**Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра математической кибернетики и информационных технологий

**Отчёт по практической работе**

на тему «Разработка Web-сайта по API фильмов»

по дисциплине «Основы UI и UX дизайна»

Выполнил студент группы БВТ2001

Казыханов Рамиль Ильясович.

Содержание

[1. Введение 3](#_Toc152855153)

[1.1. Цель работы 3](#_Toc152855154)

[1.2. Задания 3](#_Toc152855155)

[2. Теоретическая часть 3](#_Toc152855156)

[3. Выполнение 4](#_Toc152855157)

[3.1. Структура проекта 4](#_Toc152855158)

[3.2. Архитектура модулей 5](#_Toc152855159)

[3.3. Внедрение зависимостей 7](#_Toc152855160)

[4. Скриншоты приложения 9](#_Toc152855161)

[5. Вывод 13](#_Toc152855162)

1. Введение
   1. Цель работы

Необходимо разработать SPA исходя из описания задачи.

* 1. Задания
* Имеется API со списком фильмов, с которым нужно наладить взаимодействие;
* Необходимо визуализировать данные полученные от API в виде таблицы, содержащей только важную информацию о фильме;
* К каждому фильму из таблицы можно оставить комментарий (неограниченное количество), а также удалить его в любой момент времени, локальное состояние;
* Для написания сайта необходимо использовать фреймворки React и NextJS;
* По возможности избегать перезагрузки страницы;
* В приложении должна присутствовать пагинация.

1. Теоретическая часть

Next.js — открытый JavaScript фреймворк, созданный поверх React.js для создания веб-приложений, созданный компанией Vercel (ранее ZEIT). Фреймворк был предназначен для решения проблемы React.js, связанной с отрисовкой приложения на стороне сервера - SSR. Работает на сервере и в браузере.

React Query — отличная библиотека хуков для управления запросами данных, которая полностью избавляет от необходимости помещать удаленные данные в глобальное состояние. Вам просто нужно указать библиотеке, где вы хотите получить данные, и она будет обрабатывать кеширование, фоновые обновления и аннулирование данных без какого-либо дополнительного кода или конфигурации.

React Query также устраняет необходимость использовать хуки useState и useEffect и заменяет их на несколько строк логики React Query. В конечном итоге это определенно поможет вам поддерживать приложение.

1. Выполнение
   1. Структура проекта

Структура проекта представлена на следующем рисунке:

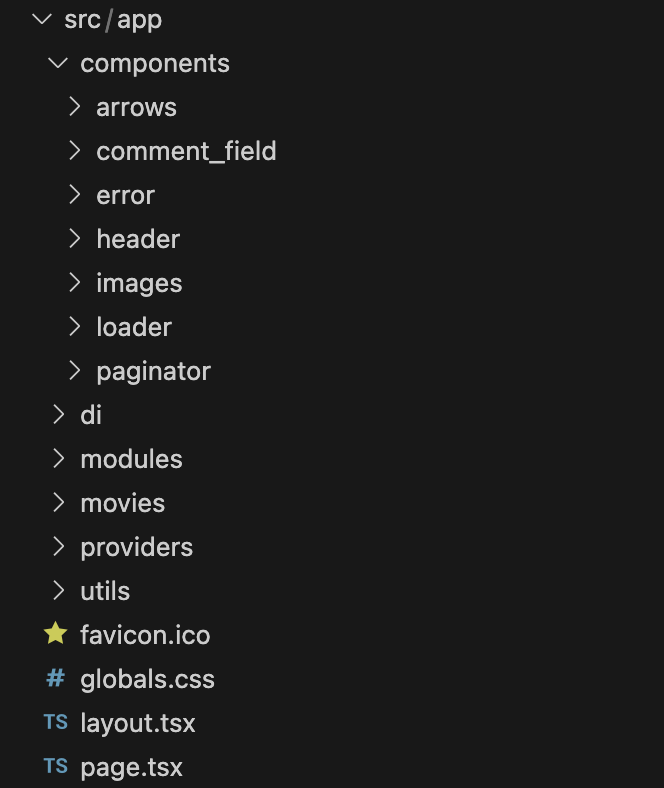


Рисунок 1 – Структура проекта

Структура содержит:

1. Компоненты (components) – атомарные единицы UI, которые могут быть переиспользованы в разных частях приложения;
2. Внедрение зависимостей (di) – конфигурация контейнера с зависимостями приложения;
3. Модули (modules) – экраны или какие-то флоу экранов;
4. Фильмы (movies) – необходимая для корректного роутинга в NextJS папка;
5. Провайдеры (providers) – различные функции-обертки страниц, позволяющие использовать, например, React Query;
6. Вспомогательные инструменты (utils) – вспомогательный код, вынесенный в отдельные файлы.
   1. Архитектура модулей

Отдельный модуль – отдельная функциональная часть приложения. Каждый модуль (в нашем приложении он только один) следует принципам чистой архитектуры:

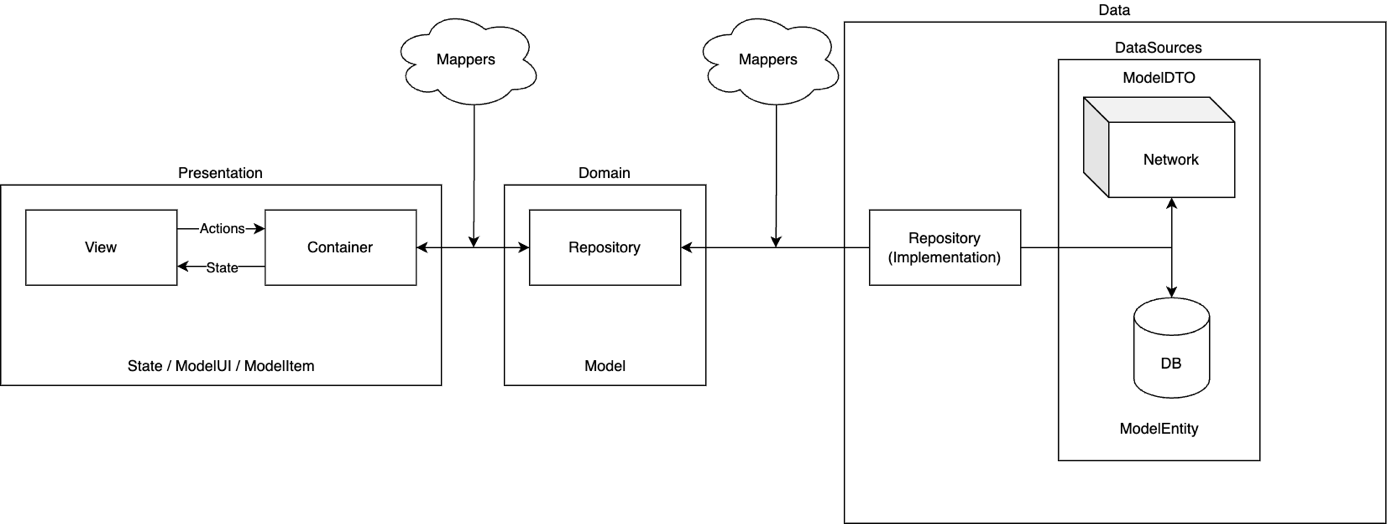


Рисунок 2 – Схема чистой архитектуры

Эта схема придерживается следующих принципов:

* Разделение модуля на три слоя: слой представления, слой бизнес-логики и слой данных;
* Слой бизнес-логики не должен ни от чего зависеть, поэтому реализация интерфейса репозитория лежит в слое данных;
* Репозиторий – это абстракция над некоторой коллекцией данных (в нашем случае коллекция фильмов). Позволяет пользователям репозитория не задумываться о том, откуда берутся данные – из базы данных, из сети или откуда-либо еще;
* Каждому слою соответствуют собственные модели данных для обеспечения независимости работы слоёв;
* Для моделей из разных слоёв существуют мапперы – сущности, превращающие модель из одного слоя в модель из другого слоя;
* Слой представления делится на два основных компонента – View и Container. View отвечает за отрисовку элементов страницы и отлавливание действий пользователя, Container отвечает за обработку действий пользователя и отправку нового состояния во View для отрисовки.

Следующим образом такая схема выглядит в файловом дереве:

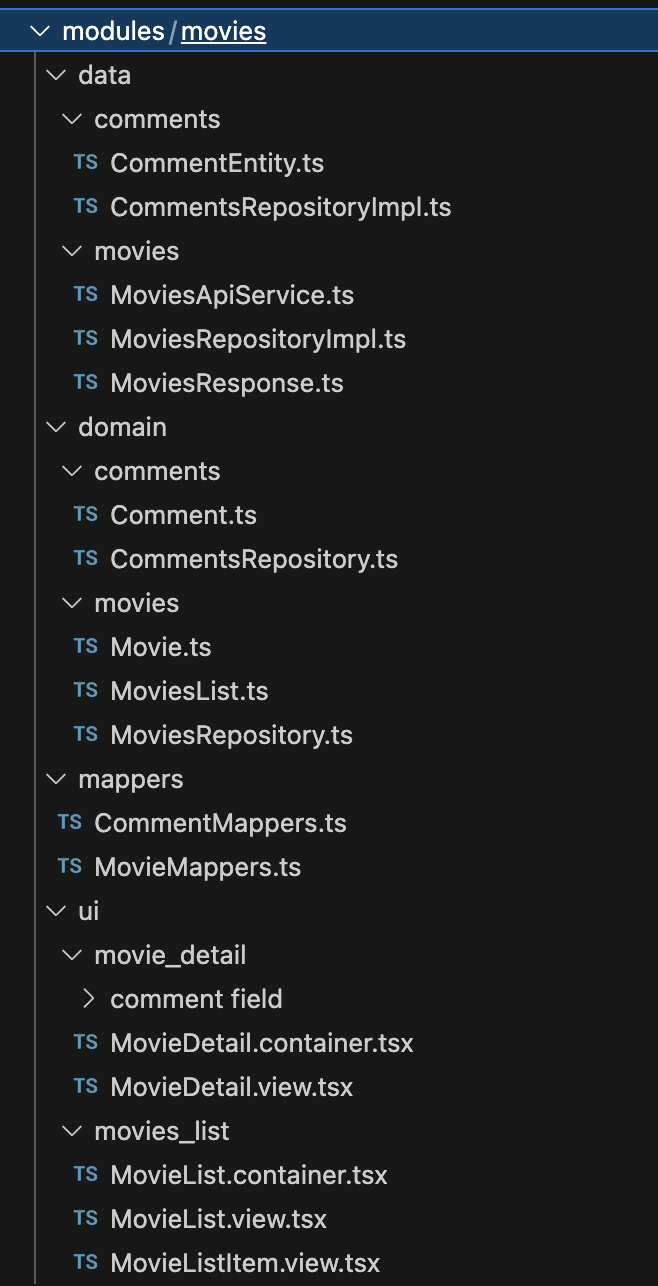


Рисунок 3 – Чистая архитектура в проекте

* 1. Внедрение зависимостей

Контейнер с зависимостями в проекте представлен в виде интерфейса:

Листинг 1 – Интерфейс контейнера с зависимостями

export interface DiContainer {

getApiService(): MoviesApiService;

getMoviesRepository(): MoviesRepository;

getCommentsRepository(): CommentsRepository;

}

Реализация этого контейнера лежит в отдельном файле и получается в коде через специальную функцию, использующую внутри себя React контекст:

Листинг 2 – Реализация контейнера с зависимостями

const appDiContainer: DiContainer = new (class implements DiContainer {

private readonly moviesApiService = new MoviesApiService();

getApiService(): MoviesApiService {

return this.moviesApiService;

}

getMoviesRepository(): MoviesRepository {

return new MoviesRepositoryImpl(this.getApiService());

}

getCommentsRepository(): CommentsRepository {

return new CommentsRepositoryImpl();

}

})();

const DIContext = React.createContext<Partial<DiContainer>>(appDiContainer);

export const useInjection = (): DiContainer => {

return useContext(DIContext) as DiContainer;

};

Функция **useInjection** используется в основном в контейнерах следующим образом (на примере списка с фильмами):

Листинг 3 – Пример использования DI

type Props = {

page: number;

};

export default function MovieListContainer(

{ page }: Props,

{ moviesRepository = useInjection().getMoviesRepository() },

) {

const limit = 10;

const { isLoading, error, data } = useQuery(

["movies\_list", page, limit],

() => moviesRepository.fetchMovies(page, limit),

);

useEffect(() => {

window.scrollTo({

top: 0,

behavior: "instant",

});

}, [page]);

if (isLoading) {

return <Loader />;

}

if (error || !data) {

return <Error />;

}

return (

<>

<MovieListView movies={data.movies} />

<PaginatorContainer

page={page}

route="/movies"

pagesCount={Math.ceil(data.movieCount / data.limit)}

/>

</>

);

}

1. Скриншоты приложения

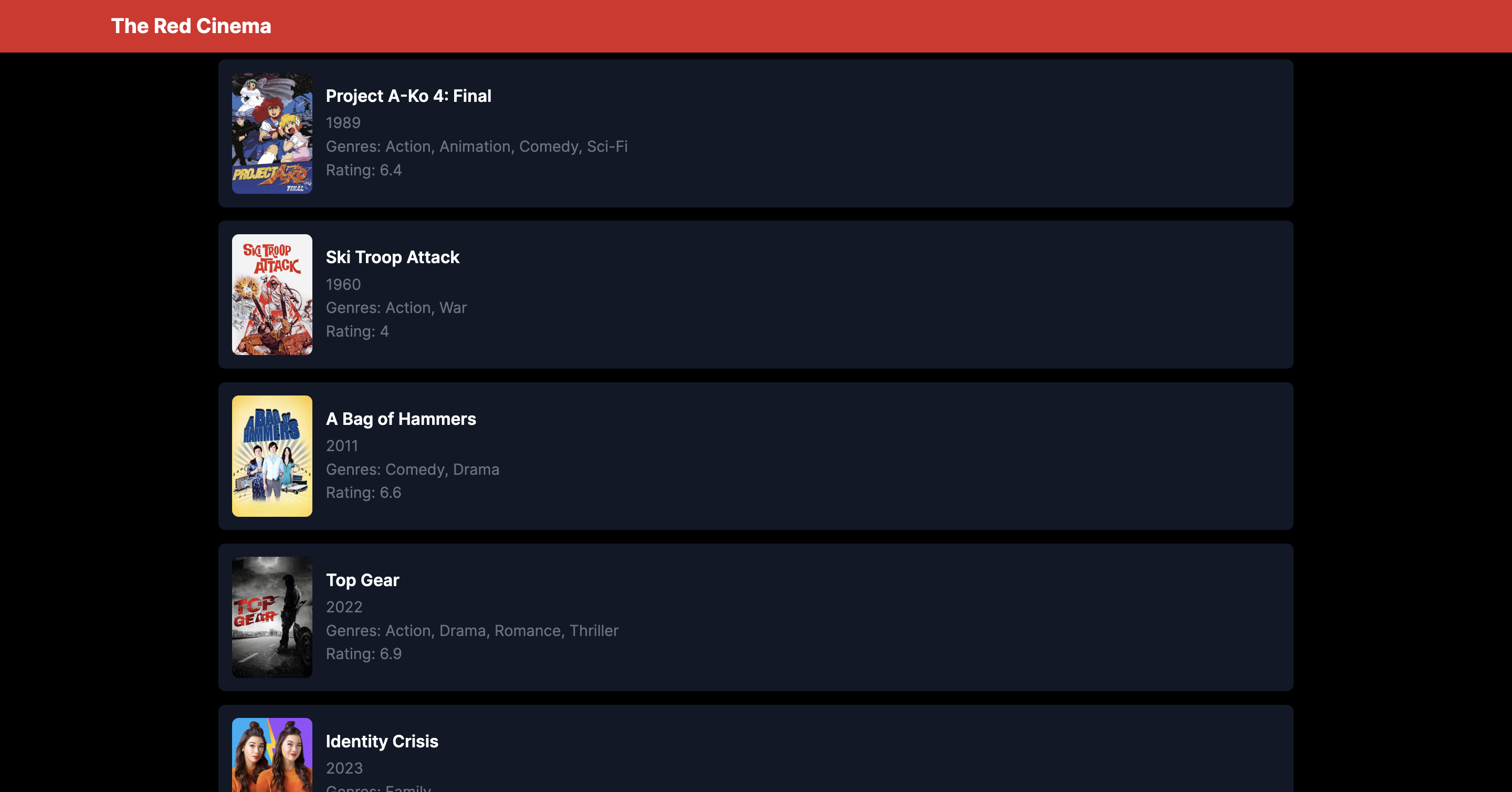


Рисунок 4 – Список фильмов

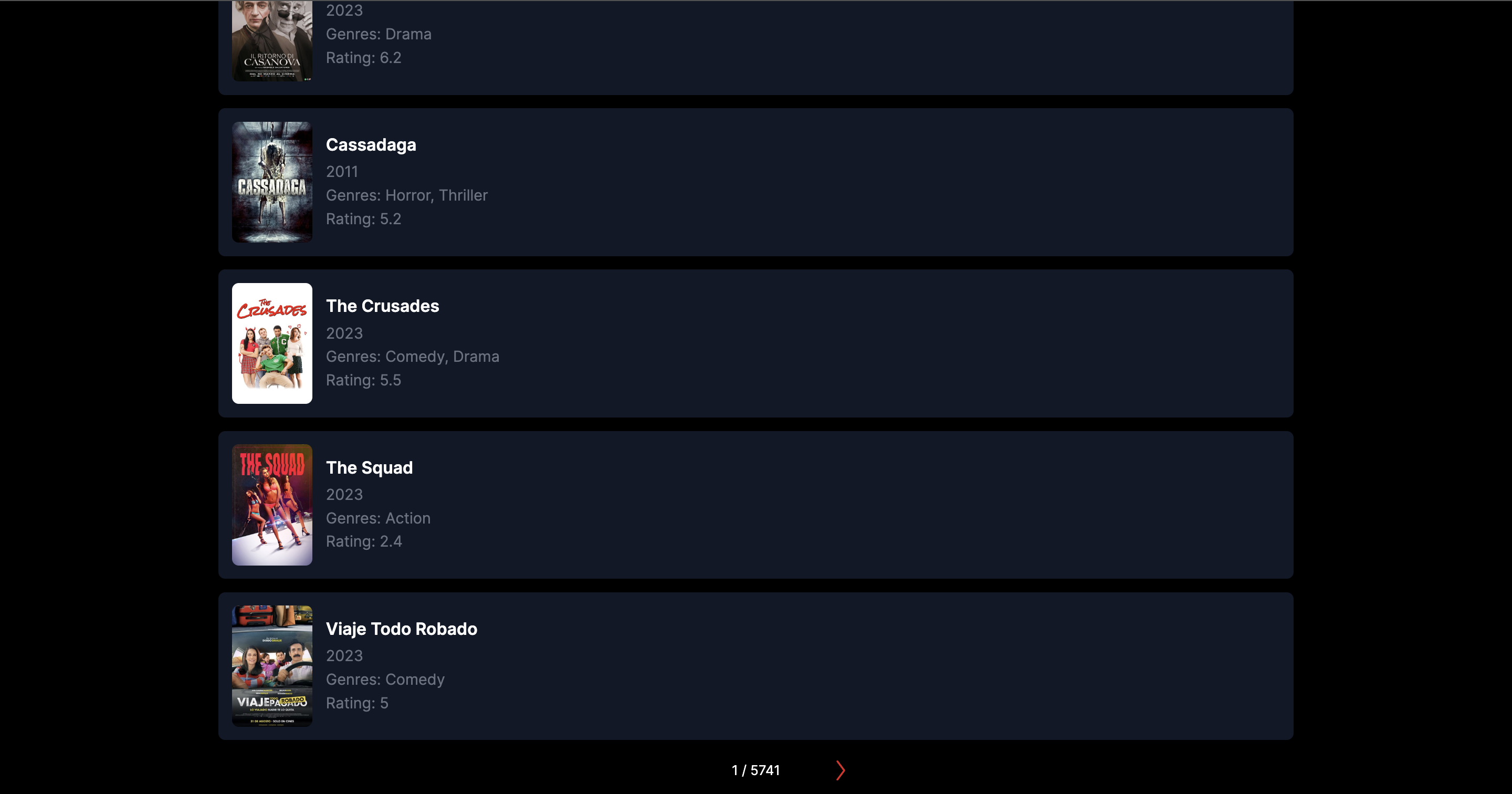


Рисунок 5 – Компонент пагинации

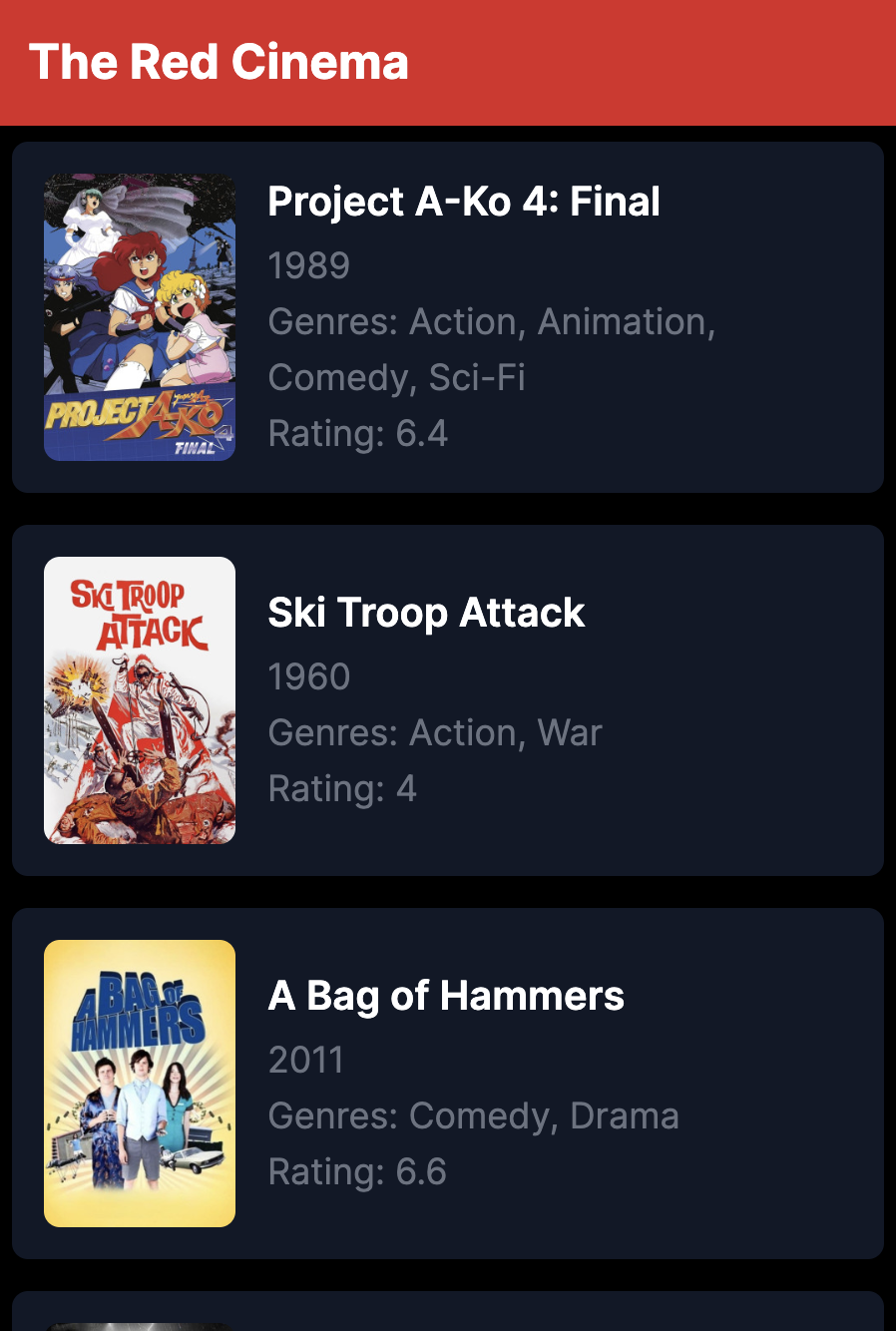


Рисунок 6 – Адаптивная вёрстка списка фильмов

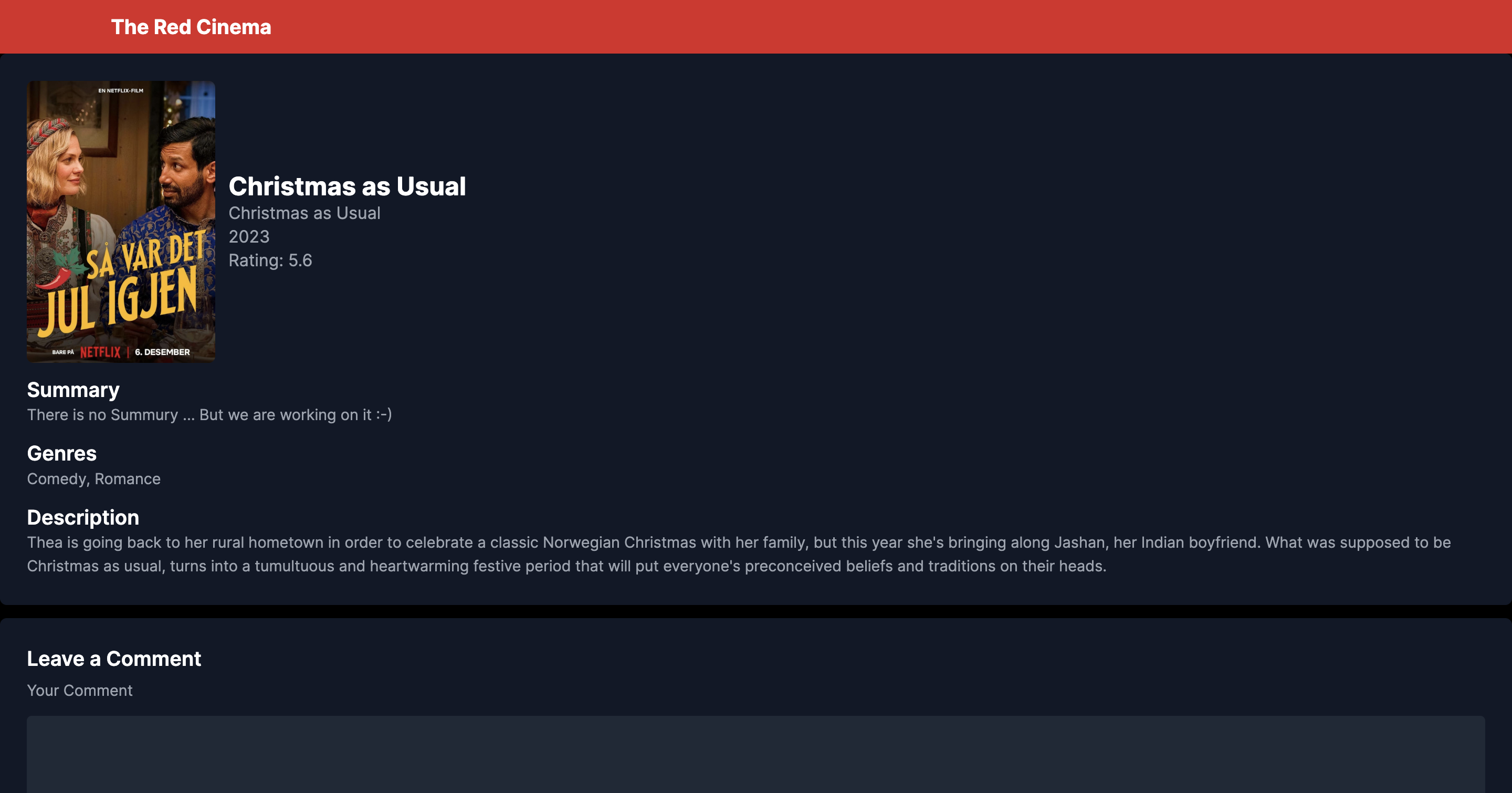


Рисунок 7 – Детальная страница фильма

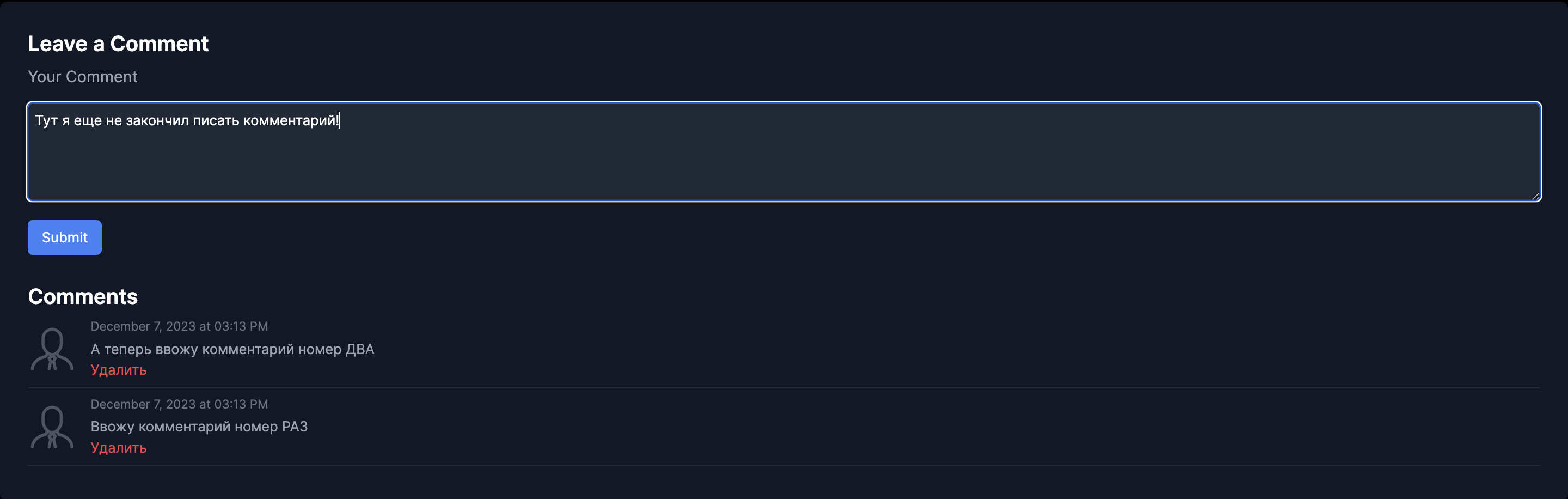


Рисунок 8 – Блок с комментариями



Рисунок 9 – Адаптивная вёрстка детального отображения

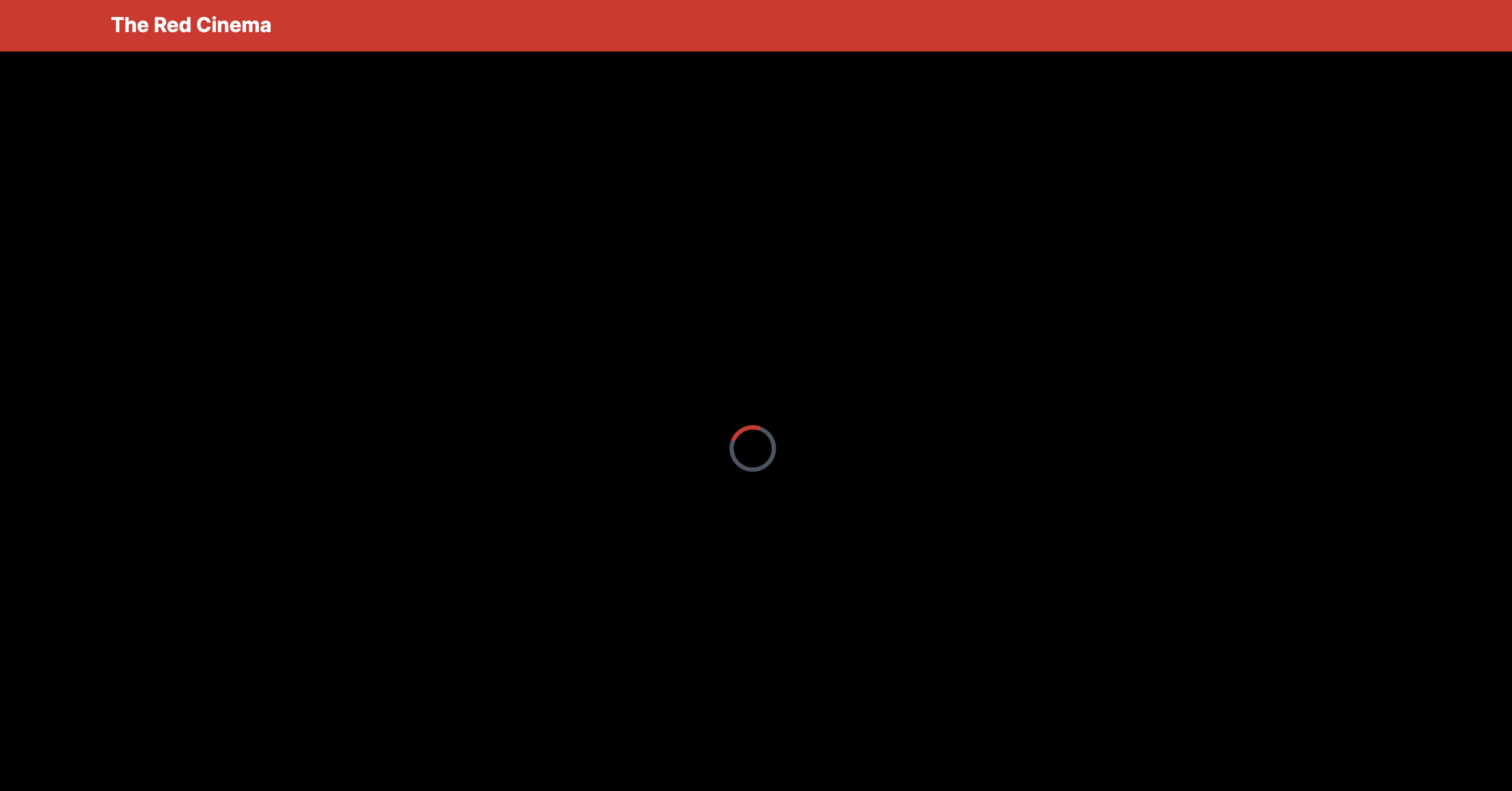


Рисунок 10 – Заглушка при загрузке данных

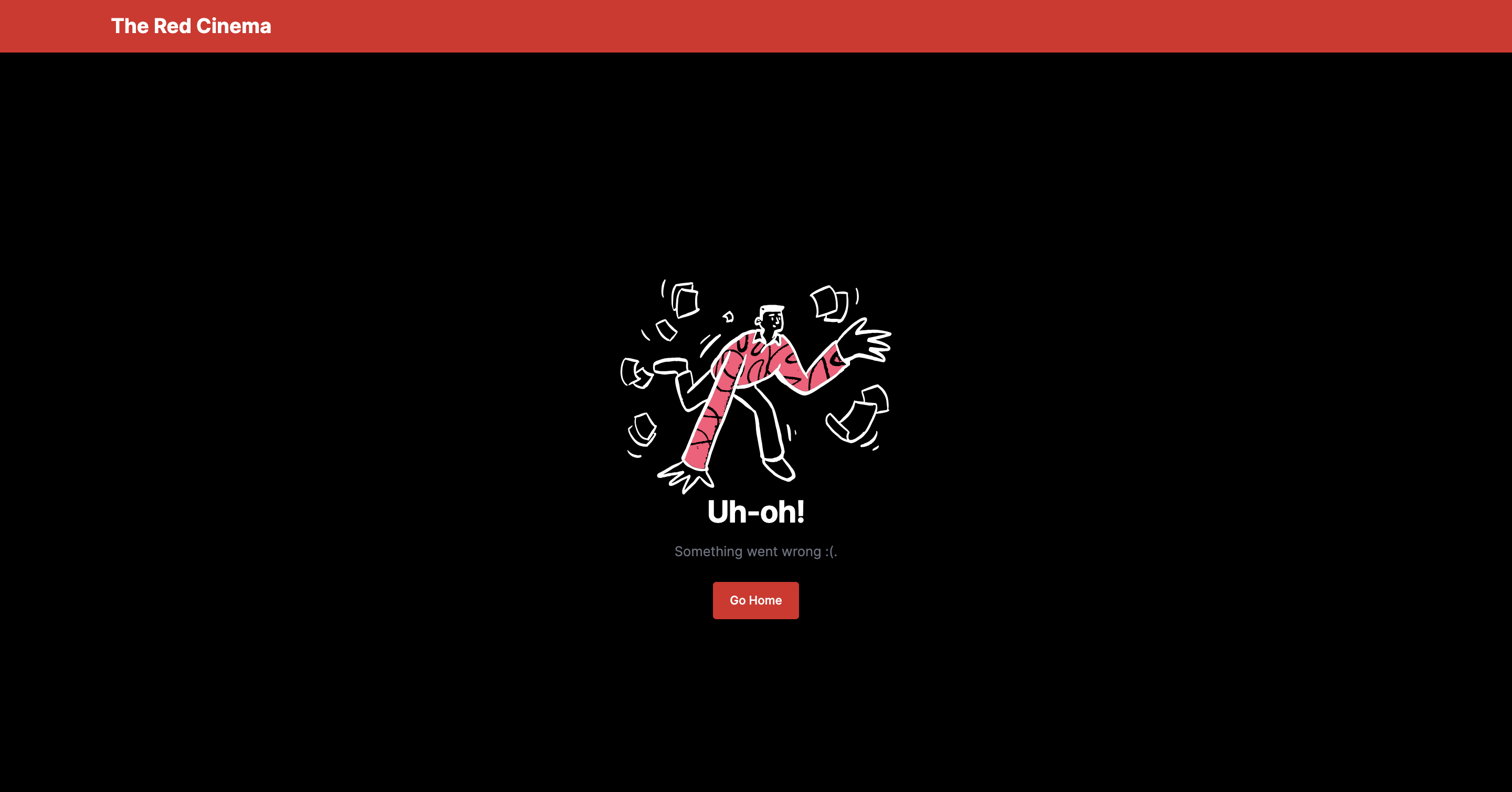


Рисунок 11 – Заглушка при ошибках загрузки

1. Вывод

В ходе проделанной работы мною был реализован веб-сайт с использованием фреймворков React и NextJS. Веб-сайт соответствует многим современным тенденциям в разработке и использует наиболее актуальный стек технологий и принципов программирования. Также реализованный сайт имеет вполне современный дизайн с адаптивной вёрсткой, что позволяет пользоваться им не только в браузере ноутбука или компьютера, но и в браузере мобильного устройства.