



Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический
университет имени Н.Э. Баумана
(национальный исследовательский
университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

ПО КУРСУ: «База данных»

НА ТЕМУ:

«Веб-приложение системы рекомендаций фильмов»

Студент ИУ9-52Б

Нгуен Т.Д. _____

(Ф.И.О)

(Подпись, дата)

Руководитель курсового проекта

Синявин А. В. _____

(Ф.И.О)

(Подпись, дата)

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего
образования
«Московский государственный технический университет имени Н.Э.
Баумана»(национальный исследовательский университет)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-9
(Индекс)

_____И.П. Иванов (И.О.Фамилия)

« ____ » _____ 2021 г.

З А Д А Н И Е

на выполнение курсовой работы

по дисциплине : «База данных»

Тема курсовой работы: Веб-приложение системы рекомендаций фильмов

Студент : Нгуен Тхань Дат ИУ9-626

График выполнения проекта: 25% к 4 нед., 50% к 8 нед., 75% к 11 нед., 100% к 14 нед.

1. Техническое задание

Этап 1: Определитесь с технологиями веб-разработки, выбрав систему управления базами данных и основной алгоритм, используемый в рекомендательных системах..

Этап 2: Разработка базы данных, создание запросов к базе данных для будущих работ.

Этап 3: Разработка пользовательских интерфейсов для веб-приложений с помощью React framework.

Этап 4: Реализация алгоритма рекомендации с подготовленной базой данных на сервере веб-приложений..

2. Оформление курсовой работы

2.1. Расчетно-пояснительная записка на 25 листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) _____

Дата выдачи задания « ____ » _____ 2021 г.

Оглавление

I. Введение

1. Система рекомендаций фильмов
2. Цель проекта
3. Обзорный подход
4. Использование технологий
5. Примечание

II. Методология

1. Описание алгоритма Matrix Factorization
2. Реализация алгоритма

III. Обзор пользовательских интерфейсов

1. Описание всех функций пользовательского интерфейса
2. Особенности пользовательского интерфейса

IV. Дизайн и характеристики сервера системы

1. Описать зачем нужен сервер и как он работает
2. Технологии, используемые для разработки сервера

V. Описание базы данных и запросы к ней.

1. Реляционная база данных
2. Некоторые запросы к базе данных для каждого отдельного действия пользователя
 - 2.1. Запрос на отдельный фильм
 - 2.2. Поиск по списку фильмов
 - 2.3. Поиск знаменитостей, связанных с отдельным фильмом
 - 2.4. Обработка некоторых основных действий пользователя
 - i. Когда пользователь регистрирует новую учетную запись
 - ii. Когда пользователи хотят добавить новый фильм в свои личные категории (понравился, не понравился, сохранен для будущего просмотра, ...)

VI. Заключение и дальнейшая работа

VII. Список литературы

I. Введение

1. Система рекомендаций фильмов

Этот проект представляет собой систему рекомендаций фильмов, которая с помощью математических уравнений дает список фильмов, которые пользователю может понравиться смотреть. Система не только рекомендует, но также помогает пользователям смотреть трейлер любого фильма, который они ищут, а также предоставляет много информации, такой как рейтинг IMDB, жанры, обзоры, актеры и т. Д. смотреть фильм, в конечном итоге спрашивать друзей и семью о предложениях, цель этого приложения - предоставить пользователю выбор фильмов, которые они могут захотеть посмотреть. Рекомендации по фильмам составляются с помощью сложных математических уравнений, чтобы попытаться определить, какие фильмы, скорее всего, понравятся пользователю, и дать им высокую оценку. Проект представляет собой приложение с простой навигацией, которое позволит пользователю искать фильмы, оценивать фильмы и получать рекомендации по фильмам на основе определенных критериев.

2. Цель проекта

Цель этого проекта - разработать систему рекомендаций, которая может узнать о своих пользователях и предоставить им набор фильмов, которые им, скорее всего, понравятся. Кроме того, помогите пользователям легче находить новые фильмы и сохранять список своих (не) любимых фильмов. Цель состоит в том, чтобы предоставить пользователю как пользовательские рекомендации, так и рекомендации на основе фильмов. Цель состоит в том, чтобы иметь приложение с простой навигацией, позволяющее пользователям легко перемещаться.

3. Обзорный подход

Этот проект будет основан на Javascript, поэтому большая часть вычислений будет производиться на Javascript. Он будет использовать SQL для хранения и извлечения данных. The front-end (или пользовательские интерфейсы) разработан с использованием React[1], а the back-end (или сервер) - с помощью Express[2]. Обе они являются популярными библиотеками для веб-разработки на Node.JS. Используемая система управления базами данных (СУБД) - SQL Server 2019. Алгоритм, используемый для рекомендательной системы под названием Matrix Factorization, который представляет собой класс алгоритмов Collaborative Filtering(CF), используемых в рекомендательных системах.

4. Использование технологий

Чтобы облегчить пользователю поиск фильмов, система позволяет смотреть трейлеры любых фильмов напрямую, без перехода на новую страницу. С помощью Youtube API[3] наша система получает видео высокого качества (почти с разрешением 1440p или 1080p) из базы данных Youtube и воспроизводит их без перенаправления на страницу Youtube. Не только трейлеры к фильмам, афиши и описания фильмов имеют решающее значение. Поблагодарить themoviedb[4], которые предоставили большое количество качественных постеров с фильмами через свой API. И последнее, но не менее важное: нам нужна вся информация о фильмах, такая как название, актеры, жанры, рейтинги ... Особая благодарность IMDb Corp.[5] для предоставленной всей этой информации точной и достоверной.

5. Примечание

В условиях ограничений по времени и объему памяти оборудования, сохранение информации всех созданных фильмов невозможно. Таким образом, в этом проекте сохраняется информация только о фильмах, созданных за последние 10 лет (2010-2020).

II. Методология

1. Описание алгоритма Matrix Factorization

- Collaborative filtering[6] - это метод, используемый рекомендательными системами. Этот метод автоматического прогнозирования (filtering) интересов пользователя путем сбора информации о предпочтениях или вкусовых предпочтениях многих пользователей (Collaborating).
- Matrix Factorization - это класс алгоритмов Collaborative filtering. Этот метод стал широко известен благодаря своей эффективности, как сообщил Simon Funk[7].
- Matrix Factorization - это способ создания скрытых функций при умножении двух разных типов сущностей. Collaborative filtering - это применение матричной факторизации для определения взаимосвязи между «элементами» и «пользователями». В нашей задаче «элементы» будут известны как «фильмы». С вводом оценок пользователей по элементам мы хотели бы предсказать, как пользователи будут оценивать элементы, чтобы пользователи могли получить рекомендацию на основе прогноза.
- Посмотрим на пример, у нас есть таблица рейтинга клиентов, состоящая из 5 пользователей и 5 фильмов, а рейтинги являются целыми числами от 1 до 5, матрица представлена в таблице ниже.

	Movie1	Movie2	Movie3	Movie4	Movie5
U1		5	4	2	1
U2	1			5	3
U3	1	4	4	1	
U4			2		2
U5	3	1	1		

Table1-Users' ratings table on movie

- Поскольку не каждый пользователь ставит оценки всем фильмам, в матрице много пропущенных значений, что приводит к разреженной матрице. Следовательно, нулевые значения, не предоставленные пользователями, будут заполнены 0, так что заполненные значения будут предоставлены для умножения. С помощью матричной факторизации мы можем обнаружить эти скрытые особенности, чтобы сделать прогноз на основе рейтинга с учетом сходства предпочтений и взаимодействий пользователей.
- С момента первоначальной работы Simon Funk[7] было предложено множество подходов к матричной факторизации. Такой как Funk MF[8],

SVD++[9], Asymmetric SVD++[10], Hybrid MF[11], Deep learning MF[12], ... В этом проекте я выбрал очень простой метод в поле машинного обучения с именем gradient descent[13], цель которого - найти локальный минимум разницы.

- Сначала определен набор пользователей (U), элементов (D), $R = |U| \times |D|$. Матрица $|U| \times |D|$ включает в себя все оценки, выставленные пользователями. Цель состоит в том, чтобы обнаружить K скрытых функций. Учитывая ввод двух матриц-матриц P ($|U| \times K$) и Q ($|D| \times K$), он сгенерирует результат продукта R. Матрица P представляет связь между пользователем и функциями, а матрица Q представляет связь между элементом и функциями. Мы можем получить прогноз рейтинга элемента путем вычисления скалярного произведения двух векторов, соответствующих P_i и P_j .

$$R \approx P * Q^T = \tilde{R}$$

$$\tilde{R}_{ij} = P_i^T Q_j = \sum_{k=1}^K P_{ik} Q_{kj}$$

- Мы определили "error" функцию, которая представляет собой разницу между реальным значением и прогнозируемым значением.

$$E_{ij}^2 = (R_{ij} - \tilde{R}_{ij})^2 = (R_{ij} - \sum_{k=1}^K P_{ik} Q_{kj})^2$$

- Наша задача - обновить матрицы P, Q, чтобы минимизировать значение "error" функции. Градиент может минимизировать это значение, и поэтому мы дифференцируем приведенное выше уравнение по отношению к этим двум переменным отдельно.

$$\frac{\partial}{\partial P_{ik}} E_{ij}^2 = -2(R_{ij} - \tilde{R}_{ij})(Q_{kj}) = -2E_{ij}Q_{kj}$$

$$\frac{\partial}{\partial Q_{kj}} E_{ij}^2 = -2(R_{ij} - \tilde{R}_{ij})(P_{ik}) = -2E_{ij}P_{ik}$$

- Из градиента математическая формула может быть обновлена как для P_{ik} , так и для Q_{kj} . α - это шаг для достижения минимума при вычислении градиента, а α обычно устанавливается с небольшим значением.

$$P'_{ik} = P_{ik} + \alpha \frac{\partial}{\partial P_{ik}} E_{ij}^2 = P_{ik} + 2\alpha E_{ij} Q_{kj}$$

$$Q'_{kj} = Q_{kj} + \alpha \frac{\partial}{\partial Q_{kj}} E_{ij}^2 = Q_{kj} + 2\alpha E_{ij} P_{ik}$$

- Из приведенного выше уравнения, P'_{ik} и Q'_{kj} могут обновляться с помощью итераций до тех пор, значение "error" достигнет своего минимума.

$$E = \sum E_{ij}^2 = \sum (R_{ij} - \sum_{k=1}^K P_{ik} Q_{kj})^2$$

2. Реализация алгоритма

- Кода из проекта:

```
const matrix_factorization = (R,K,N,M) => {
  var P = createRandomMatrix(N,K);
  var Q = createRandomMatrix(M,K);

  var alpha = 0.02;
  var beta = 0.002;
  var epsilon = 0.0001;

  Q = T(Q,M,K);
  while (true) {
    for (let i=0;i<N;i++)
      for (let j=0;j<M;j++)
        if (R[i][j] > 0) {
          var Eij = R[i][j];
          for (let t=0;t<K;t++) Eij -= P[i][t]*Q[t][j];
          for (let t=0;t<K;t++) {
            P[i][t] += alpha * (2 * Eij * Q[t][j] - beta * P[i][t]);
            Q[t][j] += alpha * (2 * Eij * P[i][t] - beta * Q[t][j]);
          }
        }
    var e = 0;
    for (let i=0;i<N;i++)
      for (let j=0;j<M;j++)
        if (R[i][j]>0) {
          var curr = R[i][j];
          for (let t=0;t<K;t++) curr -= P[i][t]*Q[t][j]
          e = e + curr*curr;
        }
    if (e<epsilon) break;
  }

  return matrix_mul(P,Q,K,N,M);
}
```

- Посмотрим небольшой пример проверки исправления алгоритма.
Приведены данные рейтинговых оценок 3 пользователей 6 фильмов (оценка от 0 до 10, оценка 0 означает, что пользователи еще не знали этот фильм)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	8	7	7	0	0	6
User 2	0	6	7	3	7	0
User 3	6	0	4	2	9	4

Table 1: Initial rating point

- Прогнозируемая матрица генерируется ниже. Матрица прогнозов имеет аналогичный результат с истинными значениями, а оценки 0 заменяются прогнозом, основанным на аналогичных предпочтениях пользователей в отношении фильмов.

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	8.001952	7.000869	6.99874	3.91094	8.87168	5.99512
User 2	7.13439	5.99828	6.99650	3.00044	7.00008	3.92563
User 3	5.99604	5.21907	4.00086	1.99841	8.99678	4.00256

Table 2: Result after executed algorithm

- Так, в примере, User1 не знал(а) Movie5, на основе предсказания результата нашего алгоритма, ему(ей) может понравиться этот фильм с довольно высокой оценкой ($\approx 8,87/10$).
- **Примечание:** Поскольку в алгоритме используется метод машинного обучения, это означает использование более количества данных, получение более точного результата.

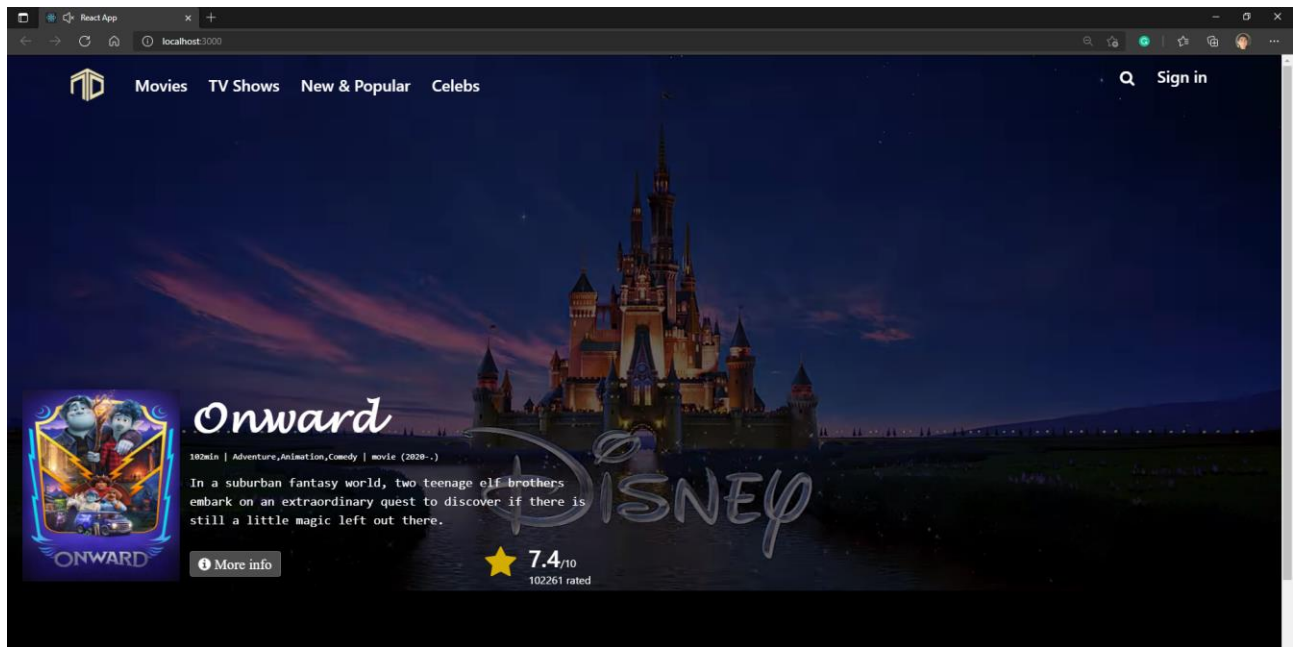
III. Обзор пользовательских интерфейсов

1. Описание всех функций пользовательского интерфейса

- Пользовательские интерфейсы (UI) нашей системы, разработанные с использованием React. По сути, UI содержит несколько компонентов, отображающих информацию. По сути, компоненты - это многоразовые пользовательские интерфейсы, которые позволяют разделить приложение на отдельные блоки, которые действуют независимо друг от друга. Компоненты принимают произвольный ввод с данными (props) и возвращают элемент React, чтобы объявить, что должно появиться на экране. Они могут взаимодействовать с другими компонентами через props для создания сложного пользовательского интерфейса.
- Пользователи могут войти в систему, если у них была зарегистрированная учетная запись, в противном случае они могут зарегистрироваться для новой.
- Пользователи могут искать любой фильм по ключевому слову. После того, как пользователи выбрали фильм, система покажет им всю информацию о нем. Пользователь может поставить оценку этому фильму или добавить его в свой сохраненный список. Система использует эту оценку для рекомендации новых фильмов, которые отображаются на главной странице.

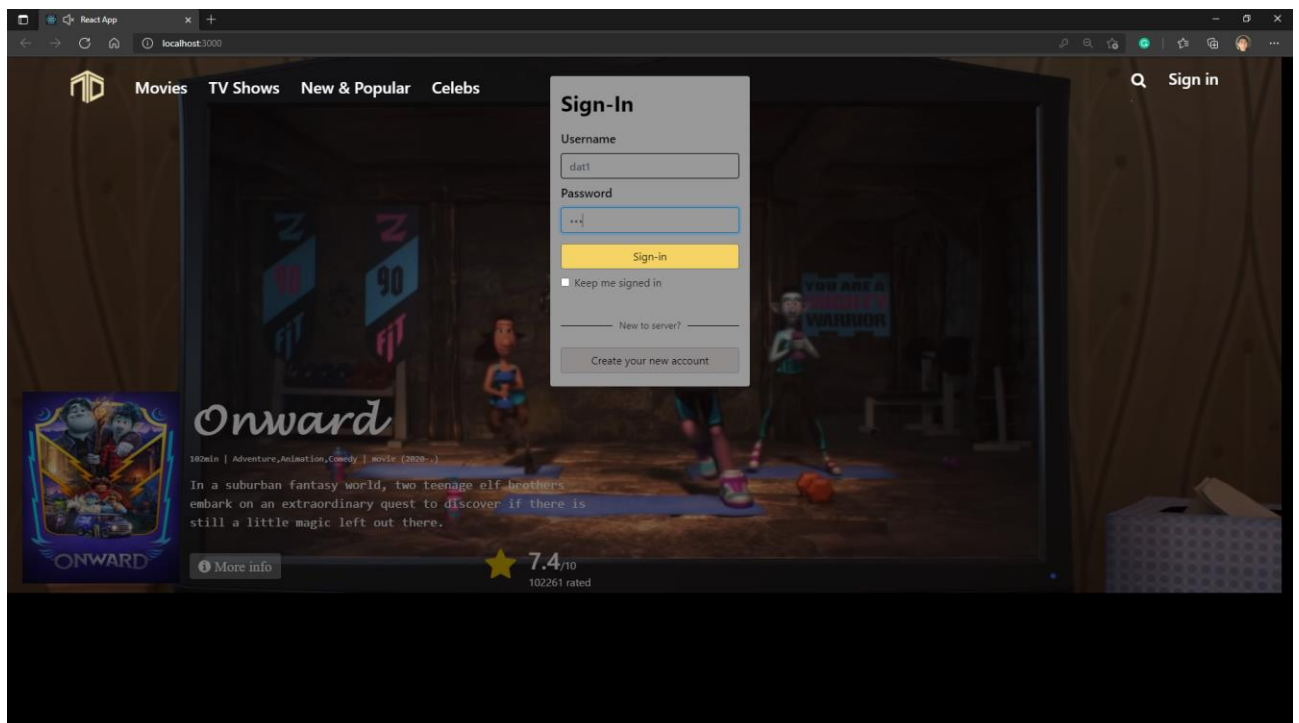
2. Особенности пользовательского интерфейса

- Главная страница, когда пользователь не авторизовался. Один из самых популярных фильмов показал свою информацию и трейлер. Пользователь может войти в свою учетную запись, нажав кнопку «Sign in» в правом верхнем углу.

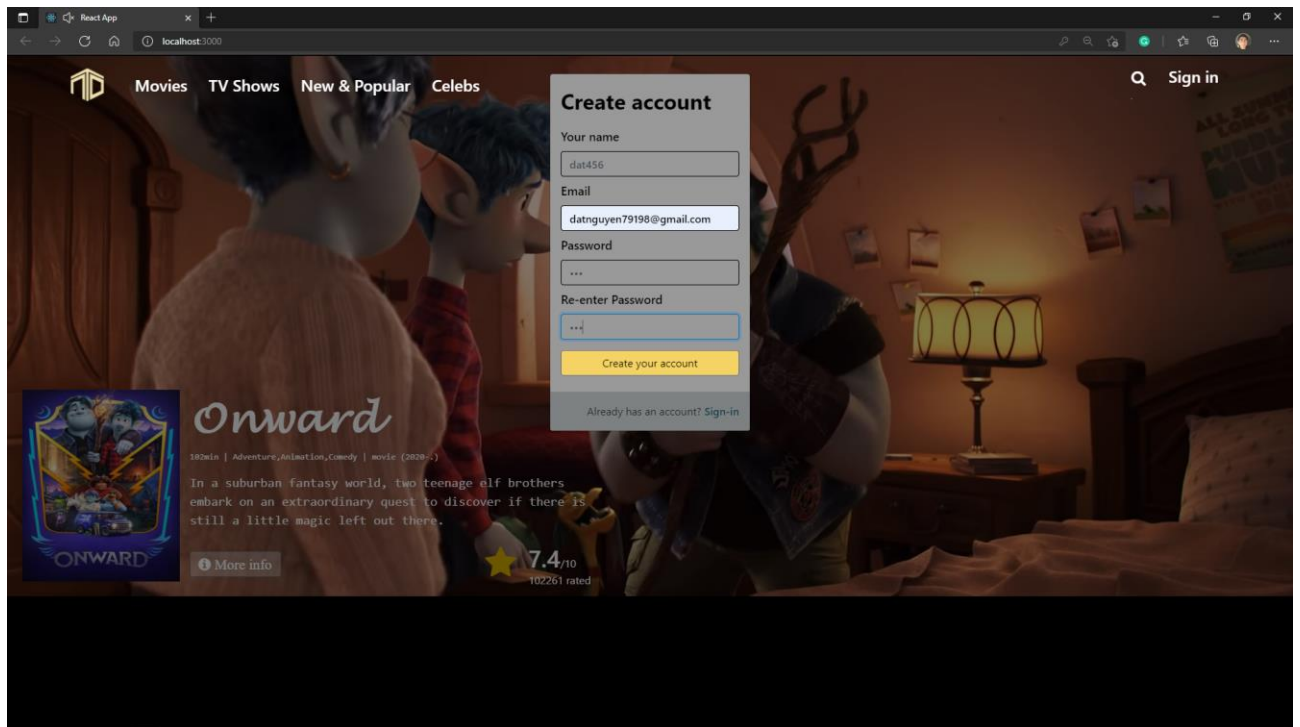


Picture 1: Main page

- Интерфейс входа и регистрации. Если у пользователей еще нет учетной записи, они могут зарегистрировать новую.

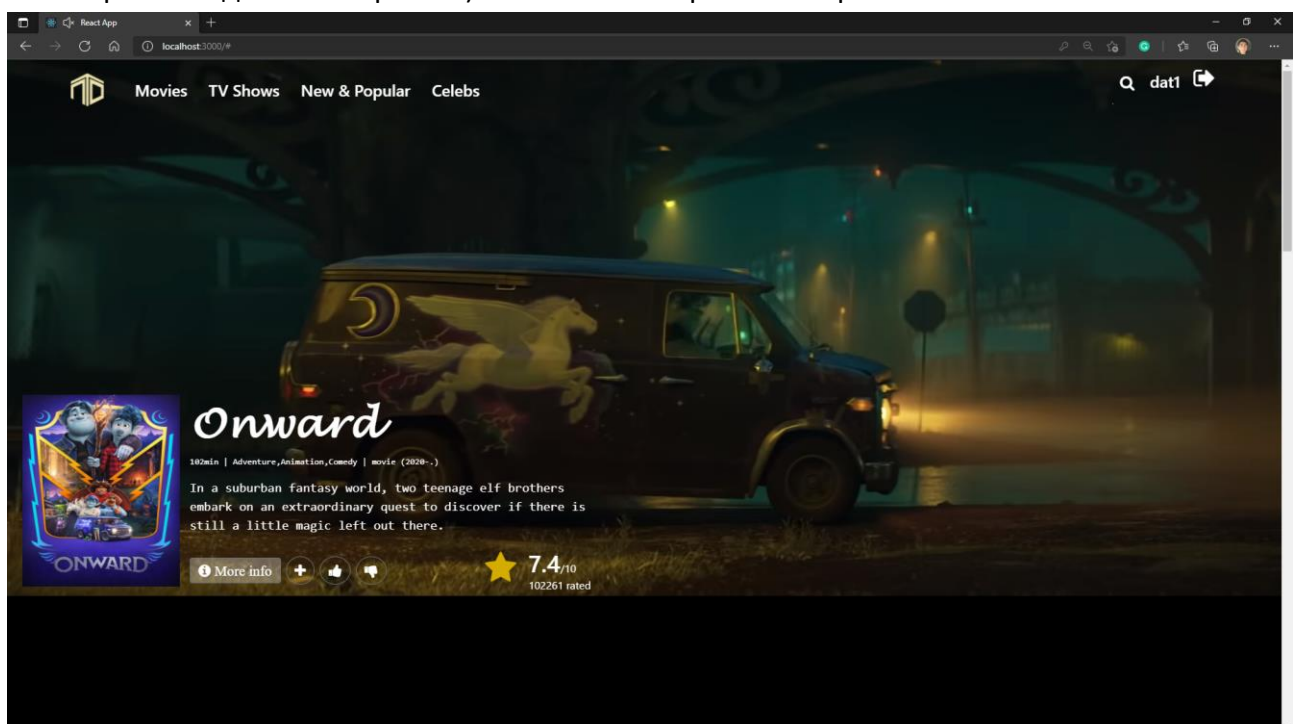


Picture 2: Sign in function

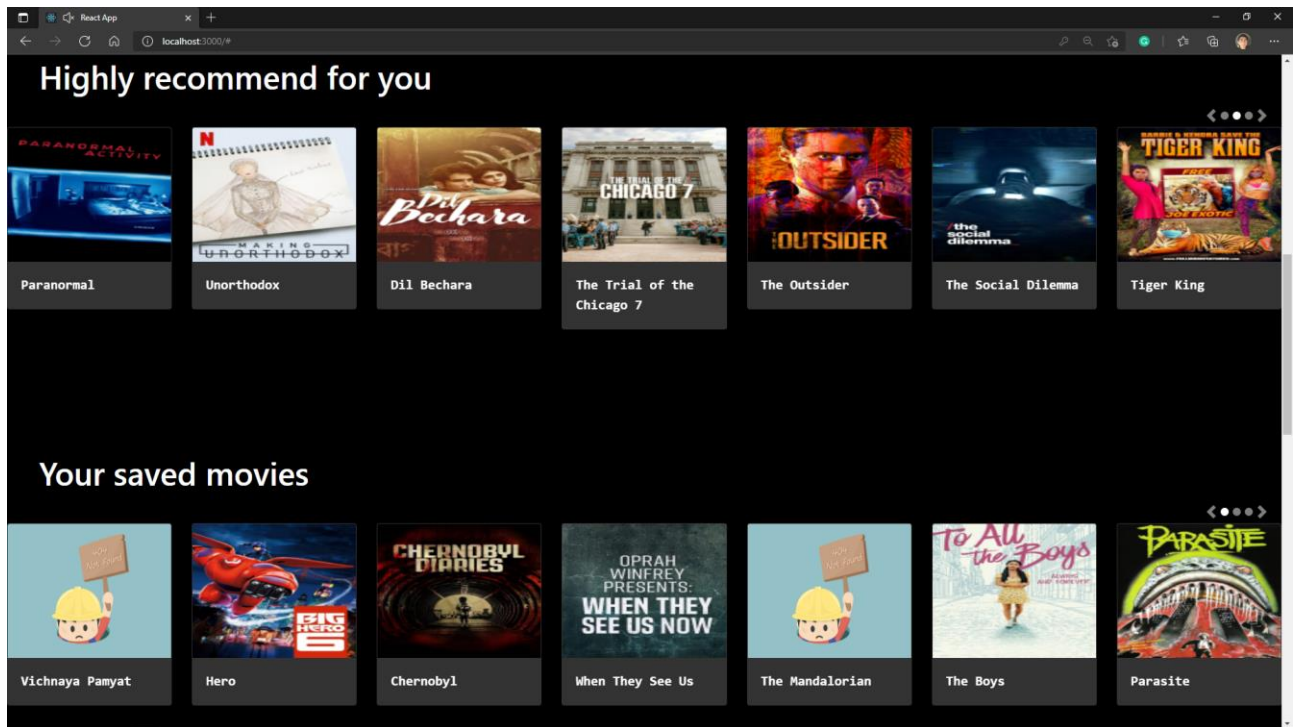


Picture 3: Sign up function

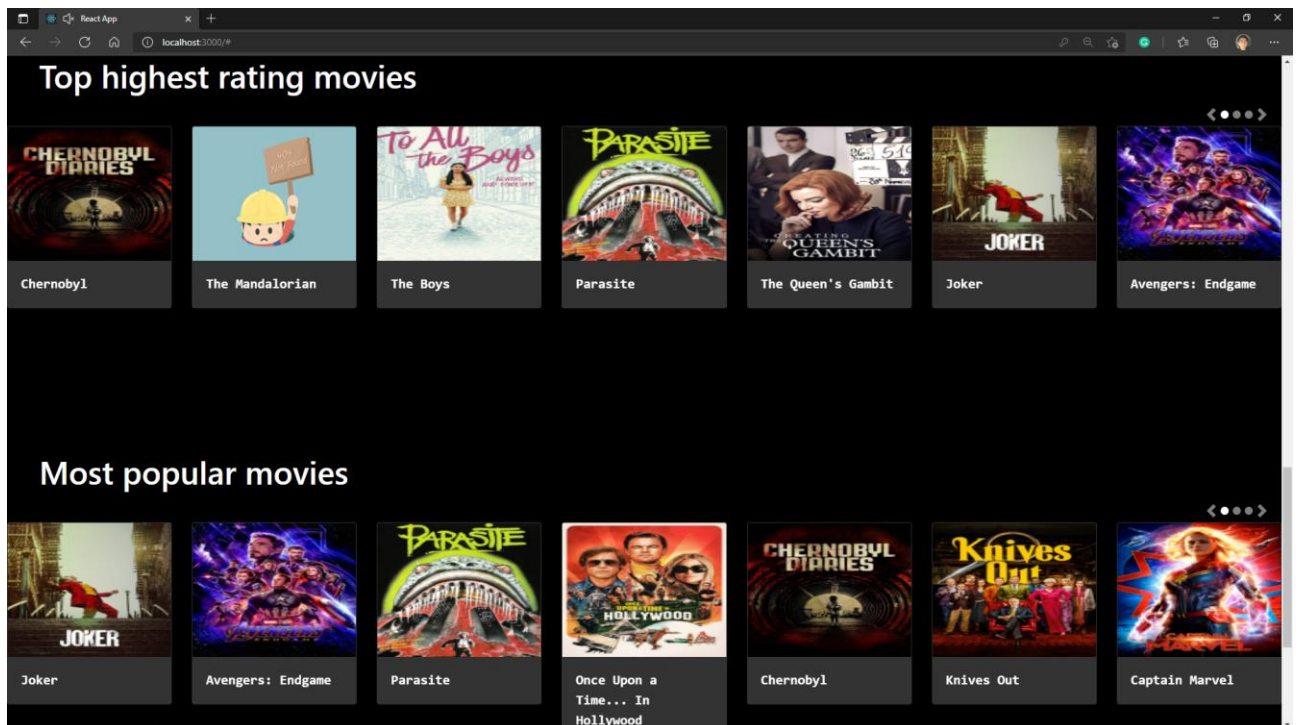
- Главная страница после того, пользователи войдут в свою учетную запись. Они также могут выйти, нажав кнопку в правом верхнем углу. Под фильмом на верхней главной странице находятся списки фильмов, рекомендованных сервером для отдельного пользователя (3 списка, в каждом списке 21 рекомендованный фильм) и их список сохраненных фильмов.



Picture 4: Main page after user signed in successfully

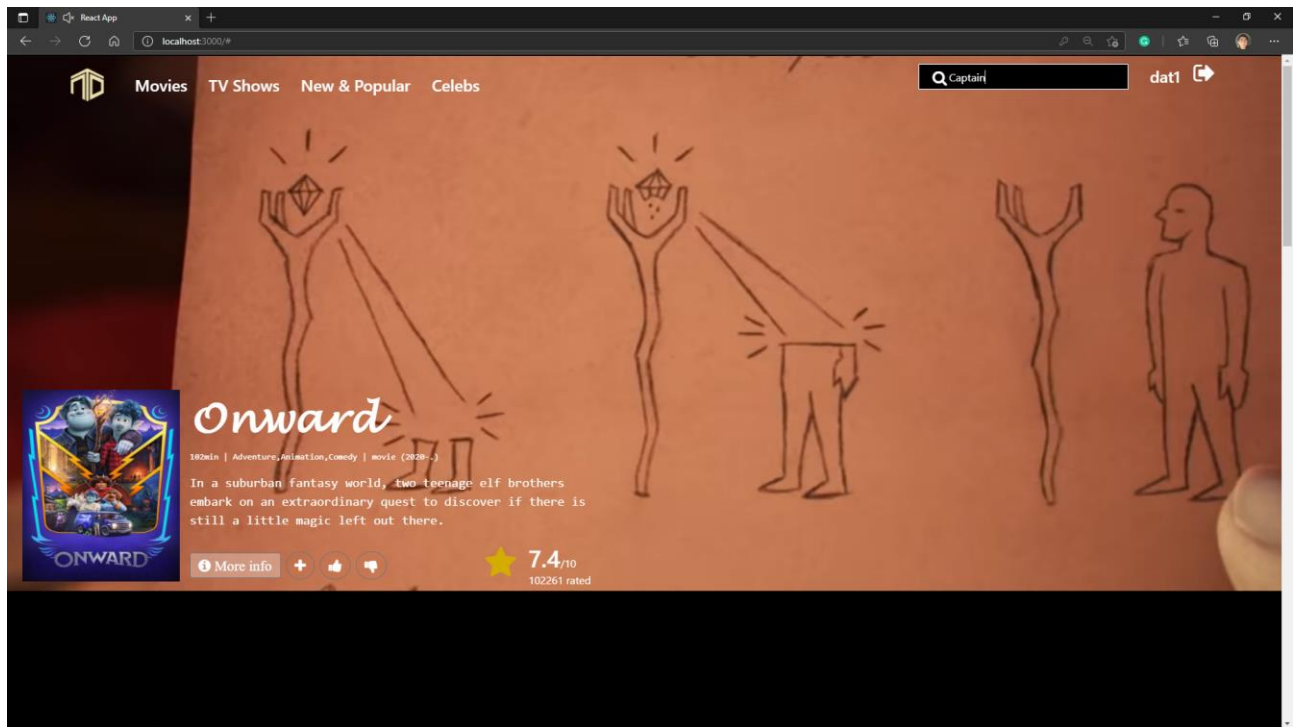


Picture 5: Recommended movies for user and their saved movies

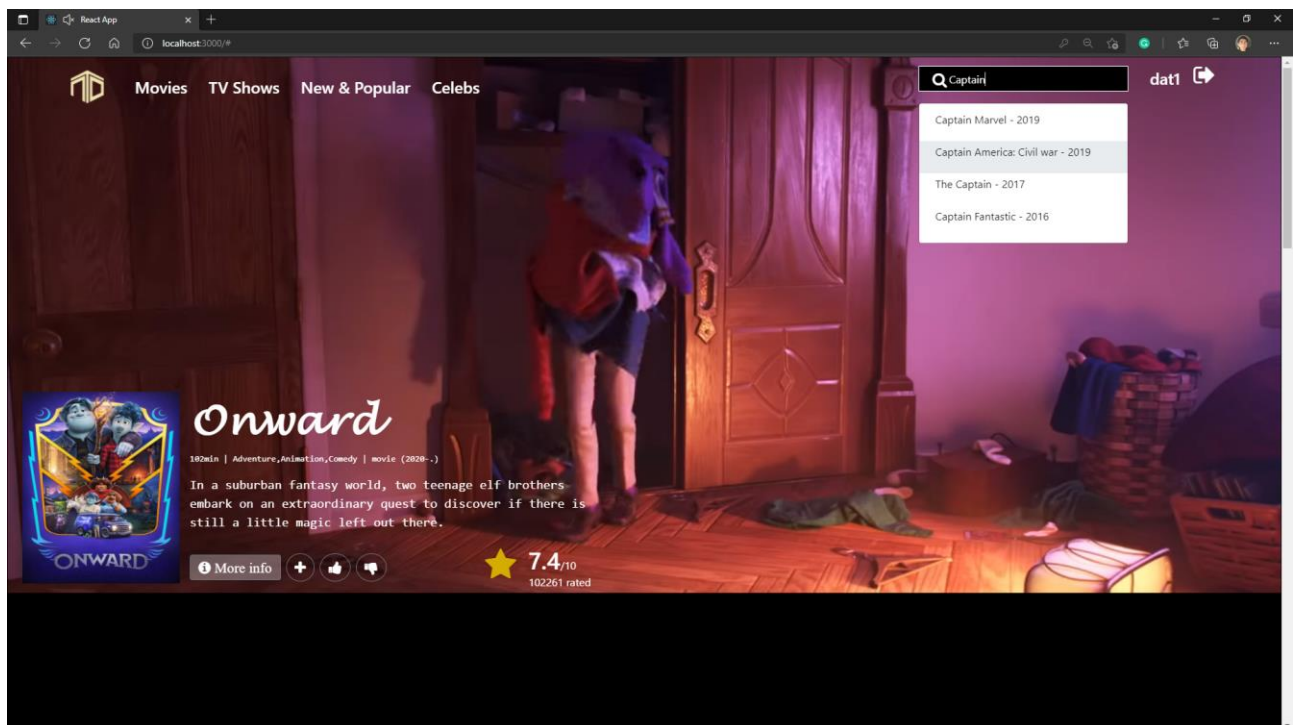


Picture 6: Recommended movies based on rating and popularity

- Система не только рекомендует, но и помогает пользователю найти фильм, который он ищет. Если щелкнуть символ "Лупа" вверху и ввести полное имя или ключевое слово, система выдаст результаты, как показано ниже.

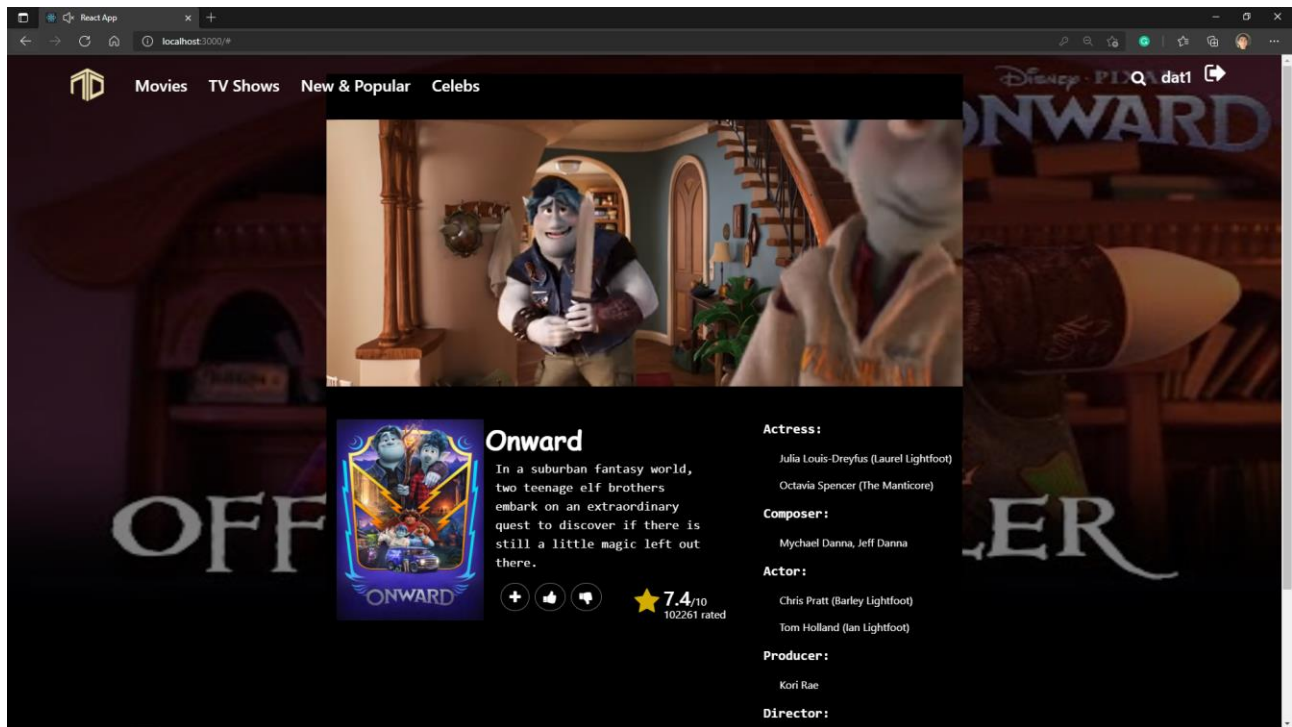


Picture 7: Searching movie function



Picture 8: Searching result for user's keyword

- Когда пользователь хочет увидеть дополнительную информацию о каком-либо фильме, щелкнув по нему, появится интерфейс, показанный ниже. Там пользователь может найти его информацию, трейлер. Кроме того, пользователи могут добавить этот фильм в свой сохраненный список и указать рейтинг для этого фильма.



Picture 9: Information of chosen movie

IV. Дизайн и характеристики сервера системы

1. Описать зачем нужен сервер и как он работает

- Всякий раз, когда пользователь выполняет действие, такое как поиск фильма, внешний интерфейс или пользовательский интерфейс представляет собой инструмент, который делает это действие визуализируемым и более простым для пользователя. Это действие можно назвать «запросом» - пользователь запрашивает часть информации. Сервер работает как «средний человек», что означает получение запросов от пользователя, обработку этих запросов и «возврат» информации, которую пользователь хочет найти.

2. Технологии, используемые для разработки сервера

- В этом проекте я использовал библиотеку Express для разработки сервера. Express помогает контролировать процесс «запрос» - «возврат» на стороне сервера, что упрощает управление и сопровождение. Library Express использует методы HTTP-запроса для приема запросов от пользователей. HTTP-запрос - это действие, которое должно быть выполнено над ресурсом, идентифицированным заданным URL-адресом запроса. Например, когда пользователь хочет выполнить поиск в фильме, пользовательский интерфейс отправляет метод HTTP-запроса GET на URL-адрес <domain>/SEARCH MOVIE?Name=<movie name want to find>. Сервер будет обрабатывать каждый HTTP-запрос и ответ на интерфейсную часть информации в теле HTML-страницы.

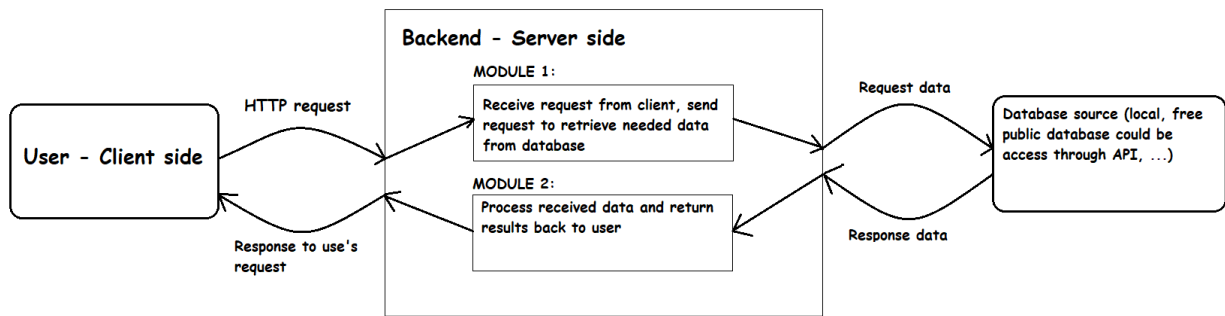


Figure 1: High-architecture system design

- В этом проекте я управлял 2 типами запросов. Запросить данные из базы данных и данные из API. Обработка данных запроса из внутренней базы данных Я объясню более подробно в части V.
- О запросе данных из API, в этом проекте я использовал 2 онлайн-источника данных. Они предоставили бесплатный API для доступа к своим данным. Во-первых, это видеоданные, предоставленные Youtube, во-вторых, данные афиши фильма, предоставленные компанией themoviedb.
- Несмотря на то, что есть 2 разных API, но один и тот же метод их использования. Использование HTTP-запроса с некоторыми настройками в URL, заголовке, ... зависит от того, какой API. После этого мы можем получить доступ к их общедоступной базе данных и получить необходимую информацию.
- Передав название фильма, который хочет найти, API Youtube вернет индекс видео в своей базе данных. Затем поместите этот индекс в URL-адрес, чтобы получить видео.

```

export default axios.create({
  baseURL : 'https://www.googleapis.com/youtube/v3/',
  params : {
    part : 'snippet',
    maxResults : 1,
    key : KEY
  }
})

/*
const res = await youtube.get('/search', {
  params : {
    q : requestApi
  }
})
*/

videoLink = "https://www.youtube.com/embed/"+youtubeId+
  "?cc_load_policy=3
  &autoplay=1
  &mute=0
  &controls=0
  &origin=http%3A%2F%2Flocalhost%3A3000
  &playsinline=1&showinfo=0&rel=0
  &iv_load_policy=3
  &modestbranding=0
  &playlist="+youtubeId+"
  &color=white&loop=1&enablejsapi=1&widgetid=1";
  
```

Picture 10: Calling method to Youtube's API

- Тот же метод, что и API YouTube, передав имя фильма, API-интерфейс themoviedb вернет индекс этого фильма в своей базе данных. С помощью этого индекса мы можем свободно извлекать эти данные.

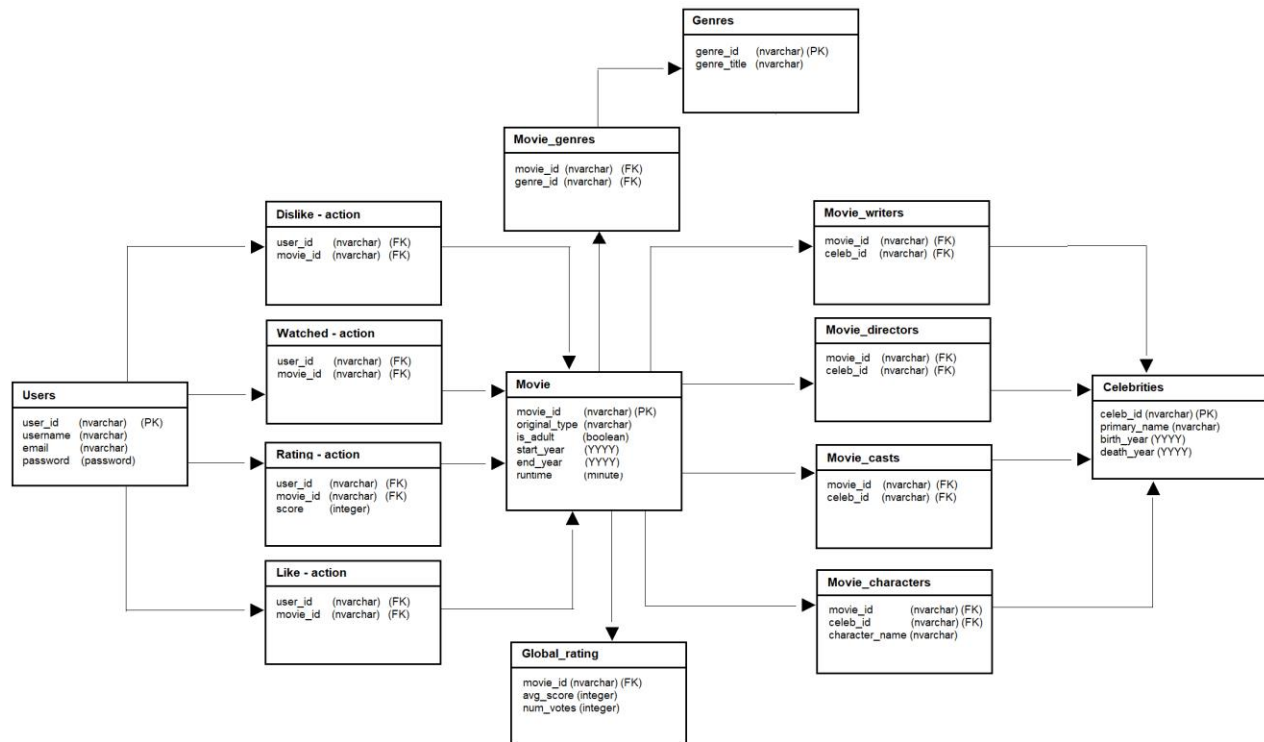
```
router.get('/search', (req,res,next) => {  
  var baseUrl = "https://api.themoviedb.org/3/search/movie?"  
  for (const key in req.query) {  
    baseUrl = baseUrl + key +"="+req.query[key]+"&";  
  }  
  baseUrl = baseUrl.slice(0,baseUrl.length-1)  
  httprequest ({  
    {url : baseUrl},  
    (error, response, body) => {  
      console.log(baseUrl);  
      res.send(JSON.parse(body));  
    }  
  })  
})
```

Picture 11: Calling method to TheMovieDB's API

V. Описание базы данных и запросы к ней.

1. Реляционная база данных

- Система, использующая реляционную базу данных и использующая SQL для запросов и обслуживания базы данных.
- Реляционная база данных - это цифровая база данных, основанная на реляционной модели данных. Эта модель организует данные в одну или несколько таблиц (или «отношений») столбцов и строк с уникальным ключом, идентифицирующим каждую строку. Чтобы система управления базами данных (СУБД) работала эффективно и точно, она должна использовать транзакции ACID.
- Сущности и их отношения могут быть построены в схеме, как показано ниже.



Picture 12: Database schema

2. Некоторые запросы к базе данных для каждого отдельного действия пользователя

2.1. Запрос на отдельный фильм

- Во-первых, нам нужно создать VIEW для будущей работы. В базе данных VIEW - это набор результатов сохраненного запроса к данным, который пользователи базы данных могут запрашивать так же, как в постоянном объекте коллекции базы данных.
- Эта предустановленная команда запроса хранится в словаре базы данных. VIEWS могут использоваться как повторно используемые разделы SELECT / CODE, которые могут быть включены в другие выборки / запросы, к которым нужно присоединиться, и использовать различные различные фильтры, без необходимости заново создавать весь SELECT каждый раз.
- View содержит всю основную информацию, относящуюся к фильму :


```

if OBJECT_ID(N'dbo.movie_title_view',N'V') is not null
    drop view [dbo].[movie_title_view]
go

create view dbo.movie_title_view (
    [movie_id],[original_type],[is_adult],
    [start_year],[end_year],[runtime],
    [num_votes], [avg_scores]
)
with schemabinding
as
select
    title.tconst as [movie_id], title.primaryTitle as [original_type],
    title.isAdult as [is_adult], title.startYear as [start_year], title.endYear as [end_year],
    title.runtimeMinutes as [runtime],
    rating.averageRating as [avg_scores], rating.numVotes as [num_votes]
from [dbo].[title.basics] as title
inner join [dbo].[title.ratings] rating
on title.[tconst] = rating.[tconst]
go

```

Picture 13: VIEW consist of movie information

- При извлечении данных из таблицы или представления в базе данных система будет последовательно искать каждую строку и проверять, соответствует ли она сформированному условию. Это вызовет своевременное выполнение при поиске по огромной таблице. По этой причине мы придумали другой метод поиска - INDEX.
- INDEX - это структура на диске, связанная с таблицей или представлением, которая ускоряет извлечение строк из таблицы или представления. Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в структуре (B-дерева), которая позволяет SQL Server быстро и эффективно находить строку или строки, связанные со значениями ключей.

```

create unique clustered index idx_movie_id
on [dbo].[movie_title_view]([movie_id])
go

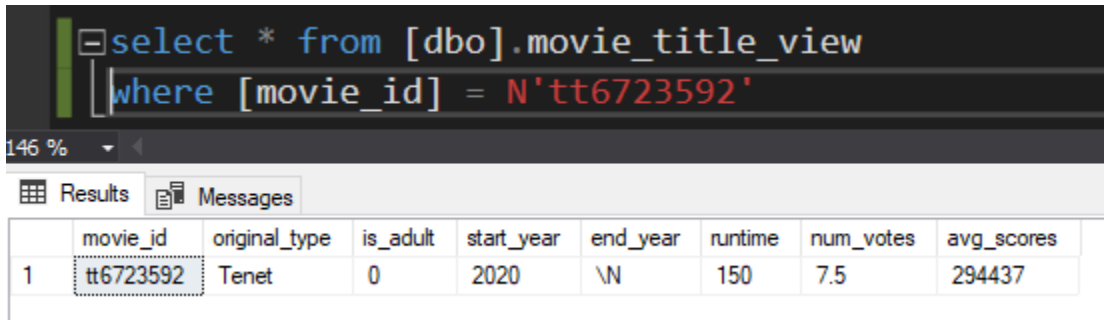
create nonclustered index idx_movie_view
on [dbo].[movie_title_view]([movie_id],[start_year])
include([original_type],[is_adult],[num_votes],[runtime],[avg_scores])

```

Picture 14: UNIQUE INDEX and NONCLUSTERED INDEX on VIEW

- Пример, когда пользователь хочет увидеть дополнительную информацию, щелкнув по нему, мы узнаем идентификатор фильма, который пользователь хочет увидеть, поэтому мы ищем его в нашем VIEW

- Запрос и результат:

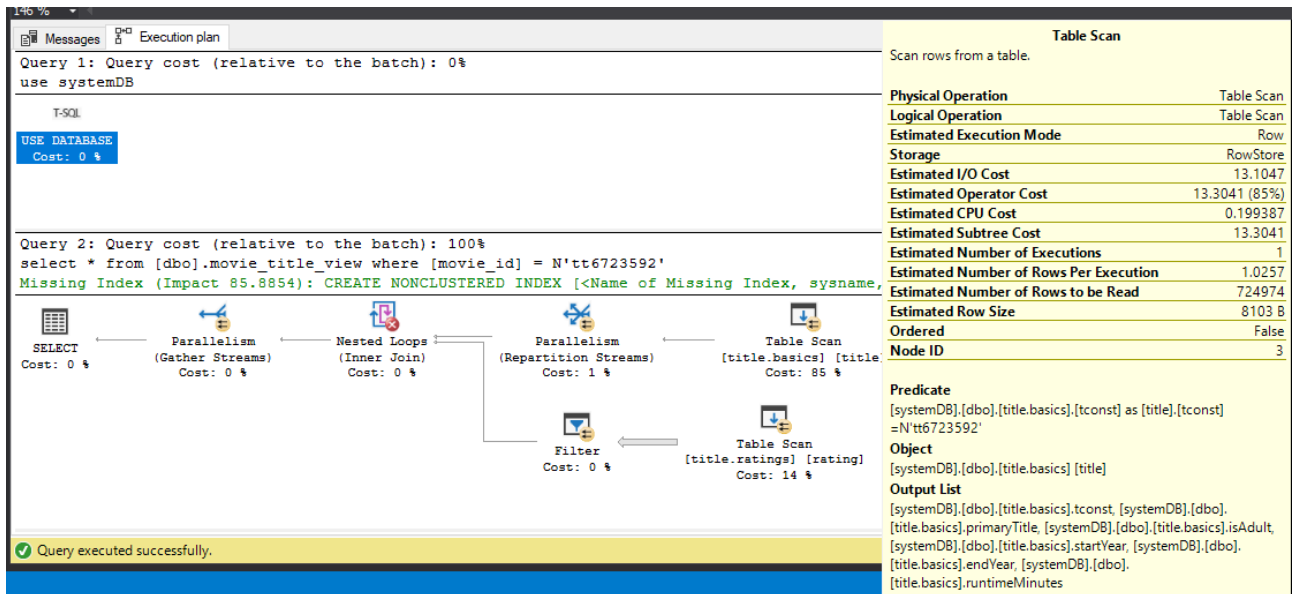


The screenshot shows a SQL query in a query editor window. The query is: `select * from [dbo].movie_title_view where [movie_id] = N'tt6723592'`. Below the query editor, the 'Results' tab is active, displaying a single row of data from the `movie_title_view` table.

	movie_id	original_type	is_adult	start_year	end_year	runtime	num_votes	avg_scores
1	tt6723592	Tenet	0	2020	\N	150	7.5	294437

Picture 15: Query and it's result

- Стоимость оборудования при использовании обычного поиска:





Picture 16: Estimated cost of table scan operation

- И стоимость оборудования при использовании поиска по INDEXES. Что намного лучше обычного поиска

Query 2: Query cost (relative to the batch)

```
select * from [dbo].[movie_title_view] where
```


SELECT
 Cost: 0 %


Clustered Index Seek (ViewClustered)
 [movie_title_view].[idx_movie_id]
 Cost: 100 %


Clustered Index Seek (ViewClustered)
Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.0032831 (100%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0032831
Estimated CPU Cost	0.0001581
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	1
Estimated Number of Rows to be Read	1
Estimated Row Size	8116 B
Ordered	True
Node ID	1

Object
[systemDB].[dbo].[movie_title_view].[idx_movie_id]

Output List
[systemDB].[dbo].[movie_title_view].movie_id, [systemDB].[dbo].[movie_title_view].original_type, [systemDB].[dbo].[movie_title_view].is_adult, [systemDB].[dbo].[movie_title_view].start_year, [systemDB].[dbo].[movie_title_view].end_year, [systemDB].[dbo].[movie_title_view].runtime, [systemDB].[dbo].[movie_title_view].num_votes, [systemDB].[dbo].[movie_title_view].avg_scores

Seek Predicates
Seek Keys[1]: Prefix: [systemDB].[dbo].[movie_title_view].movie_id = Scalar Operator(N'tt6723592')


Query executed successfully.

Picture 17: Estimated cost of index search

2.2. Поиск по списку фильмов

- Используя тот же вид, что и выше, и поиск по индексу, мы также можем получить список данных с низкой задержкой. Пример, когда система находит список рекомендательных фильмов для конкретного пользователя:

```
select title.[movie_id], title.original_type, title.avg_scores
from [dbo].[movie_title_view] as title
inner join [dbo].[recommendation] as recommended
on title.movie_id = recommended.movie_id
where recommended.user_id = N'OHqqlDr2uJm1wp21'
```

146 %

Results
Messages

	movie_id	original_type	avg_scores
1	tt2948372	Soul	176288
2	tt6723592	Tenet	294437
3	tt6751668	Parasite	561589
4	tt8110330	Dil Bechara	111835
5	tt8420184	The Last Dance	80765

Picture 18: Query for recommended movie for individual user

2.3. Поиск знаменитостей, связанных с отдельным фильмом

- Когда пользователь ищет конкретный фильм, отображается не только основная информация, но и отображаются все знаменитости.
- Поскольку у нас была связь между movie_id и Celeb_id, мы можем создать VIEW как обертку информации и запросить ее с помощью поиска по INDEX.
- VIEW содержит фильм и информацию о знаменитостях, связанных с фильмом

```
create view [dbo].[celeb_view] (  
    movie_id, [name], [birth_year], [death_year], [role], [character_name]  
)  
with schemabinding  
as  
select  
    prins.tconst as movie_id,  
    bname.primaryName as [name], bname.birthYear as [birth_year], bname.deathYear as [death_year],  
    prins.category, prins.characters as [character_name]  
from [dbo].[title.principals] as prins  
inner join [dbo].[name.basics] as bname  
on prins.nconst = bname.nconst  
go
```

Picture 19: VIEW consist of celebrity information and their role in specific movie

- Вставить UNIQUE INDEX в VIEW на основе значения [movie_id]

```
create unique clustered index idx_celeb_view  
on [dbo].[celeb_view]([movie_id])  
go
```

Picture 20: UNIQUE INDEX on VIEW

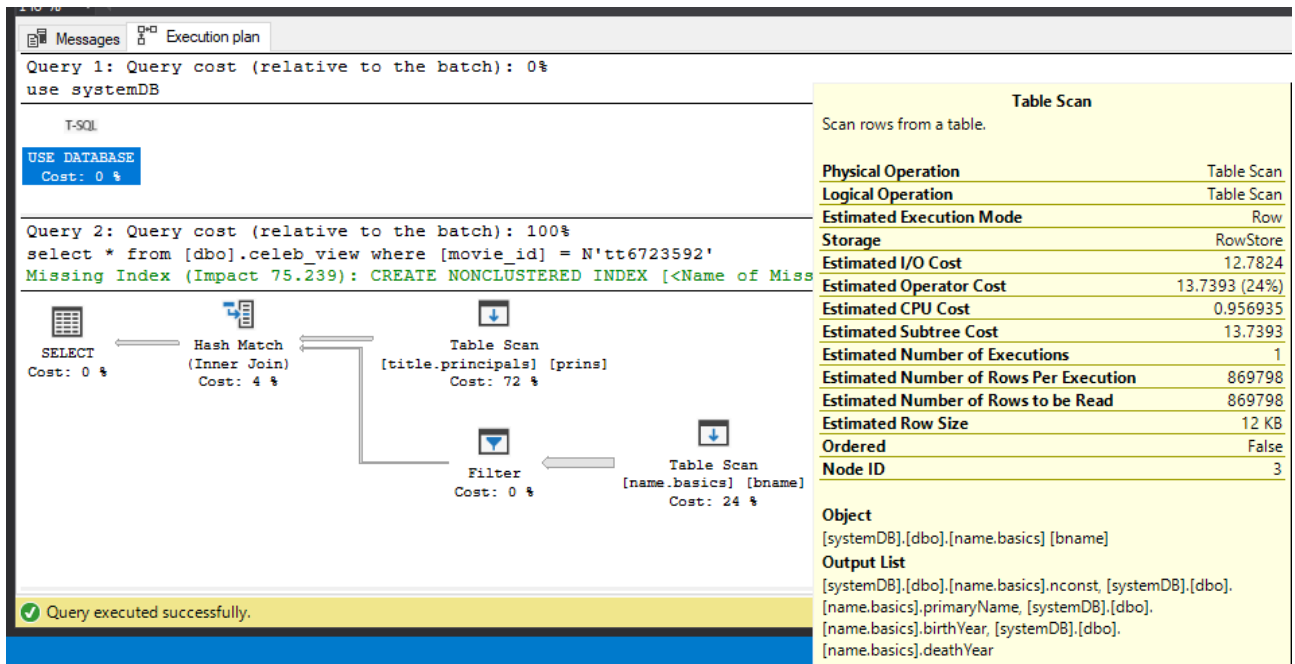
- Запрос, сравнение результатов и производительности между сканированием таблицы и поиском по INDEX

```
select * from [dbo].[celeb_view]  
where [movie_id] = N'tt6723592'
```

	movie_id	name	birth_year	death_year	role	character_name
1	tt6723592	Nathan Crowley	1966	\N	production_designer	\N
2	tt6723592	Hoyte Van Hoytema	1971	\N	cinematographer	\N
3	tt6723592	John David Washington	1984	\N	actor	["Protagonist"]
4	tt6723592	Robert Pattinson	1986	\N	actor	["Neil"]
5	tt6723592	Juhan Ulfhak	1973	\N	actor	["Passenger"]
6	tt6723592	Ludwig Göransson	1984	\N	composer	\N
7	tt6723592	Christopher Nolan	1970	\N	director	\N
8	tt6723592	Emma Thomas	\N	\N	producer	\N
9	tt6723592	Jennifer Lame	\N	\N	editor	\N
10	tt6723592	Elizabeth Debicki	1990	\N	actress	["Kat"]

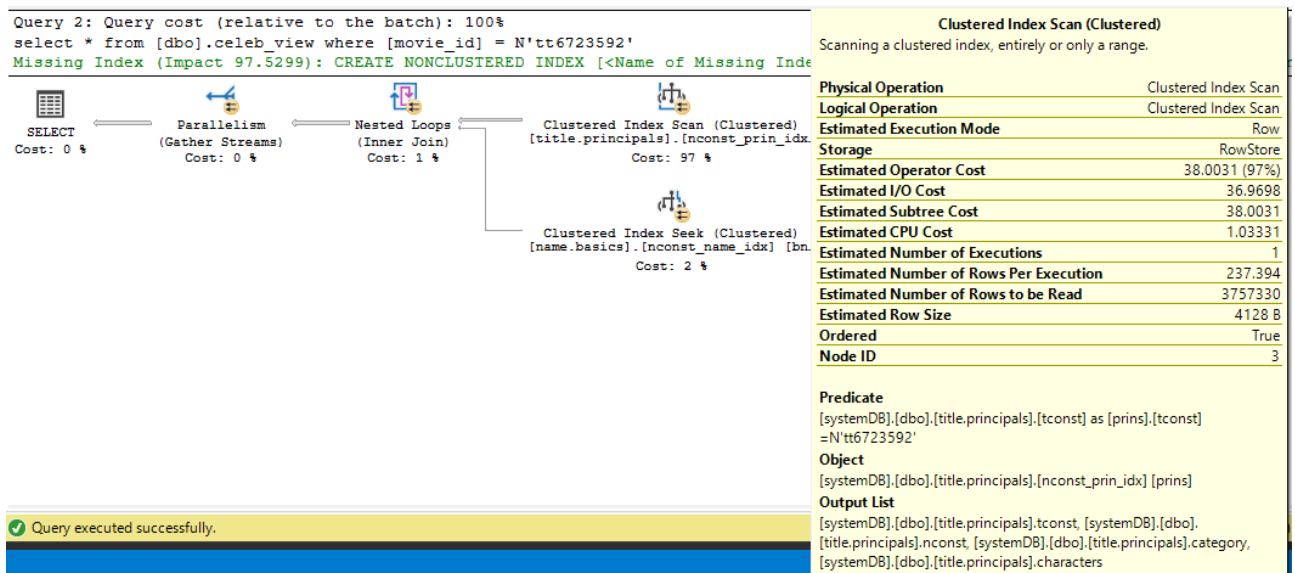
Picture 21: Query for all role in specific movie

- Сканирования таблицы



Picture 22: Estimated cost for scan table

- Поиска по INDEXES



Picture 23: Estimated cost for index search

2.4. Обработка некоторых основных действий пользователя

Поддержка SQL-сервера: мы модифицируем запрос событий с помощью моей собственной процедуры с помощью TRIGGER. Триггер - это особый тип хранимой процедуры, которая автоматически запускается при возникновении события на сервере базы данных.

2.4.1. Когда пользователь регистрирует новую учетную запись

- Когда пользователь хочет зарегистрировать свою учетную запись, сервер обрабатывает запрос и получает необходимую информацию для новой учетной записи, включая имя_пользователя, хешированный пароль (ключ для хранилища хеш-функций на сервере), ... Если [user_name] не существует в базе данных с уникальным значением [user_id] - добавьте его в нашу базу данных, иначе сервер вернет ошибку, чтобы заметить пользователя.
- Процедура триггера вместо обычного запроса вставки

```
create trigger insert_userdata_trigger
on [dbo].[users.data]
instead of insert
as
begin
    if exists(select top 1 i.[user_name]
              from inserted as i
              where i.[user_name]
              in (select [user_name] from [dbo].[users.data]))
    begin
        exec sp_addmessage 50001, 15, N'User_name exist', @lang = 'us_english', @replace='REPLACE';
        RAISERROR(50001,15,-1)
    end
    else
    begin
        insert into [dbo].[users.data] ([user_id], [user_name], [password])
        select CONVERT(nvarchar(100), NEWID()), i.[user_name], i.[password]
        from inserted as i
    end
end
go
```

Picture 24: TRIGGER for INSERT query into users database

- Если требование выполнено

```
insert [dbo].[users.data]([user_id],[user_name], [password]) values (N'',N'dat4', N'dsakgfs31df2132jdfghsd243')
go
```

	user_id	user_name	password
1	0C1ACC5A-DF76-4BCD-87E3-9FB31CCAECF1	dat4	dsakgfs31df2132jdfghsd243
2	9N3n8tk48FsfmX	dat2	654ce6bde3f0baaa4d9a86248cf7e651
3	OHqldr2uJMwpZl	dat1	654ce6bde3f0baaa4d9a86248cf7e651
4	QHmMy5IOe9yDX3C	dat3	654ce6bde3f0baaa4d9a86248cf7e651

Picture 25: Result if insert successfully

- В противном случае верните ошибку

```
insert [dbo].[users.data]([user_id],[user_name], [password]) values (N'',N'dat1', N'dsakgfs31df2132jdfghsd243')
go
```

Msg 50001, Level 15, State 1, Procedure insert_userdata_trigger, Line 13 [Batch Start Line 36]
User_name exist

(1 row affected)

Completion time: 2021-09-14T14:41:19.9736338+03:00

Picture 26: Result if insert fail

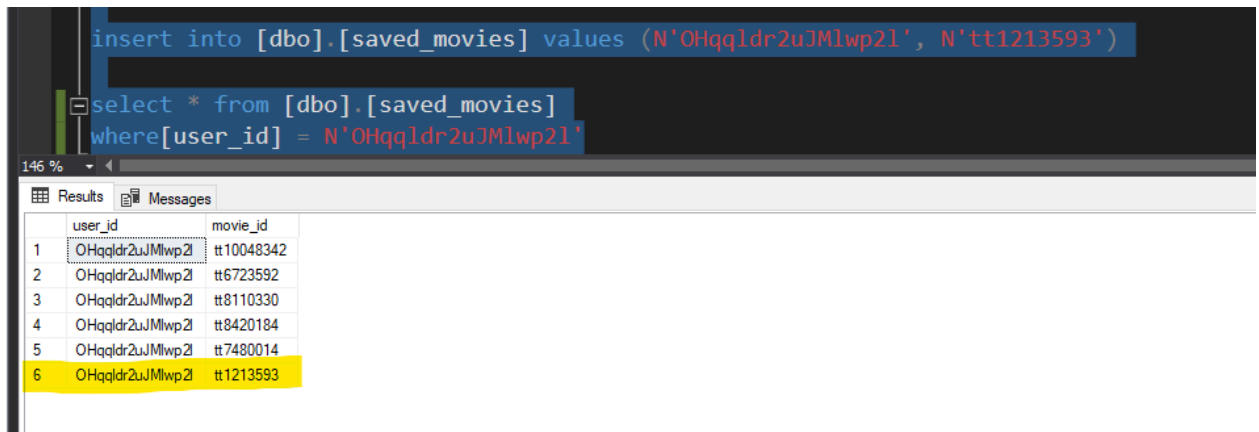
2.4.2. Когда пользователи хотят добавить новый фильм в свои личные категории (понравился, не понравился, сохранен для будущего просмотра, ...)

- Поскольку процедуры очень похожи друг на друга, просто рассмотрим пример, когда пользователь хочет добавить новый фильм в свою категорию для просмотра в будущем.
- Когда пользователь хочет добавить фильм в свой список, нажав кнопку, сервер обрабатывает этот запрос и отправляет в базу данных сигнал - [user_name] хочет добавить [movie_id] в базу данных [saved_movies]
- Если [user_name] еще не добавил [movie_id] в свой список, управление базой данных добавит новые данные и вернется к уведомлению пользователя, которое является его / ее фильмом, успешно добавленным в базу данных. В противном случае, при очень обычном UX-дизайне, руководство удалит [movie_id] из списка фильмов [user_name] и отправит обратно пользователю уведомление, которое [movie_id] было удалено из его / ее списка желаний.
- Процедура триггера вместо обычного запроса вставки

```
create trigger insert_saved_movie_trigger
on [dbo].[saved_movies]
instead of insert
as
begin
    if exists (select top 1 *
              from inserted as i
              inner join [dbo].[saved_movies] as mov
              on i.[user_id] = mov.[user_id] and i.[movie_id] = mov.[movie_id])
    begin
        exec sp_addmessage 50001, 15, N'Movie had been saved', @lang = 'us_english', @replace='REPLACE';
        delete from [dbo].[saved_movies]
        where exists (
            select *
            from inserted as i
            where i.[user_id] = [saved_movies].[user_id]
            and i.[movie_id] = [saved_movies].[movie_id]
        )
    end
else
    begin
        exec sp_addmessage 50001, 15, N'Add movie to saved list', @lang = 'us_english', @replace='REPLACE';
        insert into [dbo].[saved_movies] ([user_id], [movie_id])
        select top 1 i.[user_id], i.[movie_id]
        from inserted as i
    end
end
```

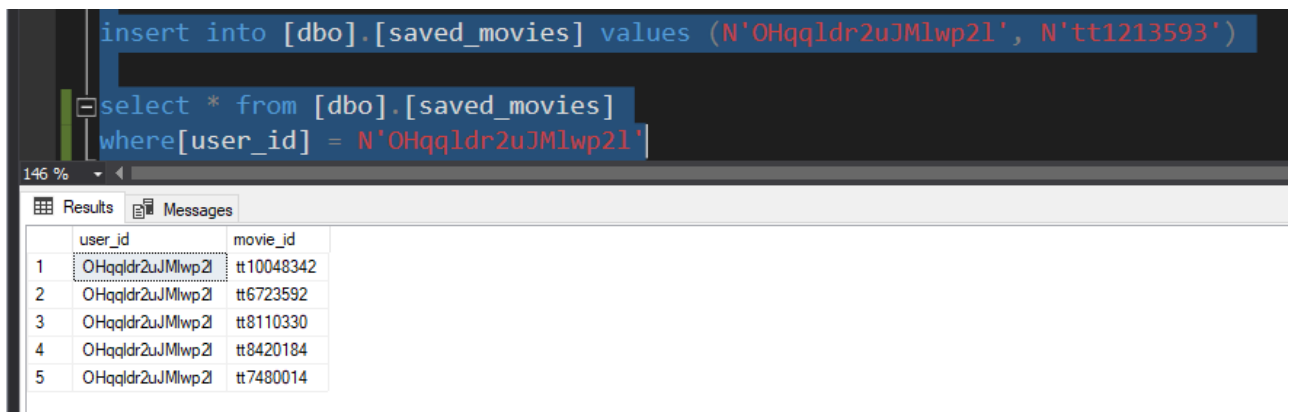
Picture 27: TRIGGER for INSERT query into personal list

- Если фильм добавлен впервые, база данных успешно добавлена в базу данных



Picture 28: Result if insert new one to database

- Если встретились старые данные, удалить их из базы данных



Picture 29: Result if insert data which already in database

VI. Заключение и дальнейшая работа

- Система рекомендаций фильмов - не новая концепция в современной индустрии высоких технологий. Таким образом, многие крупные компании ведут свой бизнес на основе такой системы, например Netflix, IMDB, Из-за ограничений технологии и времени мой проект создавался только в образовательных целях, а не как продукт, способный справиться с огромными рабочими нагрузками. Если мы поместим это в реальные обстоятельства, возникнет много проблем. Такие как:
 - Данные могут быть очень большими, поэтому одна база данных может не соответствовать всем данным. А при использовании нескольких баз данных возникает множество проблем, таких как связь между базами данных, связь между базами данных и сервером, обслуживание базы данных и т. Д.

- При большом количестве клиентов, использующих продукт, одного сервера может быть недостаточно для обработки всех запросов.
- Сложность алгоритма по времени слишком велика и не очень надежна, требуется улучшение, чтобы улучшить взаимодействие с пользователем.
- Текущая система не обладает отказоустойчивостью, поэтому, если произойдет что-то плохое, что приведет к отключению системы, единственный способ - перезагрузить всю систему, что в реальной жизни является очень плохой идеей.
- В течение последних нескольких десятилетий, с появлением множества веб-сервисов, рекомендательные системы занимают все больше и больше места в нашей жизни. От электронной коммерции (предлагать покупателям статьи, которые могут их заинтересовать) до онлайн-рекламы (предлагать пользователям правильный контент, соответствующий их предпочтениям), рекомендательные системы сегодня неизбежны в нашей повседневной работе в Интернете. Они действительно важны в некоторых отраслях, поскольку они могут приносить огромный доход, когда они эффективны, или также быть способом значительно отстать от конкурентов.

VII. Список литературы

- [1] [React – A JavaScript library for building user interfaces \(reactjs.org\)](https://reactjs.org/)
- [2] [Express - Node.js web application framework \(expressjs.com\)](https://expressjs.com/)
- [3] [YouTube Data API | Google Developers](#)
- [4] [API Overview — The Movie Database \(TMDB\) \(themoviedb.org\)](https://themoviedb.org/)
- [5] [IMDb](#)
- [6] [Collaborative filtering - Wikipedia](#)
- [7] [pres1-matrixfactorization.pdf \(hpi.de\)](#)
- [8] [Matrix factorization \(recommender systems\) - Wikipedia](#)
- [9] [Singular Value Decomposition \(SVD\) In Recommender System \(analyticsindiamag.com\)](https://analyticsindiamag.com/)
- [10] [\(PDF\) Understanding and improving relational matrix factorization in recommender systems \(researchgate.net\)](https://researchgate.net/)
- [11] [Mixed Recommender System- MF\(Matrix Factorization\) with item similarity based CF\(Collaborative Filtering\) | by Sourish Dey | Towards Data Science](#)
- [12] [An Efficient Deep Learning Approach for Collaborative Filtering Recommender System - ScienceDirect](#)
- [13] [Gradient Descent: A Quick, Simple Introduction | Built In](#)