ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ

им. проф. М. А. БОНЧ-БРУЕВИЧА»

**Разработка информационной системы по сбору и обработке данных с формированием рекомендаций для поиска видео контента в сети Интернет**

Пояснительная записка к курсовому проекту

по дисциплине

«Технологии безопасности сетевых инфраструктур»

Направление 09.03.02 – Информационные системы

и технологии

Факультет Информационных систем и технологий

Кафедра Безопасности информационных систем

Курс 4

Семестр 8

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Исполнитель

Проверил ст. преп. каф. БИС студ. гр. ИСТ-521

\_\_\_\_\_\_\_\_\_\_\_ Гвоздков И.В. \_\_\_\_\_\_\_\_\_\_\_Коргов А.М.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт Петербург

2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc5618654)

[1 Разработка основных модулей информационной системы онлайн кинотеатра для формирования рекомендаций видеоконтента 4](#_Toc5618655)

[1.1 Проектирование информационной системы 4](#_Toc5618656)

[1.2 Проектирование программной составляющей системы 6](#_Toc5618657)

[1.3 Разработка базы данных 8](#_Toc5618658)

[1.4 Создание сервера web-сервиса 11](#_Toc5618659)

[1.5 Создание главной страницы 12](#_Toc5618660)

[1.5.1 Вывод всех фильмов 12](#_Toc5618661)

[1.5.2 Фильтрация и сортировка 14](#_Toc5618662)

[1.5.3 Поиск 18](#_Toc5618663)

[1.6 Создание страницы рекомендаций пользователей 20](#_Toc5618664)

[1.7 Создание страницы рекомендаций системы: 24](#_Toc5618665)

[1.8 Создание страницы фильма 26](#_Toc5618666)

[ЗАКЛЮЧЕНИЕ 33](#_Toc5618667)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc5618668)

[ПРИЛОЖЕНИЯ 35](#_Toc5618669)

ВВЕДЕНИЕ

В настоящее время Интернет является всё более развитой средой для осуществления коммуникации с потребителями. Благодаря чему он становится удобной и достаточно дешевой альтернативой многим сферам услуг. Кинотеатры не стали тому исключением: вместо того, чтобы тратить время и деньги на походы по кинотеатрам, клиент имеет возможность удобно устроиться в своём любимом кресле с чашкой какао и насладиться любимыми фильмами или посмотреть новинки киноиндустрии.

Однако, чем дальше, тем больше – на сегодняшний день в сети Интернет доступно большое количество онлайн кинотеатров, что порождает огромную конкуренцию между ними. На сегодняшний день одного лишь доступа к фильму или сериалу недостаточно. Многие онлайн кинотеатры позволяют пользователю искать необходимый контент по названию, жанрам, году или стране выхода.

Целью данного проекта является разработка и проектирование информационной системы онлайн кинотеатра, отличительной особенностью которого будет вывод двух типов рекомендаций – рекомендаций пользователей и рекомендаций системы.

Для достижения указанных целей необходимо решить следующие задачи, связанные с разработкой:

* базы данных и осуществление её подключения;
* модуля вывода всех фильмов на главную страницу онлайн кинотеатра;
* модуля фильтрации и сортировки;
* модуля поиска фильмов;
* модулей рекомендаций пользователей и рекомендаций системы;
* страницы фильма.

Данная система будет актуальна для пользователей, которые увлекаются просмотром фильмов и облегчит для них поиск очередного фильма для просмотра.

# Разработка основных модулей информационной системы онлайн кинотеатра для формирования рекомендаций видеоконтента

## Проектирование информационной системы

Разработку информационной системы web-сайта онлайн кинотеатра необходимо начать с проектирования функционала информационной системы. Проектирование – это один из первых этапов разработки информационной системы. Он позволяет определить способы достижения поставленных целей и задач, разработать архитектуру информационного ресурса и его функционал. Данным этапом крайне не рекомендуется пренебрегать, так как он экономит время разработки системы: исправить ошибку на этапе проектирование намного проще, чем на этапе разработки, ведь для этого потребуется только изменить часть диаграмм, а не переписывать структурный код сайта.

В процессе проектирования функционала информационной системы создаётся проектная документация, включающая в себя диаграммы и текстовое описание системы. В рамках данной курсовой работы проектирование будет осуществлено с помощью UML-диаграмм. UML – унифицированный язык моделирования – это графический язык, используемый для визуализации, спецификации, конструирования и документирования сложных систем. Он позволяет описать концепцию разрабатываемой информационной системы, особенности её функционирования, взаимодействие с актёрами системы и самой системы, а также особенности её реализации.

Для создания UML-диаграмм информационной системы используется бесплатный онлайн-сервис draw.io, предоставляющий весь необходимый функционал для создания диаграмм и блок-схем.

Для описания функциональности системы, а также для описания типичных взаимодействий между актёрами системы и самой системой, необходимо построить диаграмму вариантов использования, которую иначе называют диаграммой прецедентов.

Для построения диаграммы прецедентов необходимо определить актёров системы и их функции:

Пользователь:

1. предварительный просмотр фильмов, предоставленных системой;
2. осуществление поиска по названию фильма, режиссёру или актёрам фильма;
3. фильтрация фильмов путём выбора жанров, временного отрезка, в течении которого были выпущены фильмы, а также стран-производительниц фильмов;
4. сортировка фильмов по:
   1. IMDb рейтингу фильмов;
   2. дате выхода фильмов;
   3. просмотрам фильмов;
5. просмотр страницы выбранного фильма;
6. вывод рекомендаций системы;
7. вывод рекомендаций других пользователей системы.

Диаграмма прецедентов информационной системы представлена на рисунке 1.



Рисунок 1 – Диаграмма прецедентов

При входе на сайт пользователь попадает на главную страницу, на которой автоматически выводятся все фильмы, представленные в базе данных сервиса. Он может произвести поиск по названию фильма, имени и фамилии режиссёра или конкретного актёра из актёрского состава. Для вывода фильмов, соответствующих определённым критериям, пользователь может выбрать жанры, которые фильмы будут в себя включать, отрезок времени в годах, за который фильмы были выпущены, а также страны, в которых были выпущены фильмы. Пользователю доступен инструмент сортировки фильмов, который он может настроить по своему желанию: это может быть сортировка по рейтингу IMDb или по дате выхода, выводя фильмы от новых к более старым, или по количеству просмотров фильмов на данном сервисе.

Пользователю доступно два варианта вывода рекомендуемых ему фильмов. Первый вариант учитывает его жанровые предпочтения и выводит фильмы, соответствующие его интересам, отсортированные по рейтингу IMDb. Второй вариант вывода рекомендаций учитывает жанровые предпочтения данного пользователя, а также жанровые предпочтения других пользователей и выводит список фильмов, которые понравились другим пользователям и при этом не были просмотрены данным пользователем.

## Проектирование программной составляющей системы

Web-сервис онлайн кинотеатра реализован на основе шаблона проектирования MVC, изображённый на рисунке 2. Данный шаблон предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три различных компонента: модель, представление и контроллер. Это приводит к тому, что модификация любого компонента осуществляется независимо от остальных.



Рисунок 2 – шаблон проектирования MVC

Компоненты данной системы бывают двух различных видов: внутренние и внешние. Внутренние компоненты будут разработаны в следующих подразделах курсовой работы, когда внешние подключаются извне. Ниже представлен список и краткое описание каждого используемого внешнего компонента:

* body-parser – это связующее программное обеспечение middleware, позволяющий обрабатывать POST запросы, тела application и json запросы, выставляя для них req.body;
* ejs – шаблонизатор, позволяющий получать данные из языка программирования JavaScript и передавать их в HTML-шаблоны. Полученными данными можно оперировать как и с обычными переменными, то есть сравнивать их, выводить и производить над ними какие-либо операции;
* Express – это фреймворк для приложений Node.js. Он разработан с целью создания web-приложений и API и использует для этого множество служебных методов HTTP и промежуточных обработчиков middleware;
* MongoDB – это документноориентированная СУБД с открытым исходным кодом, не требующая описания схемы таблиц;
* Nodemon – это утилита, позволяющая отслеживать любые изменения серверного кода и автоматически перезагружает сервер.

Установка внешних компонентов показана ниже на примере установки “body-parser”:

npm install body-parser --save

## Разработка базы данных

Для реализации обмена информации с базой данных необходимо запустить сервер “mongod”. Разработку базы данных необходимо начать с создания базы данных. Данная команда позволяет создать базу данных с названием “PewpoDB”:

use PewpoDB

Следующим этапом необходимо создать коллекции “films” и “users”, в которых будут храниться документы с данными о фильмах и пользователях:

db.createCollection(“films”)

db.createCollection(“users”)

После создания коллекций их необходимо заполнить документами с информацией о конкретном фильме или пользователе. Ниже представлено заполнение данных о фильме “film1” в коллекции “films”:

db.films.insertOne({

"title" : "film1",

"title\_rus" : "Фильм1",

"img" : "1.jpg",

"year" : 2010.0,

"country" : "США",

"duration" : "2 часа 9 минут",

"producer" : "Гай Ричи",

"views" : 1001.0,

"like" : 115.0,

"genre" : ["боевик", "драма", "ужасы"],

"genreN" : [0, 2, 4],

"kinopoisk" : 7.5,

"rating" : 8.1,

"actors" : [

"Пётр Петров",

"Иван Иванов"

],

"description" : "Шерлок Холмс, несомненно, является величайшим сыщиком, который когда-либо рождался на нашей планете. Обладая высочайшим уровнем интеллекта, потрясающей находчивостью и внимательностью к деталям, британец раскрыл наиболее запутанные дела, поставившие в тупик работников Скотланд-Ярда. Лучшим другом и неизменным соратником главного героя выступает доктор Ватсон, пускай не такой умный и сообразительный, однако помогающий детективу в его стремлении восстановить справедливость. Правда, после того как наш прославленный эскулап надумал жениться, дальнейшее существование легендарного дуэта оказалось под вопросом."

})

Рассмотрим ключи, которые были использованы при создании документа:

* “title” – название фильма на английском языке;
* “title\_rus” – название фильма на русском языке;
* “img” – название и формат изображения;
* “year” – год выхода фильма;
* “country” – страна выхода фильма;
* “duration” – длительность фильма в часах и минутах;
* “producer” – режиссёр фильма;
* “views” – количество просмотров фильма на данном сервисе;
* “like” – количество положительных отзывов;
* “genre” – список жанров, к которым относится данный фильм;
* “genreN” – идентификационный номер каждого жанра;
* “rating” – рейтинг фильма по версии IMDb;
* “actors” – актёрский состав фильма;
* “description” – описание фильма;

На рисунке 3 изображена заполненная коллекция фильмов:

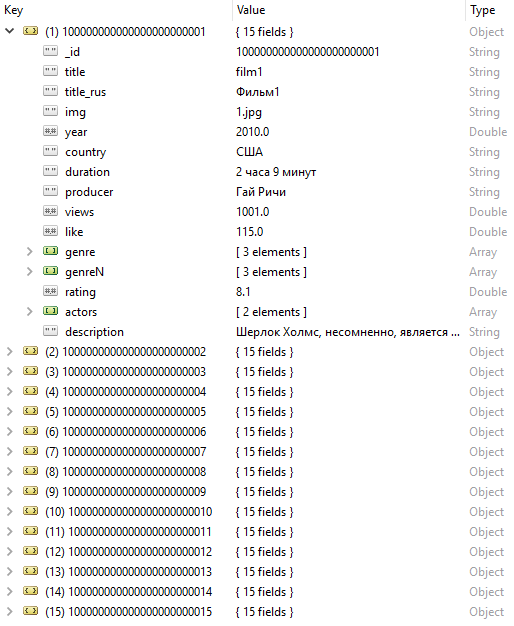


Рисунок 3 – заполненная коллекция фильмов

Создание документа для пользователя “PewpoMan” выглядит следующим образом:

{

"login" : "PewpoMan",

"genre" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

"genrep" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

"view" : [],

"like" : []

}

Рассмотрим ключи пользователей:

* “login” – имя пользователя;
* “genre” – количество каждого жанра, понравившихся пользователю;
* “genrep” – процентное соотношение каждого жанра пользователя;
* “view” – названия фильмов, просмотренных пользователем;
* “like” – названия фильмов, отмеченные как «понравившиеся» пользователем;

## Создание сервера web-сервиса

Сервер web-сервиса находится в корне директории проекта и называется “app.js”. Ниже приведена часть кода с комментариями, отвечающая за подключение к базе данных “PewpoDB” и прослушивание порта проверки работоспособности сайта через браузер:

// MongoClient – ключевой класс, через который проходят все взаимодействия с базой данных:

const MongoClient = require("mongodb").MongoClient;

// создание объекта mongoClient и передача ему строки для подключения:

const mongoClient = new MongoClient("mongodb://localhost:27017/", { useNewUrlParser: true });

// подключение к базе данных

mongoClient.connect(function (err, client) {

// проверяем, успешно ли соединение

if (err) return console.log(err);

dbClient = client

// подключение сайта к базе данных “PewpoDB”

app.locals.db = client.db("PewpoDB");

// прослушивание порта 3000

app.listen(3000, function () {

console.log("Сервер:3000");

});

});

В результате выполнения данных строк кода возможно подключение к сайту через браузер, используя следующий URL: “ <http://localhost:3000>”.

Для отображения файлов, созданных с помощью шаблонизатора ejs, в серверном файле используется следующая команда:

app.set('view engine', 'ejs');

Для того, чтобы подключить статические файлы, среди которых могут быть изображения или стили, находящиеся в папке public, необходимо прописать в файле сервера следующие строки:

app.use('/public', express.static('public'))

app.use('/js', express.static('js'))

## Создание главной страницы

### Вывод всех фильмов

Попадая на главную страницу, пользователь получает вывод всех фильмов из базы данных, отсортированный по рейтингу IMDb. Для этого необходимо в файле index.ejs создать отдельный контейнер с классом “films”, в который будут выведены данные фильмы. При этом необходимо учитывать, что фильмов в базе данных может быть десятки тысяч и, чтобы предотвратить вывод всех фильмов одновременно, необходимо ограничить поток до десяти фильмов за одну итерацию. Для этого создадим контейнер с классом “films-more”. Созданные контейнеры представлены ниже:

<div class="films"></div>

<div id="films-more"></div>

Для вывода всех фильмов используются две функции: “ GetMovies1” и “GetMovies2”. Первая функция отправляет GET запрос на сервер с помощью ajax. После получения массива объектов необходимо записать названия всех выводимых фильмов в массив “allViewFilms” и вызвать функцию “GetMovies2”:

$.ajax({

url: "/movies",

type: "GET",

contentType: "application/json",

success: function (movies) {

$.each(movies, function (index, movie) {

//записываем названия всех выводимых фильмов в allViewFilms

allViewFilms[index] = movie.title;

});

getMovies = movies;

GetMovies2();

}

});

Данная функция дублирует названия фильмов в массив “dontViewFilms”, который хранит в себе названия ещё не выведенных фильмов. Затем используется цикл while, условием выхода из которого является опустошение массива с не выведенными названиями фильмов или достижение переменной значения 10, отвечающей за одновременный вывод фильмов на страницу. Каждая итерация данного цикла записывает в строковую переменную “carts” html код, необходимый для вывода не более десяти фильмов на странице, а также удаляет из массива “dontViewFilms” название записанного фильма:

let dontViewFilms = allViewFilms;

while ((dontViewFilms.length > 0) && (viewFilms < 10)) {

flag = 0;

$.each(getMovies, function (index, movie) {

if (dontViewFilms[0] == movie.title) {

// добавляем полученные элементы

carts += cart(movie);

viewFilms++;

flag = 1;

}

})

if (flag == 1) dontViewFilms.splice(0, 1);

}

Запись производится с помощью функции “cart”, аргументом которой является массив объектов всех фильмов:

let cart = function (movie) {

return "<div class='cart' id='" + movie.\_id + "'>" +

"<a class='title' href='/movie/" + movie.title + "'>" + movie.title\_rus + "</br>" +

"<img src='/public/img/" + movie.img + "' /></a></br>" +

movie.genre + "</br> Рейтинг: " + movie.rating + "</div>";

}

Для того, чтобы пользователь мог отобразить следующие 10 фильмов, необходимо создать кнопку, позволяющую произвести данную операцию. Следующие строки кода записывают в вышеописанные контейнеры фильмы из переменной “carts” и кнопку, по нажатию на которую заново запускается функция “GetMovies2”:

$(".films").append(carts);

$('#films-more').empty();

$('#films-more').append('<button onclick="GetMovies2()"> показать больше </button>');

### Фильтрация и сортировка

Следующим модулем является модуль фильтрации и сортировки. Фильтрация необходима для отображения множества фильмов, отвечающих заданным пользователем критериям. Это позволяет минимизировать затраты времени на выбор необходимого фильма из их большого множества. Сортировка позволяет пользователю отобразить фильмы в порядке убывания рейтинга, даты выхода или популярности.

Для создания фильтрации и сортировки необходимо создать списки на странице, используя html код. Следующий пример отображает списки фильтрации и сортировки и включает в себя по одному экземпляру каждого фильтра и сортировки:

<ul class="filter-sort">

<li class="genres">Жанры

<ul class="subgenres">

<li><input type="checkbox" class="boxGenre" id="g0" value=0 />боевик</li>

</ul>

</li>

<li class="years">Год

<ul class="subyears">

<li><input type="text" class="boxYear" id="year1" placeholder="С"/></li>

</ul>

</il>

<li class="countries">Страны

<ul class="subcountries">

<li><input type="checkbox" class="boxCountry" value="Великобритания" />Великобритания</li>

</ul>

</li>

<li class="sort">Сортировка

<ul class="subsort">

<li><input type="radio" class="boxSort" name="boxSort" value="rating" checked/>Рейтинг</li>

</ul>

</li>

<button id="btn-genre">применить настройки</button>

</ul>

Представленные в базе данных фильмы имеют по три жанра, поэтому необходимо ограничить пользователя в выборе жанров до трёх критериев:

$('.boxGenre:checkbox').click(function () {

if ($('.boxGenre:checkbox:checked').length > 3) {

return false;

}})

Нажатие на кнопку «применить настройки» вызывает функцию, которая запоминает значения фильтрации и сортировки в соответствующие переменные:

let checkedGenre = [];

$('.boxGenre:checkbox:checked').each(function () {

checkedGenre.push($(this).val())

});

for (i = 0; i < checkedGenre.length; i++) {

checkedGenre[i] = Number(checkedGenre[i])

}

let checkedCountry = [];

$('.boxCountry:checkbox:checked').each(function () {

checkedCountry.push($(this).val())

});

let checkedYear = [];

$('.boxYear:text').val(function () {

checkedYear.push($(this).val())

});

for (i = 0; i < checkedYear.length; i++) {

checkedYear[i] = Number(checkedYear[i])

}

let checkedSort = "";

checkedSort = $('.boxSort:radio:checked').val();

Переменные, отвечающие за выбор жанра, временного отрезка и сортировки необходимо передать на сервер с помощью POST запроса, используя ajax. Фильтрация по странам будет реализована позже:

$.ajax({

url: "/movies",

contentType: "application/json",

method: "POST",

data: JSON.stringify({

genreM: checkedGenre,

yearM: checkedYear,

sortM: checkedSort

}),

Переданные серверу переменные подставляются в запрос к базе данных, результатом которого является массив фильмов. Необходимо учесть, что пользователь может выбрать не все критерии фильтрации. Например в том случае, если пользователь не выбрал фильтрацию по временному отрезку, то необходимо установить нижнюю границу, равную “0”, а верхнюю – “2050”. Таким образом временной фильтр не будет учтён:

const checkedYear1 = checkedYear[0]

let checkedYear2 = checkedYear[1]

if (checkedYear2 == 0) checkedYear2 = 2050;

Ниже приведён код, который отправляет один из двух запросов на вывод фильмов. Первый запрос выполняется в том случае, если пользователем выбрана фильтрация по жанрам, иначе – второй:

if (!(checkedGenre.length == 0)) {

db.collection("films").find({ "genreN": { $all: checkedGenre }, "year": { $gte: checkedYear1, $lte: checkedYear2 } }).sort({ [checkedSort]: -1 }).toArray(function (err, movies) {

if (err) return console.log(err);

res.send(movies)

});

} else {

db.collection("films").find({ "year": { $gte: checkedYear1, $lte: checkedYear2 } }).sort({ [checkedSort]: -1 }).toArray(function (err, movies) {

if (err) return console.log(err);

res.send(movies)

})}

В том случае, если в качестве фильтрации была выбрана одна или более стран, то необходимо проверить совпадение каждого полученного фильма на соответствия одной из выбранных стран и, в случае совпадения, записать в массив “countryGenre”:

//отборка N выбранных стран (в countryCheck)

if (checkedCountry.length > 0) {

$.each(movies, function (index, movie) {

flag = 0;

for (i = 0; i < checkedCountry.length; i++) {

if (movie.country == checkedCountry[i]) {

flag = 1;

countryCheck[j] = true;

break;

}

}

j++;

})

//запись выбранных стран в массив counryGenre

j = 0;

console.log('countryCheck', countryCheck)

for (i = 0; i < movies.length; i++) {

if (countryCheck[i]) {

countryGenre[j] = movies[i];

j++;

}

}

console.log('movies', movies)

} else countryGenre = movies;

Затем необходимо в массив “allTitleMovie”, отвечающий за вывод данных фильмов, записать названия полученных фильмов, а также очистить контейнер “.films” и активировать функцию “GetFilterMovie”:

$.each(countryGenre, function (index, movie) {

allTitleMovie[i] = movie.title;

i++;

})

$('.films').empty();

GetFilterMovie();

Назначение функции “GetFilterMovie” схоже с назначением функции “GetMovies2” – вывод не всех фильмов разом, а блоками, используя цикл while, в результате которого в строковую переменную “carts” записываются фильмы в формате html и выводятся блоками до пяти фильмов за один раз.

Все функции, реализованные для главной страницы, расположены в файле index.js.

### Поиск

Следующим разрабатываемым модулем является модуль поиска фильмов по их названию, имени и фамилии режиссёра или актёра из актёрского состава. Данный модуль позволяет пользователю в кротчайшие сроки найти необходимый ему фильм. Поиск необходимо отображать на каждой странице сайта, поэтому необходимо поместить его в отдельный файл search.js и подключать на каждой странице.

Для отображения поисковой строки и кнопки, подтверждающей поиск, необходимо использовать следующий код в шапке сайта:

<input type="text" id="IDsearch" placeholder="Поиск" />

<button id="btn-search">поиск</button>

При нажатии на кнопку «поиск», выполняется скрипт, который записывает в переменную “letSearch” значение текстового поля и отправляет его на сервер методом POST:

$("#btn-search").click(function () {

let letSearch = "", Scarts = "";

$('#IDsearch:text').val(function () {

letSearch += ($(this).val())

});

$.ajax({

url: "/search",

contentType: "application/json",

method: "POST",

data: JSON.stringify({

searchM: letSearch

})

Сервер, после получения переменной letSearch, выполняет запрос к базе данных для поиска совпадения её значения среди названий фильмов и, в случае успеха, записывает результат запроса в переменную “sumSearch”:

db.collection("films").find({ "title": letSearch }).sort({ "rating": -1 }).toArray(function (err, searchTitle) {

if (err) return console.log(err);

//res.send(searchTitle)

if (searchTitle.length > 0)

for (i = 0; i < searchTitle.length; i++)

sumSearch.push(searchTitle[i])

Затем, аналогично поиску по названиям, сервер производит поиск по режиссёру фильма:

db.collection("films").find({ "producer": letSearch }).sort({ "rating": -1 }).toArray(function (err, searchProducer) {

if (err) return console.log(err);

//res.send(searchTitle)

if (searchProducer.length > 0)

for (i = 0; i < searchProducer.length; i++)

sumSearch.push(searchProducer[i])

После завершения поиска по режиссёру, сервер отправляет запрос в базу данных для поиска совпадений по одному из актёров актёрского состава среди всех фильмов и отправляет результат преобразования переменной “sumSearch” обратно:

db.collection("films").find({ "actors": letSearch }).sort({ "rating": -1 }).toArray(function (err, searchActors) {

if (err) return console.log(err);

//res.send(searchActors)

if (searchActors.length > 0)

for (i = 0; i < searchActors.length; i++)

sumSearch.push(searchActors[i]);

res.send(sumSearch);

Получив результат выполнения работы сервера, функция производит запись данного результата в строковую переменную “Scarts” с помощью функции “cart” и выводит её значение в очищенный контейнер “.films”, также очистив контейнер “#films-more”, в котором хранится кнопка «показать больше»:

success: function (sumSearch) {

$.each(sumSearch, function (index, search) {

Scarts += cart(search)})

$(".films").empty();

$("#films-more").empty();

$(".films").append(Scarts);}

## Создание страницы рекомендаций пользователей

Рекомендации пользователей реализуются по следующему алгоритму: в начале система производит анализ жанров пользователя и выделяет его основные жанры, затем поочерёдно проверяет остальных пользователей, рассчитывая модуль разности каждого процента жанра от процента основных жанров пользователя, находит пользователей, жанровое соотношение которых наиболее схоже с первоначальным пользователем и выводит те фильмы, которые они отметили как понравившиеся, но при это не просмотренные пользователем.

Для того, чтобы выделить основные жанры пользователя, необходимо с помощью GET запроса получить из базы данных информацию о всех пользователей, найти в данном массиве объектов запись о текущем пользователе и сохранить массив просмотренных им фильмов в отдельную переменную:

$.ajax({

url: "/users",

type: "GET",

contentType: "application/json",

success: function (users) {

$.each(users, function (index, user) {

//находим запись о текущем пользователе

if (user.login == username) {

//сохраняем просмотренные фильмы пользователя

userView = user.view;

Вторым этапом формирования рекомендаций будет анализ основных жанров пользователей. Для этого необходимо рассчитать среднее значение каждого жанра по следующей формуле:

*1/N\*100%*, где N – количество жанров.

Количество жанров равно 10, поэтому среднее значение каждого жанра равно 10%. Следующие строки кода производят перебор значений всех процентных соотношений жанров пользователя, выделяя и записывая в массив те из них, значение которых выше среднего:

for (i = 0; i < 10; i++)

if (user.genrep[i] >= 10)

genreJ[i] = user.genrep[i];

За все вышеизложенные действия отвечает функция “CheckSort”. На третьем этапе функция “SortUsers” с помощью цикла while, условием выхода из которого является заполнения массива “betterUsers” сотней пользователей. На каждой итерации цикл запускает перебор с помощью $.each массив объектов, содержащий данные о всех пользователях, в котором он проверяет, был бы пользователь уже проверен:

while (betterUsers.length < 100) {

min = 110;

$.each(users, function (index, user) {

let sumOtklon = 0;

let flag = 0;

//проверяем, был ли пользователь уже проверен

for (i = 0; i < betterUsers.length + 1; i++)

if (betterUsers[i] == user.login) {

flag = 1;

break;

}

Если ответ положителен, то проверяется следующий пользователь, однако если пользователь ещё не был проверен, то система вычисляет суммарное отклонение процентного соотношения его жанров от соотношения жанров первоначального пользователя и записывает это значение в переменную “sumOtklon”. Затем она проверяет, является ли это значение минимальным значением суммарного отклонения среди проверенных пользователей. Если это так, то в массив “betterUsers” добавляется запись, содержащая логин данного пользователя:

if (!(user.login == username) && (flag == 0)) {

for (i = 0; i < 10; i++)

if (user.genrep[i] - genreJ[i])

sumOtklon += Math.abs(user.genrep[i] - genreJ[i]);

if (sumOtklon <= min) {

min = sumOtklon;

namemin = user.login;

}

}

})

betterUsers[betterUsers.length] = namemin;

Следующим этапом является выполнение функции “GetLikeMovies1”, которая запоминает в массив “likeFilms” названия фильмов, которые являются рекомендуемыми для пользователя. Рассмотрим подробнее работу данной функции. Получив с помощью GET запроса информацию о пользователях, она запускает цикл while, условием выхода из которого является опустошение массива “betterUsers”. Каждая итерация данного цикла производит перебор полученного массива данных о пользователей с помощью $.each и находит данные о пользователе, имя которого хранится в массиве “betterUsers” с индексом 0. Далее функция удаляет из массива “user.like”, содержащего понравившиеся данному пользователю фильмы, те фильмы, которые первоначальный пользователь уже просмотрел. После того, как все просмотренные фильмы были удалены из массива “user.like”, его значение записывается в массив “likeFilms”. После этого данные о данном пользователе удаляются из массива “betterUsers”. Как только до конца выполнится цикл while, свою работу начинает функция “ GetRecomended1”:

$.ajax({

url: "/users",

type: "GET",

contentType: "application/json",

success: function (users) {

while (betterUsers.length > 0) {

$.each(users, function (index, user) {

if (user.login == betterUsers[0]) {

console.log('login', user.login)

for (ii = 0; ii < userView.length; ii++) {

for (jj = 0; jj < user.like.length; jj++) {

if (user.like[jj] == userView[ii]) {

//удаление просмотренного фильма

user.like.splice(jj, 1);

continue;

}

}

}

for (i = 0; i < user.like.length; i++) {

likeFilms = Array.from(new Set(likeFilms.concat(user.like[i])))

}

};

});

betterUsers.splice(0, 1);

}

GetRecomended();

Функция “GetRecomended” схожа с функцией вывода всех фильмов на главной странице: в переменную “dontViewFilms” записываются понравившиеся пользователям фильмы, хранящиеся в переменной “likeFilms”, после чего функция с помощью GET запроса получает информацию о всех фильмах из базы данных. Следующим этапом она с помощью цикла while, условием выхода из которого является опустошение массива с названиями фильмов “dontViewFilms” и значение переменной “viewFilms”, равной 5. Данная переменная отвечает за количество фильмов, которое будет выведено на страницу за однократный вызов функции. Каждая итерация цикла записывает в переменную “carts” фильмы с помощью функции “cart” и увеличивает значение переменной “viewFilms” на одну единицу. В конце своей работы функция выводит не более пяти рекомендованных фильмов:

while ((dontViewFilms.length > 0) && (viewFilms < 5)) {

flag = 0;

$.each(movies, function (index, movie) {

if (dontViewFilms[0] == movie.title) {

carts += cart(movie);

viewFilms++;

flag = 1;

}

})

if (flag == 1) dontViewFilms.splice(0, 1);

}

$(".films").append(carts);

С помощью кнопки «показать больше», пользователь может ещё раз запустить работу данной функции:

$('#films-more').empty();

$('#films-more').append('<button onclick="GetRecomended()"> показать ещё </button>');

## Создание страницы рекомендаций системы:

Модуль рекомендаций системы осуществляет свою работу по следующему алгоритму: с помощью функции “CheckGenre” определяет четыре жанра пользователя, процентное соотношение которых наибольшее, затем с помощью функции “GetRecomended1” сравнивает жанры пользователя с жанрами фильмов, полученных из базы данных и с помощью функции “GetRecomended2” выводит те фильмы, которые совпадают по трём жанрам из четырёх. Рассмотрим подробнее работу каждой из функций.

При запуске страницы первой начинает свою работу функция “CheckGenre”, которая получает из базы данных информацию о всех пользователях и находит запись о текущем пользователе. Затем с помощью цикла while находит четыре жанра, значение процентного соотношения которых по отношению к дружим жанрам наибольшее и сохраняет в массив “genreJ”. После выполнения цикла функция сохраняет в массив “genres” идентификационный номер соответствующего жанра:

$.each(users, function (index, user) {

nGenres = 0;

//нахождение текущего пользователя

if (user.login == username) {

//нахождение 4х жанров с максимальным значением

while (nGenres < 4) {

max = 0;

//определение жанра с максимальным значением

for (i = 0; i < 10; i++) {

if (user.genrep[i] > max) {

max = user.genrep[i];

nmax = i;

}

}

genreJ[nmax] = max;

nGenres++;

delete user.genrep[nmax];

}

//запись ID жанра

genres = Object.keys(genreJ)

}

});

Следующей начинает свою работу функция “GetRecomended1”, которая сравнивает идентификационные номера выделенных жанров пользователя, хранящиеся в массиве “genres” с номерами жанров каждого фильма из коллекции фильмов базы данных. Если все три жанра фильмов совпадают с жанрами из массива, то название данного фильма сохраняется в массив “likeFilms”. После того, как все фильмы будут проверены, запускается функция “GetRecomended2”:

$.each(movies, function (index, movie) {

flag = 0;

for (i = 0; i < 3; i++)

for (j = 0; j < 4; j++)

if (genres[j] == movie.genreN[i]) {

flag++;

continue

}

if (flag == 3)

likeFilms = Array.from(new Set(likeFilms.concat(movie.title)));

})

GetRecomended2()

Функция “GetRecomended2” производит вывод в контейнер “.films” тех фильмов, жанры которых совпадают с основными жанрами пользователя по тому же принципу, как и функция “ GetRecomended” из предыдущего подраздела «Создание страницы рекомендаций пользователей», поэтому не имеет весомого значения описывать её работу подробно:

while ((likeFilms2.length > 0) && (vivFilms < 5)) {

flag = 0;

$.each(movies, function (index, movie) {

if (likeFilms[0] == movie.title) {

carts += cart(movie);

vivFilms++;

flag = 1;

}

})

if (flag == 1) likeFilms2.splice(0, 1);

}

$(".films").append(carts);

$('#films-more').empty();

$('#films-more').append('<button onclick="GetRecomended2()"> показать ещё</button>');

## Создание страницы фильма

Создание страницы фильма необходимо начать с передачи на неё объекта, содержащим информацию о данном фильме. За это действие отвечает следующая команда, прописываемая в серверном файле app.js:

app.get('/movie/:title', function (req, res) {

const db = req.app.locals.db;

db.collection("films").findOne({"title" : req.params.title},function (err, movie) {

res.render('movie', {movie: movie});

});

})

На странице фильма располагаются две основные функции: просмотр фильма и выставление ему оценки “like”. Рассмотрим работу каждой из них.

Функция просмотра фильма вызывается путём активирования пользователем плеера фильма, имеющего id=“play” с помощью нажатия на него. В начале своей работы функция должна считать название фильма и текущее количество просмотров:

$("#play").click(function () {

const title = $("#title\_eng").text();

let view = Number($("#span\_view").text());

Далее с помощью ajax и GET запроса она производит поиск с помощью цикла for названия фильма в просмотренных фильмах пользователя:

flag = 0;

$.each(users, function (index, user) {

if (user.login == userlogin)

for (i = 0; i < user.view.length; i++)

if (user.view[i] == title) {

flag = 1;

break;

}

})

Если данный фильм не найден, то функция передаёт переменные, содержащие название фильма и имени пользователя на сервер, используя PUT запрос:

if (flag == 0) {

$.ajax({

url: "/movie-view/:title",

contentType: "application/json",

method: "PUT",

data: JSON.stringify({

title: title,

userlogin: userlogin

})

})

Следующим шагом, получив переменные из функции, серверному приложению необходимо найти в коллекции фильмов базы данных документ о просматриваемом фильме и увеличить количество его просмотров “views” на одну единицу, а также добавить в документ текущего пользователя запись о просматриваемом фильме в массиве “view”:

const title = req.body.title;

const userlogin = req.body.userlogin;

//фильм: увеличение просмотров на одну единицу:

db.collection("films").findOneAndUpdate({"title" : title}, {$inc: {"views": 1}});

//пользователь: добавление фильма в просмотренные:

db.collection("users").findOneAndUpdate({"login" : userlogin}, {$addToSet: {"view": title}});

После завершения данных операций, функция продолжает свою работу, изменяя количество отображаемых просмотров на странице фильма на одну единицу:

view++;

$("#span\_view").empty();

$("#span\_view").append(view)

Вторая функция, выполняющая свою работу на странице фильма, увеличивает количество положительных оценок «лайков» на одну единицу при нажатии на соответствующую кнопку, добавляя жанры фильма к жанрам пользователя и рассчитывая новое процентное соотношение каждого жанра по отношению к остальным.

Данная функция срабатывает только при нажатии на кнопку “like”. Первым шагом она считывает название фильма, текущее количество «лайков» и идентификаторы его жанров, приводя их к числовому типу данных:

const title = $("#title\_eng").text()

let like = Number($("#span\_like").text());

genreN[0] = Number($("#genreN0").text());

genreN[1] = Number($("#genreN1").text());

genreN[2] = Number($("#genreN2").text());

Следующим шагом функция проверяет, был ли отмечен данный фильм пользователем ранее. Для этого необходимо получить GET запросом массив данных о пользователях из коллекции базы данных, найти в нём запись о текущем пользователе и, с помощью цикла for найти название фильма в колонке “like”:

$.each(users, function (index, user) {

//выбор записи бд текущего пользователя

if (user.login == userlogin) {

//поиск выбранного фильма среди понравившихся

for (i = 0; i < user.like.length; i++) {

if (user.like[i] == title) {

//фильм уже был отмечен пользователем

flag = 1;

break;

}

}

}});

В случае, если фильм не был отмечен пользователем, необходимо передать в серверное приложение с помощью PUT запроса переменные “title” с названием фильма, “userlogin” с именем пользователя, genreN с массивом идентификаторов жанров и переменную “status” имеющую значение «1» в случае, если пользователь ещё не оценивал фильм и значение «-1» в обратном случае:

if (flag == 0) {

//фильм не был отмечен пользователем

const status = 1;

$.ajax({

url: "/movie/:title",

contentType: "application/json",

method: "PUT",

data: JSON.stringify({

title: title,

userlogin: userlogin,

genreN: genreN,

status: status

})

});

Соответствующее серверное приложение начинает свою работу с инициализации полученных переменных и создании массива с тремя строковыми значениями по типу “genre.N”, где N – идентификационный номер жанра:

const title = req.body.title;

const userlogin = req.body.userlogin;

const status = req.body.status;

const genreN = req.body.genreN;

let genreN2 = ["genre.", "genre.", "genre."];

for (i=0; i<3; i++)

genreN2[i] += genreN[i]

Следующим шагом приложение отправляет три различных запроса в базу данных:

* изменение количества положительных оценок фильма на одну единицу;
* добавление названия фильма в список понравившихся фильмов пользователя;
* изменение количества соответствующих жанров пользователя на одну единицу.

if (status == 1) {

//добавление “лайка” фильму

db.collection("films").findOneAndUpdate({"title" : title}, {$inc: {"like": 1}});

//добавление фильм в "понравившихся"

db.collection("users").findOneAndUpdate({"login" : userlogin}, {$addToSet: {"like": title}});

//добавление жанров в понравившиеся

db.collection('users').findOneAndUpdate({"login" : userlogin}, {$inc: {[genreN2[0]]: 1, [genreN2[1]]: 1, [genreN2[2]]: 1}})

}

После каждого изменения количества просмотренных жанров пользователя, необходимо заново рассчитывать процентное соотношения каждого жанра по отношению ко всем остальным:

//расчёт % соотношения каждого жанра пользователя:

db.collection("users").findOne({"login": userlogin}, function(err, user){

let sum = 0;

let newGenreP = [];

for (i=0; i<10; i++) sum += user.genre[i];

for (i=0; i<10; i++) newGenreP[i]=((user.genre[i] / sum)\*100)

Вышеизложенный код производит поиск записи о текущем пользователе, производя следующие операции: с помощью циклов от 0 до 9 в переменную “sum” записывается суммарное значение всех жанров, а затем в массив “newGenreP” с индексом, соответствующим идентификатору каждого жанра, записываются новые значения процентных соотношений:

Последним действием приложение отправляет новые значения процентных соотношений жанров в документ пользователя:

db.collection('users').findOneAndUpdate({"login" : userlogin}, {$set: {"genrep.0": newGenreP[0], "genrep.1": newGenreP[1], "genrep.2": newGenreP[2], "genrep.3": newGenreP[3], "genrep.4": newGenreP[4], "genrep.5": newGenreP[5], "genrep.6": newGenreP[6], "genrep.7": newGenreP[7], "genrep.8": newGenreP[8], "genrep.9": newGenreP[9]}})

sum = 0;

Далее функция продолжает свою работу, увеличивая количество «лайков» на странице на одну единицу:

like++;

$("#span\_like").empty();

$("#span\_like").append(like)

В том случае, если пользователь уже поставил оценку фильму ранее, повторное нажатие на кнопку “like” приведёт к тому, что фильму будет присвоена отрицательная оценка, следовательно, функция должна произвести действия, обратные тем, что были описаны выше:

else {

//фильм уже был отмечен пользователем

const status = -1;=

$.ajax({

url: "/movie/:title",

contentType: "application/json",

method: "PUT",

data: JSON.stringify({

status: status,

title: title,

userlogin: userlogin,

genreN: genreN

})

});

Серверное приложение также должно уменьшить количество положительных оценок фильму на одну единицу, удалить запись о фильме из списка понравившихся фильмов пользователя и уменьшить количество соответствующих жанров так же на одну единицу:

else {

//удаление лайка фильму

db.collection("films").findOneAndUpdate({"title" : title}, {$inc: {"like": -1}});

//удаление фильма из "понравившихся"

db.collection("users").findOneAndUpdate({"login" : userlogin}, {$pull: {"like": title}});

//удаление жанров из понравившихся

db.collection('users').findOneAndUpdate({"login" : userlogin}, {$inc: {[genreN2[0]]: -1, [genreN2[1]]: -1, [genreN2[2]]: -1}})

}

Следующим шагом приложение изменяет значения процентных соотношений жанров в документе пользователя аналогично изменениям, описанным несколькими абзацами ранее.

В конце своей работы, функция изменяет количество положительных оценок на странице фильма на одну единицу меньше:

like--;

$("#span\_like").empty();

$("#span\_like").append(like)

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы была спроектирована диаграмма прецедентов пользователя, описаны основные компоненты разрабатываемой системы, реализована база данных и подключение к ней, а также разработаны основные модули информационной системы онлайн кинотеатра для формирования рекомендаций видеоконтента.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ECMAScript® Language Specification / Ecma International [Электронный ресурс]. URL: http://www.ecma-international.org/ecma-262/5.1/ – (дата обращения: 27.03.2019)
2. Browser Statistics / W3SCHOOLS [Электронный ресурс]. URL: http:// www.w3schools.com/browsers/browsers\_stats.asp – дата обращения: 28.03.2019)
3. MongoDB / MongoDB [Электронный ресурс]. URL: https://docs.mongodb.com/ – дата обращения: 28.03.2019)

# ПРИЛОЖЕНИЯ

Приложение 1 – Проверка на антиплагиат

