному випадку, використовуючи протокол MPEG-DASH маємо набагато кращі показники та характеристики, більшу довжину буфера, швидший бітрейт та меншу затримку порівняно з використанням протоколу HLS. То ж як висновок, у випадку проєктування та розробки системи відео на замовлення буде використана техніка потокового передавання з адаптивним бітрейтом MPEG-DASH.

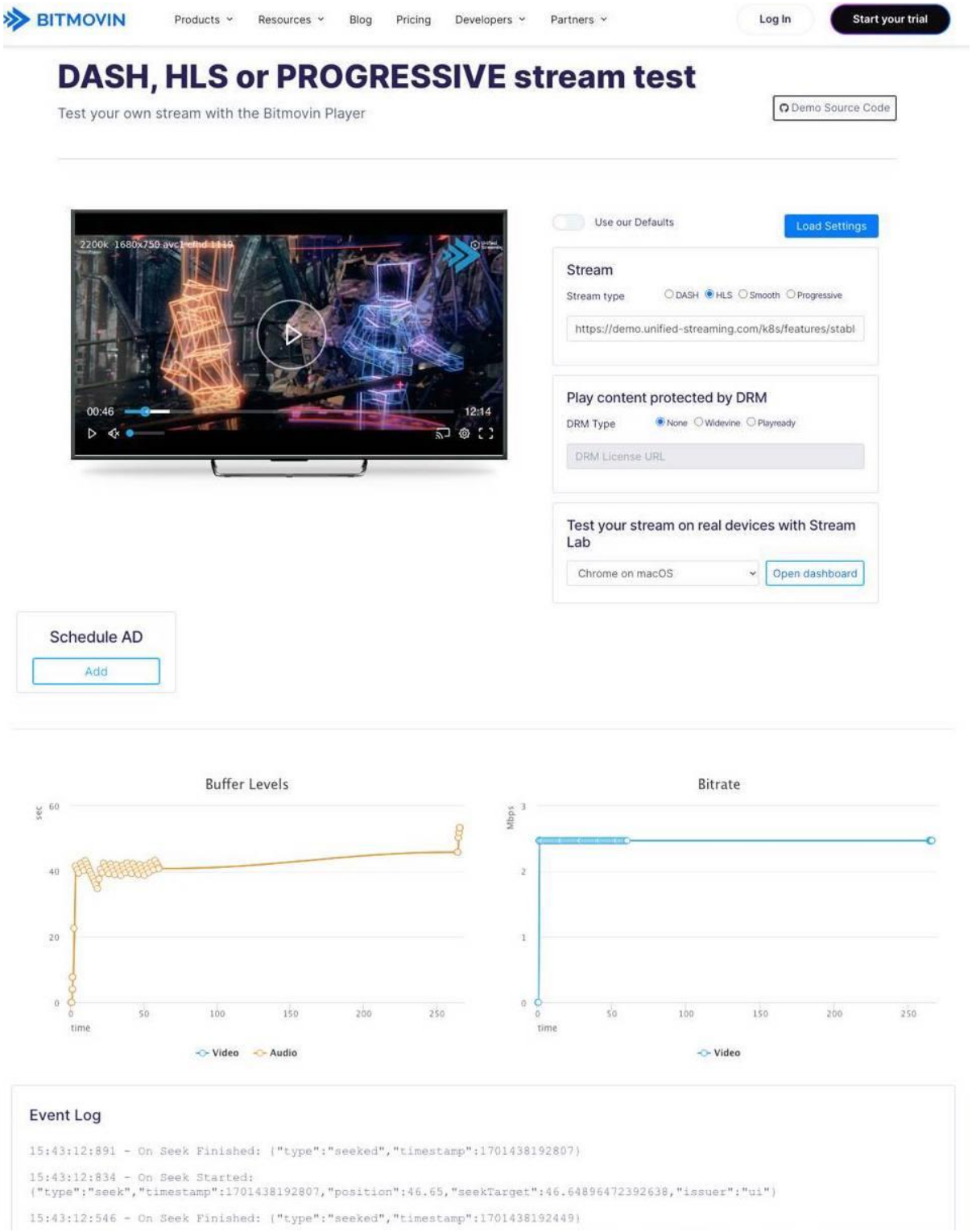

Рисунок 1.8 – Аналітичні метрики відтворення HLS протоколу.

1.6 Постановка задачі

Метою даної дипломної роботи є створення інформаційної технології проектування системи відео на замовлення з порівнянням та аналізом

протоколів потокових зв’язків з адаптивним бітрейтом та використання найбільш підходящого протоколу для даного додатку.

Даний додаток буде виконано з використанням новітніх передових технологій та принципами інтерфейсу користувача та користувацького досвіду (UX / UI), матиме зручний функціонал та чудовий перформанс.

Користуючись даним додатком кінцеві користувачі зможуть використовувати пошук та додаткові фільтри щоб знайти серед бібліотеки світового кінопрокату цікавлячі серіали та фільми, інформацію про них, дивитись трейлери, тощо.

На основі проаналізованої та отриманої інформації було визначено наступні задачі для реалізації:

- пошук та аналіз інформації необхідної для виконання роботи;
- проведення аналізу та отримання характеристик протоколів потокового зв’язку;
- вибір засобів та підходів реалізації платформи;
- реалізація додатку за допомогою мови програмування JavaScript, фреймворка React.JS, React Router, Styled Components та SASS, HTML;
- виконане розгортання платформи (deploy);
- виконане кінцеве тестування та виконаний аналіз кінцевих результатів.

Поставлена задача створення конкурентоспроможного та гарного за перформансом додатка, з використанням адаптивного стрімінгового протоколу MPEG-DASH для пошуку фільмів та серіалів використовуючи відкриту базу даних.

2 ВИБІР МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Вибір мови програмування та фреймворку для розробки

При розробці стримінгової платформи для пошуку фільмів та серіалів, мною було використано HTML, CSS/SASS, мову програмування JavaScript, фреймворк React.JS та React Router, Styled Components та REST API для виконання запитів.

HTML

HyperText Markup Language, або HTML, є основним компонентом Всесвітньої павутини (Інтернету). HTML визначає структуру та значення веб-вмісту яке буде відображено. Стилі та функціональність веб-сторінки зазвичай описуються іншими технологіями, окрім HTML, також за допомогою CSS та JavaScript. Варто розуміти, що посилання, які об'єднують онлайн-сторінки всередині одного веб-сайту або між веб-сайтами, називаються гіпертекстовими.

HTML теги, які оточують назву елемента символами «<» elem «>», служать для того, щоб відрізнити його від іншого тексту на сторінці. Великі літери не впливають на назву елемента всередині тегу. Тобто він може бути написаний у поєднанні малих і великих літер [5].

При розробці додатків, розробник у файлі HTML, пише код описуючи структуру веб-сайту. У файлах HTML код та текст можна форматувати до поставлених задач (змінювати текст, розмітку, шрифт, тощо). Також присутня можливість використання посилань на зовнішні веб-сторінки через HTML.

Всі відомі сучасні браузери, як Mozilla Firefox або Google Chrome, Safari, або Internet Explorer, 'парсять' HTML файл та відображають код описаний всередині.

CSS

Консорціум World Wide Web (W3C) створив CSS (Cascading style sheets) у 1996 році з дуже простою метою. Теги, які допомагали б форматувати сторінку, не мали бути частиною елемента HTML. Все, що вам потрібно було зробити, це написати розмітку сторінки. З появою таких тегів, як <font>, у HTML версії 3.2 веб-розробники зіткнулися з великими труднощами [6]. Написання нового коду для веб-сайтів вимагало багато часу та сил, також було фінансово невигідною задачею, оскільки вони мали доволі різні стилі, шрифти

та кольори. У результаті консорціум розробив каскадні стилі CSS для вирішення цієї проблеми. За допомогою CSS на веб-сайтах та платформах контент

SASS

SASS виступає у ролі пре-процесора CSS. Перш ніж елементи на веб-сторінці будуть скомпільовані та відображені, він використовується для аналізу складного синтаксису CSS. Процес розробки додатків є більш оптимізованим, ефективним та витрачається набагато менше часу на розробку, якщо використовується SASS. Даний пре-процессор працює з усіма версіями CSS [18]. Додаткові можливості, які пропонує SASS CSS, включають змінні, вкладення, міксіни, імпорти, константи тощо. Використання SASS має велику кільки бенефітів, таких як:

- Організованість. Оскільки код SASS є краще структурованим, ніж звичайний код написаний на CSS, ви можете використовувати менше коду для виконання тих самих завдань. Тому він важливий для масштабних проектів із залученням кількох розробників. Кожен розробник може легко зрозуміти код, створений і виконаний іншими завдяки добре організованому макету.
- Перевикористання. Аспект багаторазового використання мови програмування, який відсутній у CSS, представлений у SASS. Це дозволяє використовувати змінні та блоки коду, які розробники можуть застосовувати до інших частин проекту. Як результат, це зменшує ймовірність помилок і полегшує модифікацію коду.
- Легкий до вивчення. Щоб вивчити SASS, не потрібно багато часу. Будь-хто, хто знайомий із CSS, може швидко та без особливих зусиль освоїти SASS, витративши небагато часу чи зусиль.

Як висновок код написаний за допомогою SASS — це просто код CSS без крапок з комою та фігурних дужок, що робить код легшим для розуміння

та впорядкування, гарно адаптований до перевикористання та добре структурно організований [23].

JavaScript

Веб-розробники часто використовують JavaScript, легку мову програмування (мова сценаріїв), щоб забезпечити динамічну взаємодію із серверами, іграми, додатками або веб-сторінками. JavaScript бездоганно працює з HTML і CSS, допомагаючи CSS форматувати HTML-компоненти та забезпечуючи взаємодію з користувачем. CSS сам по собі не може забезпечити даного інтерактиву [21]. JavaScript — наразі є одною із найпопулярніших мов програмування, оскільки її багато використовують у розробці ігор, мобільних додатків, веб сторінок, тощо.

Дана мова програмування спочатку була призначена в основному для внутрішнього використання. JavaScript очолював шлях у публікації ECMAScript, як стандартна специфікація для усіх веб-браузерів, тобто дана мова повинна була стати мовою сценаріїв загального призначення [7]. Вона розроблялась для забезпечення сумісності кінцевого юзера та пристрою на якій він використовувався. З того моменту JavaScript розвивався разом із усіма створеними браузерами, такими як Firefox від Mozilla та Chrome від Google. Для інформації Google Chrome почав працювати над двигуном V8, першим двигуном JavaScript сьогодні, який перетворює байт-код у машинний код. Наразі JavaScript пропонує безліч фреймворків та інструментів, таких як React.JS, Angular та інші розробка платформ та додатків була набагато простіша.

JavaScript є кращим варіантом, ніж його конкуренти (інші мови програмуавння), завдяки численним перевагам [14]:

- Швидкість. Запуск програми ввідбувається безпосередньо в самому браузері. Компіляція відбувається під час роботи платформи.

- Простота. Дана мова програмування працює набагато швидше ніж інші мови програмування. Його легше зрозуміти та застосувати завдяки його простоті, та набагато легше дебажити (виправляти помилки).
- Завантаження сервера. JavaScript максимально мінімізує кількість запитів, які надсилаються на сервер (API Calls), що є додатковою перевагою роботи на стороні кінцевого клієнта.
- Також варто перечислити популярність даної мови програмування, її універсальність та наявність постійних оновленнь.

React.JS

React — це фрейморк, який використовує Webpack для обробки префіксів файлів CSS і автоматичного створення коду React, JSX та ES6. React — це бібліотека розробки інтерфейсу користувача, створена за допомогою JavaScript. Даний фреймворк є дуже популярною бібліотекою для веб-розробки, незважаючи на те, що це не мова програмування. З моменту свого дебюту в травні 2013 року бібліотека стала однією з найбільш широко використовуваних інтерфейсних бібліотек для веб-розробки. Крім інтерфейсів користувача, він надає інші розширення для повної підтримки архітектури програми, включаючи Flux і React Native [17].

Зараз React є більш популярним, ніж будь-який інший інтерфейсний фреймворк розробки. Розглянемо ж переваги використання [22]:

- Проста конструкція динамічних програм. На відміну від JavaScript, де код часто дуже швидко ускладнюється, React потребує менше кодування та надає більше функціональних можливостей, що полегшує створення динамічних онлайн-програм.
- Швидкість роботи. React використовує віртуальний DOM, щоб швидше створювати веб-програми. На відміну від традиційних веб-додатків, які неодноразово оновлюють кожен компонент, віртуальний DOM перевіряє минулі стани компонентів і оновлює лише ті речі в реальному DOM, які були змінені.

- Перевикористання компонентів. Одна додаток React часто складається з кількох компонентів. Компоненти є основними одиницями будь-якої програми React. Час розробки програми значно скорочується, оскільки ці компоненти можуть повторно використовуватися в ній і зберігають свою логіку та елементи керування.
- Односпрямований потік даних. Потік даних у React є односпрямованим. Це означає, що розробники часто розміщують дочірні компоненти всередині батьківських компонентів під час створення проекту React. Оскільки дані надходять лише в одному напрямку, усунення несправностей і визначення точного місця проблеми в програмі в будь-який момент часу стало простіше.
- Мінімальний час на вивчення. React простий для розуміння, оскільки він здебільшого поєднує фундаментальні ідеї JavaScript і HTML з кількома корисними вдосконаленнями.
- Він застосовується для створення як онлайнових, так і мобільних додатків. React не обмежується його використанням у розробці веб-додатків, як ми добре знаємо. Красиві мобільні програми можна створювати за допомогою фреймворку React Native, який базується на самому React. Він неймовірно популярний.
- Спеціальні інструменти для простого налагодження. аддон Chrome, розроблений Facebook, можна використовувати для налагодження додатків React. Це полегшує та прискорює процес налагодження веб-програм React.

React є найпопулярнішою бібліотекою для розробки зовнішніх додатків через свої виняткові можливості. Першою із ключових характеристик є наявність JSX. JSX — це синтаксичне розширення для JavaScript. У React ця фраза стосується бажаного вигляду інтерфейсу користувача. JSX дозволяє писати HTML-структури в одному файлі з кодом JavaScript [8]. Згаданий

нижче код демонструє, як JSX використовується в React. Це не HTML і не строка. Навпаки, він включає HTML у код JavaScript.

```
const city = 'Kyiv'
const greeting = `I was born in the ${city}, the capital of Ukraine`
return (
    <div>
        {greeting}
    </div>
)
```

Рисунок 1.8 – Приклад синтаксичного розширення JSX

Полегшена версія реального DOM у React називається віртуальним DOM. Порівнюючи віртуальні та реальні маніпуляції DOM, реальні маніпуляції DOM рухаються набагато повільніше. Віртуальний DOM оновлює лише певний об’єкт у реальному DOM, а не всі, коли змінюється стан об’єкта. Об’єктна модель документа (DOM) розглядає документ XML або HTML як деревовидну структуру, де кожен вузол є об’єктом, який представляє розділ сторінки.

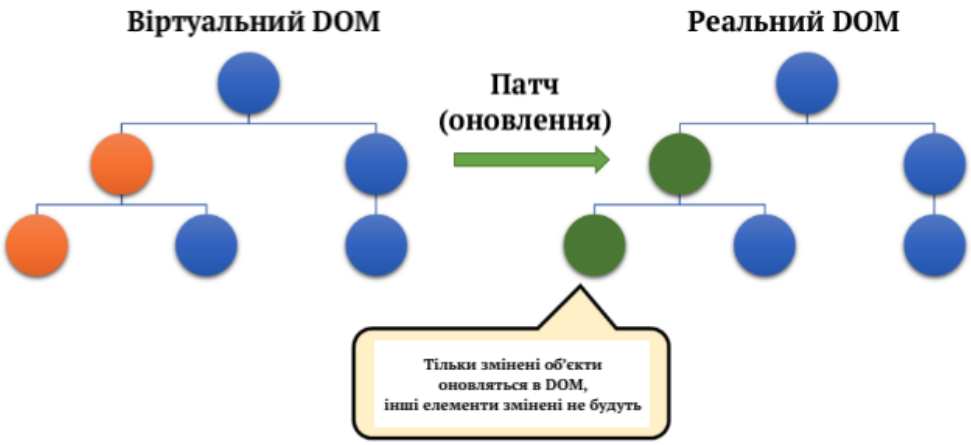

Рисунок 1.9 – Приклад роботи реального та віртуального DOM

Віртуальний DOM оновлюється в програмі React, коли змінюється стан об’єкта. Потім, замість того, щоб оновлювати кожен елемент у фактичному DOM, він порівнює його попередній стан і змінює лише ці речі. Це прискорює процес, особливо на відміну від інших зовнішніх технологій, які потребують

оновлення кожного елемента у веб-додатку, навіть якщо змінюється лише один елемент.

Будівельні елементи програми React, кожен з яких представляє частину інтерфейсу користувача, називаються компонентами [11]. Використовуючи даний фреймворк дебагінг стає легшим, оскільки інтерфейс користувача розділений на кілька компонентів, кожен з яких має власний набір функцій і можливостей.

Плюси використання компонентів очевидні:

- Повторне використання. Компонент програми, який використовується в одній області, може бути застосований в іншій. Це сприяє швидшому процесу розробки.

- Вкладені компоненти. Компонент може мати всередині кілька дочірніх компонентів.

- Метод візуалізації компонента, який описує, як компонент візуалізується в DOM, має бути визначений у найпростішій формі. Реквізити також можуть бути надіслані до компонента шляхом передачі властивостей. Це атрибути, які його батько передав для визначення значень [16].

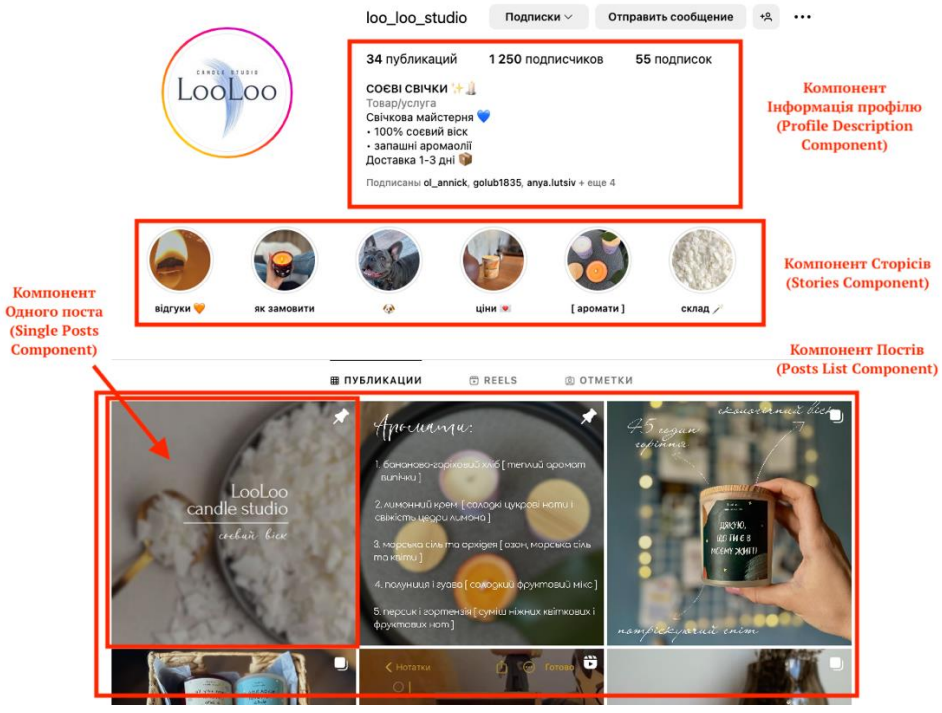

Рисунок 1.10 – Приклад компонентів React в додатку Instagram

З виходом версіїї React 16.8 на початку 2019 року команда React спростила розробку та додата підтримку ‘хуків’, що вирішило деякі наявні проблеми з класами та методами життєвого циклу. Наразі ми можемо відокремити логіку стану та побічні ефекти від функціонального компонента. Методи JavaScript, звані хуками, використовуються для ізоляції та контролю поведінки та побічних ефектів стану всередині компонента.

Розглянемо найбільш популярні React Hooks [19]:

- useState – використовується для управління станом (стейтом). Даний метод повертає ‘метод оновлення’ для оновлення значення стану компонента. Відмінність від класових компонентів в тому, що поточний стан не об’єднується з минулим станом;
- useEffect – використовується для керування різними побічними ефектами, такими як підписками та таймерами, API запитами та різноманітними мутаціями;
- useContext – використовується, щоб надати поточне значення контексту компонента, мати глобальне керування та доступ до даних з будь якої іншої частини додатка;
- UseReducer є схожим хуком useState – деякі називаються альтернативою даного хука. Він може допомогти в управлінні складними станами компонента;
- useCallback – використовується та допомагає дочірньому компоненту уникнути непотрібного повторного рендерингу. Він повертає зворотний виклик функції, які була закешована.
- useMemo використовується для повернення мемоізованої версії даних. Він допомагає оптимізувати та покращити швидкість роботи компонентів;
- useRef – використовється для створення об’єкту ref, тобто reference, посилання на компонент. Об'єкт посилання можна також змінити. Також

використовується для зберігання даних, при цьому відсутній ре-рендер компонента [26].

React Router

Стандартна бібліотека для маршрутизації в React називається React Router. Вона дозволяє перемикатися між переглядами різних компонентів у програмі React, змінювати URL-адресу браузера та підтримувати синхронізацію між інтерфейсом користувача та URL-адресою. React Router, як і інші модулі в екосистемі React, доступний для встановлення як пакет Node Package Manager. Також React Router можна встановити за допомогою будь-якого засобу керування пакетами, такими як pnpm або yarn. Необхідною командою для інсталювання є npm install react-router-dom.

Концептуально робота раутера базується на тому, що браузер робить запит сторінки з веб-сервера, завантажує та парсить файли CSS і JavaScript, а також відображає HTML, наданий із сервера на традиційних веб-сайтах. Коли користувач натискає посилання, процес перезапускається для нової сторінки [9]. Маршрутизація на стороні клієнта дозволяє вашій програмі коригувати URL-адресу після натискання посилання, не запитуючи інший документ із сервера. Натомість ваша програма може швидко відобразити оновлений інтерфейс користувача та використати вибірку для виконання викликів даних для оновлення сторінки новою інформацією. Оскільки веб-переглядачу не потрібно запитувати повністю новий документ або повторно оцінювати ресурси CSS і JavaScript для наступної сторінки, це забезпечує швидшу роботу користувачів. Це також забезпечує більш динамічну взаємодію користувача з елементами [24].

Головними компонентами роботи даної бібліотеки є:

- ‘Route’ (маршрут) - є найважливішим компонентом програми React Router. Дані маршрути пов’язують сегменти URL-адрес із компонентами, а також завантаження та модифікацію даних. Складні

макети додатків і залежності даних стають простими та декларативними завдяки вкладеності маршрутів;

- NavLink — це свого роду посилання, яке знає, чи воно «активне», «очікує» чи «переходить». Це корисно в різних ситуаціях:

1) під час створення навігаційного меню, або групи вкладок, ви можете вказати, яка з них вибрана на даний момент.

2) компонент пропонує контекст для допоміжних технологій, таких як програми зчитування з екрана.

- Navigate - під час візуалізації, елемент змінює поточне розташування. Це обгортка для useNavigate, яка приймає всі ті самі параметри, що й властивості.

Styled Components

Стилізовані компоненти є спеціальним для React рішенням стилю CSS-in-JS, яке використовує літерали шаблону з тегами та функції стрілок у ES6+ і CSS, щоб запропонувати розробникам платформу для написання фактичного CSS-коду для стилізації компонентів React, а також React Native. Використання styled-components спрощує створення стилів компонентів за допомогою фактичного CSS шляхом написання коду JavaScript. Вони відомі як «компоненти зі стилями», і насправді це компоненти React зі стилями.

Окрім покращеної та оптимізованої розробки, styled-components пропонує дані переваги використання:

- Генерація CSS на льоту — застосовує лише стилі відображених компонентів сторінки. Інструменти розробника React є чудовим підходом для дослідження цього концепту;
- Розрізнені імена класів – кожному елементу DOM присвоєно окреме ім’я класу, що запобігає проблемам і зіткненням імен класів і дозволяє нам з великої швидкістю розробляти додатки;
- Спрощене видалення CSS — стилі прив’язані до певного компонента, а не додаються як ім’я класу, що спрощує оптимізацію коду CSS.

- Динамічний стиль - дозволяє стилізувати компонент на основі атрибутів або глобальної теми;
- Підтримка компонентів є простою і доступною, незалежно від того, наскільки великою є кодова база, оскільки ніколи не доведеться переглядати численні файли, щоб знайти стиль, який впливає на потрібний компонент;
- Автоматичне додавання префіксів — обробляє префікси постачальників для підтримки браузера, щоб розробники писали більш простий код CSS.

REST API

Повна назва Representational State Transfer, або REST, — це підхід до веб-архітектури, який дозволяє клієнтам та серверам обмінюватися стандартами. Коли клієнт запитує ресурс, сервер відповідає зрозумілим представленням цього ресурсу, тобто зрозумілими даними, які надалі можуть бути використані. REST API — це свого роду API, який відповідає на запит клієнта в стандартизованому форматі, який легко читати через обмеження компонентів, які є уніфікованими [12]. Формат відповіді REST – HTTP, і це може бути JSON (об’єктна нотація JavaScript), HTML, XLT, Python, PHP або звичайний текст. Більшість розробників обирають формат JSON, оскільки він є більш універсальним. Запит до REST API, також відомого як RESTful API, має надходити через HTTP до протоколу без збереження стану, який не зберігає інформацію клієнта протягом сеансу отримання запиту. Концепт роботи пов’язаний з запитом клієнта та відповіддю сервера.

Запит клієнта містить суб’єкт, програму або особу, яка використовує служби API. Як приклад, припустімо, що є запит опублікувати фотографію з стороннього додатка на сторінці у Facebook. Буде використаний Facebook API, щоб допомогти у цьому процесі.

Відповідь сервера REST передає із сервера запитуваний клієнтом ресурс. Однак ресурс надходить не в його справжньому стані, а скоріше в стандарті

представлення, який доступний і зрозумілий як людям, так і машинам. JSON, HTML, XML, CSV, або YAML, та навіть звичайний текст є прийнятними форматами.

2.2 Середовище розробки, сервери та плагіни

Середовище розробки

Працюючи над платформ відео на замовлення, я отримав можливість ознайомитися та вивчити численні середовища розробки (IDE). Використовуючи операційну систему macOS важливою була повна сумісність обраної Integrated Development Environment з наявною системою.

Мною було обрана інтегрована середа розробки VSCode. Даний IDE є безкоштовним та легким, а також в свою чергу дуже потужним редактором коду, який доступний для багатьох операційних систем, таких як macOS, Windows, Linux або ОС Raspberry Pi [15]. Він включає вбудовану підтримку JavaScript та Node.js, TypeScript , а також надійну систему розширення для інших мов програмування (зокрема C++ та PHP, Java та Python), середовища виконання (включаючи Unity або .NET), середовищ (Kubernetes або Docker) і хмарних сховищ (Amazon Web Services, Google Cloud Platform та Microsoft Azure).

Плагіни

NPM — це абревіатура від Node Package Manager. Це менеджер пакетів платформи Node JavaScript [13]. NPM загальновизнаний як найбільший реєстр програмного забезпечення у світі. NPM використовується розробниками для публікації та розповсюдження свого відкритого вихідного коду у всьому світі.

NPM складається з трьох частин:

- Можливе використання веб-сайту для пошуку пакетів третіх сторін, керування своїми пакетами та створення профілів;
- інтерфейс командного рядка, або NPM CLI, дозволяє спілкуватися NPM з терміналом.
- Реєстр — масивна загальнодоступна база даних коду JavaScript.

Сервери

Для більш комфортної розробки проекту був використаний localhost. Localhost можна розглядати як «цей комп'ютер» у комп'ютерній мережі. Це ім'я за замовчуванням, яке використовується для підключення до вашого комп'ютера через мережу петлевої адреси.

Адреса петлі має стандартну IP-адресу (127.0.0.1) і зручна для тестування програм на комп'ютері без передачі даних через Інтернет. Це корисно під час тестування програм, які ще не готові для загального використання. При виконанні запиту за IP-адресою зі свого комп'ютера, зазвичай виконується спроба підключитися до іншого комп'ютера через мережу Інтернет. Однак, коли використовується петлева адреса, викликається локальний хост, іноді відомий як цей (даний) комп'ютер.

Однією з найпоширеніших цілей для розробників є використання локального хосту, особливо під час розробки веб-додатків або програм, які потребують підключення до Інтернету. Під час розробки проводяться тести, щоб переконатися, що програми працюють належним чином. Розробники можуть перевірити їх, створивши з'єднання з локальним хостом, яке потім можна перевірити на машині та системі, які вони зараз використовують. Коли ініціюється петля, операційна система перетворюється на імітований веб-сервер. Також можливо перевірити роботу програмного забезпечення, завантаживши відповідні файли на веб-сервери.

Готовий стримінговий додаток буде розгорнуто на хмарній платформі Netlify. Netlify — це компанія з хмарних обчислень, яка надає платформу для розробки веб-додатків і динамічних веб-сайтів, яка включає в себе створення, розгортання та безсерверні серверні послуги. Платформа базується на відкритих веб-стандартах, що дозволяє інтегрувати інструменти для створення, веб-фреймворки, API та різноманітні веб-технології в єдиний робочий процес розробки.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Налаштування проекту

Перший кроком практичного виконання роботи, мною було створено проект за допомогою менеджера пакетів NPM для запуску віддаленого сценарію. Сценарій скопіював необхідні файли в новий каталог і встановив усі залежності. NPM встановив пакети JavaScript у моєму проекті, а також надалі відстежуватиме деталі проекту.

Виконуваний пакет запустить установку create-react-app у вказаний мною каталог. Він розпочнеться зі створення нового проекту в каталозі, який матиме назву “sumdu-vod-system-design”. Знову ж таки, цей каталог не повинен існувати заздалегідь, виконуваний пакет створить його з нуля. Сценарій також запустить npm install у каталозі проекту, який завантажить усі додаткові залежності.

Виконуємо команду в терміналі Visual Studio Code:

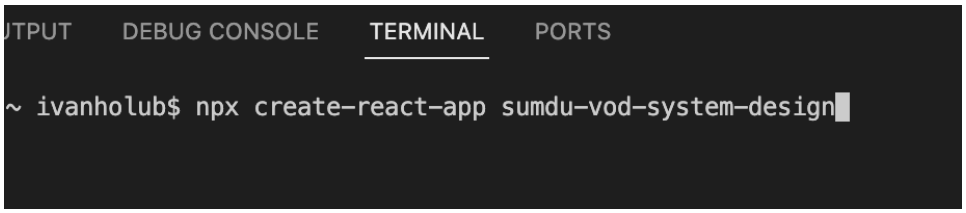

Рисунок 3.1 − створення React проекту

Create React App — це офіційно затверджений метод розробки односторінкових React-додатків. Він забезпечує сучасне налаштування збірки, яке вимагає мінімальної конфігурації. Щоб покращити додаток, використовується webpack, Babel, ESLint та інші плагіни. Надалі використовуючи термінал VSCode виконую команду npm run start — за допомогою цього сценарію запускається сервер розробки. Наприклад, nodemon можна використовувати для керування сервером у проекті Node.

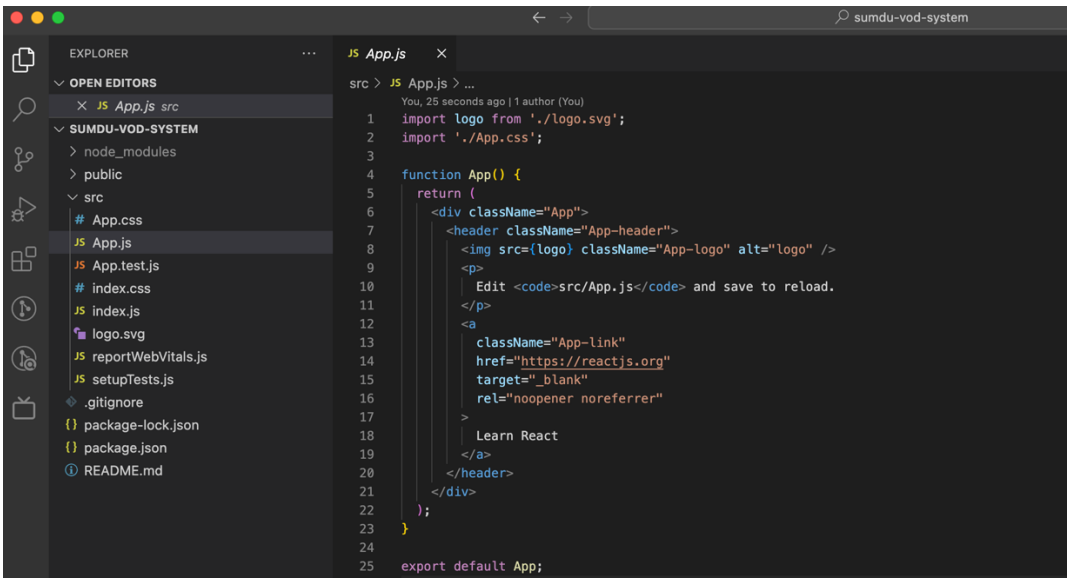

Рисунок 3.2 – Створенний базовий React проект

За допомогою терміналу та NPM встановлюємо додаткові плагіни та розширення для розробки, які нам знадобляться в майбутньому:

1) node-sass;
2) react-router-dom;
3) styled-components;
4) react-infinite-scroll-component;
5) react-dom.

Рисунок 3.3 – package.json файл з встановленими залежностями

Всі розширення були встановлені за допомогою команди ‘npm install ….’ та додана назва потрібного плагіну.

Також варто пам’ятати, що з ініціалізацією проекту та встановленням пакетів в нашому проекті з’явилась папка ‘node_modules’. Модулі Node дозволяють повторно використовувати код у моєму додатку. У деяких аспектах їх можна порівняти з класами в інших мовах, таких як C# або Java. Вони значно відрізняються від класу за багатьма параметрами. При роботі з Git та майбутньому відправленню гілки нам треба, щоб ‘node_modules’ були проігноровані та не відправлені до нашого репозиторія, тому створюємо файл ‘.gitignore’ та додаємо ‘node_modules’ для ігнорування.

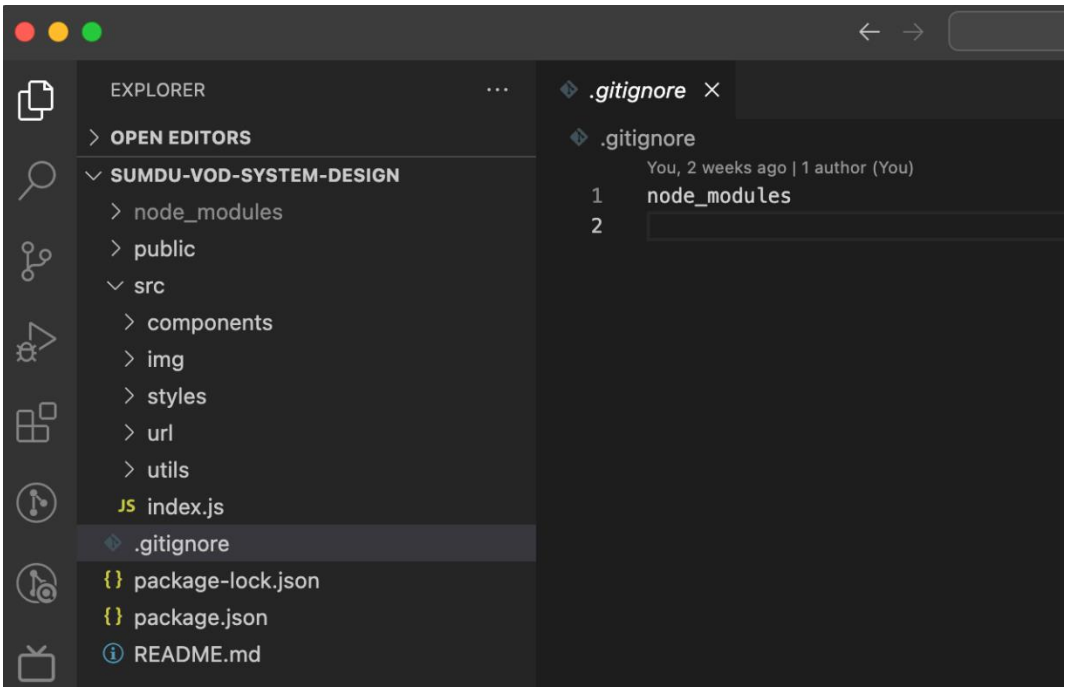

Рисунок 3.4 – Налаштування файлу ‘.gitignore’

3.2 Проектування стримінгової платформи

Перед створенням проекту мною було виконано аналіз майбутної архітектури платформи та створено діаграму потрібних компонентів та залежностей між ними.

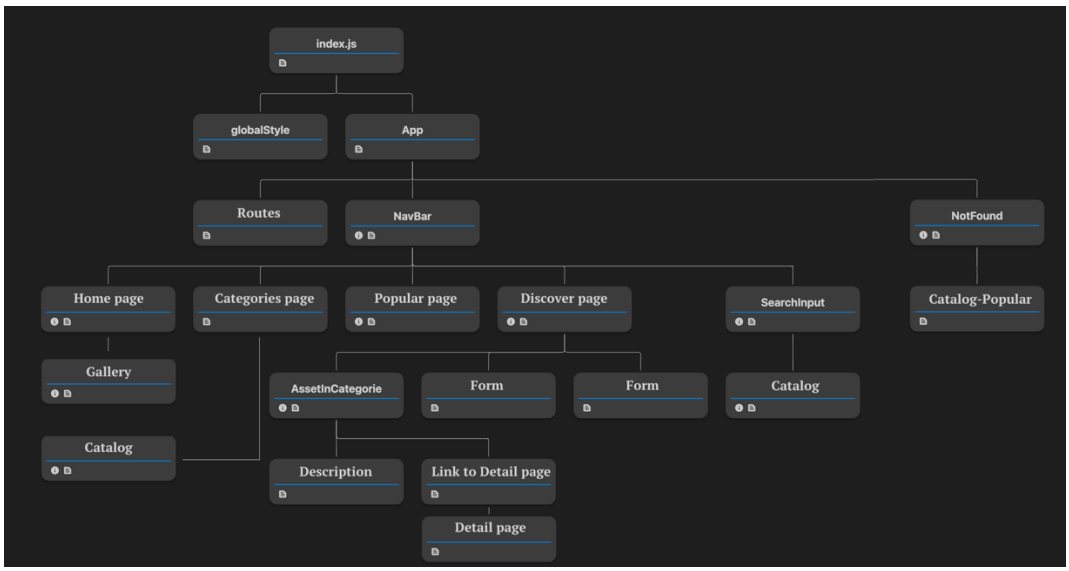

Рисунок 3.5 – Архітектура майбутньої платформи

З діаграми можна побачити що в index файлі будуть знаходитись компонент глобальних стилів та App – вхідний та головний компонент – обгортка для всього додатка. В ньому буди описаний компонент Navigation та Routes – які будуть відповідати за навігацію по додатку та компонент Not Found – у випадку помилкового вводу url або відсутніх даних даний компонент буде визваний автоматично. Компонент NavBar матиме всередині описані головні сторінки платформи, які також матимуть свої Child компоненти всередині – такі як компонент детальної сторінки, колекції, тощо. То ж загалом проект буде складатися з таких компонентів, як:

- Header;
- Головні сторінки;
- Галерея;
- Детальні сторінки;
- Пошук;
- Колекція елементів (фільмів та серіалів);
- Колекція жанрів;
- Компонент Discover – з фільтрами та пошуком;
- Компонент Not Found для Error Handling.

Також була продумана кросплатформенна сумісність. Була визначена підтримка і сумісність з різними веб-браузерами, та можливість в майбутньому масштабувати дану платформу на мобільні програми (iOS та Android) та смарт-телевізори. Визначена стратегія монетизації - були розглянуті різні моделі монетизації, як на основі підписки, з підтримкою реклами або комбінацію так і початково безкоштовні з майбутньою можливістю масштабування. Можливість в майбутньому застосування вставки реклами в стримінг, якщо бууде належна підтримка зі сторони стримінгово провайдеру. Але на початку розробки – MVP - Minimum viable product – було обрано безкоштовний тип користування платформою.

3.3 Програмна реалізація платформи

Першим кроком, за допомогою Figma – онлайн сервісу для розробки інтерфейсів, який використовує векторне прототипування, мною було розроблено інтерфейс користувача (UI). Я створив візуально привабливий інтерфейс користувача, який буде інтуїтивно зрозумілим для кожного глядача даної платформи. Також було продумано та впроваджено адаптивний дизайн, для забезпечення сумісності на різних пристроях з різною роздільною здатністю. Дизайн має вирішальне значення для розробки програми, оскільки він має великий вплив на взаємодію з користувачем, зручність використання та загальний успіх програми. Ось деякі з причин, чому дизайн важливий під час розробки програми:

- Взаємодія з користувачем: дизайн програми безпосередньо впливає на те, як люди з нею взаємодіють. Добре розроблена програма забезпечує сприятливий і зрозумілий досвід користувача, збільшуючи ймовірність того, що користувачі повернуться до програми.

- Візуальні компоненти, макет і загальна естетика програми мають вирішальне значення для залучення та підтримки користувачів. Добре розроблений інтерфейс користувача покращує загальний вигляд і відчуття програми.

- Зручність використання: добре розроблене програмне забезпечення просте у використанні та навігації. Чітка навігація, прості елементи керування та логічний потік — усе це покращує зручність використання та допомагає людям швидше навчатися.

3.3.1 Навігація, header та глобальні стилі

За допомогою інтегрованого пакету react-dom, маємо доступ до рендер функції. React-dom надає специфічні для DOM методи, які можна використовувати на верхньому рівні мого додатка, щоб вийти за межі моделі React. В компоненті App мною було описано головну логіку маршрутів (‘routes’) додатка та елементів меню. Маршрути (routes) являють собою статичний массив об’єктів з id, route та label.

```
{ id: "discover", route: "/discover", label: "Discover" },
{ id: "movie", route: "/movie/:movie_id", label: "Movie" },
{ id: "movies", route: "/movies/:category_id", label: "Movies-by-Categorie" },
```

Рисунок 3.6 − Routes (маршрути) використані в платформі

Використовуючи імпортовані методи Router, Switch, Routes з react-router мною був описаний кожен route – тобто шлях, коли юзер буде знаходитися на конкретному маршруті – в цей час буде відкритий відповідний компонент. Також звертаю увагу, що якщо ми матимемо довільний раут – який не буде описаний в кодовій базі – буде визваний компонент NotFound з описаною логікою всередині [27].

Мною додатково була використана техніка контексту – React Context. Контекст забезпечує спосіб передавачі даних через дерево компонентів без необхідності передавати атрибути вручну на кожному рівні. У типовій програмі React дані передаються зверху вниз (батьківський до дочірнього) через властивості, але таке використання може бути громіздким для певних типів властивостей (наприклад, налаштування локалі, тема інтерфейсу користувача), які потрібні багатьом компонентам програми. Контекст надає спосіб обмінюватися значеннями, подібними до цих, між компонентами без

необхідності явно передавати властивість через кожен рівень дерева. За допомогою контексту були передані дані для ‘теми’ додатку.

```
You, 2 weeks ago | 1 author (You)
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      menu: menuItems,
      routes: routes,
    };
  }

  render() {
    return (
      <ThemeProvider theme={theme}>
        <Wrapper>
          <Router>
            <NavBar menu={this.state?.menu ? this.state?.menu : false} />
            <Switch>
              {this.state?.routes &&
                this.state?.routes?.map((el) => {
                  const { id, exact, route, Component } =
                    getProperComponentData(el);

                  return (
                    <Route
                      key={id}
                      exact={exact}
                      path={route}
                      component={Component}
                    />
                  );
                })}
              <Route path="*" component={NotFound} />
            </Switch>
          </Router>
        </Wrapper>
      </ThemeProvider>
    );
  }
}

export default App;
```

Рисунок 3.7 – Компонент App з маршрутами та компонентом NavBar

Додатково в компонент App було імпортовано NavBar (компонент) в який за допомогою пропсів (props) – було передано елементи за якими ми зможемо проводити навігацію по застосунку. Дані елементи меню були імпортовані із файлу Constants.js та описані в ньому (масив об’єктів з id, route та label для кожного елемента меню).

```
{ id: "home", route: "/", label: "Home" },
{ id: "categories", route: "/categories", label: "Categories" },
{ id: "popular", route: "/popular", label: "Popular" },
```

Рисунок 3.8 – Приклад елементів меню використаних в додатку

За допомогою написаного коду можемо при оновленні сторінки побачити елементи меню в Header додатку. При кліках по даним елементам буде оновлюватись маршрут та відповідний url, надалі буде відкритий відповідний компонент.

Рисунок 3.9 – Header та елементи меню

У головному файлі index.js імпортуємо компоненти GlobalStyles та App. Компонент GlobalStyles відподає за глобальні стилі мого застосунку, та надалі кожен компонент матиме свої індивідуальні стилі:

globalStyle.js

```
src > styles > globalStyle.js > ...
      You, 2 weeks ago | 1 author (You)
1     import {createGlobalStyle} from 'styled-components';      You, 2 weeks ago • vod-system-design
2
3     const GlobalStyles = createGlobalStyle`
4         *,
5         *::after,
6         *::before {
7             margin: 0;
8             padding: 0;
9             box-sizing: inherit;
10        }
11
12        html {
13            // This defines what 1rem is
14            font-size: 62.5%; //1 rem = 10px; 10px/16px = 62.5%
15            font-family: 'Quicksand', sans-serif;
16        }
17
18        body {
19            box-sizing: border-box;
20        }
21    `;
22
23    export default GlobalStyles;
```

Рисунок 3.10 – Глобальні стилі додатку

3.3.2 Створення DataLoader класу та підключення Movie DB AP

Для доступу до бази даних фільм та серіалів мною було використано Movie DB API. Даний API надає вичерпний список доступних наразі методів до кіно, телебачення, акторів і зображень. Movie DB — це механізм, який також дозволяє програмно отримувати та використовувати наші дані та/або фотографії. Для користування даним API першочергово треба зареєструватися на платформі та створити API ключ, який надалі буде використовуватися в

header requests для кожного API запиту. Після створення даного ключа залишається тільки зробити API запит та отримати дані з response.

Мною був написаний Loader клас, який приймає ‘url’ та параметри, та виконує запит з методом ‘GET’ або методом ‘POST’. Використана конструкція ‘async/await’ для асинхронного запиту та конструкція ‘try/catch’ для відловлювання помилок.

```
export default class DataLoader {
  constructor(url, param) {
    this.url = url;
    this.param = param;
  }
  async get() {
    try {
      const request = await fetch(this.url, {
        method: "GET",
        headers: {
          Accept: "application/json",
          "Content-Type": "application/json",
        },
      });
      const data = await request.json();
      return data;
    } catch (error) {
      console.log(error);
    }
  }

  async post() {
    try {
      const request = await fetch(this.url, {
        method: "POST",
        headers: {
          Accept: "application/json",
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          filters: this.param,
        }),
      });
      const data = await request.json();
      return data;
    } catch (e) {
      console.log(e);
    }
  }
}
```

Рисунок 3.11 – DataLoader клас для виконання API запитів

Використовуючи документацію, ключ API та відповідний URL виконуємо тестовий запит для отримання даних для HOME сторінки. В response бачимо відповідний результат, масив із 20 популярних фільмів:

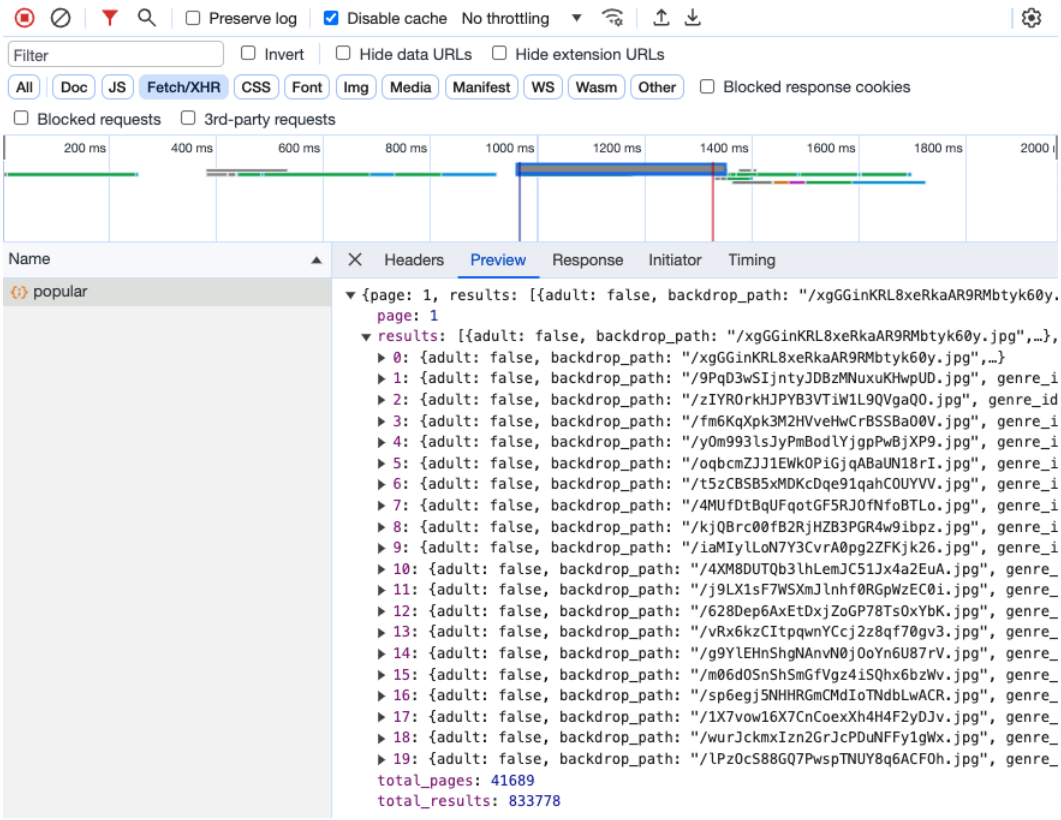

Рисунок 3.12 – Response для Home page

Надалі за допомогою даного DataLoader та відповідних URL's будемо виконувати запити та використовувати дані з response у відповідних компонентах.

3.3.3 Головна сторінка Home та Галерея

При кліку по елементу Home який знаходиться на Header додатку React Router за допомогою мною описаної функції getProperComponentData – викличе компонент Home. Дана функція по кліку на кожен елемент перевіряє label елемента та за допомогою конструкції switch/case повертає відповідний компонент. Ми можемо побачити, що відповідний компонент відкривається за відповідним лейблом кожного елемента меню.

```
export default function getProperComponentData(el) {
  switch (el?.label) {
    case "Home":
      return { ...el, exact: true, Component: Home };
    case "Categories":
      return { ...el, Component: Categories };
    case "Popular":
      return { ...el, Component: Popular };
    case "Movies-by-Categorie":
      return { ...el, Component: MoviesByCategory };
    case "Asset":
      return { ...el, Component: AssetDetails };
    case "Discover":
      return { ...el, Component: Discover };
    case "Movie":
      return { ...el, Component: AssetDetails };
    default:
  }
}
```

Рисунок 3.13 – Головна функція вибірки відповідного компонента

Компонент Home являє собою функціональний компонент з використанням useState та useEffect хуків.

Хук — це спеціальна функція, яка дозволяє «підключатися» до функцій React. Наприклад, useState — це хук, який дозволяє додавати стан React до функціональних компонентів. useEffect дозволяє виконувати побічні ефекти у функціональних компонентах, такі як:

- Отримання даних;
- Налаштування підписки
- Ручна зміна DOM у компонентах React.

За допомогою useEffect хука виконано REST API запит на бекенд для отримання популярних фільмів з прикладу вище. Після отримання response з бекенду їх було додано в стейт компоненту за допомогою хука useState. Надалі ми передаємо ці фільми до слайдеру у рендер функції.

```
const Home = () => {
  const [popularFilms, setPopularFilms] = useState(null);

  useEffect(() => {
    async function getPopularFilms(url) {
      const getFilms = await servicePopular(url);
      setPopularFilms(getFilms);
    }
    getPopularFilms(popularHomePageTMDB_URL);
  }, []);

  if (popularFilms) {
    return (
      <>
        <Header2 style={{ color: "#fff" }}>
          WHAT TO SEE
          <ImageRotated src={Logo} />
        </Header2>
        <SliderBox popularFilms={popularFilms} />
        <SummaryLinks />
      </>
    );
  }
  return null;
};
```

Рисунок 3.14 – Компонент Home з виконаним API запитом та слайдером у рендер функції

Отриманий результат при кліку на елемент меню Home та відкритті відповідного компоненту HOME можна побачити на скріншоті:

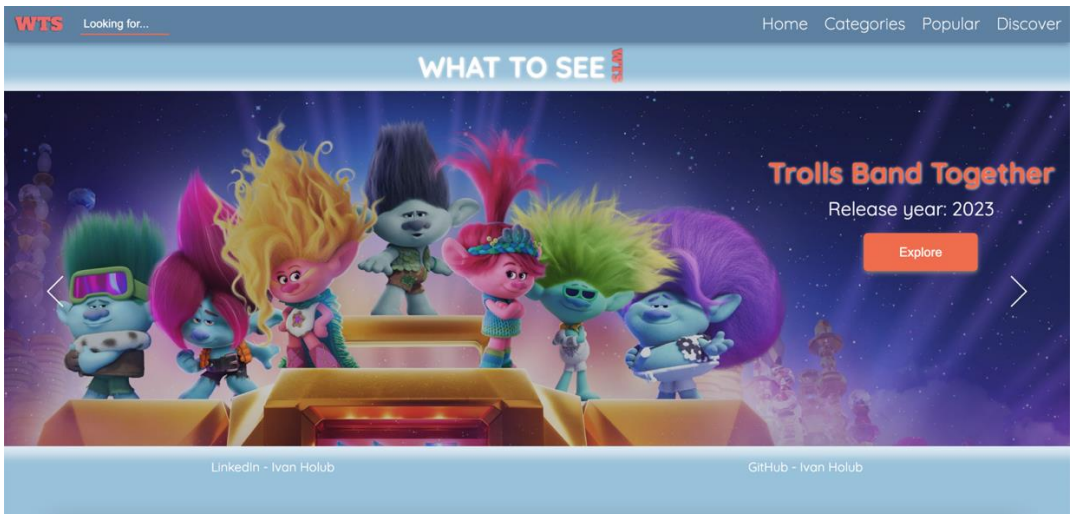

Рисунок 3.15 – Головна сторінка додатку – Home

3.3.4 Колекції, детальні сторінки та пошук

Велику кількість фільмів або серіалів мною було вирішено показувати користувачу за допомогою колекцій. Щоб оптимізувати швидкість завантаження та рендерингу було використано метод ‘lazy loading’.

Lazy loading (також відоме як завантаження за вимогою) — це стратегія оптимізації онлайн-вмісту, наприклад веб-сайту або веб-додатка. Замість того, щоб завантажувати всю веб-сторінку та показувати її користувачеві за один раз, як у масовому завантаженні, поняття відкладеного завантаження допомагає завантажувати лише необхідну частину та відкладати решту, поки користувач не зажадає цього. Для відтворення колекцій використовується один написаний компонент, який перевикористовується в декількох частинах даної платформи, що знову ж таки підтверджується правильність використання фреймворку React. Колекції ми можемо побачити натиснувши наприклад на елемент меню “Popular” або ‘Discover’.

Розглянемо концепт створення жанрів та категорій. Натиснувши на елемент меню ‘Categories’ буде визвано однойменний компонент ‘Categories’ з виконаним API запитом ‘/list’ для отримання доступних жанрів. В response ми можемо побачити структуру даних, яку я перебираю за допомогою методу ‘.map’ та використовуючи ‘useState’ хук записуємо дані в стейт компонента для для його рендера. Також одразу можемо побачити результат роботи.
Також варто зазначити, що хоч категорії жанрів є колекцією, в даному випадку lazy loading не використовувався. Даний концепт слід використовувати лише для прискорення завантаження сторінки або зменшення використання сервера, та не для того, щоб компенсувати поганий веб-код або слабкий веб-сервер. Відкладене завантаження не повинно мати візуального впливу на ваші веб-сторінки, якщо воно виконується належним чином. У даному випадку кількість елементів всього двадцять, тому воно наразі не доречне.

```jsx
const Categories = () => {
  const [categoriesData, setCategoriesData] = useState(null);

  // effect to get categories of movies
  useEffect(() => {
    (async function getCategories() {
      const dataCategories = await serviceCategories(categoriesTMDB_Url);
      setCategoriesData(dataCategories);
    })();
  }, []);

  if (categoriesData) {
    const arrayOfCategoriest = getArrayOfCategories(categoriesData);

    return (
      <>
        <Header1>Categories</Header1>
        <CategoriesWrapper>{arrayOfCategoriest}</CategoriesWrapper>
      </>
    );
  } else {
    return null;
  }
};
```

Рисунок 3.15 – Компонент Categories з виконанням API запиту

Після оновлення сторінки ми можемо побачити результат роботи даного коду. Надалі при кліку на ‘будь’ який жанр буде відпрацьований компонент Колекції з наявними фільмами.

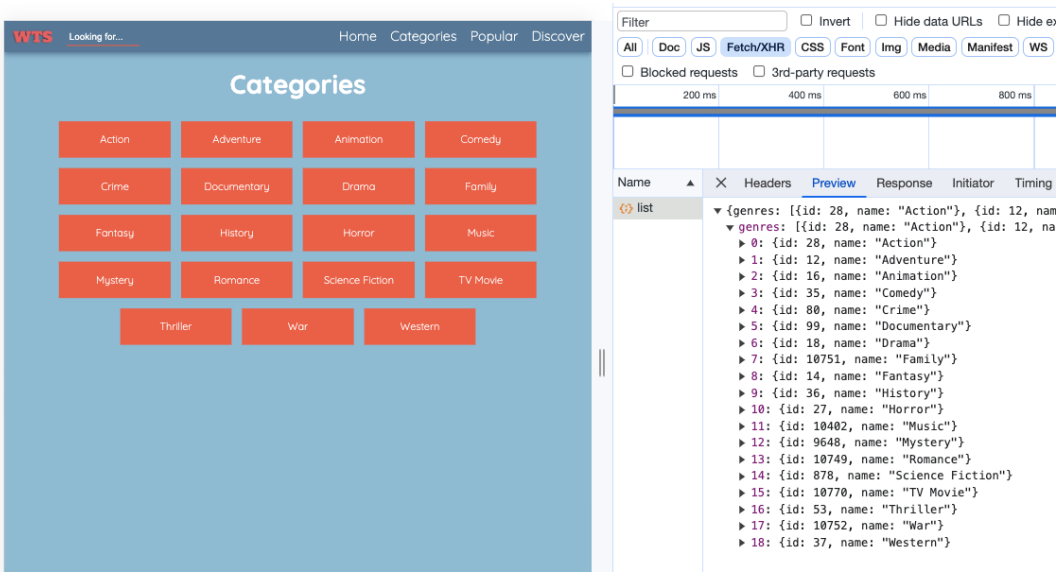

Рисунок 3.16 – Response API запиту категорій та результат

Для відображення колекцій було створено компонент MoviesByCategory. В даному компоненті за допомогою хука ‘useEffect’ виконується API запит. Варто зазначити, що використання lazy loading в моєму випадку передбачає завантаження тільки 20 фільмів та відображення їх за допомогою колекції. У випадку, якщо користувач буде скролити сторінку вниз, я виконаю додатковий API запит на наступні 20 фільмів та вони будуть додані до минулих в стейт компоненту – завдяки цьому ми оптимізуємо роботу сторінки та пришвидшуємо рендеринг. Для даного концепту була використана бібліотека ‘react-infinite-scroll-component’. В даному випадку тег ‘InfiniteScroll’ виступає обгорткою для колекції фільмів.

```
const MoviesByCategory = () => {
  const { category_id } = useParams();
  // eslint-disable-next-line no-unused-vars
  const [idOfCategorie, setIdOfCategorie] = useState(category_id);
  const [filmsOfCategorie, setFilmsOfCategorie] = useState(null);
  const [page, setPage] = useState(1);
  //effect to get assets of categorie
  useEffect(() => {
    if (idOfCategorie) {
      async function moviesOfCategorie(idOfCategorie) {
        const url = categorieDetailTMDB_URL(idOfCategorie, page);
        const dataMovies = await serviceMoviesOfCategorie(url);
        setFilmsOfCategorie(
          filmsOfCategorie
            ? [...filmsOfCategorie, ...dataMovies?.items]
            : dataMovies?.items
        );
      }
      moviesOfCategorie(idOfCategorie);
    }
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [idOfCategorie, page]);

  if (filmsOfCategorie) {
    const filmsArray = getFilms(filmsOfCategorie);

    return (
      <>
        <AssetsWrapper>
          <ScrollComponent
            filmsArray={filmsArray}
            setPage={setPage}
            page={page}
            filmsOfCategorie={filmsOfCategorie}
          />
        </AssetsWrapper>
      </>
    );
  }
  return null;
};
```

Рисунок 3.17 – Компонент колекції та виконання API запиту

Також варто зазначити що кожен раз коли глядач доходить до нижньої частини сторінки я перевіряю, чи є наступна сторінка (тобто наступні 20 фільмів) до завантаження. Це виконується за допомогою Boolean значення ‘hasMore’ та надсилання в обгортку ‘InfiniteScroll’ наступної сторінки до завантаження setPage(page + 1).

Прочитавши код можна помітити використання таких стилізованих компонентів, як:

- Paragraph;
- ParagraphBig;
- ParagraphLeft;
- ParagraphLeftOrange.

Це параграфи які були описані за допомогою Styled Components та код для них був один раз. Надалі вони були використані неодноразово по різних компонентах додатку. То ж ми бачимо ідентичну перевагу використання стилізованих компонентів, як і компонентів React.

Виконавши збереження та виконавши перезавантаження додатку, можемо побачити вигляд колекцій:

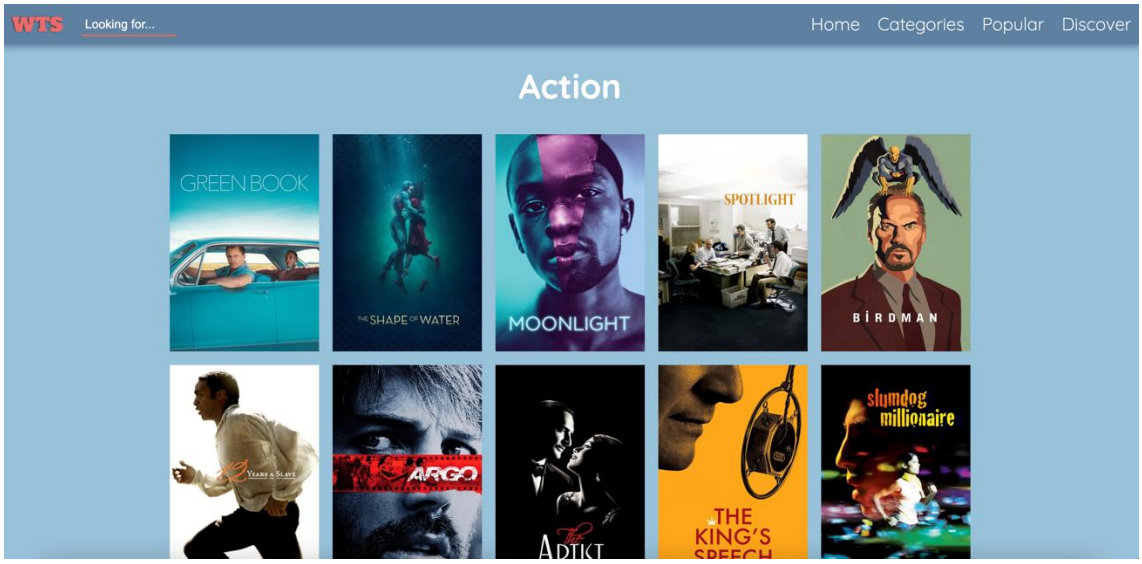

Рисунок 3.18 – Компонент Collection

Наступним кроком реальний глядач повинен обрати зацікавлений фільм із колекції на натиснувши на нього опиниться на детальній сторінці, де зможе

прочитати інформацію про фільм або серіал та почати відтворення відео. Для цього мною було створено базовий компонент детальної сторінки під назвою AssetDetails який потім був перевикористаний для інших компонентів додатку, щоб опинитися на детальній сторінці, таких як галерея, компонентах пошуку та фільтрів, сторінці популярних. В даному компоненті коли він виконується, першочергово отримуючи ‘id’ з URL я виконую запит в хуку useEffect для отримання інформації про фільм. Як тільки прийшов response я записую отримані дані в стейт компонента за допомогою хука useState та починаю парсінг даних для рендерингу їх на сторінці використовуючи Styled Components. Також дуже важливим на цій стрінці є отримання ‘id’ для створеного модального вікна, щоб надалі запустити плейбек (відео).

```
// effect to get data of film
useEffect(() => {
  async function getFilmDetails(url) {
    const dataFilm = await serviceAssetDetail(url);
    setAssetDetail(dataFilm);
  }
  getFilmDetails(assetUrl);
}, [assetUrl]);

const onAssetButtonClick = (e) => {
  e.preventDefault();
  setIsModalShow(true);
};

const onCloseVideoClick = (e) => {
  setIsModalShow(false);
};

if (assetDetail) {
  const imagePath = getImageOfAsset(assetDetail);
  const countries = getCountries(assetDetail);
  return (
    <>
      <AssetWrapper>
        <ImageOfAsset src={imagePath}></ImageOfAsset>

        <Description>
          <Header3>
            {assetDetail?.original_title} (
            {assetDetail?.release_date?.split("-")[0]})
          </Header3>
          <Div>
            <ParagraphLeftOrange>
              Rating:{" "}
              <Span>
                {assetDetail?.vote_average}/10{" "}
                <SpanSmall>({assetDetail?.vote_count} votes)</SpanSmall>
              </Span>
            </ParagraphLeftOrange>
          </Div>
          <ParagraphLeftOrange>
            Release date: <Span>{assetDetail?.release_date}</Span>
          </ParagraphLeftOrange>
          <Div>
            <ParagraphLeftOrange>Countries:</ParagraphLeftOrange> {countries}
          </Div>
          <ParagraphLeftOrange>Overview:</ParagraphLeftOrange>
          <ParagraphLeft>
            {assetDetail?.overview?.substring(0, 350)}...
          </ParagraphLeft>
        </Description>
        <Button onClick={onAssetButtonClick}>Watch Trailer</Button>
        {isModalShow && (
          <ModalWindow
            display={isModalShow}
            assetDetail={assetDetail}
            onCloseVideoClick={onCloseVideoClick}
          />
        )}
      </AssetWrapper>
```

Рисунок 3.19 – Код компоненту детальної сторінки

На рівні з параграфами були описані стилізоавні компоненти для заголовків, опису, кнопок, модальних вікон, галереї та майже всіх компонентів додатку, що каже про зменшену кількість boilerplate коду.

Також були використані utils функціх для, такі як getCountries або getImageOfAsset, для парсингу даних та отримання конкретної інформації. Їх було винесено в окремий файлі файли для зменшення розміру базого компонента. Отриманий вид даних та вигляд детальної сторінки можна побачити на скріншоті:

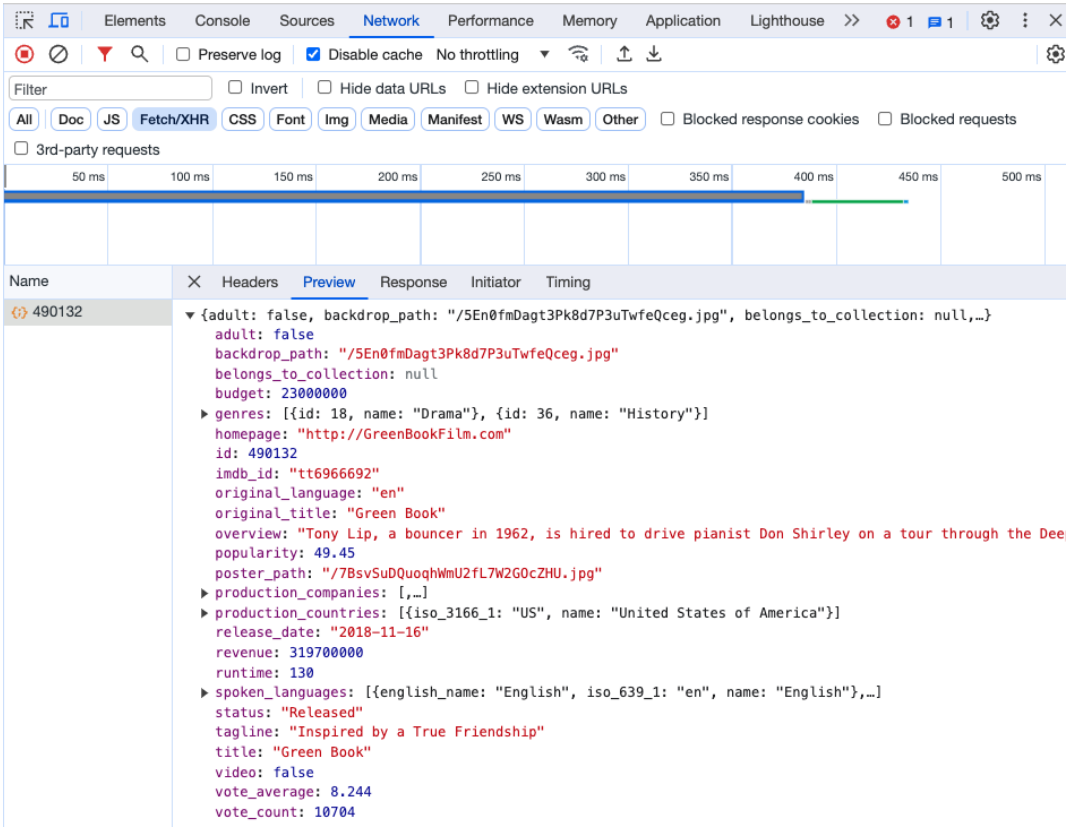

Рисунок 3.20 – Response детальної сторінки

На детальній сторінці також ми можемо побачити кнопку ‘Watch Trailer’. При кліку на дану кнопку буде визваний компонент ModalWindow. Цей компонент є відер вікном - ‘popup’, в якому почнеться відтворення трейлеру даного фільму чи серіалу.

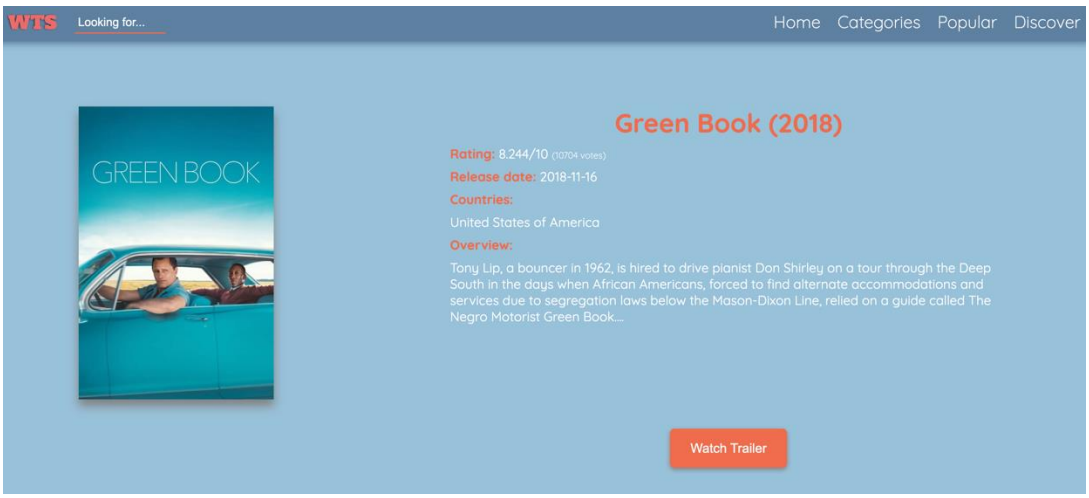

Рисунок 3.21 − Компонент детальної сторінки

3.3.5 Компонент відео відтворення

З вищезазначеної інформації для відтворення відео на детальній сторінці використаний компонент ModalWindow. Принцип роботи даного компоненту полягає в тому, що я отримую ‘id’ фільма чи серіала який хочу подивитись, робимо запит в хуку useEffect з цим ‘id’ для отримання наступного ‘id’ який буде використаний для відтворення даного відео за допомогою платформи youtube. При отриманні даного ‘id’ я використовую HTML елемент ‘iframe’ та в property ‘src’ додаємо посилання на youtube, та за допомогою літералів ‘Template literals’ додаємо отриманий id в URL та рендерим компонент.

Вбудований фрейм (iframe) — це елемент HTML, який містить іншу сторінку HTML. По суті, він вбудовує іншу сторінку в батьківську сторінку. Реклама, вбудовані фільми, онлайн-аналітика та інтерактивний вміст — все це часто використовується для них. iframe підтримують усі основні веб-браузери, і він включений до найновіших стандартів HTML5. Коли веб-браузер знаходить елемент iframe, він створює нове середовище HTML-документа, у якому завантажується вміст. Він створює код із пов’язаного src або srcdoc як власну веб-сторінку, яка потім повністю розміщується на батьківській сторінці перегляду. Оскільки він виглядає користувачем як одна веб-сторінка, його називають вбудованим фреймом.

Також використані Літерали шаблонів (Template Literals) — це літерали, розділені символами зворотної галочки (`), які дозволяють використовувати

багаторядкові рядки, інтерполяцію рядків із вбудованими виразами та шаблони з тегами. Шаблонні літерали також відомі як шаблонні рядки, оскільки вони широко використовуються для інтерполяції рядків (створення рядків шляхом заміни заповнювачів). Літерал шаблону з тегами, з іншого боку, може не створити рядок; натомість його можна використовувати з користувальницькою функцією тегу, щоб виконувати будь-які дії, які ви хочете, з різними елементами літералу шаблону.

```
const ModalWindow = ({ display, assetDetail, onCloseVideoClick }) => {
  const trailerId = assetDetail?.id;
  const videosOfFilmYoutubeUrl = getTrailerTMDB_URL(trailerId);
  const [keyOfVideos, setKeyOfVideos] = useState(null);

  useEffect(() => {
    async function getKeyForYoutube(url) {
      const data = await serviceKeyForYoutube(url);
      setKeyOfVideos(data);
    }
    getKeyForYoutube(videosOfFilmYoutubeUrl);
  }, [videosOfFilmYoutubeUrl]);

  if (display && keyOfVideos) {
    return (
      <ModalWrapper onClick={onCloseVideoClick}>
        <VideoWrapper>
          <Iframe
            title="video"
            id="ytplayer"
            type="text/html"
            frameBorder="0"
            src={`http://www.youtube.com/embed/${keyOfVideos}`}
            autoPlay="1"
            crossOrigin="anonymous"
          ></Iframe>
          <CloseImage src={closeImage} onClick={onCloseVideoClick} />
        </VideoWrapper>
      </ModalWrapper>
    );
  }
  return null;
};
```

Рисунок 3.22 – Компонент відтворення відео

Платформа youtube використовує потокове передавання з адаптивним бітрейтом MPEG-DASH, використання якого було проаналізовано мною, та є ідеальним кейсом використання в моїй стримінговій платформі. Адаптивна зміна якості відео, яка залежить від типу девайсу, пропускної можливості, швидкості Інтернета та використання браузера – є запорукою гарних відгуків від реальних глядачів.

Також варто зазначити, що можливе використання стримінгових сервісів на комерційній основі, тобто наприклад для легального відтворення повноцінного фільму чи серіалу, потрібно буде лише змінити ‘id’ та URL

компоненту ModalWindow та при кліку на кнопку Watch на детальній сторінці буде відтворено повноцінний відео контент.

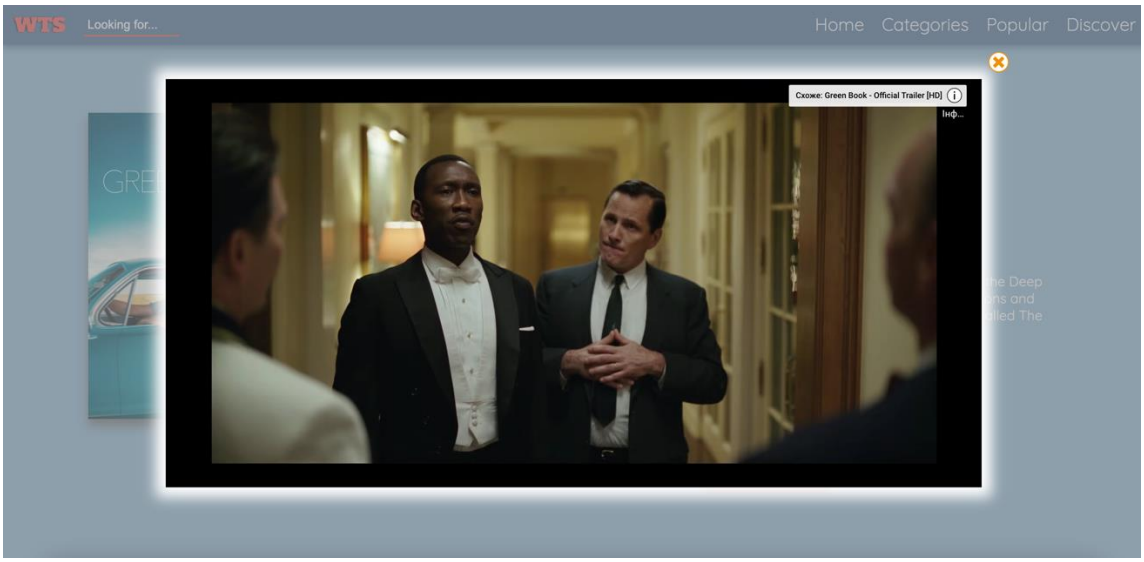

Рисунок 3.23 – Детальна сторінка з відтворенням відео

3.3.6 Пошук

Пошук може бути виконаний з будь якої сторінки даної стримінгової платформи. Для цього в Header був доданий стилізований HTML тег input та інформація про нього ‘Looking for…’. Натиснувши на даний елемент меню глядач побачить визване popup вікно поверх сторінки на якій він знаходиться. Почавши писати заголовок фільму чи серіалу відбудеться рендер колекції елементів. Логіка даного компоненту була описана в компоненті NavBar та створені для рендерингу компоненти SearchInput та SearchInputPop. Концепт роботи полягає у створенні підписки лісенером на клік глядача. Як тільки буде введений символ виконується запит на бекенд та ми отримуємо колекцію фільмів чи серіалів по даному запиту. Також мною було додано затримку за допомогою функції setTimeout у 500 мілісекунд між клікам глядача для більш плавної роботи пошуку. Наступним кроком отримані дані додаються в стейт компонента та відображаються в рендері компонента.

```
const onInputChange = (e) => {
  setInputValue(e.target.value);
  if (e.target.value) {
    setPopupShow(true);
  } else {
    setPopupShow(false);
    inputRef.current.value = "";
  }
};

useEffect(() => {
  if (inputValue) {
    setTimeout(() => {
      async function getSearchRequest() {
        const url = searchTMDB_URL(inputValue);
        const data = await serviceInputSearch(url);
        setFilmsOnInput(data);
      }
      getSearchRequest();
      window.addEventListener("click", onInputChange);
    }, 500);
  }
  return () => {
    window.removeEventListener("click", onInputChange);
  };
}, [inputValue]);

  return (
    <>
      <Nav>
        <ImageLogo src={Logo} alt="logo"></ImageLogo>

        {popupShow && filmsOnInput ? (
          <SearchInputPopup filmsOnInput={filmsOnInput} />
        ) : null}

        <SearchInput onInputChange={onInputChange} inputRef={inputRef} />
        <Ul>{lists}</Ul>
      </Nav>
    </>
  );
}
return null;
```

Рисунок 3.24 – Код пошуку фільмів та серіалів

При кліку на будь який фільм відкриється відповідна детальна сторінка, а для того щоб закрити пошукове вікно треба виконати клік на будь якій довільній зоні додатку.

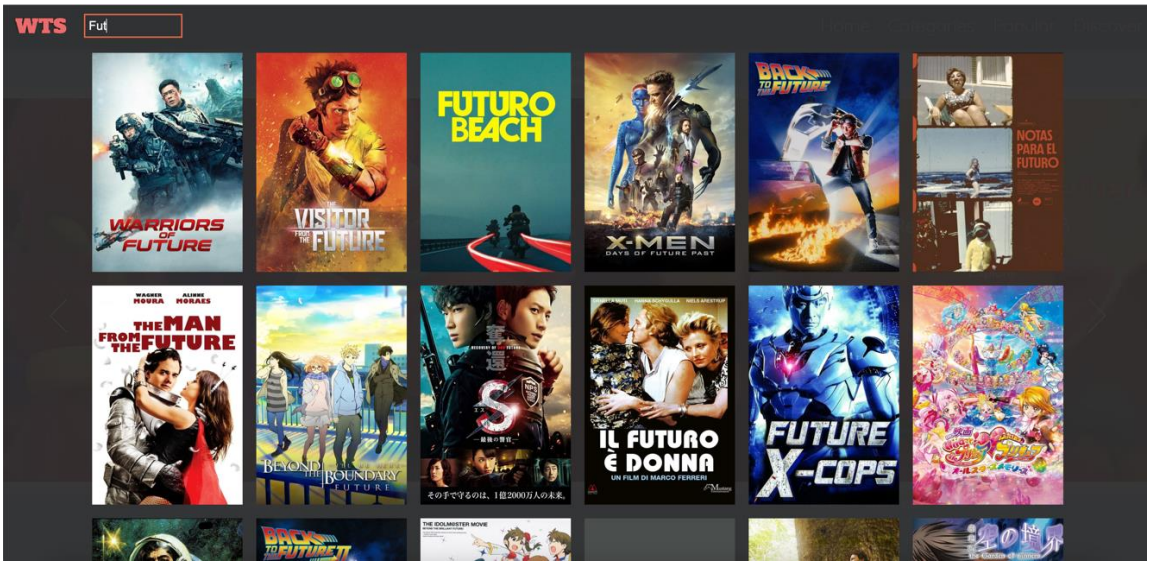

Рисунок 3.25 – Колекція фільмів в пошуку

3.4 Розгортання платформи

Мною було обрано хмарну платформу Netlify для розгортання моєї стримінгової платформи. Netlify — це популярна хмарна платформа серед розробників для створення високопродуктивних і динамічних веб-сайтів, компаній електронної комерції та онлайн-додатків.

Netlify полегшує розробникам розгортання та розміщення веб-сайтів. Дана платформа створює власний репозиторій і передає його на Github, а також власні мікросервіси. Потім він виконує та розповсюджує матеріал через велику CDN, щоб надати відвідувачам попередньо створені статичні веб-сторінки. Найприємніша частина полягає в тому, що платформа вибирає найкращий CDN і поширює вміст через нього. Як наслідок, попередньо створені веб-сайти завантажуються швидше, ніж веб-сайти в типових мережах хостингу. Замість того, щоб перезавантажувати сайт під час кожного відвідування, відвідувач отримує попередньо завантажену версію з найближчого сервера. Це значно скорочує час завантаження.

Першим кроком мною було виконано команду ‘npm run build’ в терміналі VSCode для того щоб зробити фінальний білд додатку.

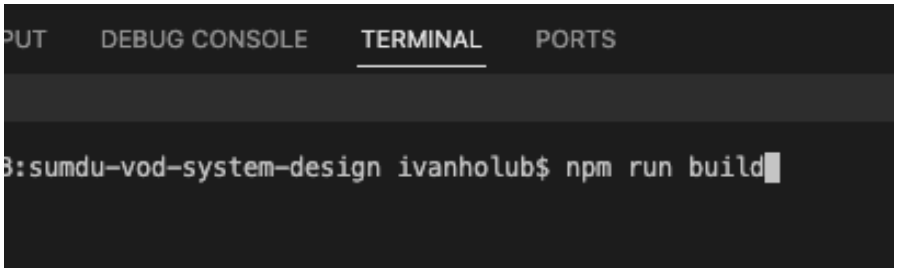

Рисунок 3.26 − Створення білда React проекту

Наступним кроком переносимо папку build з проектом на платформу Netlify у розділ Builds, через декілька хвилин проект буде доступним по відповідному отриманому URL, який надалі можна змінити.

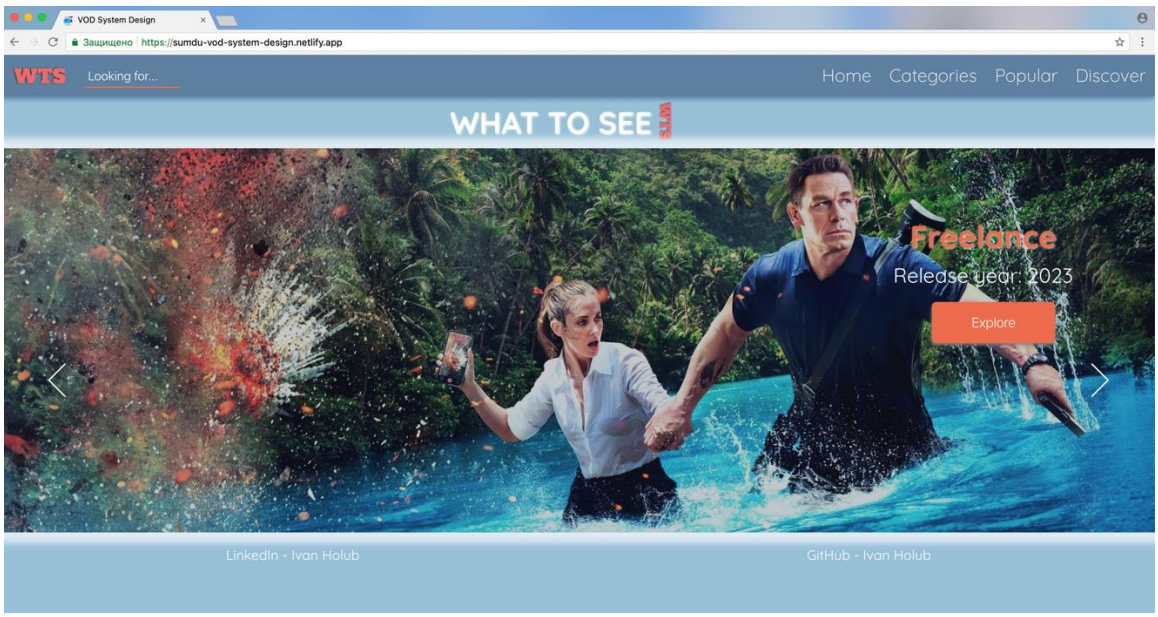

Рисунок 3.27 – Розгорнутий проект What To See

3.5 Тестування платформи: StreamTest та Google Lighthouse

Мною було виконано тестування додатку за допомогою двох платформ, StreamTest для аналізу стримінгу, та Google LightHouse для тестування якості веб-сторінок.

StreamTest від Fora Soft – це додаток для браузера. Він сумісний із будь-якою програмою відеочату WebRTC, зокрема Google Meet, iMind і ProVideoMeeting. Аналіз відбувається такими характеристиками:

- Частота кадрів, часто відома як FPS (чим більше, тим краще).

- Затримка відео та аудіо: затримка ніколи не є бажаною (чим менша, тим краще).

- Втрата пакетів: вказує на те, чи були втрачені пакети на шляху до сервера (чим менше число, тим краще).

- Роздільна здатність: це значною мірою визначається камерою учасника та гаджетом, який використовується для розмови (чим вище, тим краще).

- Зависання та зупинки: відображає відсоток вашого виклику, який було втрачено через затримки (чим менше число, тим краще).

- Бітрейт: визначає, наскільки плавне зображення (чим більше, тим краще).

- Відео та аудіокодеки.

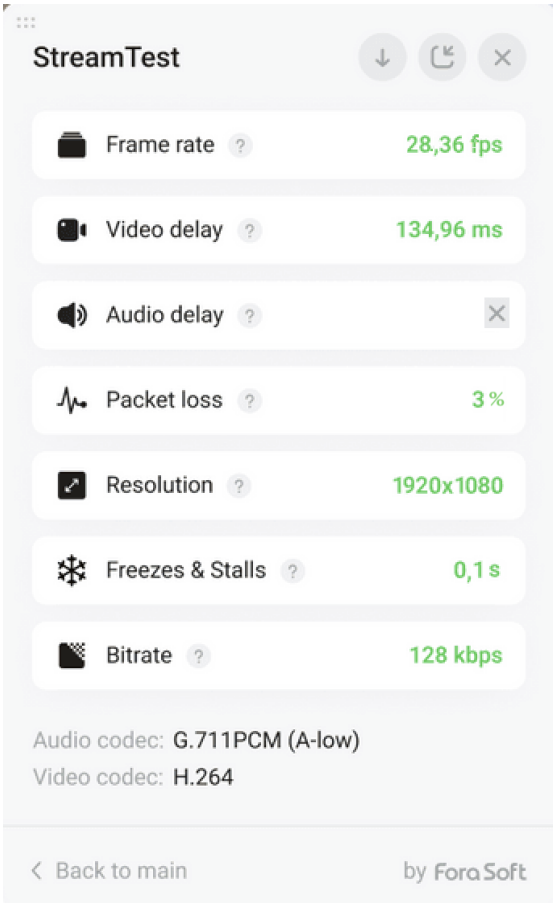

Рисунок 3.28 – Тестування за допомогою StreemTest

Отримані результати свідчать про вірне використання даного потокового передавання з адаптивним бітрейтом MPEG-DASH, так як видно з отриманих результатів - затримка відео мінімальна, присутня гарна якість та бітрейт, та наявна висока частота кадрів.

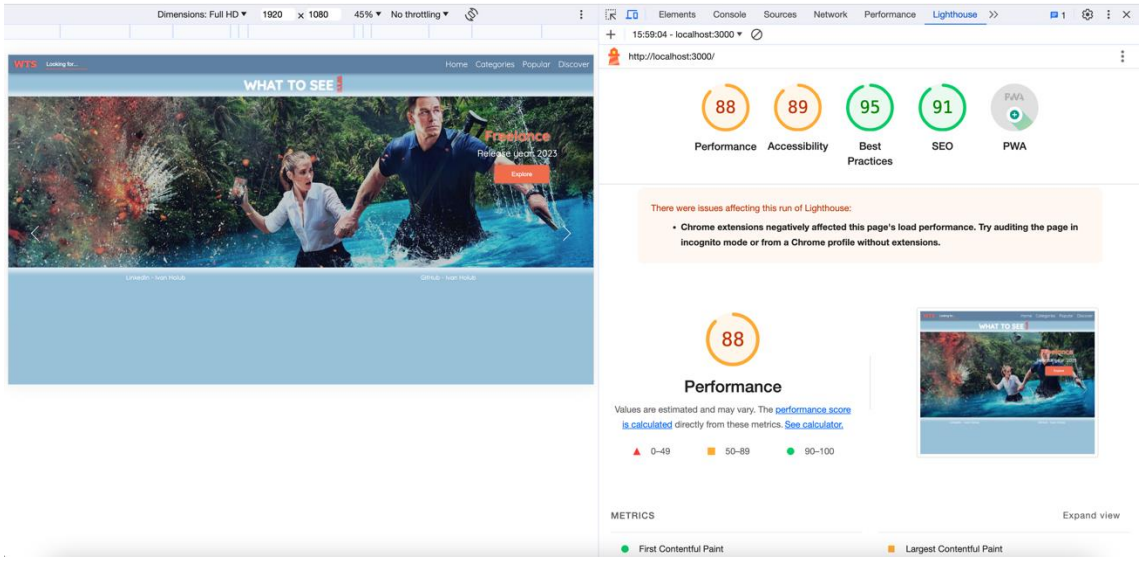

Рисунок 3.29 – Тестування за допомогою Google Lighthouse

Після тестування стримінгового додатку за допомогою Google Lighthouse можна також побачити чудові отримані результати. Додаток має дуже гарний performance та доступність, а використання найкращих практик розробки рівне 95 відсоткам.

ВИСНОВКИ

В процесі виконаної роботи, мною було розроблено інформаційну технологію проєктування системи відео на замовлення. Для відтворення відео було використано техніку потокового передавання з адаптивним бітрейтом MPEG-DASH. Платформа має гарний перформанс, інтуїтивний функціонал та при розробці були враховані сучасні техніки UX/UI для приємного користування додатком.

Виконані етапи при розробці додатка:

- Пошук, аналіз та дослідження всієї необхідної інформації;
- Було проведено аналіз протоколів потокового передавання відео та проведено аналіз додатків конкурентів;
- Обрано засоби реалізації додатка (мова програмування, плагіни та розширення);
- Реалізація платформи за допомогою HTML, CSS, SASS, Styled Components, JavaScript, React.JS, React Router;
- Розгортання додатка (deploy);
- Тестування фінальної версії додатку та аналіз отриманих результатів;

Головною задачею була розробка платформи, яка буде виконана з урахуванням усіх сучасних тенденцій UI/UX, мати гарний performance та якісний відео контент.

Завдяки розробці цієї платформи глядачі можуть знаходити популярні фільми та серіали використовуючи пошук та пошукові фільтри, ознайомлюватись з інформацію про актуальні кіно новинки та будь які види відео контенту, дивитись ознайомлювальні відео. В додаток до цього платформа досить швидко працює (гарний перформанс), використані практики UX/UI, що додає безліч переваг для комфортного користування платформою в порівнянні з конкурентами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Content dependent spatial resolution selection for MPEG DASH segmentation / Jelena Vlaović, Snježana Rimac-Drlje, Drago Žagar, Luka Filipović // Journal of Industrial Information Integration. – 2021. - № 24. - P. 12-19.

2. Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing / Tyler Akidau // 2018. – P. 25 – 72.

3. Visualizing Streaming Data: Interactive analysis beyond static limits / Anthony Aragues // 2018. – P. 78 – 102.

4. Streaming Video (Critical Cultural Communication) / Amanda D. Lotz, Ramon Lobato // 2023. – P. 19 – 58.

5. HTML5 and the evolution of HTML; tracing the origins of digital platforms / Raúl Tabarés // Technology in Society. – 2021. - № 65. - P. 15-22.

6. Responsive Web Design with HTML5 and CSS / Ben Frain // 2022. – P. 37 – 61.

7. Eloquent JavaScript - 3rd edition / Marijn Haverbeke // 2018. – P. 56 – 98.

8. Learning React: Functional Web Development with React and Redux / Alex Banks, Eve Porcello // 2017. – P. 35 – 88.

9. React Router Quick Start Guide: Routing in React applications made easy / Sagar Ganatra // 2018. – P. 12 – 43.

10. HLS Vs. DASH: Which Streaming Protocol is Right for You? / Rahul Nanwani // 2023. - https://imagekit.io/blog/hls-vs-dash/

11. Learning React: Modern Patterns for Developing React Apps / Alex Banks, Eve Porcello // 2020. – P. 127 – 178.

12. RESTful Web API Patterns and Practices Cookbook: Connecting and Orchestrating Microservices and Distributed Data / Mike Amundsen // 2022. – P. 47 – 82.

13. Learning Node.js Development / Andrew Mead // 2018. P. 18 – 55.

14. JavaScript from Beginner to Professional: Learn JavaScript quickly by

building fun, interactive, and dynamic web pages / Rob Percival // 2021. – P. 51 – 88.

15. Hands-On Visual Studio 2022: A developer's guide to exploring new features and best practices in VS2022 for maximum productivity / Miguel Angel Teheran Garcia, Hector Uriel Perez Rojas // 2022. – P. 35 – 78.

16. Better Data Visualizations: A Guide for Scholars, Researchers, and Wonks / Jonathan Schwabish // 2021. – P. 128 – 157.

17. Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker / Frank Zammetti // 2020. – P. 145 – 178.

18. CSS in Depth, First Edition / Keith J. Grant // 2018. – P. 34-78.

19. Learn React Hooks: Build and refactor modern React.js applications using Hooks / Daniel Bugl // 2019. – P. 227 – 274.

20. Video Coding and Online Streaming Technologies: Principles and Practice of VVC, AV1, HEVC, AVC, HLS, MPEG-DASH, and MSS / Benny Bing // 2022. – P. 38 – 89.

21. Head First Design Patterns. Software 2nd Edition / Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra // 2020. – P. 81 – 114.

22. React Cookbook: Recipes for Mastering the React Framework / David Griffiths, Dawn Griffiths // 2021. – P. 76 – 98.

23. Sass for Beginners: The Ultimate Beginner's Guide to Learn Sass Step by Step / John Bach // 2020. – P 42 – 67.

24. React in Action, First Edition / Mark Tielens Thomas // 2018. – P. 121 – 142.

25. Optimize Video Streaming Delivery / Derek DeJonghe // 2021. – P. 67- 102.

26. Beginning React with Hooks / Greg Lim // 2020. – P. – 101 – 110.

27. React Key Concepts: Consolidate your knowledge of React’s core features / Maximilian Schwarzmuller // 2021. – P. 342 – 378.

ДОДАТОК А

Файл index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/App';
import GlobalStyles from './styles/globalStyle';

ReactDOM.render(
<>
 <GlobalStyles />
 <App />
</>,
document.getElementById('root')
);
```

Файл App.js

```
import React from "react";
import { ThemeProvider } from "styled-components";
import { BrowserRouter as Router, Switch, Route } from
"react-router-dom";
import NavBar from "./NavBar/NavBar";
import theme from "./../styles/theme";
import { Wrapper } from "./../styles/StyledComponents";
import NotFound from "./NotFound/NotFound";
import getProperComponentData from
"./../utils/getProperComponentData";
import { menuItems, routes } from "./../utils/Constants";

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      menu: menuItems,
      routes: routes,
    };
  }

  render() {
    return (
      <ThemeProvider theme={theme}>
        <Wrapper>
          <Router>
            <NavBar menu={this.state?.menu ? this.state?.menu
: false} />
            <Switch>
              {this.state?.routes &&
                this.state?.routes?.map((el) => {
                  const { id, exact, route, Component } =
                    getProperComponentData(el);
```

```
                    return (
                      <Route
                        key={id}
                        exact={exact}
                        path={route}
                        component={Component}
                      />
                    );
                  })}
                <Route path="*" component={NotFound} />
              </Switch>
            </Router>
          </Wrapper>
        </ThemeProvider>
      );
    }
  }

  export default App;
```

Файл globalStyles.js

```
import {createGlobalStyle} from 'styled-components';

const GlobalStyles = createGlobalStyle`
    *,
    *::after,
    *::before {
        margin: 0;
        padding: 0;
        box-sizing: inherit;
    }

    html {
        // This defines what 1rem is
        font-size: 62.5%; //1 rem = 10px; 10px/16px = 62.5%
        font-family: 'Quicksand', sans-serif;
    }

    body {
        box-sizing: border-box;
    }
`;

export default GlobalStyles;
```

Файл AssetDetails.js

```
import React, { useState, useEffect } from "react";
```

```
import { useParams } from "react-router";
import serviceAssetDetail from
"./../Services/serviceAssetDetail";
import {
  AssetWrapper,
  ImageOfAsset,
  Description,
  Header3,
  ParagraphLeft,
  ParagraphLeftOrange,
  Button,
  Div,
  Span,
  SpanSmall,
} from "./../../styles/StyledComponents";
import ModalWindow from "./../ModalWindow/ModalWindow";
import getImageOfAsset from "./../../utils/ImageOfAsset";
import { getDetailsTMDB_URL } from "./../../url/url";

function getCountries(el) {
  return el?.production_countries?.map((el) => {
    return <ParagraphLeft
key={el?.name}>{el?.name}</ParagraphLeft>;
  });
}

const AssetDetails = () => {
  const { movie_id } = useParams();
  const assetUrl = getDetailsTMDB_URL(movie_id);

  const [assetDetail, setAssetDetail] = useState(null);
  const [isModalShow, setIsModalShow] = useState(false);

  // effect to get data of film
  useEffect(() => {
    async function getFilmDetails(url) {
      const dataFilm = await serviceAssetDetail(url);
      setAssetDetail(dataFilm);
    }
    getFilmDetails(assetUrl);
  }, [assetUrl]);

  const onAssetButtonClick = (e) => {
    e.preventDefault();
    setIsModalShow(true);
  };

  const onCloseVideoClick = (e) => {
    setIsModalShow(false);
  };

  if (assetDetail) {
```

```
    const imagePath = getImageOfAsset(assetDetail);
    const countries = getCountries(assetDetail);
    return (
      <>
        <AssetWrapper>
          <ImageOfAsset src={imagePath}></ImageOfAsset>

          <Description>
            <Header3>
              {assetDetail?.original_title} (
              {assetDetail?.release_date?.split("-")[0]})
            </Header3>
            <Div>
              <ParagraphLeftOrange>
                Rating:{" "}
                <Span>
                  {assetDetail?.vote_average}/10{" "}
                  <SpanSmall>({assetDetail?.vote_count}
votes)</SpanSmall>
                </Span>
              </ParagraphLeftOrange>
            </Div>
            <ParagraphLeftOrange>
              Release date:
<Span>{assetDetail?.release_date}</Span>
            </ParagraphLeftOrange>
            <Div>

<ParagraphLeftOrange>Countries:</ParagraphLeftOrange>
{countries}
            </Div>

<ParagraphLeftOrange>Overview:</ParagraphLeftOrange>
            <ParagraphLeft>
              {assetDetail?.overview?.substring(0, 350)}...
            </ParagraphLeft>
          </Description>
          <Button onClick={onAssetButtonClick}>Watch
Trailer</Button>
          {isModalShow && (
            <ModalWindow
              display={isModalShow}
              assetDetail={assetDetail}
              onCloseVideoClick={onCloseVideoClick}
            />
          )}
        </AssetWrapper>
      </>
    );
  } else {
    return null;
  }
```

```
};

export default AssetDetails;
```

Файл AssetInCategorie.js

```
import React from "react";
import { Link } from "react-router-dom";
import {
  PopularFilm,
  Image,
  DescriptionPopular,
  Paragraph,
  ParagraphBig,
} from "../../styles/StyledComponents";
import "./AssetInCategorie.scss";
import getImageOfAsset from "./../../utils/ImageOfAsset";

const AssetInCategorie = ({ filmData }) => {
  const path = `/movie/${filmData?.id}`;
  return (
    <PopularFilm>
      <Link to={path} className="film-asset">
        <Image src={getImageOfAsset(filmData)} alt="no-
image"></Image>
        <DescriptionPopular>

<ParagraphBig>{filmData?.original_title}</ParagraphBig>
          <Paragraph>Rating:
{filmData?.vote_average}/10</Paragraph>
          <Paragraph>Date:
{filmData?.release_date}</Paragraph>
        </DescriptionPopular>
      </Link>
    </PopularFilm>
  );
};

export default AssetInCategorie;
```

Файл Categories.js

```
import React, { useState, useEffect } from "react";
import serviceCategories from
"./../Services/serviceCategories";
import { categoriesTMDB_Url } from "./../../url/url";
import {
  Header1,
  CategoriesWrapper,
  Categorie,
  CategorieName,
} from "./../../styles/StyledComponents";
```

```
function getArrayOfCategories(el) {
  return el?.map((el) => {
    const path = `/movies/${el?.id}`;
    return (
      <Categorie key={el?.id} data-id={el?.id} data-
name={el?.name}>
        <CategorieName to={path}>{el?.name}</CategorieName>
      </Categorie>
    );
  });
}

const Categories = () => {
  const [categoriesData, setCategoriesData] = useState(null);

  // effect to get categories of movies
  useEffect(() => {
    (async function getCategories() {
      const dataCategories = await
serviceCategories(categoriesTMDB_Url);
      setCategoriesData(dataCategories);
    })();
  }, []);

  if (categoriesData) {
    const arrayOfCategoriest =
getArrayOfCategories(categoriesData);

    return (
      <>
        <Header1>Categories</Header1>

<CategoriesWrapper>{arrayOfCategoriest}</CategoriesWrapper>
      </>
    );
  } else {
    return null;
  }
};

export default Categories;
```

Файл DataLoader.js

```
export default class DataLoader {
  constructor(url, param) {
    this.url = url;
    this.param = param;
  }
  async get() {
    try {
      const request = await fetch(this.url, {
        method: "GET",
        headers: {
          Accept: "application/json",
          "Content-Type": "application/json",
        },
      });
      const data = await request.json();
      return data;
    } catch (error) {
      console.log(error);
    }
  }

  async post() {
    try {
      const request = await fetch(this.url, {
        method: "POST",
        headers: {
          Accept: "application/json",
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          filters: this.param,
        }),
      });
      const data = await request.json();
      return data;
    } catch (e) {
      console.log(e);
    }
  }
}
```

Файл Discover.js

```
import React, { useState, useEffect } from "react";
import { discoverTMDB_URL, categoriesTMDB_Url } from
"./../../url/url";
import serviceDiscovery from
"./../Services/serviceDiscovery";
import { AssetsWrapper, Header1, Form, Paragraph,
InputDiscover, Select, DivCentered,
```

```
} from "./../../styles/StyledComponents";
import AssetInCategorie from
"./../AssetInCategorie/AssetInCategorie";
import serviceCategories from
"./../Services/serviceCategories";
import getOptionsOfSelect from
"./../../utils/getOptionsOfSelect";
import { getDiscoverURL } from "./../../utils/utils";

function getAssetsOfFilms(data) {
  return data?.map((el) => {
    return <AssetInCategorie filmData={el} key={el?.id} />;
  });
}

const Discover = () => {
  const [filmsData, setFilmsData] = useState(null);
  const [optionsOfSelect, setOptionsOfSelect] =
useState(null);
  const [params, setParams] = useState({
    "vote_average.gte": "",
    "vote_average.lte": "",
    "release_date.gte": "",
    "release_date.lte": "",
    with_genres: [],
    without_genres: [],
  });

  useEffect(() => {
    async function getDiscover() {
      const data = await serviceDiscovery(
        getDiscoverURL(discoverTMDB_URL, params)
      );
      const categories = await
serviceCategories(categoriesTMDB_Url);
      setFilmsData(data);
      setOptionsOfSelect(categories);
    }
    getDiscover();
  }, [params]);

  const onInputChange = (e) => {
    setParams({
      ...params,
      [e.target.getAttribute("name")]: e.target.value,
    });
  };

  const onSelectChange = (e) => {
    if
(!params[e.target.getAttribute("name")].includes(e.target.val
ue)) {
```

```
      setParams({
        ...params,
        [e.target.getAttribute("name")]: [
          ...params[e.target.getAttribute("name")],
          e.target.value,
        ],
      });
    }
  };

  if (filmsData && optionsOfSelect) {
    const films = getAssetsOfFilms(filmsData);
    const options = getOptionsOfSelect(optionsOfSelect);
    return (
      <>
        <Header1>Discover world movies</Header1>
        <Form>
          <DivCentered>
            <Paragraph>Rating</Paragraph>
            <InputDiscover
              name="vote_average.gte"
              placeholder="From"
              type="number"
              onChange={onInputChange}
            />
            <InputDiscover
              name="vote_average.lte"
              placeholder="To"
              type="number"
              onChange={onInputChange}
            />
          </DivCentered>
          <DivCentered>
            <Paragraph>Release Date</Paragraph>
            <InputDiscover
              name="release_date.gte"
              placeholder="From"
              type="number"
              onChange={onInputChange}
            />
            <InputDiscover
              name="release_date.lte"
              placeholder="To"
              type="number"
              onChange={onInputChange}
            />
          </DivCentered>
          <DivCentered>
            <Paragraph>Include Genres</Paragraph>
            <Select name="with_genres"
onChange={onSelectChange} multiple>
              <option value hidden>
```

```
                Choose..
              </option>
              {options}
            </Select>
          </DivCentered>
          <DivCentered>
            <Paragraph>Exclude Genres</Paragraph>
            <Select name="without_genres"
onChange={onSelectChange} multiple>
              <option value hidden>
                Choose..
              </option>
              {options}
            </Select>
          </DivCentered>
        </Form>
        <AssetsWrapper>{films}</AssetsWrapper>
      </>
    );
  }
  return null;
};

export default Discover;
```

Файл Home.js

```
import React, { useState, useEffect } from "react";
import { Header2, ImageRotated } from
"./../../styles/StyledComponents";
import { popularHomePageTMDB_URL } from "./../../url/url";
import servicePopular from "./../Services/servicePopular";
import SliderBox from "./../SliderBox/SliderBox";
import Logo from "./../../img/WTS.png";
import SummaryLinks from "./../SummaryLinks/SummaryLinks";

const Home = () => {
  const [popularFilms, setPopularFilms] = useState(null);

  useEffect(() => {
    async function getPopularFilms(url) {
      const getFilms = await servicePopular(url);
      setPopularFilms(getFilms);
    }
    getPopularFilms(popularHomePageTMDB_URL);
  }, []);

  if (popularFilms) {
    return (
      <>
        <Header2 style={{ color: "#fff" }}>
          WHAT TO SEE
```

```
          <ImageRotated src={Logo} />
        </Header2>
        <SliderBox popularFilms={popularFilms} />
        <SummaryLinks />
      </>
    );
  }
  return null;
};

export default Home;
```

Файл ModalWindow.js

```
import React, { useState, useEffect } from "react";
import { ModalWrapper, Iframe, CloseImage, VideoWrapper,
} from "./../../styles/StyledComponents";
import serviceKeyForYoutube from
"./../Services/serviceKeyForYoutube";
import closeImage from "./../../img/close.png";
import { getTrailerTMDB_URL } from "./../../url/url";

const ModalWindow = ({ display, assetDetail,
onCloseVideoClick }) => {
const trailerId = assetDetail?.id;
const videosOfFilmYoutubeUrl = getTrailerTMDB_URL(trailerId);
const [keyOfVideos, setKeyOfVideos] = useState(null);

  useEffect(() => {
    async function getKeyForYoutube(url) {
      const data = await serviceKeyForYoutube(url);
      setKeyOfVideos(data);
    }
    getKeyForYoutube(videosOfFilmYoutubeUrl);
  }, [videosOfFilmYoutubeUrl]);

  if (display && keyOfVideos) {
    return (
      <ModalWrapper onClick={onCloseVideoClick}>
        <VideoWrapper>
          <Iframe
            title="video"
            id="ytplayer"
            type="text/html"
            frameBorder="0"

src={`http://www.youtube.com/embed/${keyOfVideos}`}
            autoPlay="1"
            crossOrigin="anonymous"
          ></Iframe>
          <CloseImage src={closeImage}
onClick={onCloseVideoClick} />
```

```
        </VideoWrapper>
      </ModalWrapper>
    );
  }
  return null;
};

export default ModalWindow;
```

Файл MoviesByCategorie.js

```
import React, { useState, useEffect } from "react";
import { useParams } from "react-router";
import AssetInCategorie from
"../AssetInCategorie/AssetInCategorie";
import serviceMoviesOfCategorie from
"./../Services/serviceMoviesOfCategorie";
import { AssetsWrapper, Header1 } from
"./../../styles/StyledComponents";
import ScrollComponent from
"./../ScrollComponent/ScrollComponent";
import serviceCategories from
"./../Services/serviceCategories";
import { categoriesTMDB_Url, categorieDetailTMDB_URL } from
"./../../url/url";
import getProperlyNameOfCategorie from
"./../../utils/NameOfCategoryUtil";

function getFilms(el) {
return el?.map((el) => {
  return <AssetInCategorie filmData={el} key={el?.id} />;
  });
}

const MoviesByCategory = () => {
const { category_id } = useParams();

// eslint-disable-next-line no-unused-vars
const [idOfCategorie, setIdOfCategorie] =
useState(category_id);
const [filmsOfCategorie, setFilmsOfCategorie] =
useState(null);
const [page, setPage] = useState(1);
const [nameOfCategorie, setNameOfCategorie] =
useState("AAA");

  //effect to get assets of categorie
  useEffect(() => {
    if (idOfCategorie) {
      async function moviesOfCategorie(idOfCategorie) {
        const url = categorieDetailTMDB_URL(idOfCategorie,
page);
```

```
        const dataMovies = await
serviceMoviesOfCategorie(url);
        setFilmsOfCategorie(
          filmsOfCategorie
            ? [...filmsOfCategorie, ...dataMovies?.items]
            : dataMovies?.items
        );
      }
      moviesOfCategorie(idOfCategorie);
    }
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [idOfCategorie, page]);

  //effect to get name of categorie
  useEffect(() => {
    if (idOfCategorie) {
      (async function getCategories() {
        const dataCategories = await
serviceCategories(categoriesTMDB_Url);
        const getProperlyName = getProperlyNameOfCategorie(
          dataCategories,
          idOfCategorie
        );
        setNameOfCategorie(getProperlyName);
      })();
    }
  }, [idOfCategorie]);

  if (filmsOfCategorie) {
    const filmsArray = getFilms(filmsOfCategorie);

    return (
      <>
        <Header1>{nameOfCategorie}</Header1>
        <AssetsWrapper>
          <ScrollComponent
            filmsArray={filmsArray}
            setPage={setPage}
            page={page}
            filmsOfCategorie={filmsOfCategorie}
          />
        </AssetsWrapper>
      </>
    );
  }
  return null;
};
export default MoviesByCategory;
```

Файл NavBar.js

```
import React, { useState, useEffect, useRef } from "react";
import { Nav, Ul } from "./../../styles/StyledComponents";
import "./NavBar.scss";
import { List, ImageLogo } from
"../../styles/StyledComponents";
import { NavLink } from "react-router-dom";
import Logo from "./../../img/WTS.png";
import SearchInput from "./../SearchInput/SearchInput";
import { searchTMDB_URL } from "./../../url/url";
import serviceInputSearch from
"./../Services/serviceInputSearch";
import SearchInputPopup from
"./../SearchInputPopup/SearchInputPopup";

const NavBar = ({ menu }) => {
  const inputRef = useRef();
  const [inputValue, setInputValue] = useState(null);
  const [popupShow, setPopupShow] = useState(false);
  const [filmsOnInput, setFilmsOnInput] = useState(null);

  const onInputChange = (e) => {
    setInputValue(e.target.value);
    if (e.target.value) {
      setPopupShow(true);
    } else {
      setPopupShow(false);
      inputRef.current.value = "";
    }
  };

  useEffect(() => {
    if (inputValue) {
      setTimeout(() => {
        async function getSearchRequest() {
          const url = searchTMDB_URL(inputValue);
          const data = await serviceInputSearch(url);
          setFilmsOnInput(data);
        }
        getSearchRequest();
        window.addEventListener("click", onInputChange);
      }, 500);
    }
    return () => {
      window.removeEventListener("click", onInputChange);
    };
  }, [inputValue]);

  if (menu) {
    const lists = menu?.map((el) => {
```

```
        return (
          <List key={el?.id}>
            <NavLink to={el?.route} className="nav__link">
              {el?.label}
            </NavLink>
          </List>
        );
      });

      return (
        <>
          <Nav>
            <ImageLogo src={Logo} alt="logo"></ImageLogo>

            {popupShow && filmsOnInput ? (
              <SearchInputPopup filmsOnInput={filmsOnInput} />
            ) : null}

            <SearchInput onInputChange={onInputChange}
inputRef={inputRef} />
            <Ul>{lists}</Ul>
          </Nav>
        </>
      );
    }
    return null;
};

export default NavBar;
```

Файл NotFound.js

```
import React, { useState, useEffect } from "react";
import { Header1, AssetsWrapper } from
"./../../styles/StyledComponents";
import servicePopular from "../Services/servicePopular";
import { popularHomePageTMDB_URL } from "./../../url/url";
import { getThreePopularFilms } from
"./../../utils/getThreePopularFilms";

const NotFound = () => {
  const [popularData, setPopularData] = useState(null);

  useEffect(() => {
    (async function getPopularFilms() {
      const popularFilms = await
servicePopular(popularHomePageTMDB_URL);
      setPopularData(popularFilms);
    })();
  }, []);

  if (popularData) {
```

```
    const films = getThreePopularFilms(popularData);

    return (
      <>
        <Header1>Not found...sorry!</Header1>
        <AssetsWrapper>{films}</AssetsWrapper>
      </>
    );
  }
  return null;
};

export default NotFound;
```

Файл NotFound.js

```
import React, { useState, useEffect } from "react";
import { Header1, AssetsWrapper } from
"./../../styles/StyledComponents";
import servicePopular from "../Services/servicePopular";
import { popularHomePageTMDB_URL } from "./../../url/url";
import { getThreePopularFilms } from
"./../../utils/getThreePopularFilms";

const NotFound = () => {
  const [popularData, setPopularData] = useState(null);

  useEffect(() => {
    (async function getPopularFilms() {
      const popularFilms = await
servicePopular(popularHomePageTMDB_URL);
      setPopularData(popularFilms);
    })();
  }, []);

  if (popularData) {
    const films = getThreePopularFilms(popularData);

    return (
      <>
        <Header1>Not found...sorry!</Header1>
        <AssetsWrapper>{films}</AssetsWrapper>
      </>
    );
  }
  return null;
};

export default NotFound;
```

Файл ScrollComponent.js

```
import React, { useState, useEffect } from "react";
import InfiniteScroll from "react-infinite-scroll-component";
import { ParagraphBig } from
"./../../styles/StyledComponents";
import "./ScrollComponent.scss";
import servicePopular from "../Services/servicePopular";
import { popularHomePageTMDB_URL } from "./../../url/url";
import {
  Header1,
  Header2,
  AssetsWrapper,
} from "./../../styles/StyledComponents";
import { getThreePopularFilms } from
"./../../utils/getThreePopularFilms";

const ScrollComponent = ({ filmsArray, page, setPage,
filmsOfCategorie }) => {
  const [popularData, setPopularData] = useState(null);

  useEffect(() => {
    if (filmsArray?.length === 0) {
      (async function getPopularFilms() {
        const popularFilms = await
servicePopular(popularHomePageTMDB_URL);
        setPopularData(popularFilms);
      })();
    }
  }, [filmsArray]);

  if (filmsArray?.length === 0 && filmsOfCategorie?.length
=== 0) {
    if (popularData) {
      const films = getThreePopularFilms(popularData);

      return (
        <div className="wrapper-not-found">
          <Header1>Not found...sorry!</Header1>
          <Header2>Check 3 popular films</Header2>
          <AssetsWrapper>{films}</AssetsWrapper>
        </div>
      );
    }
  }

  return (
    <InfiniteScroll
      dataLength={filmsOfCategorie?.length}
      next={() => {
        setPage(page + 1);
      }}
```

```
      hasMore={true}
      loader={
        <ParagraphBig style={{ gridColumn: "1/6"
}}>Loading...</ParagraphBig>
      }
      endMessage={
        <ParagraphBig style={{ gridColumn: "1/6" }}>
          Yay! You have seen it all
        </ParagraphBig>
      }
      className="scroll-component"
    >
      {filmsArray}
    </InfiniteScroll>
  );
};

export default ScrollComponent;
```

Файл SearchInput.js

```
import React from "react";
import { Input } from "./../../styles/StyledComponents";

const SearchInput = ({ onInputChange, inputRef }) => {
  return (
    <form style={{ height: "0" }}>
      <Input
        type="text"
        placeholder="Looking for..."
        onChange={onInputChange}
        ref={inputRef}
      />
    </form>
  );
};

export default SearchInput;
```

Файл SearchInputPopup.js

```
import React from "react";
import { InputPopupWrapper, PopupFilmsWrapper,
} from "./../../styles/StyledComponents";
import AssetInCategorie from
"./../AssetInCategorie/AssetInCategorie";

const SearchInputPopup = ({ filmsOnInput }) => {
  const films = filmsOnInput?.map((el) => {
    return <AssetInCategorie filmData={el} key={el?.id} />;
  });
  return (
```

```
    <InputPopupWrapper>
      <PopupFilmsWrapper>{films}</PopupFilmsWrapper>
    </InputPopupWrapper>
  );
};

export default SearchInputPopup;
```

Файл Slider.js

```
import React from "react";
import { Slide, DescriptionPopular, ParagraphBig, SlideImage,
  DescriptionBlock, Button, Header2,
} from "../../styles/StyledComponents";
import { Link } from "react-router-dom";
import "./Slider.scss";

const Slider = ({ filmObject }) => {
  const imagePath =
`https://image.tmdb.org/t/p/original/${filmObject?.backdrop_p
ath}`;
  const linkPath = `/movie/${filmObject?.id}`;

  return (
    <Slide>
      <SlideImage src={imagePath} alt="no-image"
className="fadeIn" />
      <DescriptionPopular
        style={{ display: "block", backgroundColor:
"#29324145" }}
      >
        <DescriptionBlock>
          <Header2>{filmObject?.original_title}</Header2>
          <ParagraphBig>
            Release year: {filmObject?.release_date.split("-
")[0]}
          </ParagraphBig>
          <Link to={linkPath} className="button-link">
            <Button style={{ marginTop: "1rem"
}}>Explore</Button>
          </Link>
        </DescriptionBlock>
      </DescriptionPopular>
    </Slide>
  );
};

export default Slider;
```

Файл SliderBox.js

```
import React, { useState, useEffect } from "react";
import { SliderWrapper, ImageArrowLeft, ImageArrowRight,
} from "./../../styles/StyledComponents";
import Slider from "./../Slider/Slider";
import ArrowLeft from "./../../img/carousel-left.svg";
import ArrowRight from "./../../img/carousel-right.svg";

const SliderBox = ({ popularFilms }) => {
  const [index, setIndex] = useState(0);
  const [filmObject, setFilmObject] = useState(null);

  useEffect(() => {
    setFilmObject(popularFilms?.results?.[index]);
  }, [index, popularFilms]);

  const onArrowClick = (e) => {
    if (e.target.getAttribute("data-arrow") === "left") {
      if (index === 0) {
        setIndex(popularFilms?.results?.length - 1);
      } else {
        setIndex(index - 1);
      }
    } else {
      if (index > popularFilms?.results?.length - 2) {
        setIndex(0);
      } else {
        setIndex(index + 1);
      }
    }
  };

  if (filmObject) {
    return (
      <>
        <SliderWrapper>
          <Slider filmObject={filmObject} />
          <ImageArrowLeft
            onClick={onArrowClick}
            src={ArrowLeft}
            alt="arrow-left"
            data-arrow="left"
          />
          <ImageArrowRight
            onClick={onArrowClick}
            src={ArrowRight}
            alt="arrow-right"
            data-arrow="right"
          />
        </SliderWrapper>
      </>
```

```
    );
  }
  return null;
};

export default SliderBox;
```

Файл SummaryLinks.js

```
import React from "react";
import { SummaryLinksWrapper, ExternalLink,
} from "./../../styles/StyledComponents";

const SummaryLinks = () => {
  return (
    <SummaryLinksWrapper>
      <ExternalLink
        href="https://www.linkedin.com/in/ivan-holub-
48344b155/"
        target="_blank"
      >
        LinkedIn - Ivan Holub
      </ExternalLink>
      <ExternalLink href="https://github.com/HolubIvan"
target="_blank">
        GitHub - Ivan Holub
      </ExternalLink>
    </SummaryLinksWrapper>
  );
};

export default SummaryLinks;
```