НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія

програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Система стрімінгу медіаконтенту

Виконала: студентка 4 курсу, групи ІП-73

Сергієнко Олеся Олексіївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Волокита І. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Сергієнко Олеся Олексіївна

1. Тема проєкту *Система стрімінгу медіаконтенту*
керівник проєкту Волокита Іван Миколайович, асистент
Затверджені наказом по університету від 11.05.2021 року № 1139-с
2. Термін здачі студентом закінченого проєкту *2 червня 2021 р.*
3. Вихідні дані до проєкту: *технічна документація, теоретичні та статистичні дані;*
4. Зміст пояснювальної записки: *опис предметної області, дослідження засобів організації стрімінгу медіаконтенту, написання та тестування програми;*
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): *структурна схема системи, принципова схема системи, функціональна схема системи.*

6. Консультант проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	д.т.н., проф. Сімоненко В.		

7. Дата видачі завдання 22 вересня 2020 року

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2020-30.01.2021*	
2.	*Вивчення та аналіз завдання*	*30.01.2021-1.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*1.03.2021-12.04.2021*	
4.	*Розробка структур окремих підсистем*	*12.04.2021-26.04.2021*	
5.	*Програмна реалізація системи*	*26.04.2021-16.05.2021*	
6.	*Оформлення пояснювальної записки*	*16.04.2021-23.05.2021*	
7.	*Захист програмного продукту*	*25.05.2021*	
8.	*Передзахист*	*6.06.2021*	
9.	*Захист*	*18.06.2021*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

В бакалаврському дипломному проєкті розглянуто системи стрімінгу медіаконтенту. Практичною частиною роботи є реалізація програми, що виконує потокову передачу відео.

Програма дозволяє переглядати медіаконтент, спочатку не завантажуючи його повністю при цьому. Реалізована аутентифікація та реєстрація користувачів. Є можливість шукати необхідне відео за ключовими словами по опису, назві або тематиці.

Система розроблена у вигляді веб-сайту мовою програмування JavaScript в середовищі розробки Atom.

Annotation

In this project for a Bachelor's Degree, media streaming system is considered. The practical part of the work is the implementation of a program that performs video streaming.

The program allows you to view media content without first downloading it completely. Implemented authentication and user registration. It is possible to search for the desired video by keywords by description, title or topic.

The system is designed as a website in the JavaScript programming language in the Atom development environment.

Опис альбому

до дипломного проекту

на тему: «Система стрімінгу медіаконтенту»

Київ – 2021

Справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467800.001 ОА***	Опис альбому	1		
	А4	***ІАЛЦ.467800.002 ТЗ***	Технічне завдання	3		
	А4	***ІАЛЦ.467800.003 ПЗ***	Пояснювальна записка	70		
	А4	***ІАЛЦ.467800.004 Д1***	Додаток А	1		
			Структурна схема програми			
	А4	***ІАЛЦ.467800.005 Д2***	Додаток Б	1		
			Функціональна схема програми			
	А4	***ІАЛЦ.467800.004 Д3***	Додаток В	1		
			Принципова схема програми			
	А4	***ІАЛЦ.467800.006 Д4***	Додаток Г	13		
			Лістинг коду програми			

					ІАЛЦ.467800.001 ОА	
Зм	***Лист***	***№ докум.***	***Підп.***	***Дата***		
Розроб		***Сергієнко О.О.***			*Система стрімінгу медіаконтенту Опис альбому*	***Літ.*** / ***Аркуш*** / ***Аркушів***
Перев		***Волокита І.М.***				***1*** / ***1***
Реценз.					***НТУУ "КПІ" ФІОТ ІП-73***	
Н. контр.		***Сімоненко В.П.***				
Затв.						

Технічне завдання

до дипломного проекту

на тему: «Система стрімінгу медіаконтенту»

Київ – 2021

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Найменування: система стрімінгу медіаконтенту.

Область застосування: альтернатива існуючим сервісам потокової передачі медіаконтенту.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту по освітньо-професійної програми “Інженерія програмного забезпечення комп’ютерних систем” за спеціальністю 121 Інженерія програмного забезпечення, затверджене кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є вивчення та аналіз програмних засобів для організації систем стрімінгу медіаконтенту, проектуванння та реалізація даної системи.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, технічна документація, публікації в періодичних виданнях, довідники, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Відносно простий і інтуїтивно зрозумілий інтерфейс системи.
- Можливість перегляду медіаконтенту.
- Можливість аутентифікуватись та зареєструватись в системі.
- Незалежність - система повинна бути автономна і не залежати від встановленого програмного забезпечення.
- Технічна коректність та актуальність інформації, що становить наповнення системи.

5.2. Вимоги до програмного забезпечення

- Google Chrome 16.0 / Apple Safari 5.1 / Mozilla Firefox 9.0 та вище
- Node.js версії 12.9.0 та вище
- Windows/Linux/macOS

6. Етапи розробки

	Дата
Вивчення літератури	11.01.2021
Складання і узгодження технічного завдання	15.03.2021
Створення модулів розробленої системи	29.03.2021
Тестування модулів розробленої системи	26.04.2021
Доопрацювання і виправлення помилок	17.05.2021
Оформлення документації дипломного проєкту	24.05.2021

Пояснювальна записка

до дипломного проекту

на тему: «Система стрімінгу медіаконтенту»

Київ – 2021

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

VOD (Video-On-Demand)	Відео на запит
LAN(Local Area Network)	Логкальна мережа
RTMP (Real-Time Messaging Protocol)	Протокол обміну повідомленнями у реальному часі
CDN (Content Delivery Network)	Мережа доставки контенту
WebRTC (Web Real-Time Communication)	Комунікації в реальному часі
API (Application Programming Interface)	Прикладний програмний інтерфейс

ВСТУП

Розвиток технологій за останні п’ятнадцять років суттєво вплинув на те як ми отримуємо та споживаємо інформацію. Зберігання на фізичних носіях, перегляд телебачення, а наразі вже і завантаження медіаконтенту відходять у минуле у зв’язку з появою стрімінгових сервісів. Сьогодні ми сприймаємо контент як потік чи ефір, зокрема не завантажуємо фільми та музику, а дивимось та слухаємо їх онлайн.

Стрімінг медіаконтенту або потокове мультимедіа – це мультимедіа, яке користувач безперервно отримує від провайдеру потокового мовлення. Для того, щоб почати відтворення контенту користувачвикористовує свій медіаплеєр, при цьому не дочікуючись передачі всього файлу. Це альтернатива завантаженню файлу, коли користувачу необхідно отримати файл цілком аби переглянути його.

Відповідно до досліджень Global Internet Phenomena Report[1] у 2018 році від компанії Sandvine Incorporated на стрімінговий відеосервіс Netflix витрачається 15% споживаного інтернет-трафіку в усьому світі. На відео (наприклад записи або прямі трансляції) припадає 58% від всього трафіку, який користувачі використовують в мережі. У США частка інтернет-трафіку Netflix з загальносвітового показника в 15% збільшується до 19,1%.

Отже, можна сказати, що на перше місце як в Україні так і в усьому світі виходять системи для потокової передачі. Завдяки ним користувачі можуть слухати улюблених виконавців, дивитися фільми та серіали, телешоу, новини чи спортивні події незалежно від пристрою або часу, та в зручному для них темпі.

Завдання, що поставлені для даної роботи – це вивчення систем стрімінгу медіаконтенту, а саме основних принципів та ключових моментів їх роботи; вивчення програмних засобів для організації таких систем; проектування та реалізація такої системи.

Метою дипломного проекту є реалізація системи стрімінгу медіаконтенту.

Даний дипломний проект складається з чотирьох розділів. В кінці кожного розділу містяться висновки.

У першому розділі розглянуто загальні риси систем стрімінгу медіаконтенту, принципи їх роботи, переваги та недоліки. Проаналізовано вже наявні популярні приклади таких систем.

У другому розділі розглянуто варіанти основних компонентів, протоколів та програмних засобів, що використовуються для стрімінгу. Проаналізовано випадки їх використання з різними цілями у різних проектах.

У третьому розділі спроектовано та реалізовано власну систему стрімінгу медіаконтенту. Обґрунтовано вибір стеку технологій та архітектури.

У четвертому розділі продемонстровано роботу системи. Протестовано основні її модулі.

По завершенню роботи зроблені загальні підсумки усього дипломного проекту.

РОЗДІЛ 1. СИСТЕМИ СТРІМІНГУ МЕДІАКОНТЕНТУ

1.1. Принцип роботи

Стрімінг або потокове мультимедіа можна вважати важливою частиною інформаційної революції, що відбулася за останні два десятиліття. Історія розвитку систем стрімінгу медіаконтенту починається приблизно у 2006 році, коли стартував відеосервіс Amazon Prime Video. Менше ніж за рік компанія Netflix запустила власну платформу потокового медіа. Звичайно, що йдеться про стрімінг або потокову передачу медіаконтенту у тому вигляді з яким ми знайомі сьогодні. Адже в загальному випадку історія потокової передачі сягає початку 1970-х років, коли дослідницькі лабораторії в США вперше продемонстрували технологію потокового аудіо.[2]

Стрімінг або потокове медіа (від англ. «stream media») – це медіаконтент, що безупинно отримується від провайдера потокового мовлення. Це може стосуватися будь-якого медіаконтенту, що генерується просто зараз наживо чи вже записаному, що використовується для абсолютно різних цілей в різноманітних сферах життя людини.

Принцип роботи потокового мовлення можна коротко описати таким чином, що медіа файл безперервно відправляється на користувацький пристрій поступово, а не увесь відразу. Вихідний контент в залежності від типу стрімінгу або зберігається віддалено, або, в разі прямої трансляції, відтворюється в реальному часі з віддалених пристроїв (камери, мікрофони тощо). Таким чином вдається уникнути попереднього завантаження цілого файлу локально користувачем.

Подібно будь-яким іншим даним, що надсилаються через Інтернет, медіаконтент сегментується або розбивається на пакети даних. Кожен з цих пакетів вміщує в собі частину файлу, а медіаплеєр в браузері з клієнтської сторони приймає цей потік та інтерпретує їх як відео чи аудіо.

Потокова передача медіа вимагає певної швидкості для оптимальної

продуктивності. Часто стрімінгові сервіси пропонують рекомендації з мінімальної швидкості в залежності від якості контенту. Чим вища якість, тим більша швидкість необхідна для коректної передачі фільмів або музики.

1.2. Вплив COVID-19

Пандемія COVID-19 в 2020-2021 роках в негативному ключі вплинула майже на кожен бізнес. Мільйони людей залишилися замкненими в своїх будинках. Але, через те що потокове медіа це повністю про онлайн, ця сфера тільки виграла. Відчутно збільшилась актуальність стрімінгових сервісів, адже попит на нові розважальні послуги різко зріс. Один з найкращих прикладів – потокове відео за запитом, включаючи Netflix, Hulu, Amazon Prime Video, YouTube, Disney+.

Згідно досліджень[3] 2021 року кількість підписників американських стрімінгових сервісів Netflix та Hulu різко збільшилася з 2019 по 2020 рік. Наприклад, у Netflix зростання кількості абонентів було 4,36% з 2018 по 2019 рік. З 2019 року до третього кварталу 2020 року їх річний приріст підписників становив 19,72%. А в Hulu з 19,63% зростання абонентів з 2018 по 2019 рік до 26,95% зростання з 2019-2020. У Дісней+ не має даних за рік, оскільки 2020 рік став першим фінансовим роком. Однак їх зростання за четвертий квартал 2020 року було неймовірним - 178,11%. Інші потокові сервіси, що випускали щоквартальні номери абонентів, також спостерігали аномально сильний квартальний ріст після четвертого кварталу 2019 року. Це сильне квартальне зростання збігається з початком пандемії COVID-19 та наступними блокуваннями.

Люди також використовують такі платформи аби грати в ігри, спілкуватися, розважатися, виходити в прямі ефіри у таких сервісах як Twitch, YouTube Live, Facebook Live.

1.3. Переваги та недоліки

Основні переваги стрімінгу медіаконтенту:

- Потокове передавання відтворюється у даний момент - потоковий вміст починає відтворюватися майже миттєво, незалежно від того, наскільки великим є візуальний або аудіоматеріал. Користувачеві не потрібно чекати до моменту, коли він завантажиться повністю.
- Потокова передача не вимагає місця для зберігання. Вам не потрібен великий жорсткий диск, аби завантажити контент, який ви хочете дивитись або слухати, на відміну від способу, яким потрібно завантажувати програми, і мати достатньо місця для їх зберігання.
- Більшість потокових програм дозволяють вам вибирати, що ви хочете дивитись або слухати вільно, а це означає, що вам не доведеться дотримуватися розкладу телебачення чи радіо.
- Так звана «мультиекранність» або можливість переглядати контент з різних пристроїв: комп’ютер, телефон, планшет, ігрова консоль тощо.
- Захист від піратства. Дозвіл відвідувачам веб-сайту завантажувати відео або аудіо значно спрощує піратство.

З іншого боку, стрімінг медіаконтенту має деякі обмеження:

- Вам потрібно активне підключення до Інтернету. Ви можете транслювати лише в тому випадку, якщо ви підключені до Інтернету, тому ви не можете користуватися послугами потокового передавання без стільникових даних, Wi-Fi або Ethernet-з'єднання.
- Трансляція - це діяльність у режимі реального часу; якщо ви захочете переглянути той самий фільм пізніше, наприклад, вам потрібно мати підключення до Інтернету кожного разу, коли ви його вмикаєте. Завантажений файл, навпаки, потрібно завантажити лише один раз для відтворення довільної кількості разів.

- Вас може обмежувати швидкість підключення до Інтернету. Ваше з’єднання повинно бути досить швидким та надійним, інакше відтворення потокового відео чи аудіо буде невдалим.

1.4. Фактори сповільнення

Розберемо основні причини, що можуть сповільнити потокову передачу медіаконтенту як зі сторони мережі, так і зі сторони користувача.

З боку мережі:

- Мережева затримка – на затримку впливає безліч факторів, у тому числі місце зберігання контенту, до якого користувачі намагаються отримати доступ.
- Перевантаження мережі – у випадку передачі по мережі занадто великої кількості даних, може знизитись продуктивність потокової передачі.

З боку користувача:

- Проблеми з WiFi – перезапуск LAN-маршрутизатора або перемикання на Ethernet замість WiFi може допомогти поліпшити продуктивність потокової передачі.
- Повільно працюють клієнтські пристрої – для відтворення відео потрібно багато обчислювальної потужності. Якщо на пристрої, що відео, запущено багато інших процесів або воно працює повільно, це може вплинути на продуктивність потокової передачі.
- Недостатньо пропускної спроможності – для потокової передачі відео домашнім мережам потрібно пропускна здатність близько 4 Мбіт/с; для відео високої чіткості їм, ймовірно, буде потрібно більше.

1.5. Буферизація

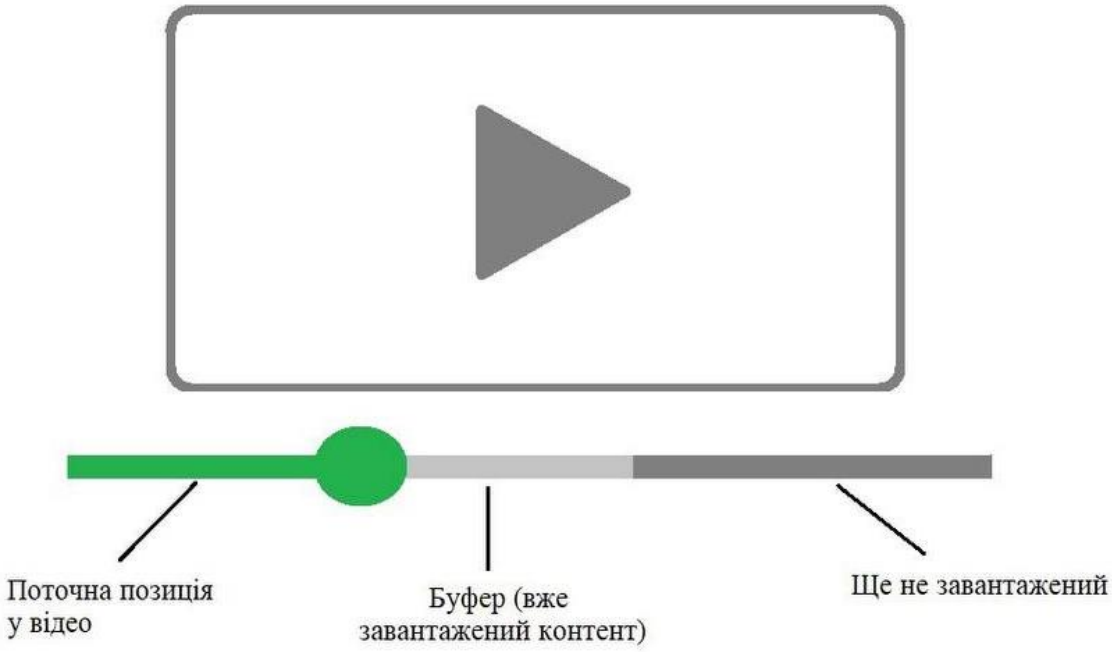


Рис. 1.1 – Приклад буферизації

Програвачі потокового мультимедіа завантажують потік на кілька секунд завчасно, щоб відео або аудіо могли продовжити відтворення, якщо з'єднання буде ненадовго перервано. Це називається буферизацією(рис. 1.1). Буферизація забезпечує плавне і безперервне відтворення відео. Однак через повільні підключення або при великій затримці в мережі буферизація відео може зайняти багато часу.

1.6. Система рекомендацій відео

Розглянемо алгоритми побудови системи рекомендацій на прикладі стрімінгової платформи Netflix.

Якщо користувач хоче знайти якийсь контент або відео на Netflix, система рекомендацій Netflix допомагає користувачам знайти свої улюблені фільми чи відео. Для побудови цієї рекомендаційної системи Netflix повинен передбачити зацікавленість користувачів та збирати різні типи даних від користувачів, такі як:

- Взаємодія користувача зі службою (перегляд історії та те, як користувач оцінив інші заголовки)
- Інші учасники зі схожими смаками та уподобаннями.
- Інформація про метадані з раніше переглянутих відео для користувача, таких як заголовки, жанр, категорії, актори, рік випуску тощо.
- Пристрій користувача, в який час користувач активніший і як довго користувач активний.

Netflix використовує два різні алгоритми для побудови системи рекомендацій.

1. Спільна фільтрація. Ідея цієї фільтрації полягає в тому, що якщо двоє користувачів мають схожу історію рейтингів, вони будуть поводитися подібним чином і в майбутньому.
2. Фільтрація на основі вмісту. Ідея полягає у фільтруванні тих відео, які схожі на відео, яке сподобалось користувачеві раніше. Фільтрація на основі вмісту сильно залежить від інформації з таких продуктів, як назва фільму, рік випуску, актори, жанр. Отже, для реалізації цієї фільтрації важливо знати інформацію, що описує кожен елемент, і якийсь профіль користувача, що описує те, що подобається користувачеві, також бажано.

1.7. Типи потокових додатків

Більшість систем стрімінгу медіаконтенту можна поділити на такі основні групи:

Потокове відео за запитом (Video on Demand, VOD). В таких системах є каталог, з якого користувач може вибрати фільми, серіали, мультфільми тощо за допомогою пошукової системи. В сучасних додатках часто наявні механізми рекомендацій, що засновані на особистих уподобаннях.

Приклади: Netflix, HBO Max, Disney Plus, Amazon Prime, Hulu, YouTube.

Потокове аудіо. За платною підпискою користувачі слухають музику не зберігаючи її на свій пристрій. Приклади: Spotify, Deezer, iTunes, YouTube Music, Apple Music.

Телетрансляції в прямому ефірі. Потреба телевізора дома зникає, але потрібні телепрограми можна подивитися онлайн зі смартфону чи ноутбука. Приклади: Hulu, YouTube TV, Amazon Prime Video, Sling TV. Більшість з них можна визначити як гібридні, адже здебільшого це платформи для відео за запитом, але в них є прямі трансляції з телебачення. Існує також більш гнучка версія моделі стрімінгу телевізійного контенту з функцією time-shift. Користувачі мають змогу переглядати телешоу та використовувати при цьому «Паузу» та «Перемотку».

Додатки для прямих трансляцій (Live Broadcasting). Наразі один з найбільш популярних типів потокового медіа. Це перегляд відео в режимі реального часу, що записується та транслюється одночасно. Найбільшого поширення такий тип отримав у геймерів (або стрімерів, коли транслюється як користувач грає в комп'ютерні ігри), у блогерів (коли просто відбувається розмова або відповідають на питання в прямому ефірі) та при передачі відео зі спортивних подій. Приклади: Twitch, Instagram Live, Facebook Live, YouTube Live.

В даному дипломному проекті розроблено додаток, що відноситься до першої групи.

1.8. Порівняння існуючих систем стрімінгу медіаконтенту

Розглянемо більш детально вже наявні приклади сервісів стрімінгу медіаконтенту, що відносяться до групи потокового відео за запитом.

1.8.1. Netflix

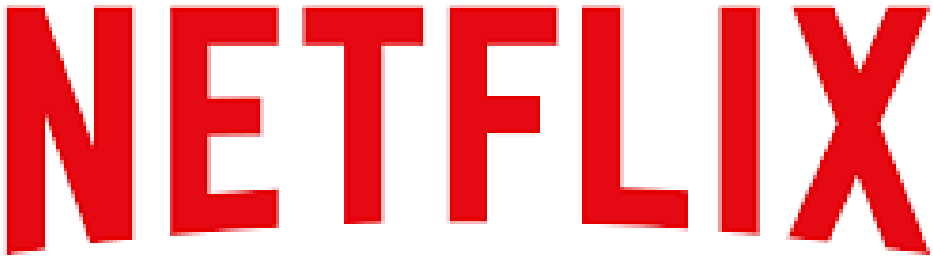


Рис. 1.2 – Логотип Netflix[4]

Компанія Netflix (рис. 1.2), що починалася у 1997 році як магазин DVD-прокату після того, як перейшла в онлайн, стала сьогодні одною з найвпливовіших компаній у світі. У 2007 році був створений стрімінговий відеосервіс. Кількість активних користувачів сайту швидко зростає завдяки хорошій системі рекомендацій серіалів та фільмів.

Netflix використовує технологію адаптивного стримінгу, який регулює якість аудіо та відео відповідно до швидкості інтернету користувача, а компанія також пропонує клієнтам можливість встановити якість відео самостійно. Велике число поєднань кодеків та знайомих бітрейтів у Netflix означає «обов'язковість кодування одного фільму 120 різними способами перед його передачами на будь-яку стрімінгову платформу».

Через те, що процес кодування відео здійснюється у хмарах, компанія отримує більші можливості по масштабуванню: якщо різко потребує необхідності в обробці великої кількості фільмів, то хмарні технології дозволяють легко задіяти додаткові ресурси. Справедливо і зворотнє – якщо потрібно, обсяг обчислювальних потужностей можна і зменшити. Netflix використовує протокол DASH (Dynamic Streaming over HTTP) потокового передавання.[5]

У 2019 році до впровадження прогресивного обмеження навантажень у роботі службової мережі стрімінгу Netflix здійснили переходи, з-за яких значне число підписників не могло переглядати відео протягом деякого часу. У 2020 році, інженерна група почала відмічати переваги його введення.[6] У системі Netflix відбулася подібна проблема, потенційним наслідком якої могло стати такий же перебій у роботі сервісу, як і в 2019 році. Однак система прогресивного обмеження навантаження Zuul включилася і стала обмежувати трафік, оскільки ефективність служби не була відновлена в повному об'ємі. При цьому можливість підписників переглядати відео не постраждала.

Платформи і пристрої: Android, iOS, веб-браузер, Apple TV, Chromecast, Fire TV, Nvidia Shield, Roku, Smart TV провідних брендів, ігрові консолі PS4 і Xbox One, смарт тв-приставки різних виробників.

1.8.2. Amazon Prime Video

Рис. 1.3 – Логотип Amazon Prime Video[7]

Amazon Prime Video(рис. 1.3), частина послуги Інтернет-відео Amazon Video на замовлення, пропонує вибір оригінального вмісту та ліцензованих фільмів та телешоу, які можна транслювати або завантажувати в рамках передплати на Amazon Prime.[8]

Був заснований в 2006 році. Сервіс дозволяє змінювати якість під час

перегляду, а 95% контенту вже доступно в Full HD.

Prime Video підтримує низку відео- та аудіоформатів, включаючи високий динамічний діапазон (HDR), 4K Ultra HD та Dolby Atmos.

Служба потокової передачі дозволяє передавати низку програм у форматі 4K Ultra HD, але, по-перше, ви захочете відфільтрувати інші програми. Для цього у веб-браузері просто знайдіть програму, яку ви хочете переглянути на Prime Video. Далі вам потрібно буде встановити прапорець "4K Ultra High Definition". Prime Video також пропонує спеціальні розділи для вмісту 4K через свої програми. Таким чином, ви побачите лише фільми та шоу в цій якості відео.

Платформи і пристрої: Fire TV, планшети Fire, iPhone, iPad, Chromecast, ігрові консолі Xbox, PlayStation 4, Smart TV провідних брендів, Blu-ray плеєри, Apple TV, Roku.

1.8.3. Hulu

Рис. 1.4 – Логотип Hulu[9]

Стрімінговий сервіс Hulu(рис. 1.4) створений у 2007 році медіахолдингом News Corporation та NBCUniversal Media.

Відео Hulu транслюється зі швидкістю 480 і 700 кбіт /с. Кілька відео також можна транслювати зі швидкістю 1000 кбіт/с. Абоненти HuluPlus також можуть отримати доступ до відео у високій якості, коли вони доступні. Клієнти можуть перемикатися між бітрейтами під час відтворення на основі доступної смуги пропускання.[10]

Hulu використовує ті самі три CDN, що і Netflix, для доставки відеовмісту користувачам. На основі файлів маніфесту клієнту Hulu спочатку присвоюється пріоритетне ім'я хосту CDN, а потім використовує DNS для вибору IP-адреси сервера. Протокол потокового передавання: Hulu використовує зашифрований протокол обміну повідомленнями в реальному часі (RTMP) для доставки фільмів у браузери настільних ПК. Відео Hulu можна передавати через необроблений RTMP через порт 1935 або RTMP, тунельований через HTTP (RTMPT).

Hulu використовує лише один CDN-сервер протягом усього відео. Тим не менше що цікаво, зазвичай він перемикається на інший CDN для наступного відео.

Платформи і пристрої: Android, Android TV, Apple TV, Chromecast, Fire Tablets, Fire TV, iPhone і iPad (yci), LG TV (окремі моделі), Nintendo Switch, Mac і PC, PlayStation 3 (без live), PlayStation 4 , Roku, Samsung TV (окремі моделі), VIZIO SmartCast TV, Xbox

1.8.4. YouTube

Рис. 1.4 – Логотип YouTube[11]

YouTube(рис. 1.4) був створений у лютому 2005 трьома колишніми

працівниками Paypal для обміну відео з друзями. В результаті зростання популярності та величезного потенціалу цей сайт був придбаний Google у 2006 році.

Подібно до інших систем стрімінгу медіаконтенту за запитом, YouTube дозволяє користувачам переглядати, завантажувати, коментувати контент та підписуватися на інших користувачів. Під час перегляду відео глядачі можуть бачити заголовок відео, опис відео, хто завантажив відео, дату та час завантаження та теги, що були зазначені власником. Крім того, доступна інформація про кількість переглядів і оцінок відео зареєстрованими користувачами.

Технологія відтворення відео YouTube заснована на програвачі Flash Player від Macromedia та використовує відеокодек Sorenson Spark H.264 з розмірами пікселів 320 на 240 і 25 кадрів у секунд.[12] Ця технологія дозволяє YouTube відображати відео з якістю порівняно більш усталеною, під час відтворення відео. YouTube надає можливість завантажувати відео у форматах WMV, AVI, MOV та MPEG, які перетворюються у формат .FLV після завантаження.

YouTube в основному використовує протокол Dynamic Adaptive Streaming, що заснований на протоколі HTTP.

Для мобільних пристроїв іноді сервери Youtube надсилають дані за допомогою RTSP, який є протоколом прикладного рівня.

На транспортному рівні RTSP використовує як TCP, так і UDP.

Всі відео, які завантажуються на YouTube, спочатку перекодовуються в різні формати та роздільну здатність, що встановлені платформою. Відео під час процесу перекодування розбивається на сегменти та конвертується у різні роздільні здатності. Обробка декількох сегментів розподіляється по декількох машинах для паралелізації процесу, збільшуючи таким чином пропускну здатність. Якщо відео стає популярним, воно підлягає черговому етапу стиснення відео. Цей другий раунд стиснення забезпечує однакову візуальну якість відео при значно меншому розмірі.

При кодуванні відео YouTube вибирає бітрейт у межах, дозволених кодеком. Відео з високим бітрейтом має кращу якість, але інколи навіть за збільшенні бітрейту не відбувається значного візуального покращення якості відео, хоча розмір відео в процесі збільшується.

Платформи і пристрої: Amazon Fire TV, програвачі та телевізори Roku, Apple TV та Apple TV 4K, телевізори Vizio SmartCast, смарт-телевізори Samsung та LG, телевізори HiSense, Xbox One тощо.

Висновок до розділу 1

В результаті аналізу та досліджень систем стрімінгу медіаконтенту, що були проведені у першому розділі, можна сформулювати такі висновки:

- Сформульовано основні принципи роботи досліджуваних систем.
- З’ясовано, що системи стрімінгу медіаконтенту – це одна з небагатьох сфер, що стала стрімко розвиватися та набула тільки ще більшого поширення під час пандемії COVID-19.
- Були наведені головні переваги та недоліки, що стосуються таких систем.
- Визначені фактори сповільнення, що можуть вплинути на перегляд відео чи прослуховування аудіо зі сторін мережі, а також користувача.
- Виокремлені типи потокових додатків, особливості їх реалізації, спільні риси та основні моменти роботи.
- Були проаналізовані найбільш успішні існуючі системи стрімінгу медіаконтенту.

РОЗДІЛ 2. ПРОГРАМНІ ЗАСОБИ ОРГАНІЗАЦІЇ СТРІМІНГУ

2.1. Мережа розповсюдження контенту

Рис. 2.1 – Мережа розповсюдження контенту

Технологія віддалених серверів потокової передачі медіаконтенту може прискорити доставку інформації за допомогою процесу, що називається HTTP-кешуванням. Це коли візуальні або аудіоматеріали тимчасово зберігаються на декількох серверах в мережі розповсюдження контенту (Content Delivery Network, CDN). Мережі розповсюдження контенту – це фізично розподілена мережа проксі-серверів та центрів обробки їх даних. Основною метою є забезпечення високої доступності та продуктивності за рахунок географічного розподілення послуг по відношенню до клієнтів.

В момент, коли користувач робить запит на контент, він направляється на найближчий сервер CDN з кешованим контентом, потім сервер доставляє його на пристрій користувача. Відправка кешованої інформації пришвидшується у

порівнянні з доставкою з вихідного серверу, бо шлях коротший. Таким чином контент тільки деякий час зберігається в кеші, а не постійно на жорсткому диску на пристрої користувача.

У веб-кешах зберігається інформація, що є найбільш затребуваною, до якої звертаються користувачі частіше всього. Завдяки цьому знижується навантаження на сервери та збільшується пропускна здатність мережі. А відповідно зменшується час очікування при зверненні до даних.

CDN забезпечують високоякісну потокову передачу по всьому світу. Content Delivery Network оптимізує навантаження і дозволяє транслювати відео користувачам по всьому світу з мінімальними втратами швидкості. Найбільш значні переваги CDN – це швидкість і надійність.

Деякі сервіси стрімінгу медіаконтенту навіть запускають свої власні CDN. Наприклад, Netflix побудувала власну розподілену мережу під назвою Open Connect для більш ефективної доставки свого відеоконтенту. Але це має сенс переважно в тих випадках, якщо їх каталог контенту не дуже великий. Якщо файлів мало, то весь контент можна зберігати в одному місці. У міру зростання компаній створення CDN стає все менше сенсу. Сторонні CDN дозволяють їм миттєво отримати глобальне охоплення за допомогою існуючих сервісів. Передаючи свої мережі розповсюдження на аутсорсинг, служби отримують більше часу для роботи над більш важливими проектами. Вони можуть використовувати цей час для створення алгоритмів адаптації до постійно змінюваних умов мережі.

2.2. Протоколи для стрімінгу

Більшість відеофайлів не пристосовані до потокової передачі. Їх необхідно спочатку завантажити, а потім відтворювати. Протоколи потокової передачі відео - це стандартизовані способи розбити файл на частини, відправити його до користувача та повторно його зібрати. Розмір частин, що розбиваються, залежить від використаного протоколу потокової передачі відео. Мета – доставити фрагменти у хорошій якості з мінімальним часом

буферизації.

При потоковій передачі по HTTP стандартний шаблон запит-відповідь не застосовується. З'єднання між клієнтом і сервером залишається відкритим протягом усього потоку, і сервер передає відеодані клієнту, щоб клієнтові не доводилося запитувати кожен сегмент відеоданих.

Вибір протоколу залежить від пріоритетів системи, адже деякі більш орієнтовані на збереження якості, деякі на зменшення затримки, декі працюють тільки у певних системах.

2.2.1. RTMP

Протокол обміну повідомленнями у реальному часі (Real-Time Messaging Protocol) найбільше використовується для прямих трансляцій у реальному часі. Принцип роботи полягає у тому, що медіаплеєр повинен зв’язатися з сервером, сервер реагує та надсилає відеофайл. Головною умовою є наявність високої пропускної здатності по обидва боки. Перевагою є найменша затримка з-поміж інших технологій.

Негативним моментом є те, що необхідний ще один протокол аби доставити файл до кінцевого користувача (частіше протокол HTL). Це відбувається через те, що RTMP залежить від модуля Flash, що має проблеми з безпекою, тому цей протокол не використовується зі сторони клієнтів.

Отже, використовувати його можна для прямих трансляцій та обміну даними між клієнтами, які використовують Flash.

2.2.2. MPEG-DASH

Динамічна адаптивна потокова передача через HTTP (Moving Picture Experts Group Dynamic Adaptive Streaming over HTTP) є дуже хорошим варіантом за багатьох причин. Він сумісний з усіма форматами кодування, може працювати на будь-якому сервері, адже даний метод заснований на HTTP. Розбиває відео на фрагменти по 2-4 секунди.

Основні етапи процесу потокової передачі MPEG-DASH:

- Кодування та сегментація:

Вихідний сервер ділить відеофайл на більш дрібні сегменти довжиною в кілька секунд. Сервер також створює індексний файл - наприклад, зміст для сегментів відео. Потім сегменти кодуються, тобто форматуються таким чином, щоб їх могли інтерпретувати кілька пристроїв. MPEG-DASH дозволяє використовувати будь-який стандарт кодування.

- Доставка:

Коли користувачі починають дивитися потік, закодовані сегменти відео відправляються на клієнтські пристрої через Інтернет. Майже у всіх випадках мережа доставки контенту (CDN) допомагає більш ефективно розподіляти потік.

- Розшифровка і відтворення:

Коли пристрій користувача отримує потокові дані, воно декодує дані і відтворює відео. Мультимедійний автоматично перемикається на зображення більш низької або більш високої якості, щоб пристосуватися до умов мережі - наприклад, якщо у користувача в даний час дуже мала пропускна здатність, відео буде відтворюватися з більш низьким рівнем якості, що використовують меншу пропускну здатність.

У MPEG-DASH реалізовано штучний інтелект для вибору сегментів та інтервалів в мультимедійному контенті з максимально можливим адаптивним бітрейтом, який може бути завантажений для відтворення без буферизації. Завдяки щосекундному налаштуванню усуває деякі технічні проблеми з доставкою і стисненням. Це протокол з адаптивною швидкістю передачі даних (Adaptive Bitrate Streaming, ABR). Адаптивна потокова передача з бітрейтом - це можливість регулювати якість відео в середині потоку при зміні умов мережі. Кілька протоколів потокової передачі, включаючи MPEG-DASH, HLS і HDS, дозволяють здійснювати потокову передачу з адаптивною швидкістю передачі даних.

Передача потокового з адаптивною швидкістю передачі даних можлива, оскільки вихідний сервер кодує сегменти відео з кількома різними рівнями якості. Це відбувається під час процесів кодування і сегментації. Клієнт може перемикатися з одного рівня якості на інший в середині відео, не перериваючи відтворення. Це запобігає повну зупинку відео, якщо пропускна здатність мережі раптово знижується.

2.2.3. HLS

HTTP Live Streaming на даний момент найбільш безпечний та надійний протокол. З початку розроблений для видалення Flash з iPhone, наразі він підтримується на настільних комп'ютерах, смарт-телевізорах, Android і iOS.

Протокол HLS працює наступним чином:

- Сервер:

Потік HLS виходить від сервера, на якому зберігається мультимедійний або де створюється потік. Оскільки HLS заснований на HTTP, будь-який звичайний веб-сервер може створити потік.

На сервері відбуваються два основних процеси:

- Кодування:

Відео дані переформатуються, щоб будь-який пристрій могло розпізнавати і інтерпретувати дані. HLS повинен використовувати H.264 або кодування H.265.

- Сегментування:

Відео розділене на сегменти по кілька секунд. Довжина сегментів може варіюватися, хоча за замовчуванням довжина становить 6 секунд (до 2016 року вона становила 10 секунд).

Крім поділу відео на сегменти, HLS створює індексний файл сегментів відео для запису порядку, якому вони належать.

HLS також створить кілька повторюваних наборів сегментів з різними рівнями якості: 480p, 720p, 1080p і так далі.

- Поширення:

Закодовані відеосегментів відправляються на клієнтські пристрої через Інтернет, коли клієнтські пристрої запитують потік. Як правило, мережа доставки контенту (CDN) допомагає поширювати потік в географічно рознесені області. CDN також кешує потік, щоб ще швидше надавати його клієнтам.

- Клієнтський пристрій:

Клієнтський пристрій - це пристрій, який приймає потік і відтворює відео, наприклад користувацький смартфон або ноутбук. Клієнтський пристрій використовує індексний файл як еталон для збірки відео по порядку і переключається з більш високої якості на більш низьку якість зображення (і навпаки) в міру необхідності.

HLS розбиває файли на більш дрібні завантаження HTTP і доставляє їх з використанням протоколу HTTP. За замовчуванням сегментує на блоки по 6 секунд. Менші сегменти у MPEG-DASH краще адаптуються до швидкості Інтернету. HLS як і MPEG-DASH використовує протокол HTTP, а всі підключені до Інтернету пристрої його підтримують, і тому не вимагає використання спеціалізованих серверів. Також, так само як і MPEG-DASH, потік HLS може підвищувати або знижувати якість відео в залежності від стану мережі без переривання відтворення.

Найкращий вибір у разі масштабування на велику аудиторію. Негативним моментом є відносно високий час затримки (від 15 до 30 секунд) при потокової передачі в реальному часі.

2.2.4. HDS

HTTP Dynamic Streaming не так часто використовується, як інші протоколи потокової передачі, як наприклад HLS. це метод потокової передачі з адаптивним бітрейтом, розроблений Adobe. HDS доставляє відеоконтент MP4 через HTTP- з'єднання. HDS можна використовувати для потокової передачі за запитом або для потокової передачі в реальному часі. Оскільки

вони доставляються по протоколу HTTP, потоки HDS можуть кешуватися або мережею доставки контенту (CDN).

HDS був розроблений для використання з Adobe Flash Player і Adobe AIR. Підтримка Adobe Flash Player припинена, і тепер стороння фірма підтримує AIR замість Adobe. HDS не підтримується пристроями Apple.

Процес створення і доставки HDS-потоку приблизно такий:

- Сервер:

Перед потоковою передачею відеофайлів через HDS їх необхідно перетворити зі звичайного MP4 в формат файлу F4F (фрагментований MP4). Відео F4F містять аудіо, відео та метадані. Оскільки файли «фрагментовані», ці три елементи можуть зберігатися окремо один від одного.

Відео HDS кодуються за допомогою H.264, який є поширеним стандартом кодування. Як і багато інших технології потокової передачі, HDS кодує версії відеофайлу з декількома рівнями якості і розділяє відео на більш короткі сегменти довжиною в кілька секунд. Це уможливлює потокову передачу з адаптивним бітрейтом.

- Поширення:

Сегменти відео HDS відправляються на клієнтські пристрої, які запитують потік через Інтернет. CDN зазвичай допомагає розподіляти потік, а також кеширует потік, щоб обслуговувати його швидше.

- Клієнт:

Пристрій, яка вимагала потік, використовує файл маніфесту відео, що міститься в метаданих, як посилання для збірки і відтворення відеофрагментів по порядку. При необхідності він також змінює якість зображення.

2.2.5. WebRTC

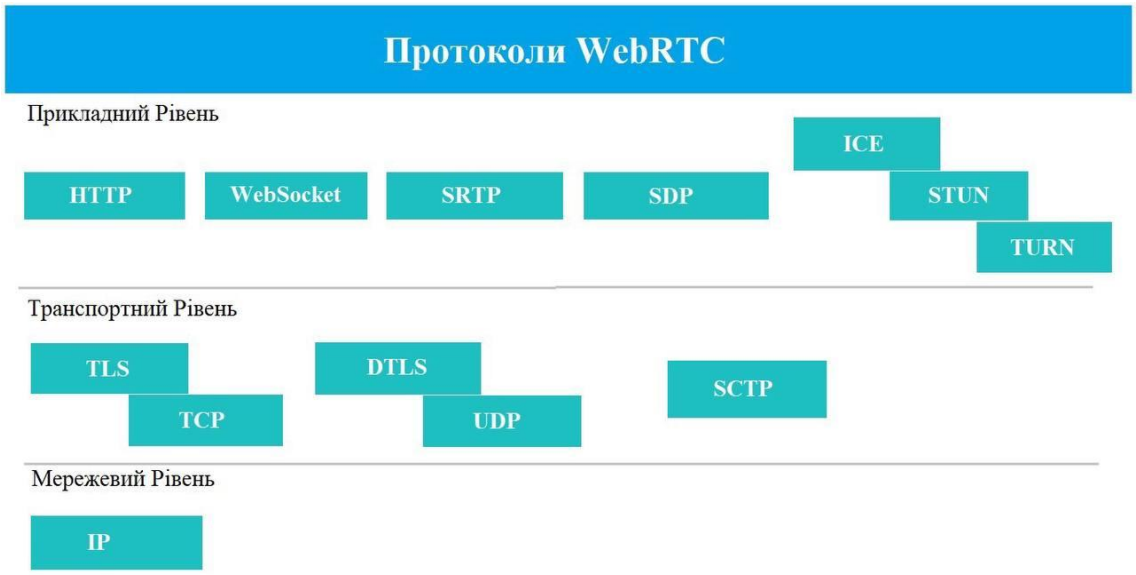


Рис. 2.2 – Протоколи WebRTC

Комунікації в реальному часі (Web Real-Time Communication) – це, насправді, не один протокол, а цілий стек протоколів. WebRTC здатний передавати контент між браузерами в режимі реального часу без будь-яких завантажень або плагінів. Він забезпечує передачу медіаконтенту (такого як голос, відео, спільне використання екрану, інтерактивна дошка та онлайн-ігри) без додаткового клієнтського програмного забезпечення, крім стандартного веб-браузера.

Заснований на трьох API:

- MediaStream. Відповідає за надання можливості веб-браузеру отримувати аудіо- та відеосигнали з камер або робочого столу користувача.
- RTCPeerConnection. Відповідає за з'єднання між браузерами для обміну медіаданими, отриманими з камери, мікрофона або робочого столу користувача. У його «обов'язки» також входить обробка сигналу (очищення його від фонового шуму, регулювання рівня мікрофона) і управління використовуваними аудіо- та відеокодеками.
- RTCDataChannel. Відповідає за двосторонню передачу даних через

встановлене з'єднання.

Хороше рішення для потокової передачі відео, а особливо відеоконференцій у реальному часі, коли невелика довжина затримки є обов’язковою.

2.3. Порівняння протоколів

Був проведений докладний порівняльний аналіз існуючих протоколів для стрімінгу медіаконтенту. Результати цього аналізу наведені в таблиці 1.1.

Таблиця 2.1 – Порівняння протоколів для стрімінгу

Протоколи для стрімінгу	RTMP	MPEG-DASH	HLS	HDS	WebRTC
Затримка	Низька затримка (менше 5 секунд)	Середня затримка (від 6 до 30 секунд)	Висока затримка (близько 10 секунд)	Низька затримка (від 6 до 8 секунд)	Низька затримка (менше 500 мс)
Сумісність відтворення	Flash Player, Adobe AIR, RTMP-сумісні плеєри	Усі пристрої Android; більшість телевізорів Samsung, Philips, Panasonic та Sony після 2012 року; Браузери Chrome, Safari та Firefox	Усі браузери Google Chrome; Пристрої Android, Linux, Microsoft та MacOS; кілька приставок, смарт-телевізорів та інших програвачів	Flash Player, Adobe AIR	Chrome, Firefox та Safari підтримують WebRTC без будь-якого плагіна

Продовження таблиці 1.1

Підтримувані відеокодеки	H.264, VP8, VP6, Sorenson Spark®, Screen, Video v1 & v2	Будь-які (немає прив'язки до кодеків)	AAC-LC, HE-AAC+ v1 & v2, xHE-AAC, Apple Lossless, FLAC	H.264, VP6	H.264, VP8, VP9
Підтримувані аудіокодеки	AAC, AAC-LC, HE-AAC+ v1 & v2, MP3, Speex, Opus, Vorbis	Будь-які (немає прив'язки до кодеків)	H.265, H.264	AAC, MP3	Opus, iSAC, iLBC
Переваги	Низька затримка і не вимагає буферизації	Незалежний від постачальника, міжнародний стандарт адаптивного бітрейту	Адаптивний бітрейт і широко підтримується	Адаптивна технологія бітрейту для Flash	Супер швидкий і браузерний
Недоліки	Не оптимізовано для якості досвіду чи масштабованості	Не підтримується iOS або Apple TV	Якість досвіду має пріоритет над низькою затримкою	Фірмова технологія з відсутністю підтримки	Призначений для відеоконференцій і не масштабується

2.4. Firebase

Рис. 2.3 – Логотип Firebase[13]

Великі корпорації сьогодні часто обирають нереляційні NoSQL бази даних, а не реляційні SQL, тому що NoSQL надає базі даних кілька переваг, а саме покращує швидкість її роботи та масштабованість системи. Такі бази даних виходять дешевшими, швидшими та більш безпечними для того, щоб розширити вже існуючу систему. Порівняння наведене у таблиці 2.1.

Згідно з дослідженнями Огівер Маргарети[14] у 2019 році статистичний тест критерій Уілкоксона показав, що Firebase є більш оптимальним варіантом для використання, ніж MySQL. Були досліджені операції створення, читання, оновлення та видалення з бази даних (CRUD). Для кожної операції Firebase показала кращий час відповіді.

Таблиця 2.1 – Порівняння реляційних та нереляційних БД

Реляційні БД	Нереляційні БД
Фіксована структура SQL ускладнює, або майже унеможливлює зміни у бізнес-логіці	Добре працює з неструктурованими та не пов’язаними між собою даними
Зміни у структурі таблиць є дуже проблематичними, трудомісткими та забирають багато часу	Втрачає деякі особливості для покращення швидкості та масштабованості
Мають недостатню продуктивність у порівнянні з новими БД та високу затримку	Значно дешевші, швидші та безпечніші для розширення вже існуючої програми

Firebase – це NoSQL безсерверна база даних, що працює у режимі реального часу, використовується для зберігання та синхронізації метаданих відео між клієнтами. Представляє собою цілу екосистему для побудови веб і мобільних додатків. Google Firebase є більш повним рішенням в порівнянні, наприклад, з популярною MongoDB. Це ідеальний варіант для розробки систем, що потребують отримання даних у режимі реального часу, таких як чати, системи з великою кількістю користувачів, програми для торгівлі акціями або програми для оновлення спортивних балів. Компанії, що використовують Firebase: Shazam, Alibaba.com, Duolingo, The Economist, The New York Times, Venmo та інші.

Розробка додатків без використання серверу має величезні переваги у часі для розробки та масштабованості програми. Завдяки цьому забезпечується можливість більше зосередитись на системі стрімінгу медіаконтенту та на покращенні взаємодії з користувачем, а не на розробці та налаштуванні серверу. Firebase дозволяє синхронізувати дані між клієнтами без кодування серверної частини, включаючи функції автономної синхронізації (досить складні у реалізації).

Зручно відбувається синхронізація даних між користувачами в режимі реального часу. Firebase повідомляє всі пристрої протягом короткого періоду. Коли відбувається зміна, усі користувачі отримують ці оновлення. Це надає гнучкість доступу до даних з будь-якого пристрою (через Інтернет або з мобільного). Оскільки дані розміщуються в хмарі, не потрібно підтримувати сервер. Ще одна перевага полягає в тому, що його можна використовувати і в автономному режимі. Коли з'єднання втрачено, база даних використовує локальний кеш на пристрої для зберігання змін.

Рис. 2.4 – Схема взаємодії Firebase з пристроями

Firebase від компанії Google дозволяє просто реалізовувати такий функціонал:

- Керує усіма даними в режимі реального часу в базі даних. Отже, обмін даними відбувається просто та швидко, а тому є сенс використовувати Firebase, якщо при розробці додатків, такі як пряма трансляція, повідомлення в чаті тощо.
- Дозволяє синхронізувати дані в режимі реального часу на всіх пристроях - Android, iOS та у браузерах, без потреби оновлювати сторінку або екран.
- Забезпечує інтеграцію з DoubleClick, Google Ads, AdMob, BigQuery, Play Store, Data Studio та Slack, аби зробити вашу програму ефективною та більш точною для управління та обслуговування.

Google надає набір засобів розробки (SDK) для входу у систему. Firebase надає багато серверних послуг, однією з яких є аутентифікація. Firebase підтримує аутентифікацію за допомогою паролів, електронної пошти, номерів телефону, а також сторонніх сервісів, що пропонують ідентифікацію, таких як Google, GitHub, Twitter тощо. Firebase також може пов’язати вашу власну існуючу систему аутентифікації, так що користувачам не потрібно буде

реєструватися знову, якщо ви вирішите використовувати Firebase, а також анонімну систему аутентифікації, яка дозволяє створювати тимчасові облікові записи, щоб користувачі, які ще не зареєструвались, мали можливість працювати з даними, захищеними правилами безпеки.[15] Цей модуль містить основні функції (Вхід, Реєстрація, Вхід за допомогою соціальних мереж, Скидання/зміна пароля, Скидання/зміна електронної пошти, перевірка SMS). Особливістю аутентифікації Firebase є те, що він дозволяє легко виконувати безпечні входи, що займає відносно багато часу для правильної реалізації власноруч. Firebase дає можливість мати повністю інтегровану аутентифікацію користувача у конкретному проекті.

Хмарне сховище Firebase (Firebase Cloud Storage) створено для розробників програмного забезпечення, яким потрібно зберігати та обробляти створений користувачем контент, як правило, це великі файли, такі як фотографії чи відео. Воно в основному використовується саме для медіаконтенту (такого як фотографії або відео), але може бути містити також й інші дані, наприклад текстові файли. Firebase Storage зберігає файли у Google Cloud Storage, спільному з додатком за замовчуванням Google App Engine, що дозволяє отримати доступ до них як за допомогою Firebase, так і через API Google Cloud. Це робить систему більш гнучкою, тобто дає можливість завантажувати файли з мобільних клієнтів за допомогою Firebase та виконувати обробку на стороні сервера, наприклад можна фільтрувати зображення та перекодовувати відео за допомогою Google Cloud Platform. Технологія, яка дозволяє зберігати та управляти різним медіаконтентом, що створюється користувачами інформаційної системи. Це модель хмарних обчислень, яка зберігає дані в Інтернеті через постачальника хмарних обчислень, який управляє та адмініструє сховище даних як послугу. Хмарне сховище Firebase виключає необхідність придбання та управління власною інфраструктурою для зберігання даних. Воно надає нам швидкість, глобальний масштаб, довговічність та доступ до даних будь-де і будь-коли.

Хмарне сховище Firebase здатне забезпечити виконання наступних дій:

- Надійність - одна з найбільших переваг Cloud Firestore. SDK Firebase здійснює завантаження медіаконтенту незалежно від якості мережевого з'єднання. Надійні завантаження означає, що за умови зупинки, відео перезапуститься з того місця, де воно зупинилося, та заощадить користувачеві час і пропускну здатність.
- Для забезпечення простої та інтуїтивно зрозумілої аутентифікації розробнику, SDK Firebase для хмарного сховища інтегрується з автентифікацією Firebase. Для дозволу доступу на основі імені файлу, розміру, типу вмісту та інших метаданих ми можемо використовувати декларативну модель безпеки.
- Хмарне сховище створено для масштабу Exabyte, коли наш додаток стає популярним. Легко переходити від прототипу до виробництва, використовуючи ту саму структуру, яка забезпечує Spotify та Google Photos.

Хмарне сховище закуповується у сторонніх постачальників хмарних послуг, які володіють та експлуатують ємність для зберігання даних та розповсюджують його через Інтернет за моделлю оплати. Ці постачальники хмарних сховищ керують безпекою, ємністю та стабільністю, щоб зробити дані доступними для наших додатків у всьому світі.

Хмарні функції для Firebase надають можливість писати код, який відповідає на події в елементах Firebase. Наприклад, зміни в базі даних реального часу, аутентифікації або інших елементах можуть бути використані для активації функції.[16]

Google Analytics для Firebase може використовуватися для відстеження подій відтворення відео, що відбуваються у стрімінговій системі. Основною метою можна назвати відстеження того, наскільки популярний контент у додатку. Висновки робляться на більш глибокому рівні, ніж звичайний підрахунок переглядів. Можна також побачити скільки користувачів переглядає контент майже в режимі реального часу. За допомогою функцій

Firebase Analytics можна підвищити якість взаємодії з користувачем програми. Проаналізувавши точну звітність про потік користувачів, їх діяльність, місцезнаходження та іншу інфографіку, порівняно легше відстежувати аудиторію та розуміти її уподобання.

На функціях Firebase зручно створювати систему стрімінгу медіаконтенту. Використовуючи такі засоби як Zencoder або Bitmovin можна перекодувати формат, аби подрібнити завантажене відео на менші частини з різною роздільною здатністю відео (та різними форматами, так, наприклад, для різних iOS вимагає HLS для потокової передавачі медіаконтенту). Клієнт може тим часом зберігати інформацію про фрагменти в базі даних реального часу (Realtime Database) (назва фрагмента, доступні роздільні здатності, кількість фрагментів) і може завантажувати ці фрагменти із сховища по мірі перегляду відео.

Firebase надає розробникам гнучкість та правила на основі виразів із синтаксисом, подібним до JavaScript, щоб визначити, як ваші дані мають бути структуровані, як їх слід індексувати та коли користувач може читати та додавати дані.[17]

Наведемо список основних переваг Firebase:

- Розробка програми без сервера.
- Синхронізація даних системи в режимі реального часу.
- Швидке відображення даних у системі.
- Більш швидка робота у порівнянні з серверними веб-сервісами.
- Доступ до веб-сайту може бути наданий через соціальні мережі.
- Аналітика.
- Хмарне сховище
- Тестування.
- Динамічне зв’язування.
- Автозавантаження.

Таблиця 2.2 – Порівняння Firebase та MongoDB

Критерій	Firebase	MongoDB
Загальні риси	Ідеальна база даних для зберігання та синхронізації даних у режимі реального часу	Безкоштовний відкритий код із високопродуктивною базою даних на основі документів
Продуктивність	Firebase має нижчу продуктивність, ніж MongoDB	MongoDB забезпечує високу продуктивність для додатків із великим трафіком
Ким розроблено	Google	MongoDB
Мови програмування	Objective-C, PHP, NodeJS, JavaScript, Swift, C ++ тощо	Java, JavaScript, PHP, NodeJS, C, C #, Perl, Python тощо
Безпека	Firebase не така безпечна, як MongoDB	Безпечніша за Firebase
Переваги	Миттєве оновлення даних без оновлення Легко синхронізувати кілька пристроїв з базою даних Хмарна черга подій Push-повідомлення Firebase у режимі реального часу Пропонує дуже швидкий CDN для статичних веб-сайтів	Динамічний - відсутність жорсткої структури Гнучкість - додавання та видалення полів має менший вплив на програму або не впливає на неї Представлення даних у JSON або BSON

Продовження таблиці 2.2

Застосування	Firebase ідеально підходить для невеликих програм Додатки, яким потрібні дані в режимі реального часу В перспективі необхідно масштабувати систему легко і часто Додатки для обміну миттєвими повідомленнями, онлайн-ігор та соціальних мереж Синхронізація в режимі реального часу між пристроями та браузерами	MongoDB найкраще підходить для масштабних програм Повне управління конфігурацією Ведення даних на основі місцезнаходження Управління даними великих підприємств
Недоліки	У Firebase немає реляційних запитів Складна міграція даних Ви не є власником серверів, на яких розміщені ваші дані, тому неможливо експортувати користувацькі дані.	Індексація та пошук у MongoDB не дуже потужні Не існує жодної функції чи збереженої процедури, де ви можете прив'язати логіку

2.5. Node.js

У якості мови для розробки системи стрімінгу медіаконтенту була обрана JavaScript.

Node.js чудово підходить для створення додатків з важким рендерингом на стороні клієнта, кількома одночасними запитами та частим перемішуванням даних з клієнта на сервер. Щоразу, коли ви замислюєтесь про створення важких додатків вводу-виводу та керування даними, Node.js однозначно повинен бути першим варіантом у вашому списку. Тим не менш,

Node.js однозначно повинен бути першим варіантом у вашому списку. Тим не менш, Node.js не настільки хороший для розробки інтенсивних процесорів, які передбачають генерацію та обробку зображень, аудіо чи відео. Будучи однопотоковим рішенням, Node.js може не реагувати та повільно обробляти великі файли. У цьому випадку найкращим варіантом будуть звичайні багатопотокові рішення.

Це дозволяє миттєво запустити власну багатоекранну платформу Video On Demand (VOD), включаючи хостинг, Video CMS, перекодування, Інтернет-програвач відео, веб-сайт та програми для мобільних пристроїв.

Висновок до розділу 2

Після ретельного огляду програмних засобів організації стрімінгу медіаконтенту в другому розділі даного дипломного проєкту можна зробити такі висновки:

- Досліджено необхідність та корисність існування мереж розповсюдження контенту.
- Розглянуто основні протоколи, що використовуються для стрімінгу медіаконтенту, такі як RTMP, MPEG-DASH, HLS, HDS, WebRTC.
- Сформована детальна порівняльна таблиця стрімінгових протоколів.
- Досліджено базу даних Firebase, її основні сервіси та застосування, порівняно її з іншими сучасними рішеннями.
- Оглянуто засоби та технології для розробки програмного забезпечення, зокрема обрано мову програмування JavaScript.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ

Використовуючи наведені вище програмні засоби можна розробити систему стрімінгу медіаконтенту. У даному розділі детально опишемо проектування та реалізацію нашого додатку.

3.1. Середовище розробки

Atom (рис. 3.1) - це безкоштовний редактор тексту та вихідного коду з відкритим кодом, розроблений GitHub (Atom - Hackable Text and Source Code Editor for Linux). Його розробники називають його "текстовим редактором для 21 століття, що можна зламати". Atom дозволяє користувачам встановлювати сторонні пакети та теми для налаштування функцій та зовнішнього вигляду редактора, тому ви можете налаштувати його відповідно до своїх уподобань та з легкістю. Це так само приємно для новачка, як і для досвідченого розробника.

Пакети за замовчуванням Atom можуть застосовувати підсвічування синтаксису для багатьох мов програмування та форматів файлів.

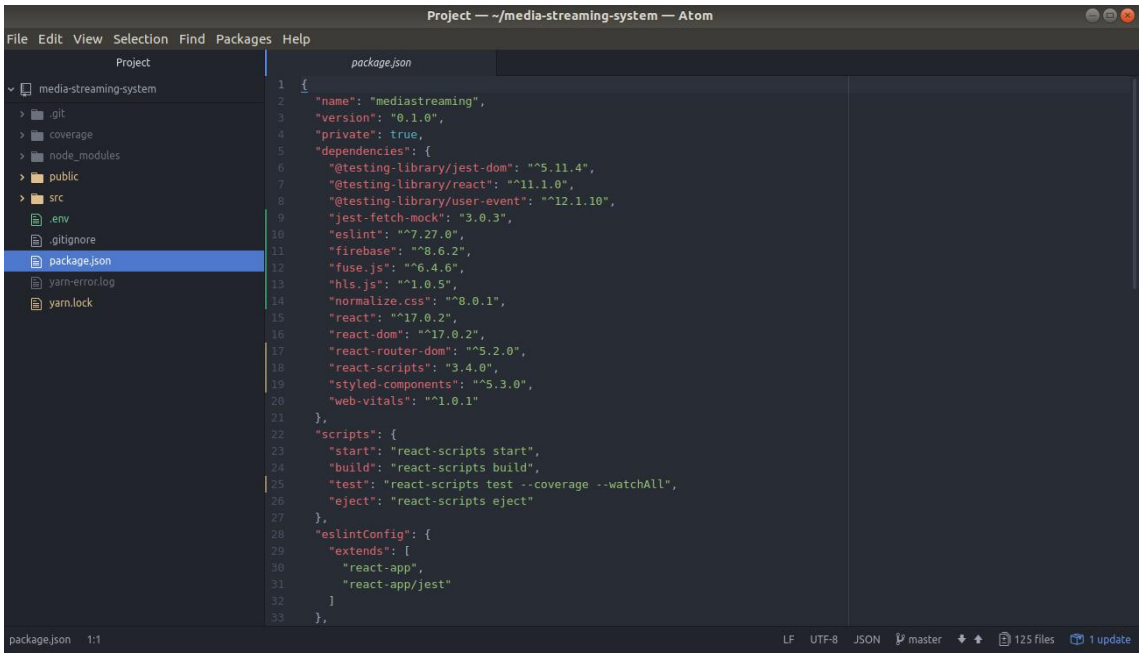


Рис. 3.1 – Середовище розробки

3.2. Розробка програмного коду

Беручи до уваги аналіз програмних особливостей реалізації спроектуємо та розробимо систему.

3.2.1. Components

У директорії компонентів (components) (рис. 3.2) містяться необхідні для системи модулі.

А саме: компонент картки (card), компонент опису (feature), компонент форми (form), компонент заголовку (header), компонент програвача (player), компонент профілів (profiles).

Рис. 3.2 – Структура Components

У файлі index.js (рис. 3.2), що знаходиться також у директорії з компонентами, експортуються усі модулі компонентів для того, аби зручно використовувати їх у інших частинах програмного коду.

index.js

```
1  export { default as Header } from './header';
2  export { default as Feature } from './feature';
3  export { default as Form } from './form';
4  export { default as Profiles } from './profiles';
5  export { default as Card } from './card';
6  export { default as Player } from './player';
7
```

Рис. 3.3 – Експорт компонентів

Важливим моментом є те, що модулі ES6 експортують прив'язки, а не значення або посилання.

Перейдемо до першого компоненту – компонент заголовку (header) (рис. 3.4). Він буде використовуватися на всіх сторінках системи стрімінгу медіаконтенту.

Компонент заголовку містить у собі наступні елементи: Picture (для відображення фото користувача), TextLink (для відображення переходу між різними категоріями відео), ButtonLink (для відображення кнопки авторизації), Text (для відображення тексту) та інші.

index.js — components | index.js — components/header

```
68    return <Profile {...restProps}>{children}</Profile>;
69  };
70
71  Header.Feature = function HeaderFeature({ children, ...restProps }) {
72    return <Feature>{children}</Feature>;
73  };
74
75  Header.Picture = function HeaderPicture({ src, ...restProps }) {
76    return <Picture {...restProps} src={`/images/users/${src}.png`} />;
77  };
78
79  Header.Dropdown = function HeaderDropdown({ children, ...restProps }) {
80    return <Dropdown {...restProps}>{children}</Dropdown>;
81  };
82
83  Header.TextLink = function HeaderTextLink({ children, ...restProps }) {
84    return <Link {...restProps}>{children}</Link>;
85  };
86
87  Header.PlayButton = function HeaderPlayButton({ children, ...restProps }) {
88    return <PlayButton {...restProps}>{children}</PlayButton>;
89  };
90
91  Header.FeatureCallOut = function HeaderFeatureCallOut({ children, ...restProps }) {
92    return <FeatureCallOut {...restProps}>{children}</FeatureCallOut>;
93  };
94
95  Header.Text = function HeaderText({ children, ...restProps }) {
96    return <Text {...restProps}>{children}</Text>;
97  };
98
99  Header.ButtonLink = function HeaderButtonLink({ children, ...restProps }) {
100   return <ButtonLink {...restProps}>{children}</ButtonLink>;
101 };
102
```

LF UTF-8 JavaScript master 123 files 1 update

Рис. 3.4 – Компонент header

А також в ньому присутній елемент Header.Search (рис. 3.5), який представляє собою пошукову стрічку та дозволить виконувати пошук посеред доступних відео.

```
Header.Search = function HeaderSearch({ searchTerm, setSearchTerm, ...restProps }) {
  const [searchActive, setSearchActive] = useState(false);

  return (
    <Search {...restProps}>
      <SearchIcon onClick={() => setSearchActive((searchActive) => !searchActive)} data-testid="search-click">
        <img src="/images/icons/search.png" alt="Search" />
      </SearchIcon>
      <SearchInput
        value={searchTerm}
        onChange={({ target }) => setSearchTerm(target.value)}
        placeholder="Search videos"
        active={searchActive}
        data-testid="search-input"
      />
    </Search>
  );
};
```

Рис. 3.5 – Елемент Search

Розглянемо детальніше компонент форми (form) (рис. 3.6). Компонент цієї форми використовується як для аутентифікації користувача, так і для його реєстрації.

Складається з таких елементів: Error (для відображення помилок при некоректному вводі даних при реєстрації або при аутентифікації), Title (для відображення заголовку форми), Link (для відображення посилання на форму аутентифікації, якщо користувач уже є в системі, та навпаки), Input (для відображення полів вводу даних), Submit (для відправлення форми), Text (для відображення тексту) та інші.

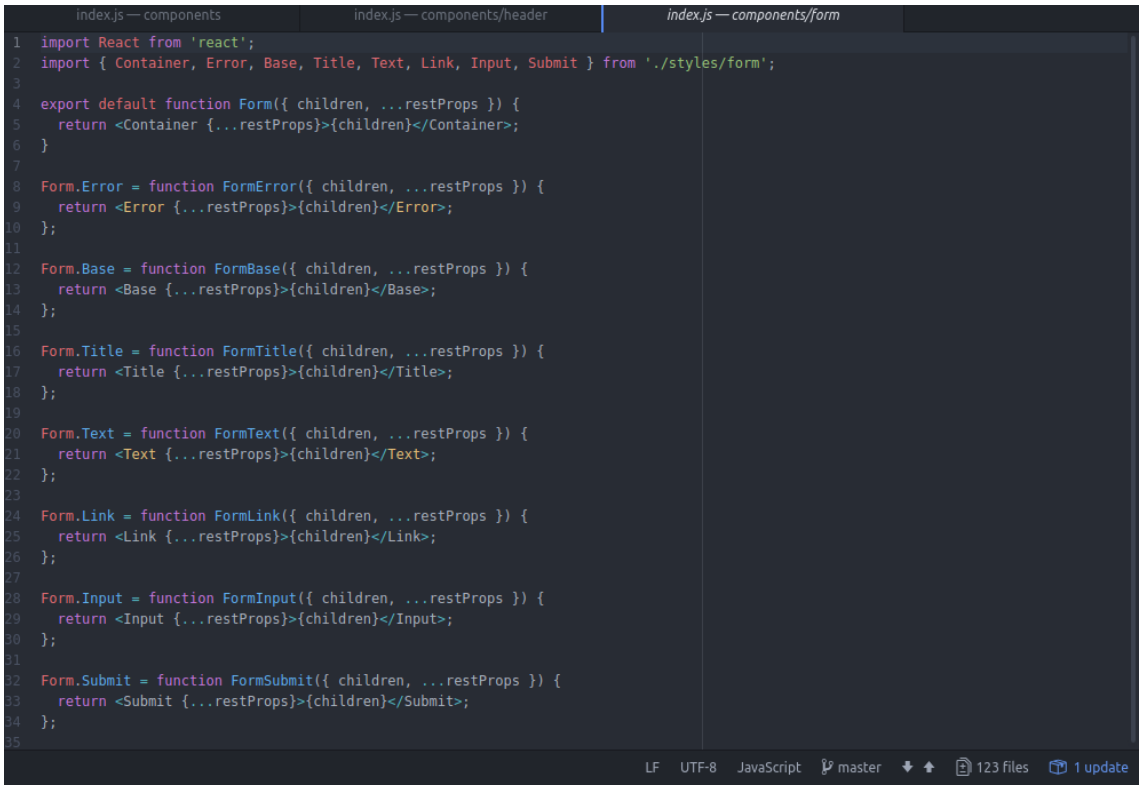


Рис. 3.6 – Компонент form

Цікавим є також компонент картки (card) (рис. 3.7). Компонент картки дозволяє лаконічно та впорядковано відображати відео та пов’язану з ними інформацію (обкладинка, опис, назва тощо).

Складається з таких елементів: SubTitle (для відображення опису відео), Title (для відображення заголовку форми), Text (для відображення тексту), Item (для відображення карток з медіаконтентом, також при натисканні відображується опис відео та з’являється можливість його переглянути) та інші.

Відображення опису відео у елементі Item реалізовано у функціях setShowFeature() та setItemFeature().

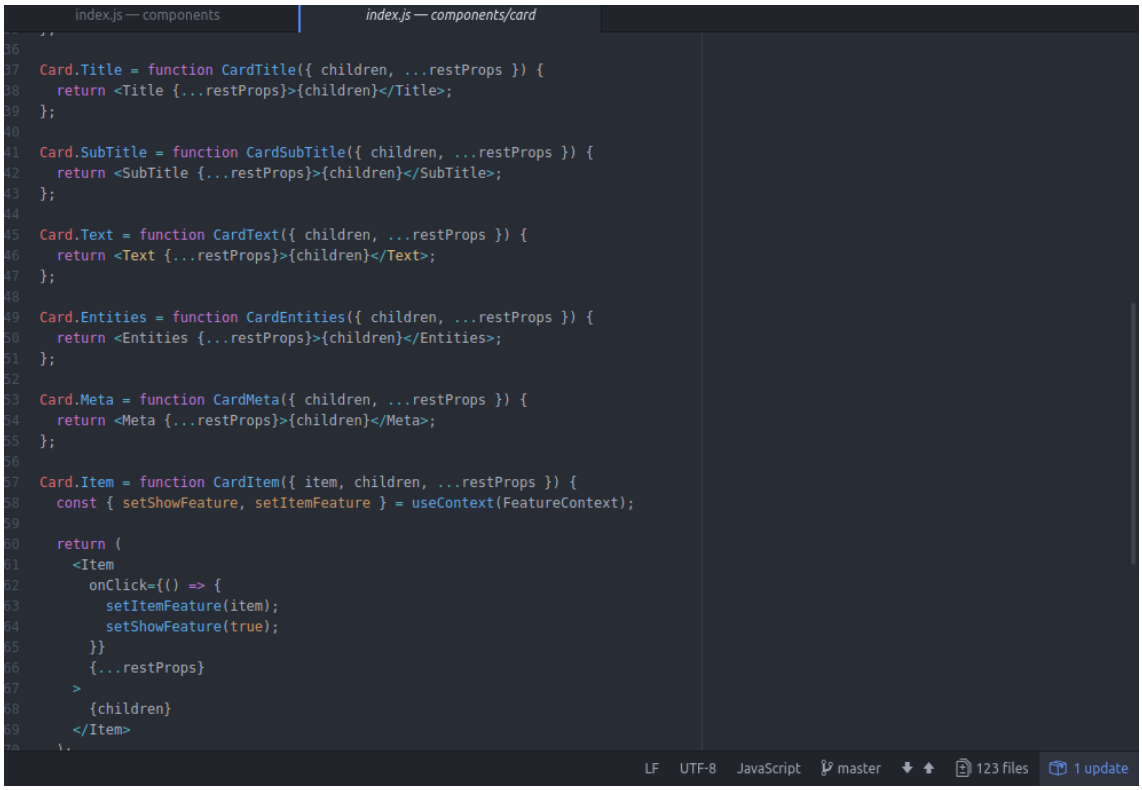


Рис. 3.7 – Компонент card

А також в ньому присутній елемент Header.Feature (рис. 3.8), який представляє собою опис обраного відео та можливість переглянути його.

```
Card.Feature = function CardFeature({ children, category, ...restProps }) {
  const { showFeature, itemFeature, setShowFeature } = useContext(FeatureContext);

  return showFeature ? (
    <Feature {...restProps} src={`/images/${category}/${itemFeature.genre}/${itemFeature.slug}/large.jpg`}>
      <Content>
        <FeatureTitle>{itemFeature.title}</FeatureTitle>
        <FeatureText>{itemFeature.description}</FeatureText>
        <FeatureClose onClick={() => setShowFeature(false)}>
          <img src="/images/icons/close.png" alt="Close" />
        </FeatureClose>
        {children}
      </Content>
    </Feature>
  ) : null;
};
```

Рис. 3.8 – Елемент Feature

3.2.2. Containers

Програма містить два контейнери (containers) – це контейнер заголовку (header) (рис. 3.9) та контейнер пошуку або перегляду (рис. 3.10) (browse).

Обидва контейнери описують вміст веб-сторінок та описують комплексні елементи, у які ще можна передавати аргументи (children).

header.js

```
import React from 'react';
import { Header } from '../components';
import * as ROUTES from '../constants/routes';
import logo from '../logo.svg';

export function HeaderContainer({ children }) {
  return (
    <Header>
      <Header.Frame>
        <Header.Logo to={ROUTES.HOME} src={logo} alt="home-bg" />
        <Header.ButtonLink to={ROUTES.SIGN_IN}>Sign In</Header.ButtonLink>
      </Header.Frame>
      {children}
    </Header>
  );
}
```

Рис. 3.9 – Контейнер Header

header.js browse.js

```
return profile.displayName ? (
  <>
    <Header src = "ntu_kpi" dontShowOnSmallViewPort>
      <Header.Frame>
        <Header.Group>
          <Header.Logo to={ROUTES.HOME} src={logo} alt="home-bg" />
          <Header.TextLink active={category === 'kpi' ? 'true' : 'false'} onClick={() => setCategory('kpi')}>
            KPI
          </Header.TextLink>
          <Header.TextLink active={category === 'lectures' ? 'true' : 'false'} onClick={() => setCategory('lectures')}>
            Lectures
          </Header.TextLink>
        </Header.Group>
        <Header.Group>
          <Header.Search searchTerm={searchTerm} setSearchTerm={setSearchTerm} />
          <Header.Profile>
            <Header.Picture src={user.photoURL} />
            <Header.Dropdown>
              <Header.Group>
                <Header.Picture src={user.photoURL} />
                <Header.TextLink>{user.displayName}</Header.TextLink>
              </Header.Group>
              <Header.Group>
                <Header.TextLink onClick={() => firebase.auth().signOut()}>Sign out</Header.TextLink>
              </Header.Group>
            </Header.Dropdown>
          </Header.Profile>
        </Header.Group>
      </Header.Frame>

      <Header.Feature />
    </Header>

    <Card.Group>
      {slideRows.map((slideItem) => (
```

LF UTF-8 JavaScript master 122 files 1 update

Рис. 3.10 – Контейнер Browse

Для імплементації функціоналу пошуку у нашій системі стрімінгу медіаконтенту (рис. 3.11) використовується інструмент Fuse.js. Fuse.js - це бібліотека JavaScript, яка надає можливості нечіткого пошуку (толерантний до друкарських помилок) програм та веб-сайтів. Це приємно і легко

використовувати з коробки, але також включає параметри конфігурації, які дозволяють налаштувати та створювати потужні рішення.

Fuse.js надає "нечіткий пошук", що означає, що він намагається допомогти вам у випадку, якщо ви не впевнені, що шукаєте, або неправильно пишете свій запит.

Внутрішній досвід користувачів, який ми прагнули створити, включав пошук під час введення за маршрутом глосарію та ресурсів, відображаючи всі відповідні екземпляри ключового слова в різних полях глосарію або елементів ресурсів, інакше відображаючи повний глосарій або список ресурсів, якщо результатів не знайдено.

```
18  useEffect(() => {
19    setSlideRows(slides[category]);
20  }, [slides, category]);
21
22  useEffect(() => {
23    const fuse = new Fuse(slideRows, { keys: ['data.description', 'data.title', 'data.genre'] });
24    const results = fuse.search(searchTerm).map(({ item }) => item);
25
26    if (slideRows.length > 0 && searchTerm.length > 3 && results.length > 0) {
27      setSlideRows(results);
28    } else {
29      setSlideRows(slides[category]);
30    }
31  }, [searchTerm]);
```

Рис. 3.11 – Функція пошуку

3.2.3. Routes

У файлі routes.js (рис. 3.12) зберігаються усі шляхи, що використовуються у проекті у вигляді констант: «/» (домашня сторінка), «/browse» (сторінка перегляду медіаконтенту), «/signup» (сторінка реєстрації), «/signin» (сторінка аутентифікації).

Організація компонентів Route в одному конфігураційному файлі є оптимальним рішенням, так як в цьому разі можна легко знаходити всі вкладені маршрути та налаштовувати їх, змінювати посилання, перевіряти їх приватність, налаштовувати переспрямування прямо з цього файлу.

routes.js

```
export const HOME = '/';
export const BROWSE = '/browse';
export const SIGN_UP = '/signup';
export const SIGN_IN = '/signin';

```

Рис. 3.12 - Шляхи

3.2.4. Utils

Основною утилітою у нашому програмному коді є фільтрація за вибором (selectionFilter()), що реалізована у модулі selection-filter.js (рис. 3.13).

Дана функція дозволяє нам відфільтрувати об’єкти у базі даних за категоріями.

selection-filter.js

```
export default function selectionFilter({ kpi, lectures } = []) {
  return {
    kpi: [
      { title: 'Institutes and Faculties of University', data: kpi?.filter((item) => item.genre === 'faculties') },
      { title: 'EMI/EFVV', data: kpi?.filter((item) => item.genre === 'evi') },
      { title: 'Students Organizations', data: kpi?.filter((item) => item.genre === 'circle') },
      { title: 'Entertaining', data: kpi?.filter((item) => item.genre === 'entertaining') },
    ],
    lectures: [
      { title: 'What is Data Science', data: lectures?.filter((item) => item.genre === 'data') },
      { title: 'History', data: lectures?.filter((item) => item.genre === 'history') },
      { title: 'Learn SQL and Databases for Data Science with IBM', data: lectures?.filter((item) => item.genre === 'sql') },
    ],
  };
}

```

Рис. 3.13 - Фільтрація

3.2.5. Pages

Наразі система стрімінгу медіаконтенту складається з п’яти сторінок (рис. 3.15): домашня сторінка, сторінка аутентифікації, сторінка реєстрації, сторінка перегляду медіаконтенту.

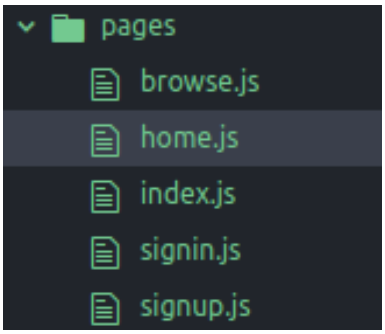


Рис. 3.14 – Сторінки веб-сайту

Також у директорії pages наявний файл index.js (рис. 3.15), у якому експортуються усі сторінки для подальшого використання зовні.

index.js

```
export { default as Home } from './home';
export { default as Browse } from './browse';
export { default as SignIn } from './signin';
export { default as SignUp } from './signup';
```

Рис. 3.15 – Експорт сторінок

Домашня сторінка веб-сайту (home.js) (рис. 3.16) використовує компонент опису (feature) для відображення основної інформації про систему, а також створений раніше контейнер заголовку (header).

Функціонал стартової сторінки доволі невеликий, адже переглядати відео можуть лише авторизовані користувачі. А зі стартової сторінки користувачі можуть перейти лише на сторінки аутентифікації та реєстрації.

home.js

```
import React from 'react';
import { Feature } from '../components';
import { HeaderContainer } from '../containers/header';

export default function Home() {
  return (
    <>
      <HeaderContainer>
        <Feature>
          <Feature.Title>MEDIA STREAMING SYSTEM</Feature.Title>
          <Feature.SubTitle>On this platform, you can browse media content. The content is broadcast in real time, the download spe
        </Feature>
      </HeaderContainer>
    </>
  );
}
```

Рис. 3.16 – Код домашної сторінки

Сторінки аутентифікації (signin.js) (рис. 3.17) та реєстрації (signup.js) (3.18) виглядають та проектуються приблизно однаковим чином.

Для аутентифікації та реєстрації використовуються функції Firebase signInwithEmailAndPassword для аутентифікації та createUserWithEmailandPassword для реєстрації.

Обидві сторінки використовують створений раніше компонент форми з тією відмінністю, що сторінка реєстрації має поле «Ім’я», яке не має форма аутентифікації.

signin.js

```
import React, {useState, useContext} from 'react';
import {useHistory} from 'react-router-dom';
import {FirebaseContext} from '../context/firebase';
import { HeaderContainer } from '../containers/header';
import {Form} from '../components';
import * as ROUTES from '../constants/routes';

export default function SignIn() {
  const history = useHistory();
  const {firebase} = useContext(FirebaseContext);
  const [emailAdress, setEmailAddress] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const isInvalid = password === '' || emailAdress === '';
  const handleSignIn = (event) => {
    event.preventDefault();
    firebase
      .auth()
      .signInWithEmailAndPassword(emailAdress, password)
      .then(() => {
        history.push(ROUTES.BROWSE);
      })
      .catch((error) => {
        setEmailAddress('');
        setPassword('');
        setError(error.message);
      });
```

Рис. 3.17 – Код сторінки аутентифікації

signup.js

```
import React, {useState, useContext} from 'react';
import {useHistory} from 'react-router-dom';
import {FirebaseContext} from '../context/firebase';
import { HeaderContainer } from '../containers/header';
import {Form} from '../components';
import * as ROUTES from '../constants/routes';

export default function SignUp() {
  const history = useHistory();
  const {firebase} = useContext(FirebaseContext);

  const [firstName, setFirstName] = useState('');
  const [emailAddress, setEmailAddress] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const isInvalid = firstName === '' || password === '' || emailAddress === '';

  const handleSignup = (event) => {
    event.preventDefault();

    return firebase
      .auth()
      .createUserWithEmailAndPassword(emailAddress, password)
      .then((result) =>
        result.user
          .updateProfile({
            displayName: firstName,
            photoURL: Math.floor(Math.random() * 5) + 1,
          })
          .then(() => {
            history.push(ROUTES.BROWSE);
          })
      )
      .catch((error) => {
```

LF UTF-8 JavaScript master 122 files 1 update

Рис. 3.18 – Код сторінки реєстрації

Сторінка перегляду медіаконтенту (browse.js) (рис. 3.19) використовує створений раніше контейнер BrowseContainer, у якому реалізовано основну структуру модуля, що розроблюється.

У вихідному коді даної сторінки ми отримуємо за допомогою функції useContent() відео за категоріями, відфільтровуємо необхідні за допомогою selectionFilter() та передаємо їх у контейнер перегляду медіаконтенту (BrowseContainer).

browse.js

```
import React from 'react';
import { BrowseContainer } from '../containers/browse';
import { useContent } from '../hooks';
import { selectionFilter } from '../utils';

export default function Browse() {
  const { kpi } = useContent('kpi');
  const { lectures } = useContent('lectures');
  const slides = selectionFilter({ kpi, lectures });

  return <BrowseContainer slides={slides} />;
}
```

Рис. 3.19 – Код сторінки перегляду медіаконтенту

3.2.6. Firebase

Оглянемо конфігурацію (рис. 3.20) підключення до бази даних Firebase. Ініціалізуємо додаток Firebase з оновленою конфігурацією.

firebase.prod.js

```
import Firebase from 'firebase/app';
import 'firebase/firestore';
import 'firebase/auth';

const config = {
  apiKey: "AIzaSyAi2PWLpg8ATeipz_Cz1y2kx8vpfFK5wSc",
  authDomain: "media-streaming-6c4e2.firebaseapp.com",
  projectId: "media-streaming-6c4e2",
  storageBucket: "media-streaming-6c4e2.appspot.com",
  messagingSenderId: "684283765266",
  appId: "1:684283765266:web:d3eab7c17e91339f152ecb"
};

const firebase = Firebase.initializeApp(config);

export { firebase };
```

Рис. 3.20 – Конфігураційний файл Firebase

```
firebase.js
1  import { createContext } from 'react';
2
3  export const FirebaseContext = createContext(null);
4
```

Рис. 3.21 – Контекст Firebase

Створена база даних «media-streaming» відображається в консолі Firebase (рис. 3.20).

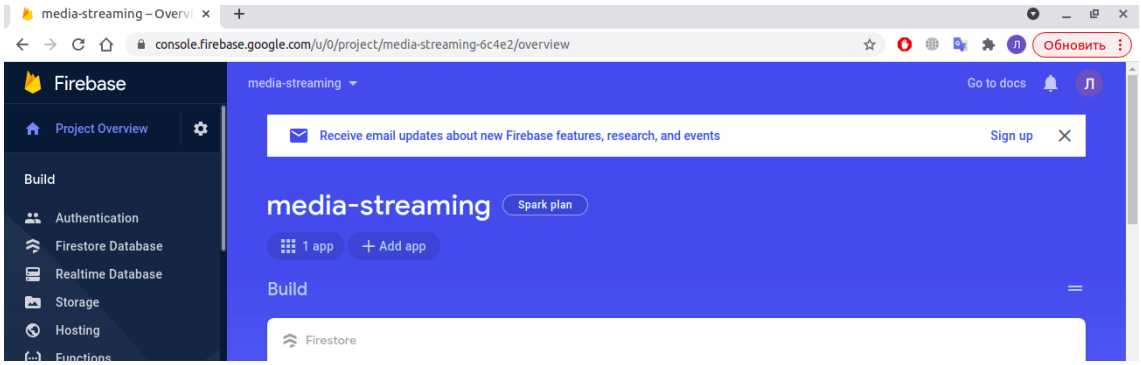


Рис. 3.20 – Консоль Firebase

Так як Firebase надає власні послуги з аутентифікації та реєстрації, то список користувачів з іменами, поштами, паролями та ідентифікаторами зберігається в окремому розділі «Authentication» в консолі Firebase.

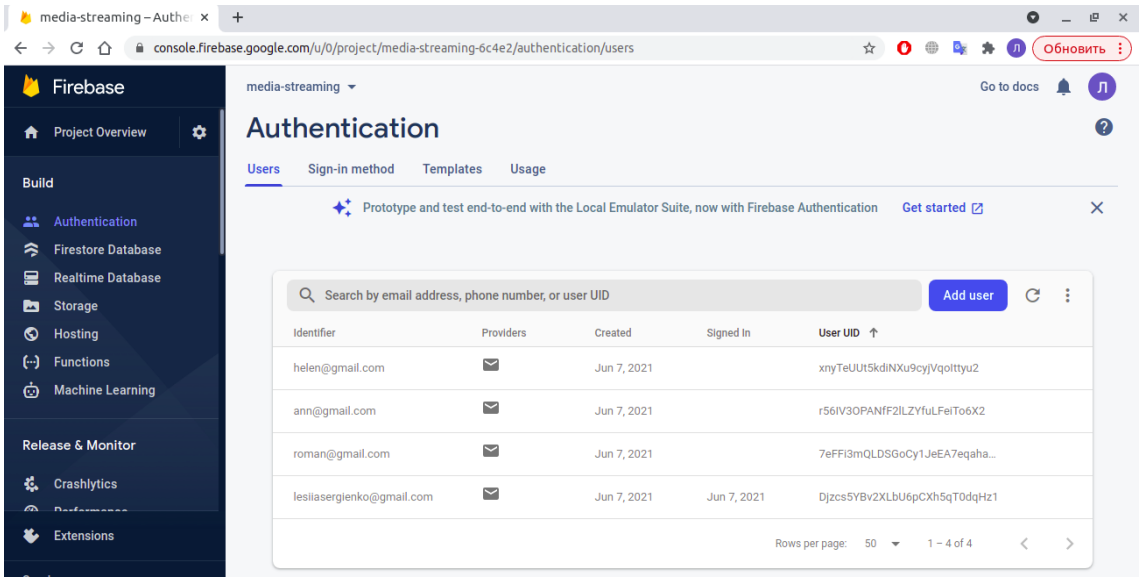


Рис. 3.21 – Список зареєстрованих користувачів Firebase

Створюємо у базі даних дві таблиці («lectures» (рис. 3.22) та «kpi» (рис.3.23)) з інформацією про медіаконтент.

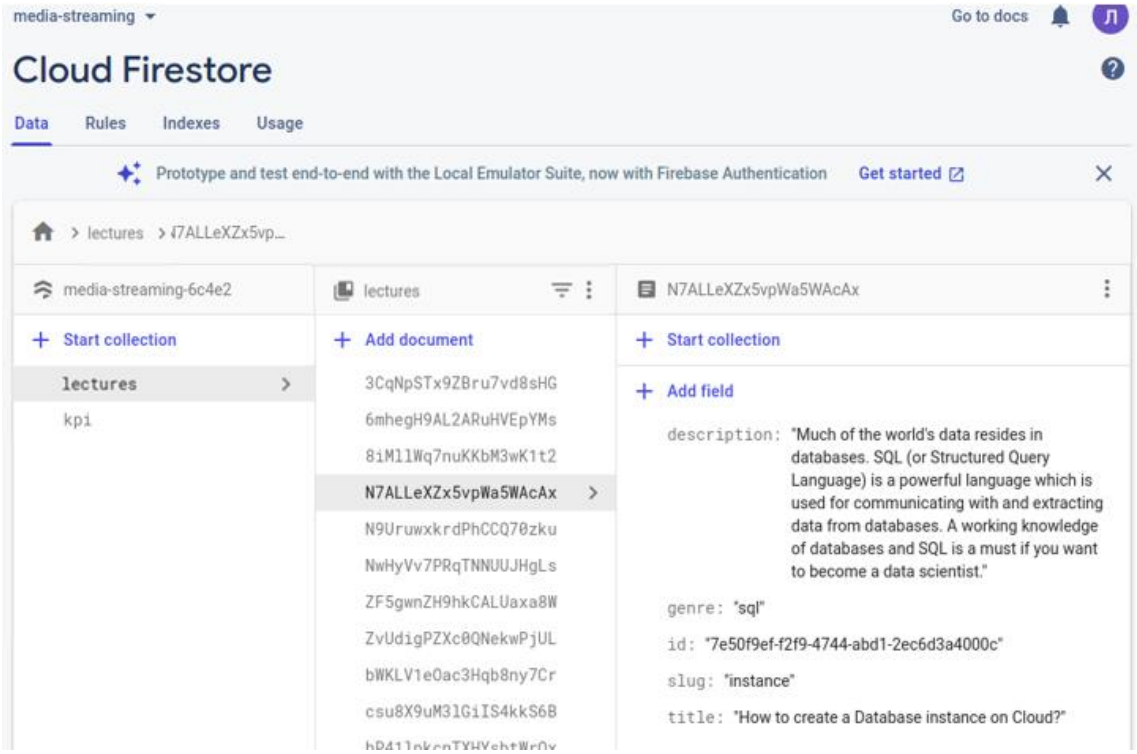


Рис. 3.22 – Таблиця «lectures» Firebase

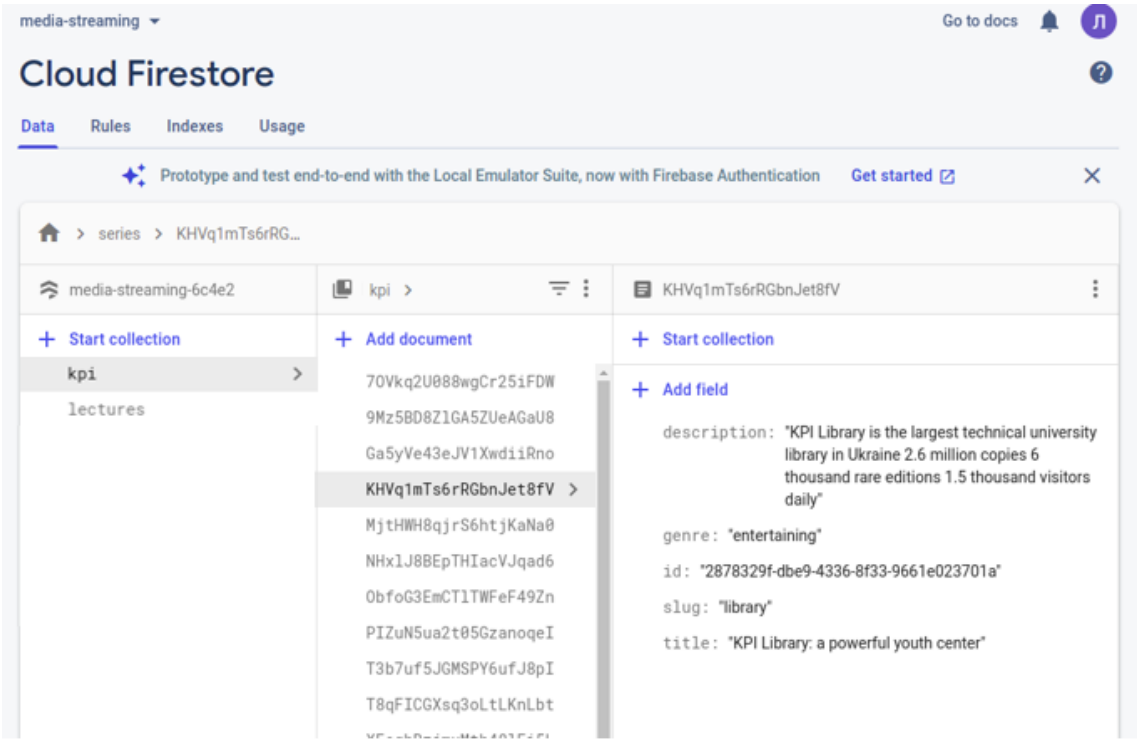


Рис. 3.23 – Таблиця «kpi» Firebase

Висновок до розділу 3

В третьому розділі було спроектовано та розроблено вихідний код системи стрімінгу медіаконтенту. Можна зробити такі висновки:

- Програмні компоненти розроблено відповідно до задачі, що була поставлена.
- Програмно реалізовано систему стрімінгу медіаконтенту.
- Інструменти, що були обрані у розділі 3, виявилися зручними, що значно полегшило та пришвидшило процес написання вихідного коду.
- Реалізовано аутентифікацію користувача, реєстрацію користувача, можливість перегляду та пошуку медіаконтенту.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ СИСТЕМИ

Отримана в результаті система представлена веб-сайтом. В даному розділі продемонструємо та оглянемо отриманий інтерфейс розробленої системи. А також проведемо тестування розроблених модулів.

4.1. Інтерфейс системи

Система складається з трьох основних компонентів: домашня сторінка, авторизація та реєстрація, сторінка перегляду медіаконтенту. Розглянемо їх більш детально.

4.1.1. Домашня сторінка

Для запуску системи стрімінгу медіаконтенту локально на своєму пристрої необхідно перейти за посиланням http://localhost:3000/. Стартова сторінка (рис. 4.1) надає нам інформацію про систему стрімінгу медіаконтенту. Також є можливість спробувати увійти до системи (за допомогою кнопки Sign In). При натисканні на логотип Media Streaming System у верхньому лівому куті користувач повертається на домашню сторінку.

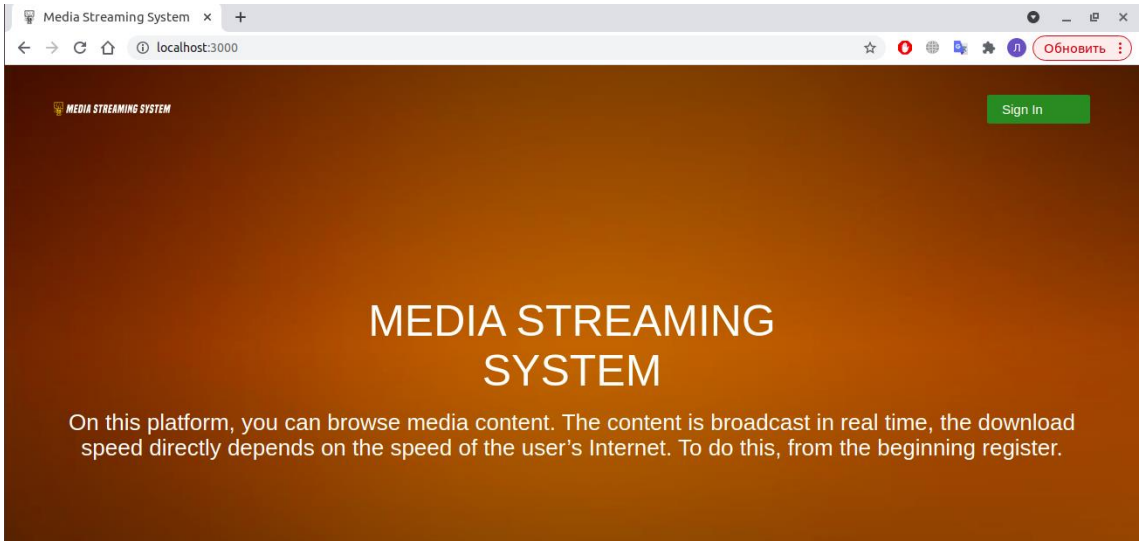


Рис. 4.1 – Стартова сторінка системи

4.1.2. Аутентифікація та реєстрація

Далі користувач може перейти на сторінку аутентифікації (рис. 4.2). Аутентифікація відбувається шляхом вводу адреси електронної пошти та паролю. Кнопку Sign In при цьому можна натиснути тільки у випадку, коли користувач почав введення у обох полях. Під формою незареєстрований користувач має можливість натиснути на посилання «Create an account» та перейти до форми реєстрації.

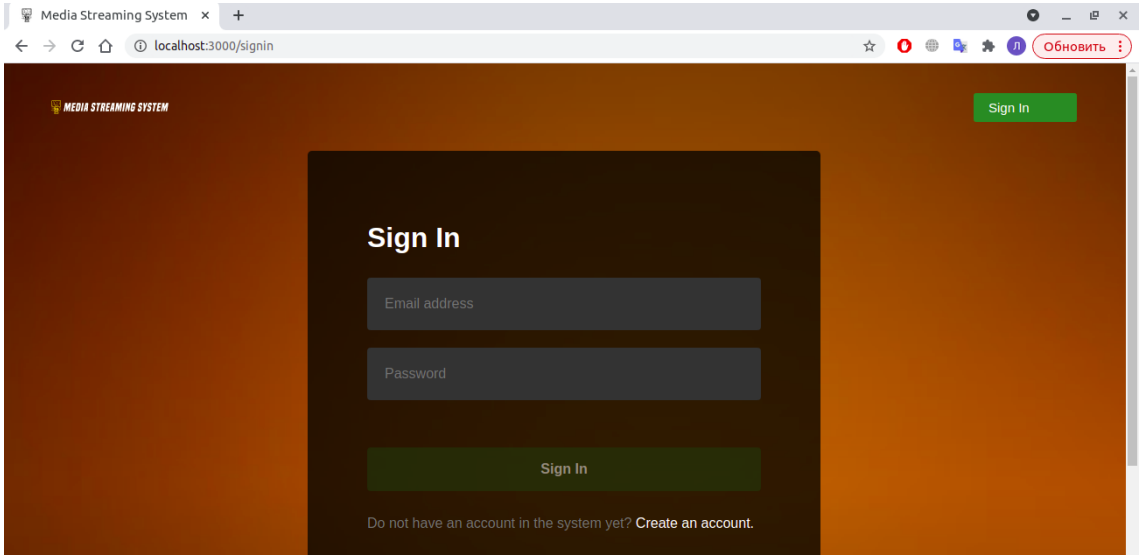


Рис. 4.2 – Форма авторизації

Якщо користувач вперше заходить до системи він має можливість зареєструватися в ній. Необхідно ввести значення до форми (рис. 4.3) у три поля: «First name»(ім'я), «Email address»(електронна пошта) та «Password» (пароль).

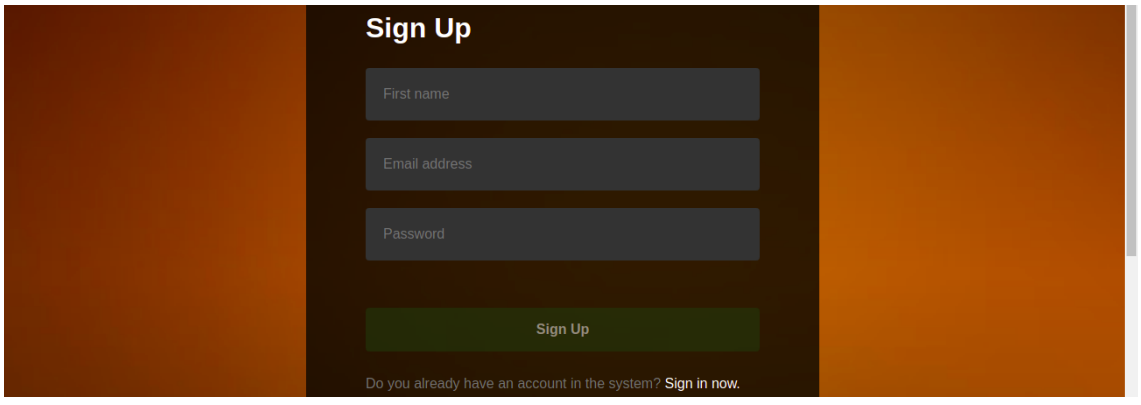


Рис. 4.3 – Форма реєстрації

У випадку введення некоректних даних, або спроби реєстрації з вже занесеною до бази даних адресою електронної пошти, реєстрація не буде успішною, а користувач побачить повідомлення про помилки (рис. 4.4).

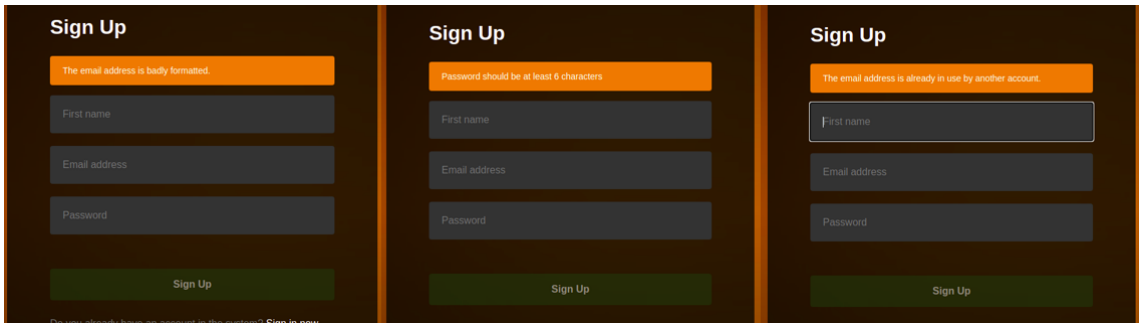


Рис. 4.4 – Приклади введення некоректних даних

За успішної реєстрації (рис. 4.5) користувач заноситься до бази даних та перенаправляється на сторінку перегляду медіаконтенту.

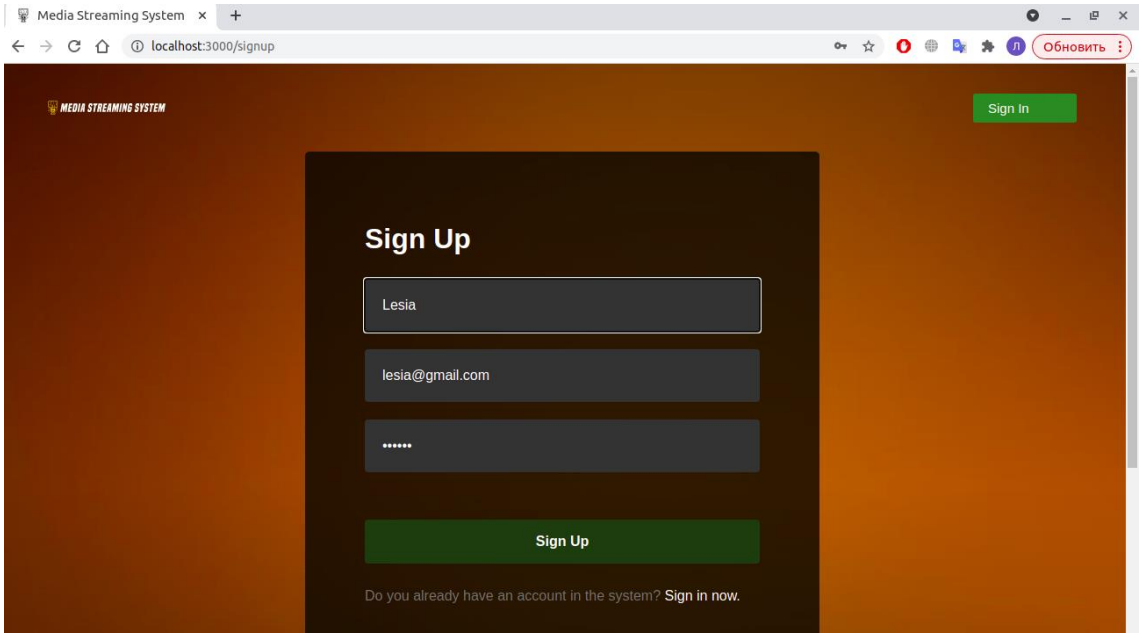


Рис. 4.5 – Приклад введення коректних даних

4.1.3. Сторінка перегляду медіаконтенту

При переході на сторінку перегляду медіаконтенту користувач має можливість спочатку обрати плейлист з контентом (наразі це «KPI» (рис. 4.6), де зберігаються відео про КПІ ім. Ігоря Сікорського, та «Lectures» (рис. 4.7), де зберігаються цікаві лекції). Також відео відокремлені за розділами. У верхньому лівому куті логотип системи, що перенаправляє користувача на головну сторінку.

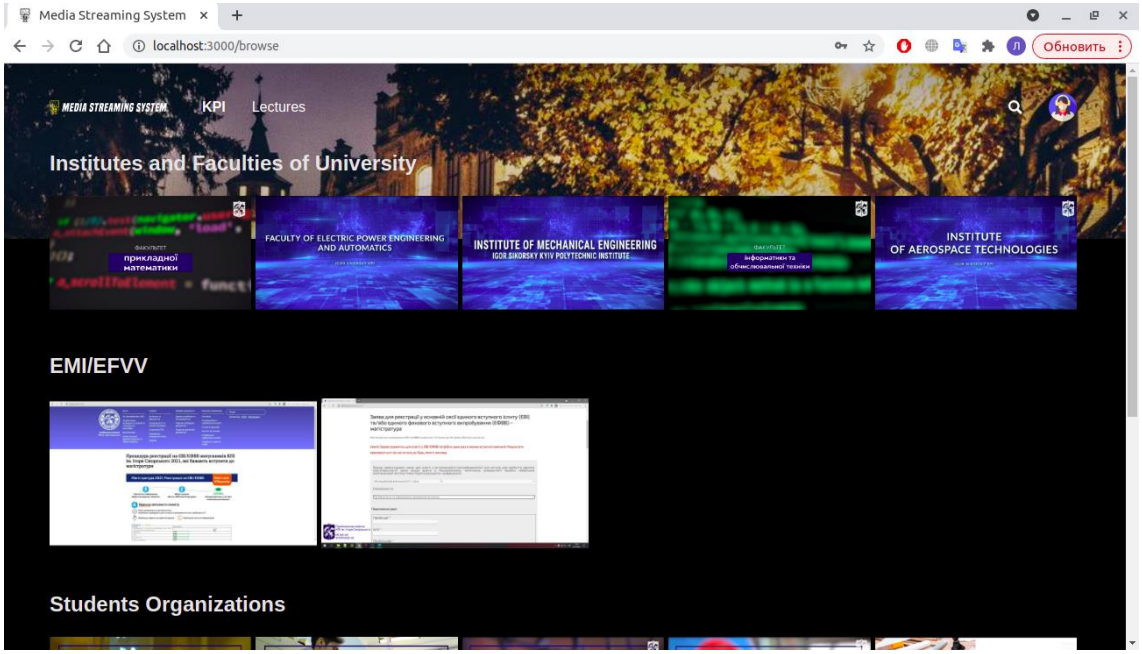


Рис. 4.6 – Сторінка перегляду медіаконтенту (KPI)

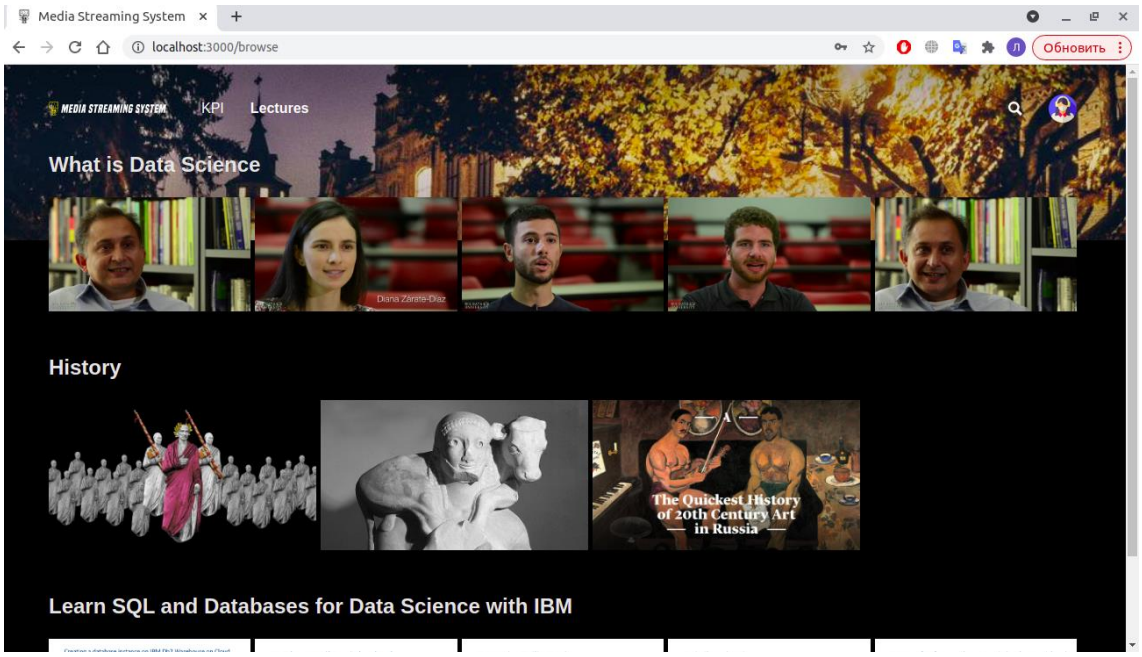


Рис. 4.7 – Сторінка перегляду медіаконтенту (Lectures)

У правому верхньому куті є поле для пошуку відео (рис. 4.8).

Рис. 4.8 – Поле для пошуку

Пошук (рис. 4.9) відбувається за ключовими словами, що вводяться у поле, відповідність буде знайдена у назвах або описах до відео. Розділи медіаконтенту, що містять відео відповідне пошуковим параметрам, піднімаються вище у пошуку. Якщо співпадіння часткові або наявні у декількох відео, то відображаються усі категорії. Наприклад, користувач вводить у пошукову стрічку «registra» і йому в першу чергу випадає відео з реєстрацією (registration) на ЄВІ.

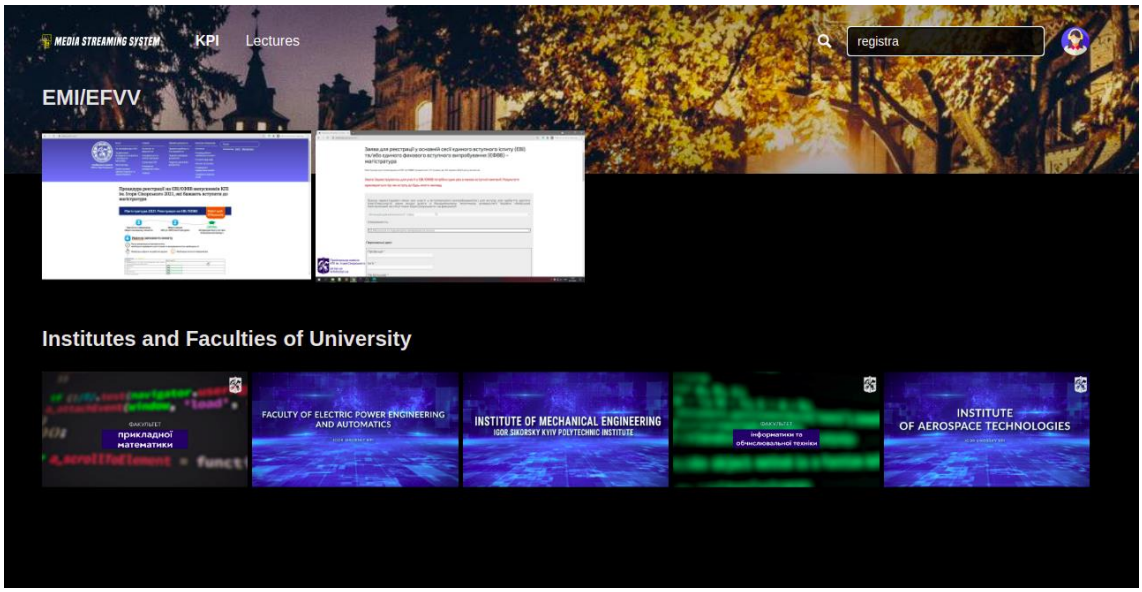


Рис. 4.9 – Пошук на сторінці перегляду медіаконтенту

При натисканні на обкладинку відео користувач побачить його опис (рис. 4.10) та зможе переглянути його, якщо натисне на кнопку «PLAY».

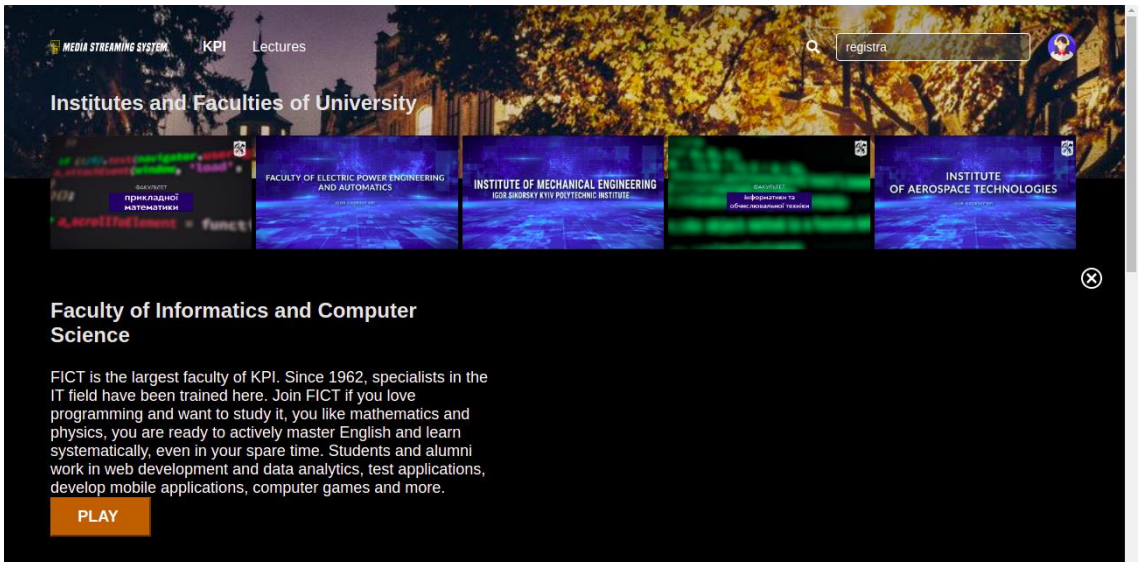


Рис. 4.10 – Опис обраного відео

При натисканні на кнопку «PLAY» користувач перенаправляється у медіаплеєр (рис. 4.7), де може переглянути конент. Закрити відео можна якщо натиснути на іконку хрестика у верхньому правому куті.

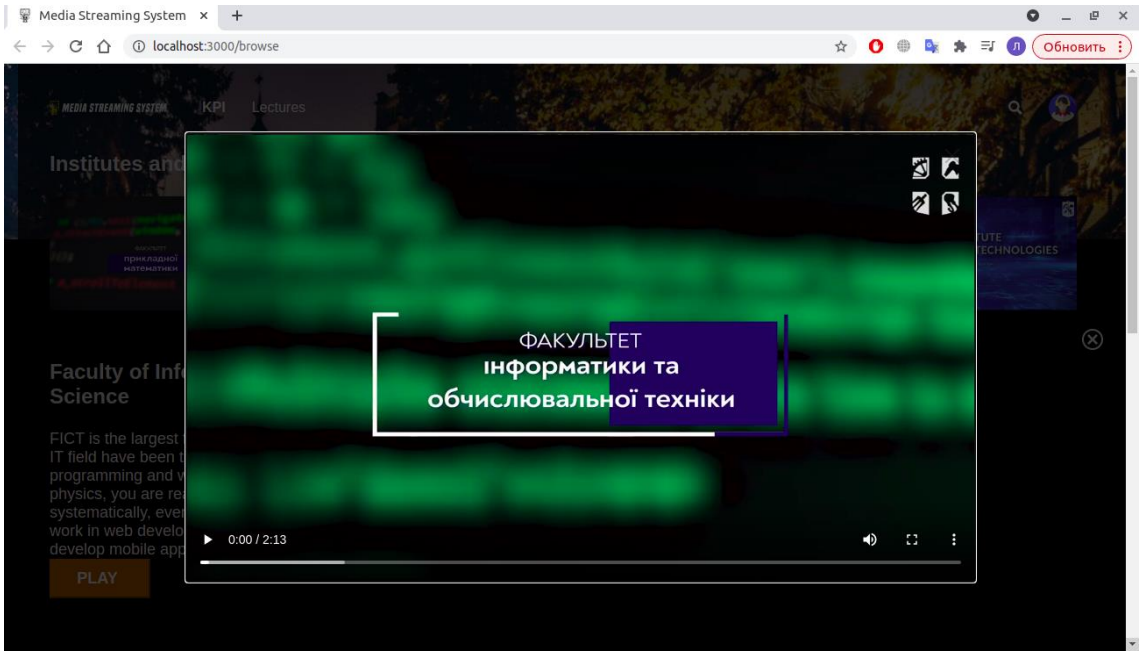


Рис. 4.7 – Плеєр

4.2. Тестування системи

Для системи написані юніт-тести за допомогою фреймворка Jest. Jest - це тестовий фреймворк з відкритим кодом, створений Facebook, який чудово інтегрується з React.js. Він має інструмент командного рядка для виконання тесту. Це також дозволяє нам створювати функції-заглушки з майже нульовою конфігурацією та забезпечує дійсно гарний набір збігів, що полегшує читання тверджень.

Крім того, він пропонує дійсно приємну функцію під назвою «тестування знімків», яка допомагає нам перевіряти та перевіряти результат візуалізації компонентів. Ми скористаємося тестуванням знімків, щоб захопити дерево компонента та зберегти його у файл, який ми можемо використати для порівняння з деревом візуалізації (або тим, що ми передаємо функції очікування як перший аргумент).

4.2.1. Написання тестів

Виконано тестування окремих модулів, таких як компоненти(components), сторінки(pages) та утиліти(utils) (рис. 4.8).

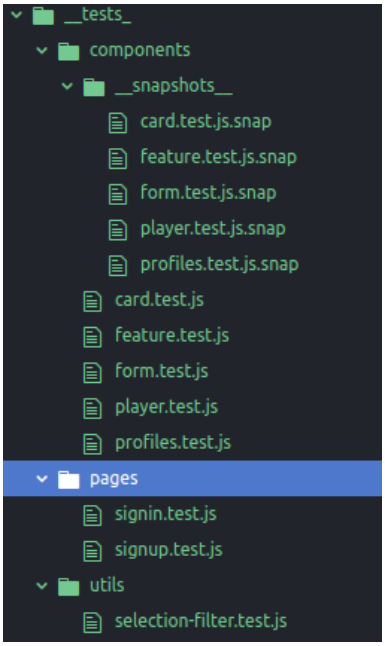


Рис. 4.8 – Структура тестових файлів

З утиліт був протестований фільтр вибору (selection-filter.test.js (рис. 4.9)). Для його тестування спочатку було створено два об’єкти (з категорій, що описані у базі даних). Вони передаються в якості аргументу в функцію selectionFilter(). Потім за допомогою функції expect() та його методу toBe() перевіряється коректне відображення створених об’єктів у слайдах (slides).

У цьому випадку toBe() є функцією, що перевіряє відповідність очікуваного результату з дійсним. Аргументом для функції expect() є значення, яке генерує код програми, а аргументом для функції toBe() є правильне значення.

Окремим тест кейсом перевіряється також передача порожніх аргументів у функцію selectionFilter().

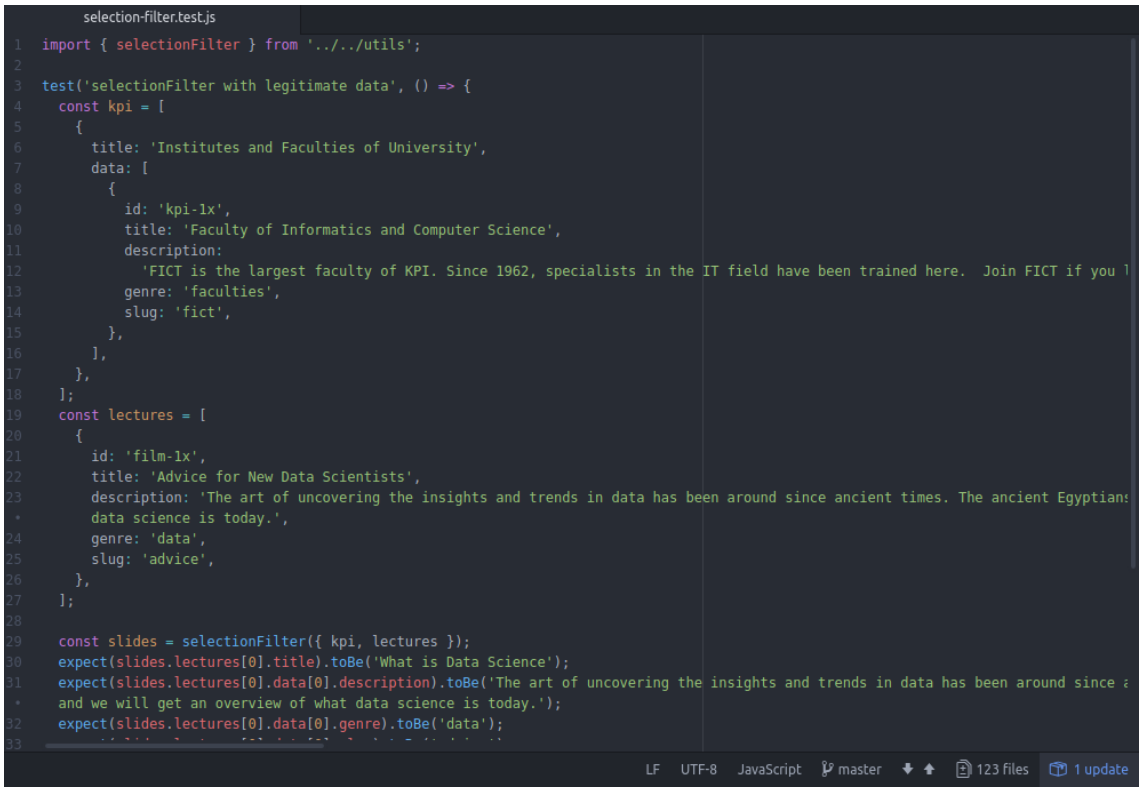


Рис. 4.9 – Тест selection-filter.test.js

Зі сторінок були протестовані сторінка аутентифікації (signin.test.js (рис. 4.10)) та сторінка реєстрації (signup.test.js). Їх тестування проводиться приблизно однаковим чином, тому розглянемо детальніше тестування сторінки аутентифікації.

Для його тестування спочатку використовується функція-заглушка (mock function). Функція-заглушки дозволяють протестувати зв’язки між кодом, стираючи фактичну реалізацію функції, фіксуючи виклики функції (і параметри, передані в цих викликах), фіксуючи екземпляри конструкторських функцій у випадку створення нових та дозволяючи конфігурацію часу перевірки повернутих значень.

У цьому випадку ми робимо заглушку на потрібну нам функцію за допомогою функції-заглушки Jest за замовчуванням jest.fn (), а потім прив'язуємо до неї реалізацію всередині кожного з наших тестових випадків. У нашому прикладі коду програми функція-заглушка використовується лише для хуку useHistory React Router. У нашому прикладі ми використовуємо Jest requireActual(), щоб переконатися, що ми робимо функцію-заглушку лише для функції useHistory і нічим іншим у модулі.

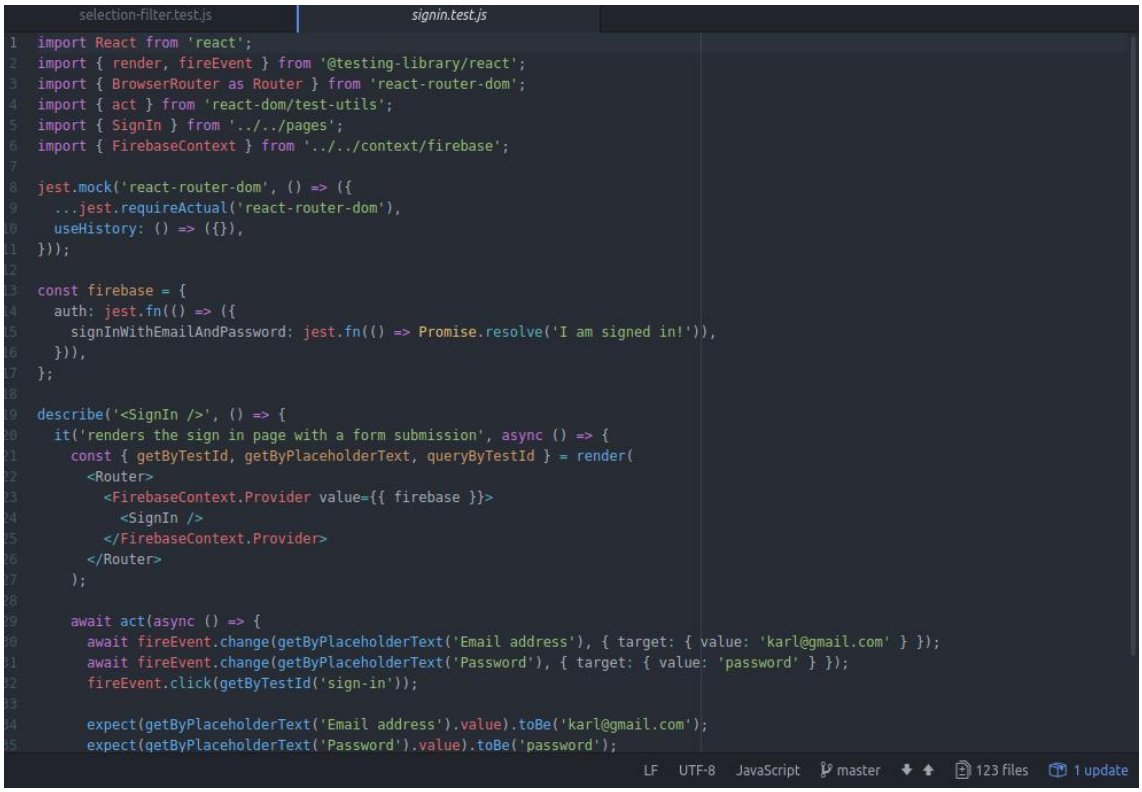


Рис. 4.10 – Тест signin.test.js

З компонентів були протестовані компонент картки (card.test.js), компонент опису (feature.test.js), компонент плеєра (player.test.js), компонент профілю (feature.test.js) та компонент форми (form.test.js).

Для тестування card.test.js, як і попередньо у випадку з selection-filter.test.js, спочатку було створено два об’єкти (з категорій, що описані у базі даних).

Для усіх компонентів також використовується тестування з використанням знімків. Це є дуже корисним інструментом, аби упевнитися, що UI не змінився несподівано.

Тест з використанням знімків робить компонент інтерфейсу користувача, робить знімок, а потім порівнює його з еталонним файлом знімка, що зберігається поряд із тестом. Тест не вдасться, якщо два знімки не збігаються: або зміна несподівана, або еталонний знімок потрібно оновити до нової версії компонента UI.

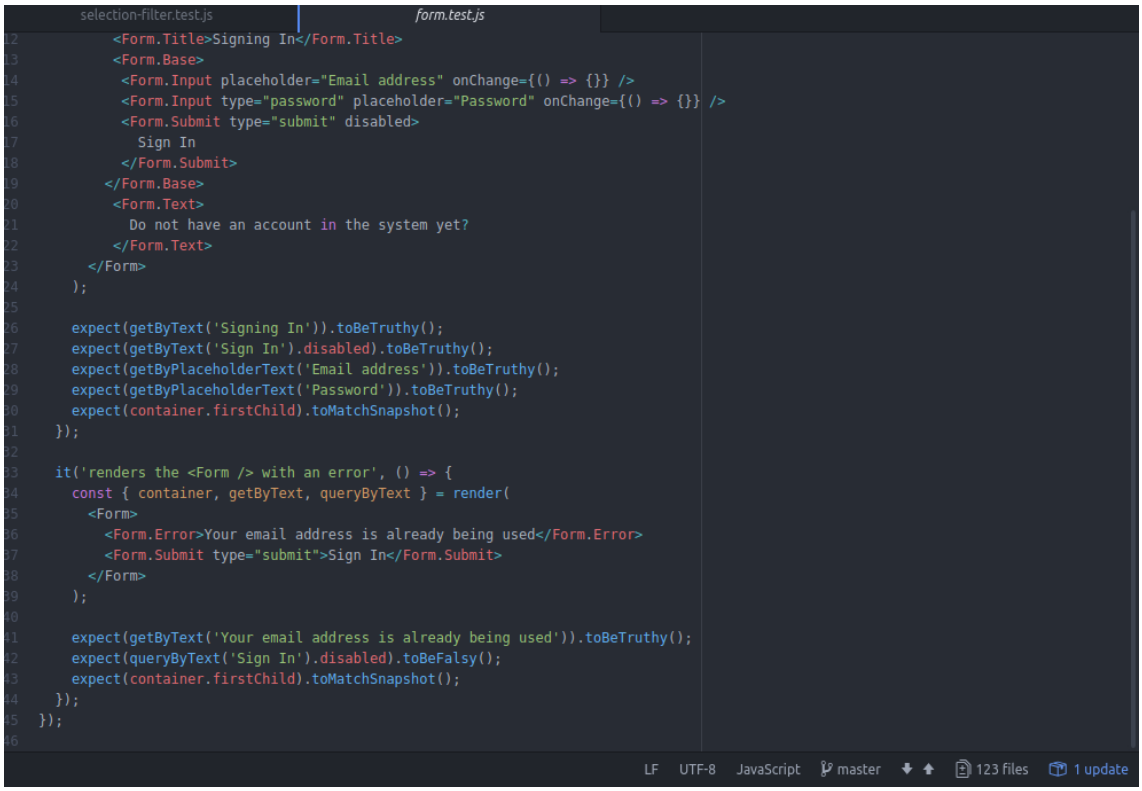


Рис. 4. - Тест form.test.js

4.2.2. Запуск та покриття

Запускаються наведені тести у терміналі за допомогою команди yarn test (рис. 4.).

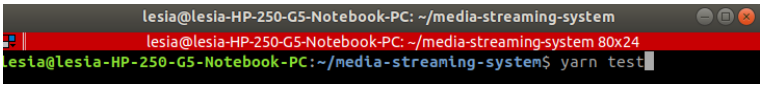


Рис. 4. – Команда для запуску тестів у терміналі

Гілки (branches) представляють, якщо твердження, умови яких були виконані принаймні один раз під час модульних тестів.

Функції (functions) представляють функції, які були викликані принаймні один раз під час модульних тестів.

Рядки (lines) представляють рядки коду, які виконувались принаймні один раз під час модульних тестів.

Твердження (statements) представляють інструкції, які виконувались принаймні один раз під час модульних тестів.

Всі написані тест-кейси виконуються успішно (рис. 4.).

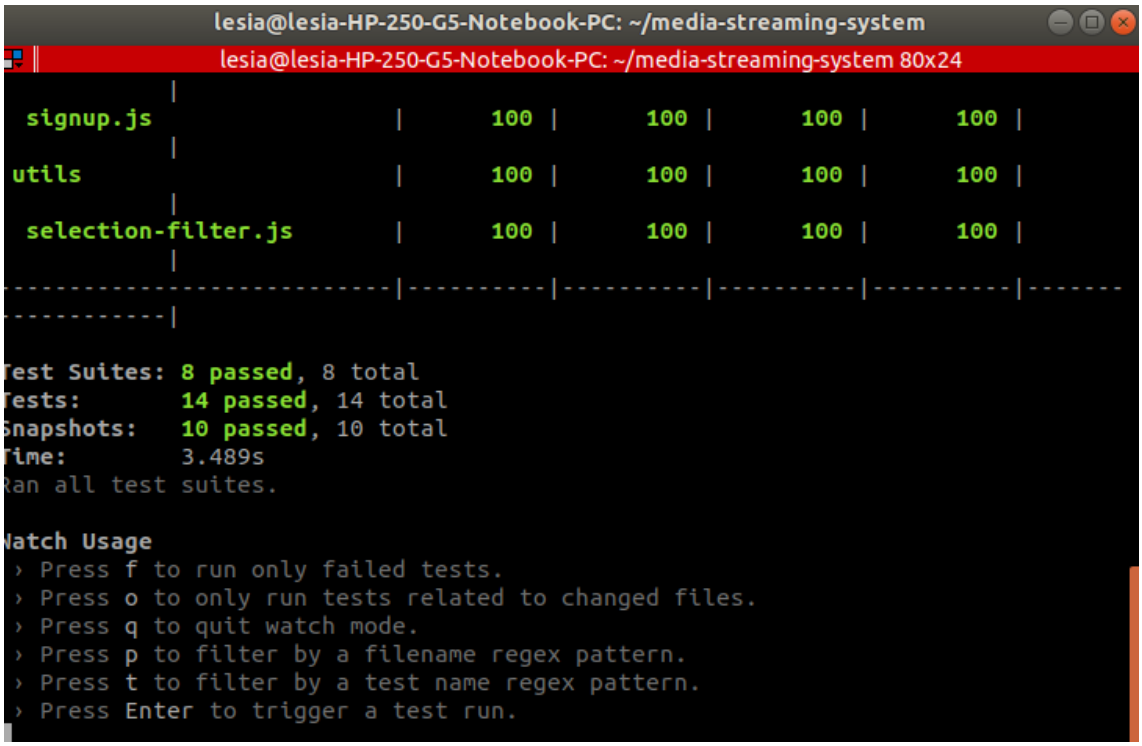


Рис. 4. – Успішне проходження тестів

Jest має функціонал формування звіту (рис. 4.), який допомагає нам зрозуміти покриття тестів. це охоплення включає твердження, функціональні, галузеві покриття. Покриття коду вашої програми - це відсоток коду, який наразі охоплюється модульними тестами.

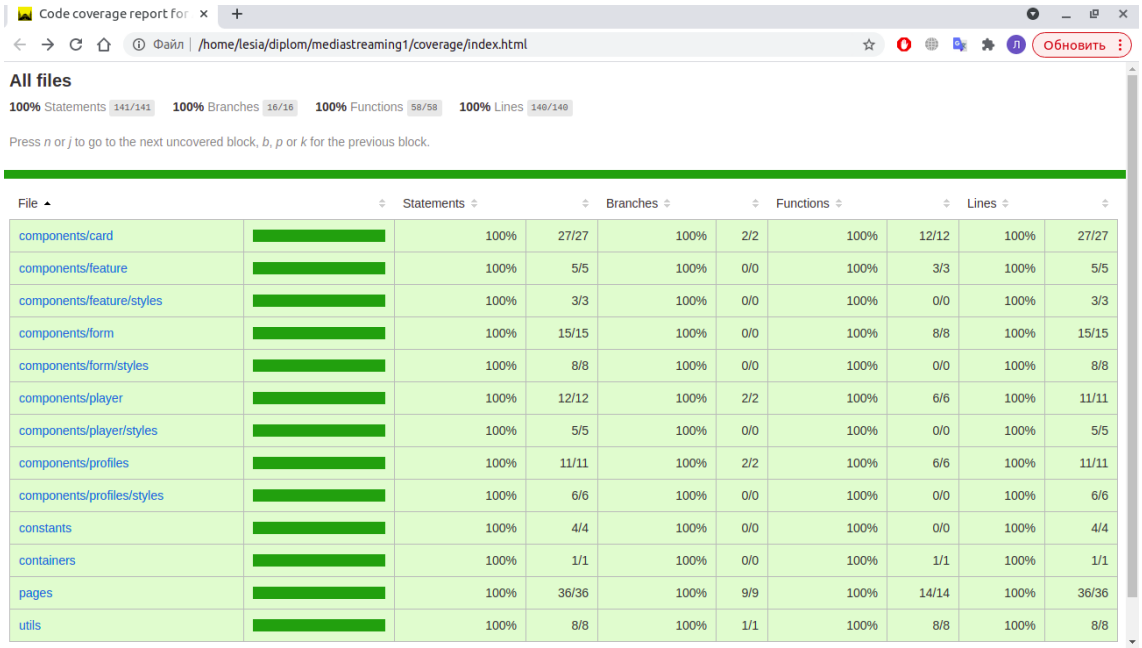


Рис. 4 – Звіт про тестове покриття

Висновок до розділу 4

У даному розділі було продемонстровано та протестовано роботу розробленої системи стрімінгу медіаконтенту. Можна зробити такі висновки:

- Розроблено зручний інтерфейс користувача, що відповідає створеним раніше вимогам.
- Написана достатня кількість тест-кейсів, які дозволяють ствердити, що система працює прогнозовано та передбачувано.
- Розроблені тести та відсоток покриття гарантують коректність роботи програми для користувача.
- Простором для покращення системи можна вважати розробку власного плеєра.

ВИСНОВОК

В ході виконання даного дипломного проєкту було розроблено систему стрімінгу медіаконтенту. Система надає можливість аутентифікації користувача, реєстрації користувача, а також пошуку та перегляду медіаконтенту.

У першому розділі були розглянуті теоретичні аспекти даного дипломного проєкту. Проведений докладний аналіз вже існуючих систем, які технології в них використовуються, їх переваги та недоліки. Визначені основні типи таких додатків, принципи їх роботи.

У другому розділі були досліджені основні програмні засоби організації стрімінгу. Визначені основні протоколи, які можуть бути при цьому задіяні. Обрані мова програмування та база даних.

У третьому розділі розглядалися ключові моменти розробки системи.

У четвертому розділі проводилися тестування та демонстрація додатку. Завдяки цьому можна сказати, що робота системи відбувається коректно та прогнозовано.

Отже, основні поставлені задачі були виконані. Розроблена система має потенціал для покращення. Наразі додаток працює стабільно та реалізує необхідний функціонал.

ЛІТЕРАТУРА

1. Cullen, C. (2018). Global internet phenomena report. Sandvine, Tech. Rep.
2. Krikke, J. (2004). Streaming video transforms the media industry. IEEE Computer Graphics and Applications, 24(4), 6-12.
3. Azhar, S., Morris, B., Casey, R., & Guillot, S. (2021). The Impact of the COVID-19 Pandemic on Subscription Video-On-Demand Services.
4. Netflix [Електронний ресурс] – Режим доступу: https://www.netflix.com/
5. Adhikari, V. K., Guo, Y., Hao, F., Varvello, M., Hilt, V., Steiner, M., & Zhang, Z. L. (2012, March). Unreeling netflix: Understanding and improving multi-cdn movie delivery. In 2012 Proceedings IEEE INFOCOM (pp. 1620-1628). IEEE.
6. Burroughs, B. (2019). House of Netflix: Streaming media and digital lore. Popular Communication, 17(1), 1-17.
7. Amazon Prime Video [Електронний ресурс] – Режим доступу: https://www.primevideo.com/
8. Cho, J. (2019). Amazon. com, Inc.
9. Hulu [Електронний ресурс] – Режим доступу: https://www.hulu.com/
10. YouTube [Електронний ресурс] – Режим доступу: https://www.youtube.com/
11. Cheng, X., Dale, C., & Liu, J. (2007). Understanding the characteristics of internet short video sharing: YouTube as a case study.
12. Gershon, R. A. (2020). Over-the-Top Video-streaming Services. In Media, Telecommunications and Business Strategy (pp. 99-111). Routledge.
13. Adhikari, V. K., Guo, Y., Hao, F., Hilt, V., Zhang, Z. L., Varvello, M., & Steiner, M. (2014). Measurement study of Netflix, Hulu, and a tale of three CDNs. IEEE/ACM Transactions on Networking, 23(6), 1984-1997.
14. MacDonald, K. M. (2020). Listening in: Investigating Social Media Activity in the Streaming Service Industry.

15. Firebase [Електронний ресурс] – Режим доступу: https://www.firebase.google.com/
16. Moroney, L. (2017). Using authentication in firebase. In The Definitive Guide to Firebase (pp. 25-50). Apress, Berkeley, CA.
17. Moroney, L. (2017). An Introduction to Firebase. In The Definitive Guide to Firebase (pp. 1-24). Apress, Berkeley, CA.
18. Ohyver, M., Moniaga, J. V., Sungkawa, I., Subagyo, B. E., & Chandra, I. A. (2019). The comparison firebase realtime database and MySQL database performance using Wilcoxon signed-rank test. Procedia Computer Science, 157, 396-405.
19. Tanna, M., & Singh, H. (2018). Serverless Web Applications with React and Firebase: Develop real-time applications for web and mobile platforms. Packt Publishing Ltd.

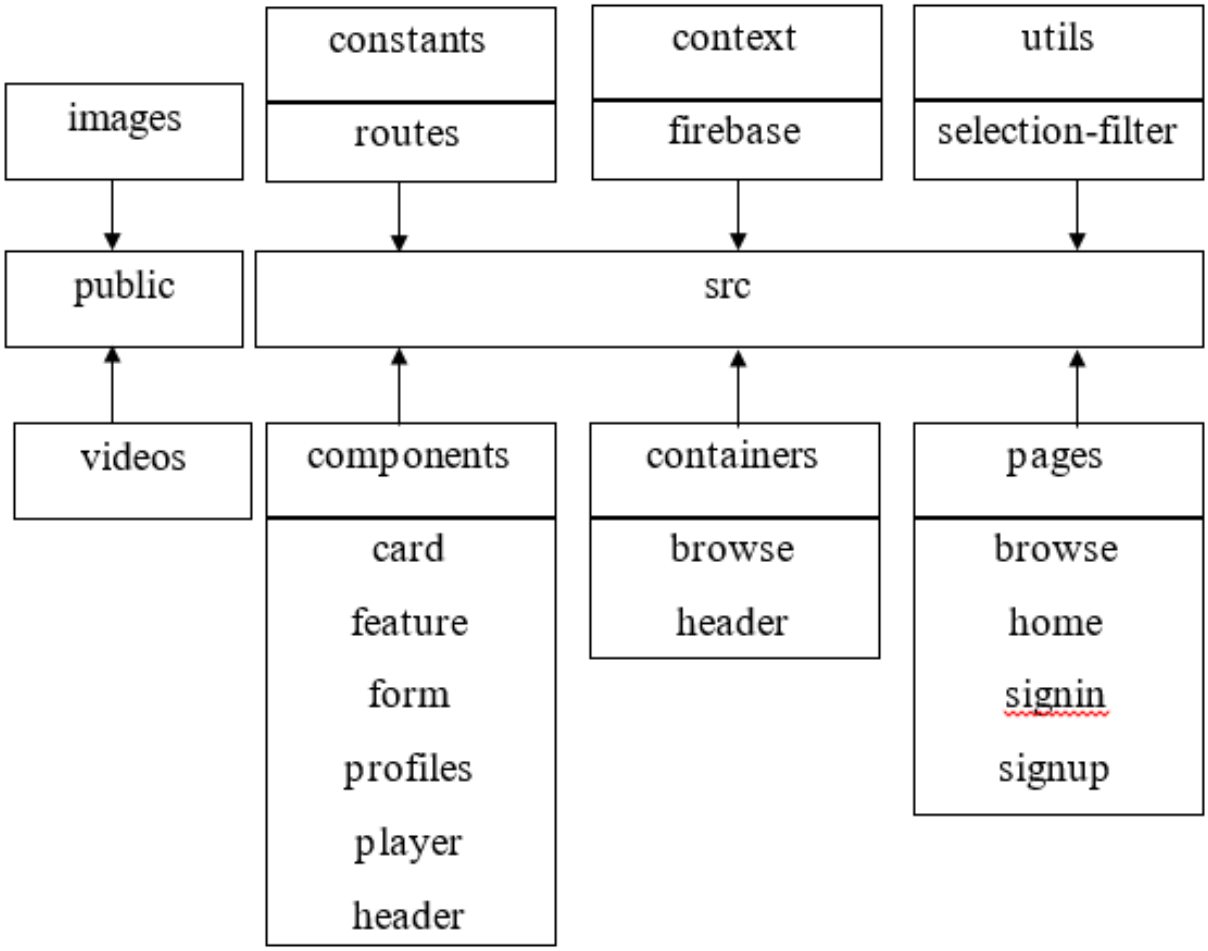
Додаток А

СТРУКТУРНА СХЕМА ПРОГРАМИ

до дипломного проекту

на тему: «Система стрімінгу медіаконтенту»

Київ – 2021



					ІАЛЦ.467800.004 Д1		
					Система стрімінгу медіаконтенту Структурна схема програми	***Літ.***	***Маса*** / ***Масштаб***
Розроб.		***Сергієнко О.О.***					
Перев.		***Волокита І.М.***				***Аркуш 1***	***Аркушів 1***
Реценз.							
Н. контр.		***Сімоненко В.П.***			*Дипломний проект*	***НТУУ "КПІ" ФІОТ ІП-73***	
Затв.							

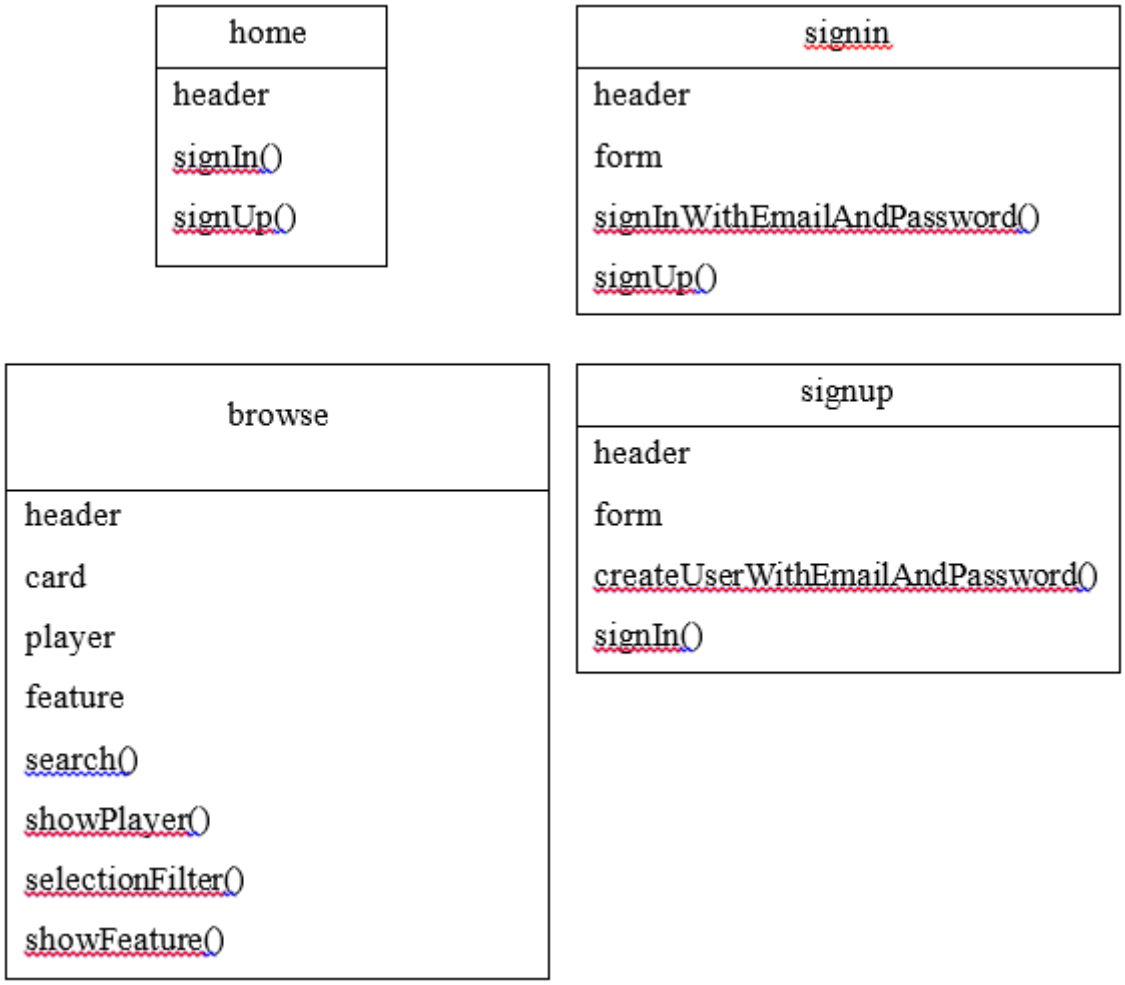
Додаток Б

ФУНКЦІОНАЛЬНА СХЕМА ПРОГРАМИ

до дипломного проекту

на тему: «Виявлення DDOS атак за
допомогою машинного навчання»

Київ – 2021



					ІАЛЦ.467800.005 Д2		
					Система стрімінгу медіаконтенту Функціональна схема програми	***Літ.***	***Маса*** / ***Масштаб***
Розроб.		***Сергієнко О.О..***					
Перев.		***Волокита І.М.***				***Аркуш 1***	***Аркушів 1***
Реценз.							
Н. контр.		***Сімоненко В.П.***			*Дипломний проект*	***НТУУ "КПІ" ФІОТ ІП-73***	
Затв.							

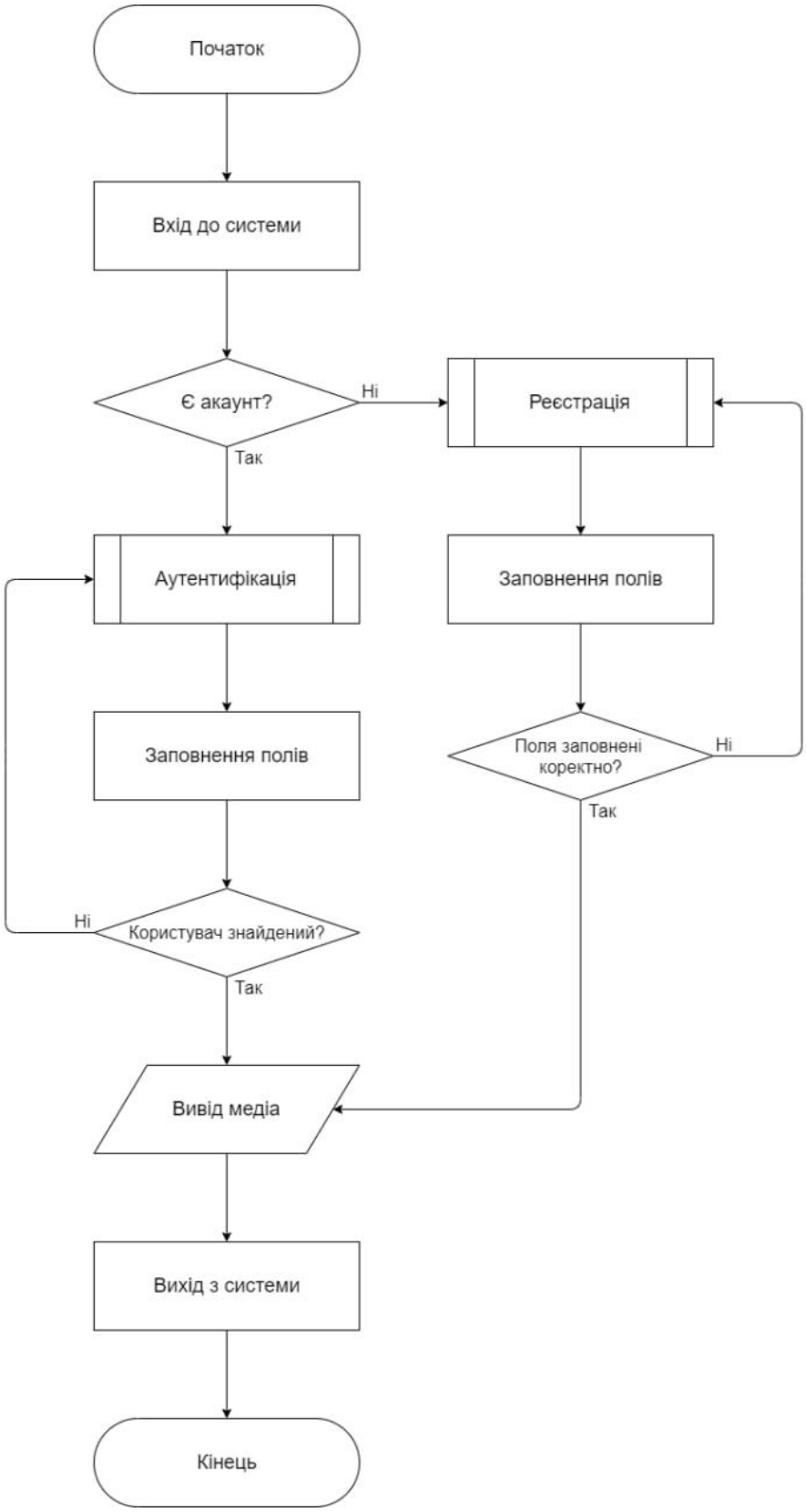
Додаток В

ПРИНЦИПОВА СХЕМА ПРОГРАМИ

до дипломного проекту

на тему: «Система стрімінгу медіаконтенту»

Київ – 2021



					ІАЛЦ.467800.006 Д3		
					Система стрімінгу медіаконтенту Принципова схема програми	***Літ.***	***Маса*** / ***Масштаб***
Розроб.		***Сергієнко О.О.***					
Перев.		***Волокита І.М.***				***Аркуш 1***	***Аркушів 1***
Реценз.							
Н. контр.		***Сімоненко В.П.***			*Дипломний проект*	***НТУУ "КПІ" ФІОТ ІП-73***	
Затв.							

Додаток Г

ЛІСТИНГ КОДУ ПРОГРАМИ

до дипломного проекту

на тему: «Система стрімінгу медіаконтенту»

Київ – 2021

ЛІСТИНГ ПРОГРАМИ

«pages/signup.js»

```
import React, {useState, useContext} from 'react';
import {useHistory} from 'react-router-dom';
import {FirebaseContext} from '../context/firebase';
import { HeaderContainer } from '../containers/header';
import {Form} from '../components';
import * as ROUTES from '../constants/routes';

export default function SignUp() {
  const history = useHistory();
  const {firebase} = useContext(FirebaseContext);

  const [firstName, setFirstName] = useState('');
  const [emailAddress, setEmailAddress] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const isInvalid = firstName === '' || password === '' || emailAddress === '';

  const handleSignup = (event) => {
    event.preventDefault();

    return firebase
      .auth()
      .createUserWithEmailAndPassword(emailAddress, password)
      .then((result) =>
        result.user
          .updateProfile({
            displayName: firstName,
            photoURL: Math.floor(Math.random() * 5) + 1,
          })
          .then(() => {
            history.push(ROUTES.BROWSE);
          })
      )
      .catch((error) => {
        setFirstName('');
        setEmailAddress('');
        setPassword('');
        setError(error.message);
      });
  };
```

```
    return (
      <>
        <HeaderContainer>
          <Form>
            <Form.Title>Sign Up</Form.Title>
            {error && <Form.Error>{error}</Form.Error>}

            <Form.Base onSubmit={handleSignup} method="POST">
              <Form.Input
                placeholder="First name"
                value={firstName}
                onChange={({ target }) => setFirstName(target.value)}
              />
              <Form.Input
                placeholder="Email address"
                value={emailAddress}
                onChange={({ target }) => setEmailAddress(target.value)}
              />
              <Form.Input
                type="password"
                value={password}
                autoComplete="off"
                placeholder="Password"
                onChange={({ target }) => setPassword(target.value)}
              />
              <Form.Submit disabled={isInvalid} type="submit" data-testid="sign-up">
                Sign Up
              </Form.Submit>
            </Form.Base>

            <Form.Text>
              Do you already have an account in the system? <Form.Link to="/signin">Sign in
now.</Form.Link>
            </Form.Text>
          </Form>
        </HeaderContainer>
      </>
    );
  }
```

«pages/signin.js»

```
import React, {useState, useContext} from 'react';
import {useHistory} from 'react-router-dom';
import {FirebaseContext} from '../context/firebase';
import { HeaderContainer } from '../containers/header';
import {Form} from '../components';
import * as ROUTES from '../constants/routes';
export default function SignIn() {
  const history = useHistory();
  const {firebase} = useContext(FirebaseContext);
  const [emailAdress, setEmailAddress] = useState('');
```

```
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const isInvalid = password === '' || emailAdress === '';
  const handleSignIn = (event) => {
    event.preventDefault();
    firebase
      .auth()
      .signInWithEmailAndPassword(emailAdress, password)
      .then(() => {
        history.push(ROUTES.BROWSE);
      })
      .catch((error) => {
        setEmailAddress('');
        setPassword('');
        setError(error.message);
      });
  };

  return (
    <>
      <HeaderContainer>
        <Form>
          <Form.Title>Sign In</Form.Title>
          {error && <Form.Error data-testid="error">{error}</Form.Error>}
          <Form.Base onSubmit={handleSignIn} method="POST">
            <Form.Input
              placeholder="Email address"
              value={emailAdress}
              onChange={({target}) => setEmailAddress(target.value)}
            />
            <Form.Input
              type = "password"
              autoComplete="off"
              placeholder="Password"
              value={password}
              onChange={({target}) => setPassword(target.value)}
            />
            <Form.Submit disabled={isInvalid} type="submit" data-testid="sign-in">
              Sign In
            </Form.Submit>
          </Form.Base>
          <Form.Text>
            Do not have an account in the system yet? <Form.Link to="/signup">Create an
account.</Form.Link>
          </Form.Text>
        </Form>
      </HeaderContainer>
    </>
  );
}
```

«pages/home.js»

```
import React from 'react';
import { Feature } from '../components';
import { HeaderContainer } from '../containers/header';

export default function Home() {
  return (
    <>
      <HeaderContainer>
        <Feature>
          <Feature.Title>MEDIA STREAMING SYSTEM</Feature.Title>
          <Feature.SubTitle>On this platform, you can browse media content. The content is
broadcast in real time, the download speed directly depends on the speed of the user&rsquo;s
Internet. To do this, from the beginning register.</Feature.SubTitle>
        </Feature>
      </HeaderContainer>
    </>
  );
}
```

«pages/browse.js»

```
import React from 'react';
import { BrowseContainer } from '../containers/browse';
import { useContent } from '../hooks';
import { selectionFilter } from '../utils';

export default function Browse() {
  const { kpi } = useContent('kpi');
  const { lectures } = useContent('lectures');
  const slides = selectionFilter({ kpi, lectures });
  return <BrowseContainer slides={slides} />;
```

index.js

```
import React from 'react';
import {render} from 'react-dom';
import 'normalize.css';
import App from './app';
import {GlobalStyles} from './global-styles';
import {firebase} from './lib/firebase.prod';
import {FirebaseContext} from './context/firebase';

render(
  <>
    <FirebaseContext.Provider value = {{firebase}}>
      <GlobalStyles />
      <App />
    </FirebaseContext.Provider>
  </>,
  document.getElementById('root')
);
```

«app.js»

```
import React from 'react';
import { BrowserRouter as Router, Switch } from 'react-router-dom';
import { Home, Browse, SignIn, SignUp } from './pages';
import * as ROUTES from './constants/routes';
import { IsUserRedirect, ProtectedRoute } from './helpers/routes';
import { useAuthListener } from './hooks';

export default function App() {
  const { user } = useAuthListener();

  return (
    <Router>
      <Switch>
        <IsUserRedirect user={user} loggedInPath={ROUTES.BROWSE}
path={ROUTES.SIGN_IN}>
          <SignIn />
        </IsUserRedirect>
        <IsUserRedirect user={user} loggedInPath={ROUTES.BROWSE}
path={ROUTES.SIGN_UP}>
          <SignUp />
        </IsUserRedirect>
        <ProtectedRoute user={user} path={ROUTES.BROWSE}>
          <Browse />
        </ProtectedRoute>
        <IsUserRedirect user={user} loggedInPath={ROUTES.BROWSE}
path={ROUTES.HOME}>
          <Home />
        </IsUserRedirect>
      </Switch>
    </Router>
  );
}
```

«utils/selection-filter.js»

```
export default function selectionFilter({ kpi, lectures } = []) {
  return {
    kpi: [
      { title: 'Institutes and Faculties of University', data: kpi?.filter((item) => item.genre
=== 'faculties') },
      { title: 'EMI/EFVV', data: kpi?.filter((item) => item.genre === 'evi') },
      { title: 'Students Organizations', data: kpi?.filter((item) => item.genre === 'circle') },
      { title: 'Entertaining', data: kpi?.filter((item) => item.genre === 'entertaining') },
    ],
    lectures: [
      { title: 'What is Data Science', data: lectures?.filter((item) => item.genre === 'data') },
      { title: 'History', data: lectures?.filter((item) => item.genre === 'history') },
      { title: 'Learn SQL and Databases for Data Science with IBM', data:
lectures?.filter((item) => item.genre === 'sql') },
    ],
  };
}
```

«lib/firebase.prod.js»

```
import Firebase from 'firebase/app';
import 'firebase/firestore';
import 'firebase/auth';

const config = {
  apiKey: "AIzaSyAi2PWLpg8ATeipz_Cz1y2kx8vpfFK5wSc",
  authDomain: "media-streaming-6c4e2.firebaseapp.com",
  projectId: "media-streaming-6c4e2",
  storageBucket: "media-streaming-6c4e2.appspot.com",
  messagingSenderId: "684283765266",
  appId: "1:684283765266:web:d3eab7c17e91339f152ecb"
};

const firebase = Firebase.initializeApp(config);

export { firebase };
```

«context/firebase.js»

```
import { createContext } from 'react';

export const FirebaseContext = createContext(null);
```

«containers/header.js»

```
import React from 'react';
import { Header } from '../components';
import * as ROUTES from '../constants/routes';
import logo from '../logo.svg';

export function HeaderContainer({ children }) {
  return (
    <Header>
      <Header.Frame>
        <Header.Logo to={ROUTES.HOME} src={logo} alt="home-bg" />
        <Header.ButtonLink to={ROUTES.SIGN_IN}>Sign In</Header.ButtonLink>
      </Header.Frame>
      {children}
    </Header>
  );
}
```

«constants/routes.js»

```
export const HOME = '/';
export const BROWSE = '/browse';
export const SIGN_UP = '/signup';
export const SIGN_IN = '/signin';
```

«containers/header.js»

```
import React from 'react';
import { Header } from '../components';
import * as ROUTES from '../constants/routes';
import logo from '../logo.svg';

export function HeaderContainer({ children }) {
  return (
    <Header>
      <Header.Frame>
        <Header.Logo to={ROUTES.HOME} src={logo} alt="home-bg" />
        <Header.ButtonLink to={ROUTES.SIGN_IN}>Sign In</Header.ButtonLink>
      </Header.Frame>
      {children}
    </Header>
  );
}
```

«components/form/header.js»

```
import React, { useState } from 'react';
import { Link as ReactRouterLink } from 'react-router-dom';
import {
  Container,
  Group,
  Background,
  Dropdown,
  Picture,
  Link,
  Search,
  Profile,
  FeatureCallOut,
  SearchIcon,
  SearchInput,
  ButtonLink,
  PlayButton,
  Text,
  Feature,
  Logo,
} from './styles/header';

export default function Header({ bg = true, children, ...restProps }) {
  return bg ? (
    <Background data-testid="header-bg" {...restProps}>
      {children}
    </Background>
  ) : (
    children
  );
}
```

```
Header.Frame = function HeaderFrame({ children, ...restProps }) {
  return <Container {...restProps}>{children}</Container>;
};

Header.Group = function HeaderGroup({ children, ...restProps }) {
  return <Group {...restProps}>{children}</Group>;
};

Header.Logo = function HeaderLogo({ to, ...restProps }) {
  return (
    <ReactRouterLink to={to}>
      <Logo {...restProps} />
    </ReactRouterLink>
  );
};

Header.Search = function HeaderSearch({ searchTerm, setSearchTerm, ...restProps }) {
  const [searchActive, setSearchActive] = useState(false);

  return (
    <Search {...restProps}>
      <SearchIcon onClick={() => setSearchActive((searchActive) => !searchActive)}
data-testid="search-click">
        <img src="/images/icons/search.png" alt="Search" />
      </SearchIcon>
      <SearchInput
        value={searchTerm}
        onChange={({ target }) => setSearchTerm(target.value)}
        placeholder="Search videos"
        active={searchActive}
        data-testid="search-input"
      />
    </Search>
  );
};

Header.Profile = function HeaderProfile({ children, ...restProps }) {
  return <Profile {...restProps}>{children}</Profile>;
};

Header.Feature = function HeaderFeature({ children, ...restProps }) {
  return <Feature>{children}</Feature>;
};

Header.Picture = function HeaderPicture({ src, ...restProps }) {
  return <Picture {...restProps} src={`/images/users/${src}.png`} />;
};

Header.Dropdown = function HeaderDropdown({ children, ...restProps }) {
  return <Dropdown {...restProps}>{children}</Dropdown>;
};
```

```
Header.TextLink = function HeaderTextLink({ children, ...restProps }) {
  return <Link {...restProps}>{children}</Link>;
};

Header.PlayButton = function HeaderPlayButton({ children, ...restProps }) {
  return <PlayButton {...restProps}>{children}</PlayButton>;
};

Header.FeatureCallOut = function HeaderFeatureCallOut({ children, ...restProps }) {
  return <FeatureCallOut {...restProps}>{children}</FeatureCallOut>;
};

Header.Text = function HeaderText({ children, ...restProps }) {
  return <Text {...restProps}>{children}</Text>;
};

Header.ButtonLink = function HeaderButtonLink({ children, ...restProps }) {
  return <ButtonLink {...restProps}>{children}</ButtonLink>;
};
```

«components/form/form.js»

```
import React from 'react';
import { Container, Error, Base, Title, Text, Link, Input, Submit } from './styles/form';

export default function Form({ children, ...restProps }) {
  return <Container {...restProps}>{children}</Container>;
}

Form.Error = function FormError({ children, ...restProps }) {
  return <Error {...restProps}>{children}</Error>;
};

Form.Base = function FormBase({ children, ...restProps }) {
  return <Base {...restProps}>{children}</Base>;
};

Form.Title = function FormTitle({ children, ...restProps }) {
  return <Title {...restProps}>{children}</Title>;
};

Form.Text = function FormText({ children, ...restProps }) {
  return <Text {...restProps}>{children}</Text>;
};

Form.Link = function FormLink({ children, ...restProps }) {
  return <Link {...restProps}>{children}</Link>;
};

Form.Input = function FormInput({ children, ...restProps }) {
  return <Input {...restProps}>{children}</Input>;
};
```

«components/card/index.js»

```
import React, { useState, useContext, createContext } from 'react';

import {
  Container,
  Group,
  Title,
  SubTitle,
  Text,
  Feature,
  FeatureTitle,
  FeatureText,
  FeatureClose,
  Content,
  Meta,
  Entities,
  Item,
  Image,
} from './styles/card';

export const FeatureContext = createContext();

export default function Card({ children, ...restProps }) {
  const [showFeature, setShowFeature] = useState(false);
  const [itemFeature, setItemFeature] = useState({});

  return (
    <FeatureContext.Provider value={{ showFeature, setShowFeature, itemFeature,
setItemFeature }}>
      <Container {...restProps}>{children}</Container>
    </FeatureContext.Provider>
  );
}

Card.Group = function CardGroup({ children, ...restProps }) {
  return <Group {...restProps}>{children}</Group>;
};

Card.Title = function CardTitle({ children, ...restProps }) {
  return <Title {...restProps}>{children}</Title>;
};

Card.SubTitle = function CardSubTitle({ children, ...restProps }) {
  return <SubTitle {...restProps}>{children}</SubTitle>;
};

Card.Text = function CardText({ children, ...restProps }) {
  return <Text {...restProps}>{children}</Text>;
};
```

```
Card.Entities = function CardEntities({ children, ...restProps }) {
  return <Entities {...restProps}>{children}</Entities>;
};

Card.Meta = function CardMeta({ children, ...restProps }) {
  return <Meta {...restProps}>{children}</Meta>;
};

Card.Item = function CardItem({ item, children, ...restProps }) {
  const { setShowFeature, setItemFeature } = useContext(FeatureContext);

  return (
    <Item
      onClick={() => {
        setItemFeature(item);
        setShowFeature(true);
      }}
      {...restProps}
    >
      {children}
    </Item>
  );
};

Card.Image = function CardImage({ ...restProps }) {
  return <Image {...restProps} />;
};
Card.Feature = function CardFeature({ children, category, ...restProps }) {
  const { showFeature, itemFeature, setShowFeature } = useContext(FeatureContext);

  return showFeature ? (
    <Feature                                                                {...restProps}
src={`/images/${category}/${itemFeature.genre}/${itemFeature.slug}/large.jpg`}>
      <Content>
        <FeatureTitle>{itemFeature.title}</FeatureTitle>
        <FeatureText>{itemFeature.description}</FeatureText>
        <FeatureClose onClick={() => setShowFeature(false)}>
          <img src="/images/icons/close.png" alt="Close" />
        </FeatureClose>
        {children}
      </Content>
    </Feature>
  ) : null;
};
```

«components/feature/index.js»

```
import React, { useState, useContext, createContext } from 'react';

import {
  Container,
  Group,
  Title,
```

```
  SubTitle,
  Text,
  Feature,
  FeatureTitle,
  FeatureText,
  FeatureClose,
  Content,
  Meta,
  Entities,
  Item,
  Image,
} from './styles/card';

export const FeatureContext = createContext();

export default function Card({ children, ...restProps }) {
  const [showFeature, setShowFeature] = useState(false);
  const [itemFeature, setItemFeature] = useState({});

  return (
    <FeatureContext.Provider value={{ showFeature, setShowFeature, itemFeature,
setItemFeature }}>
      <Container {...restProps}>{children}</Container>
    </FeatureContext.Provider>
  );
}

Card.Group = function CardGroup({ children, ...restProps }) {
  return <Group {...restProps}>{children}</Group>;
};

Card.Title = function CardTitle({ children, ...restProps }) {
  return <Title {...restProps}>{children}</Title>;
};

Card.SubTitle = function CardSubTitle({ children, ...restProps }) {
  return <SubTitle {...restProps}>{children}</SubTitle>;
};

Card.Text = function CardText({ children, ...restProps }) {
  return <Text {...restProps}>{children}</Text>;
};

Card.Entities = function CardEntities({ children, ...restProps }) {
  return <Entities {...restProps}>{children}</Entities>;
};

Card.Meta = function CardMeta({ children, ...restProps }) {
  return <Meta {...restProps}>{children}</Meta>;
};
```

```
Card.Item = function CardItem({ item, children, ...restProps }) {
  const { setShowFeature, setItemFeature } = useContext(FeatureContext);

  return (
    <Item
      onClick={() => {
        setItemFeature(item);
        setShowFeature(true);
      }}
      {...restProps}
    >
      {children}
    </Item>
  );
};

Card.Image = function CardImage({ ...restProps }) {
  return <Image {...restProps} />;
};

Card.Feature = function CardFeature({ children, category, ...restProps }) {
  const { showFeature, itemFeature, setShowFeature } = useContext(FeatureContext);

  return showFeature ? (
    <Feature {...restProps}
src={`/images/${category}/${itemFeature.genre}/${itemFeature.slug}/large.jpg`}>
      <Content>
        <FeatureTitle>{itemFeature.title}</FeatureTitle>
        <FeatureText>{itemFeature.description}</FeatureText>
        <FeatureClose onClick={() => setShowFeature(false)}>
          <img src="/images/icons/close.png" alt="Close" />
        </FeatureClose>
        {children}
      </Content>
    </Feature>
  ) : null;
};
```