Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

## Разработка информационной платформы для кроссжанрового поиска художественных произведений

Студент гр. 3530901/70203 Р.А. Калашников

Санкт-Петербург
2022

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Работа допущена к защите
директором ВШИСиСТ

_____ В.М. Ицыксон

«____» _____ 2022 г.

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

## Разработка информационной платформы для кроссжанрового поиска художественных произведений

по направлению 09.03.01 «Информатика и вычислительная техника»
по образовательной программе
09.03.01_2 «Технологии разработки программного обеспечения»

Выполнил студент гр. 3530901/70203 _____ Р.А. Калашников

Научный руководитель,
к. т. н., доц. _____ В.М. Ицыксон

Научный консультант,
ассистент _____ И.С. Егорова

Консультант по нормоконтролю,
ст. преподаватель _____ С.А. Нестеров

Санкт-Петербург
2022

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

УТВЕРЖДАЮ
Директор ВШИСиСТ

_____ В.М. Ицыксон

«___»_____2021г.

**ЗАДАНИЕ**
**на выполнение выпускной квалификационной работы**

студенту Калашникову Роману Андреевичу, гр. 3530901/70203

1. Тема работы: «Разработка информационной платформы для кроссжанрового поиска художественных произведений»
2. Срок сдачи студентом законченной работы: июнь 2021
3. Содержание работы (перечень подлежащих разработке вопросов):
   - Обзор и анализ существующих информационных платформ;
   - Проектирование информационной системы и системы рекомендаций;
   - Разработка серверной части;
   - Разработка клиентской части;
   - Тестирование информационной системы и системы рекомендаций.
4. Консультант по работе: И.С. Егорова

5. Дата выдачи задания: «___»_____2021г.

Руководитель ВКР _____ В.М. Ицыксон

Задание принял к исполнению «___»_____2021г.

Студент _____Р.А. Калашников

## РЕФЕРАТ

На 76 с., 22 рисунков, 1 приложения

VUE.JS, NODE.JS, EXPRESS, РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, ВЕБ-САЙТ, ОДНОСТРАНИЧНОЕ ПРИЛОЖЕНИЕ, АВТОМАТИЗИРОВАННЫЙ ВЫВОД РЕКОМЕНДАЦИЙ

В ходе выпускной квалификационной работы был проведен сравнительный анализ существующих информационных платформ. По его результатам были сформулированы функциональные требования к создаваемой платформе и модель взаимодействия пользователей с ней. В качестве архитектуры для платформы была выбрана трехуровневая архитектура и выбраны средства разработки платформы, а именно: Node.js сервер с использованием Express.js, общающийся с PostgreSQL базой данных в качестве серверной части, и клиент, основанный на Vue.js.

Результатом работы является веб-сайт, позволяющий осуществлять кроссжанровый поиск художественных произведений, добавлять, просматривать и редактировать информацию о них, а также участниках их создания, и получать рекомендации, основанные как на признаках конкретного произведения, так и на портрете конкретного пользователя.

## THE ABSTRACT

76 pages, 22 pictures, 1 appendicies

VUE.JS, NODE.JS, EXPRESS, RECOMMENDATION SYSTEM, WEB-SITE, SINGLE-PAGE APPLICATION, AUTOMATED RECOMMENDATION DISPLAY

In the course of the final qualifying work, a comparative analysis of existing information platforms was carried out. Based on its results, functional requirements for the created platform and a user interaciton model were formulated. As the architecture for the platform, a three-tier architecture was chosen and platform development tools were chosen, namely: a Node.js server using Express.js, exchanging with a PostgreSQL database as a server part, and a client based on Vue.js.

The result of the work is a website that allows you to carry out a cross-genre search for works of art, add, view and edit information about them, as well as participants in their creation, and receive recommendations based both on the characteristics of a particular work and on a portrait of a particular user.

## СОДЕРЖАНИЕ

# СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

| | |
|---|---|
| F-measure | Комбинированная метрика Precision и Recall |
| MAE | Средняя абсолютная ошибка |
| Precision | Доля произведений, порекомендованных пользователю и при этом действительно являющимися ему интересными |
| Recall | Доля произведений, интересных пользователю, которые были предсказаны системой |
| RMSE | Среднеквадратичная ошибка |
| Жанр | Род произведений, характеризующийся форматом самих произведений. Примеры: книга, фильм, видеоигра. |
| Роль | Род деятельности участника произведения. Примеры: режиссер, писатель, актер. |
| Тэг | Идентификатор для категоризации и описания произведений. Например, комедия или драма. |
| Участник | Человек, участвующий в создании произведения в каком-либо виде. Примеры: писатель, режиссер, актер |

# ВВЕДЕНИЕ

При просмотре фильмов или чтении книг, люди часто посещают сайты в поисках дополнительной информации о произведении, которое они прочитали или посмотрели. Там они узнают новые факты, изучают информацию об авторах, оставляют отзывы, участвуют в обсуждениях, отслеживают просмотренные произведения,а также ищут рекомендации. Одними из наиболее популярных сервисов данного типа являются англоязычные Imdb.com, goodreads.com, movielens.org или русскоязычный Кинопоиск.ру.

У всех вышеперечисленных сервисов есть большая особенность: они фокусируются лишь на одном жанре произведения, например, только на книгах или только фильмах. Это позволяет сделать платформу наиболее информативной для конкретного жанра, однако значительно урезает возможности обсуждения и рекомендаций. Помимо этого, пользователь, часто погружающийся в разные жанры произведений, при желании вести список произведений вынужден использовать различные сайты, функциональность которых для него почти не отличается. Таким образом, наиболее популярные решения обладают следующими недостатками:

- Отсутствие кроссжанровости;
- Дублирование функциональности.

Выявленные проблемы могут быть решены при помощи кроссжанровой информационной платформы. Помимо этого преимущество собрания произведений различных жанров на одной платформе состоит, в том числе, в возможности выдачи рекомендаций между жанрами, благодаря чему пользователи могут открыть для себя больше новых вещей. При этом стоит отметить, что наличие рекомендаций между жанрами никак не отменяет возможность формирования реко-

мендаций и в рамках одного жанра, то есть является не заменой а скорее расширением стандартной системы.

При этом необходимо отметить, что в существующих решениях в качестве одной из наиболее важных опций, влияющих на удобство пользовательского интерфейса, является поддержка возможности переходов между страницами произведений и их участников. Согласно опросу пользователей портала «Кинопоиск» [10], одного из крупнейших русскоязычных интернет-сервисов, предоставляющих информацию о произведениях в жанре "кино наиболее часто они посещают страницы фильмов и персоналий, где их основными целями являются: посмотреть фильмографию, найти конкретный фильм, узнать побольше о конкретной персоналии или же выбрать фильм, который можно посмотреть дальше. При этом 74% респондентов ответили, что попадают на страницы персоналий через страницы связанных фильмов, а 63% после этого переходят на страницы других фильмов с тем же участником. На основе рассмотренной информации можно сделать вывод о необходимости представления аналогичных возможностей со стороны разрабатываемой системы.

Целью данной выпускной квалификационной работы является создание кроссжанровой информационной платформы, предоставляющей возможности кроссжанровости для повышения качества опыта пользователей. В рамках выпускной работы будут рассмотрены различные существующие решения, реализующие информационные системы, поддерживающие кроссжанровость, затем на их основе будут сформулированы требования к разрабатываемой информационной платформе, после чего будут рассмотрены существующие алгоритмы выдачи рекомендаций, а затем разработаны веб-сервер и клиент информационной платформы.

# 1. ОБЗОР И СРАВНИТЕЛЬНЫЙ АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

## 1.1. Критерии сравнения

Для оценки и сравнения ранее разработанных решений были сформулированы следующие критерии:

1. Поддержка различных жанров произведений - основной критерий оценки существующих решений, поскольку новизна предлагаемого решения состоит в том, что существующие системы не поддерживают кроссжанровый поиск. Любые платформы, не удовлетворяющие этому критерию, не рассматривались как аналоги, пусть их функциональность и могла быть похожей;

2. Возможность оценивания - минимумом для данного критерия считается возможность поставить оценку «нравится» или «не нравится» произведению. При этом в идеальном виде платформа должна предоставлять возможность не только для выставления оценки в виде конкретного числа, например от 1 до 10, но и для написания детальной рецензии, предоставляя удобные инструменты для этого, предусматривающие минимальное форматирование с возможностью вставки картинок. Данный критерий важен, так как оценивание является одной из основных функций информационной платформы: люди используют платформы как для выражения собственного мнения о произведении, так и для выбора произведения на основе мнений других людей. Также при отсутствии возможностей оценивания значительно уменьшаются возможности части систем рекомендаций, так как составление портрета пользователя становится значительно более сложным;

3. Поддержание связей между произведениями различных жанров

- данный критерий подразумевает, что пользователь имеет возможность осуществлять навигацию произведениями различных жанров. В качестве примера соответствия критерию можно привести следующую ситуацию: в качестве компании по продвижению видеоигры был выпущен фильм, рассказывающий о событий до ее начала. В этом случае на странице игры должна присутствовать ссылка на страницу фильма, с пояснением, чем именно он является для нее. То же самое должно быть реализовано и для фильма. Возможность легкой навигации между произведениями различных жанров, связанных между собой является еще одним преимуществом кроссжанровой платформы, является еще одним преимуществом кроссжанровой платформы, соответственно данный критерий также играет роль при оценке соответственно данный критерий также играет роль при оценке платформы.платформы;

4. Рекомендации между жанрами - данный критерий описывает систему рекомендаций произведений между различными жанрами. Примером может служить следующая ситуация: в качестве рекомендации к фильму, являющемуся экранизацией книги, предлагаются другие книги того же писателя. Данная система может поддерживать 2 режима работы: ручной, на основе пользовательских предложений: пользователи сами предлагаю рекомендации к определенным произведениям, описывая причины такого выбора, а также автоматический: в этом случае система автоматически подбирает рекомендации к произведению на основе различных данных, например на основе людей, задействованных в проекте или на основе тегов произведений;

5. Наличие обсуждений - для этого критерия минимумом является наличие обсуждений хотя бы в каком-либо виде, пусть даже и не обособленных и не объединенных отдельной темой, например

общая лента на главной странице сайта. Сюда относится возможность создавать отдельные обсуждения самостоятельно, возможность привязывать обсуждения к определенным событиям и генерировать их автоматически, наличие инструментов написания комментариев с возможностью форматирования, как и в предыдущем пункте, и механизмом ответов на комментарии других пользователей. Обсуждения также важны, поскольку позволяют людям взаимодействовать друг с другом и вести тематические дискуссии.

Все критерии кроме первого могут иметь разную степень полноты реализации, соответственно при дальнейшем рассмотрении аналогов для каждого критерия будет описываться степень его выполнения.

## 1.2. Анализ существующих решений

Для рассмотрения были выбраны несколько платформ, удовлетворяющих первому критерию - наличие произведений различных жанров. Данные решения были выбраны как наиболее популярные решения, поддерживающие кроссжанровость. При этом за счет несколько разных приоритетов функций эти платформы разнятся между собой. Важно будет упомянуть, что не было найдено подходящей под этот критерий платформы на русском языке, поэтому все представленные платформы являются англоязычными. Для каждой платформы будет отдельно приведен результат оценки по критериям, а также обобщенная характеристика. Стоит также отметить, что все представленные платформы являются достаточно небольшими проектами, что влияет на перечень их возможностей и заполненность данными.

### 1.2.1. tastedive.com

Оценка по критериям:

1. Платформа соответствует первому критерию;
2. Система оценивания на платформе ограничена оценками «понравилось», «не понравилось» и «ничего особенного», а возможность написания текстового отзыва находится лишь на странице составления списков;
3. Система обсуждений на данном ресурсе отсутствует;
4. На данной платформе присутствует автоматическая система рекомендаций, однако рекомендации привязаны к одному жанру, соответственно полноценная поддержка кроссжанровости отсутствует;
5. Система связей между произведениями различных жанров отсутствует.

В целом можно сказать, что основной целью этой платформы является формирование рекомендаций внутри одного жанра произведений, при этом платформа содержит небольшое количество информации о произведениях, на ней отсутствуют обсуждения и используется упрощенная система оценивания.

### 1.2.2. itcher.com

Оценка по критериям:

1. Платформа соответствует первому критерию;
2. Система оценивания предоставляет возможность для оценивания произведения по пятибалльной шкале, а также для написания текстового отзыва с ограничением в 200 символов;
3. Обсуждения присутствуют только для рецензий - можно оставить комментарий к рецензии другого человека;

4. Функциональность системы автоматизированного вывода рекомендаций, которой обладает платформа, можно разделить на 2 категории: рекомендации к произведению и рекомендации для пользователя. Первая не является кроссжанровой и предлагает лишь произведения того же жанра. Вторая позволяет получать рекомендации произведений различных жанров, однако похожесть пользователей, на основе которой генерируются рекомендации, для каждого жанра считается отдельно только по нему, то есть при генерации рекомендаций какого-либо одного жанра, игнорируется информация о произведениях других жанров;

5. Система связей между произведениями различных жанров отсутствует.

В итоге, можно сказать, что данная платформа также нацелена в первую очередь на выдачу рекомендаций, которые изолированы в рамках одного жанра, однако в отличие от предыдущей содержит в себе больше информации о произведениях и более гибкую систему оценивания.

### 1.2.3. listal.com

Оценка по критериям:

1. Платформа соответствует первому критерию;

2. Система оценивания позволяет оставить рейтинг по десятибалльной шкале, а также словесный отзыв;

3. Возможность обсуждения присутствует на различных страницах (например на странице списка произведений или страницы иллюстрации к произведению), а также в рамках отдельного форума, не привязанного к произведениям;

4. Присутствует автоматическая система рекомендаций, однако она ограничена фильмами и сериалами и не является кроссжанровой;

5. Система связей между произведениями различных жанров отсутствует.

Данная платформа фокусируется на создании различных списков, однако ее система рекомендаций не является в полной мере кроссжанровой, хоть платформа и содержит информацию о произведениях разных жанров.

### 1.2.4. rate.house

Оценка по критериям:

1. Платформа соответствует первому критерию;

2. Система оценивания предоставляет возможность для оценки произведения по десятибалльной шкале и написания словесного отзыва;

3. Обсуждение присутствует в виде единой ленты с возможностью отвечать на сообщения других людей на странице произведения или на странице списка, что ограничивает возможности обсуждения конкретных тем, связанных с произведениями, так как, например, при обсуждении двух различных тем комментарии об одной и о другой будут постоянно перемешиваться, мешая ходу дискуссии;

4. Присутствует ручная кроссжанровая система рекомендаций: пользователь может на странице произведения добавить новую рекомендацию к нему или согласиться с уже существующей. Дополнительную информацию о причинах выбора добавить невоз-

можно, то есть пользователь не может понять, почему другой пользователь выдал именно такую рекомендацию;

5. Система связей между произведениями различных жанров отсутствует.

Данную платформу можно назвать наиболее близкой к удовлетворению всех критериев, пусть она в этом и не преуспевает. Наиболее слабыми местами данной платформы являются отсутствие тематических обсуждений и автоматической системы рекомендаций.

### 1.2.5. Сравнение

В таблице на рис.1.1 показан результат анализа существующих решений. Под «+» в таблице понимается присутствие реализации критерия на достаточно высоком уровне, например, для системы оценивания это будет означать наличие шкалы оценивания и возможность написания словесного отзыва, «+/-» означает, что решение удовлетворяет критерию неполностью, а «-» полное отсутствие выполнения критерия.

| | Наличие различных жанров произведений | Система оценивания | Обсуждения | Система рекомендаций | | Система связей |
|---|---|---|---|---|---|---|
| | | | | Ручная | Автоматическая | |
| tastedive | + | +/- | - | - | - | - |
| itcher | + | + | +/- | - | +/- | - |
| listal | + | + | +/- | - | - | - |
| rate.house | + | + | +/- | + | - | - |

Рис.1.1. Результаты сравнения существующих решений

Основываясь на результатах проведенного сравнительного анализа, наиболее успешной можно назвать платформу rate.house . Для со-

ответствия всем критериям ей не хватает автоматической системы рекомендаций, системы связей между произведениями различных жанров и наличия тематических обсуждений.

В целом по таблице можно заметить, что платформы Поддерживают в достаточно урезанном формате системы рекомендаций и связей между произведениями различных жанров.

# 2. ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## 2.1. Формулирование требований

Были сформированы следующие требования к проекту:

- Платформа должна реализовывать стандартные функции информационной системы: выдача информации о произведении или персоналии, поиск;
- Должна быть реализована балльная система оценивания с возможностью написания текстовых отзывов;
- Должна быть реализована система обсуждений с возможностью создавать новые темы, а также автоматической генерацией тем;
- Должна поддерживаться кроссжанровая система рекомендаций: изначально, основывающаяся на вручную добавляемых пользователями связях особого вида между произведениями, далее автоматическая на основе конкретного произведения и профиля конкретного пользователя;
- Должна быть реализована система связей между произведениями разных жанров, имеющих отношение друг к другу;
- Платформа должна обладать русскоязычным интерфейсом.

## 2.2. Варианты взаимодействия с системой

Далее представлены диаграммы прецедентов, описывающие взаимодействие с системой с точки зрения трех различных акторов: пассивного пользователя, активного пользователя и администратора системы. На рис.2.1 и 2.2 представлены диаграммы вариантов взаимо-

действия для двух сущностей, о которых можно найти информацию на платформе: произведений и участников.

С точки зрения активного пользователя, добавляющего новые записи, стандартный алгоритм действий будет следующим:

1. Вход на сайт;
2. Переход на страницу добавления произведения;
3. Заполнение информации о произведении;
4. Выбор участника и его роли из выпадающего списка, либо сначала добавление нового участника.

В свою очередь для администратора стандартным порядком действий будет:

1. Вход на сайт;
2. Переход на страницу панели администрирования;
3. Просмотр списка ожидающих одобрения произведений или участников;
4. Переход на конкретную ожидающую одобрения запись;
5. Одобрение записи с изменением данных или без или удаление ее.

Для пассивного пользователя стандартным порядком действий будет:

1. Вход на сайт;
2. Поиск произведения;
3. Переход на страницу произведения и чтение информации о нем;

4. Выставление оценки произведению;
5. Переход в профиль и просмотр списка рекомендаций, сгенерированного системой.

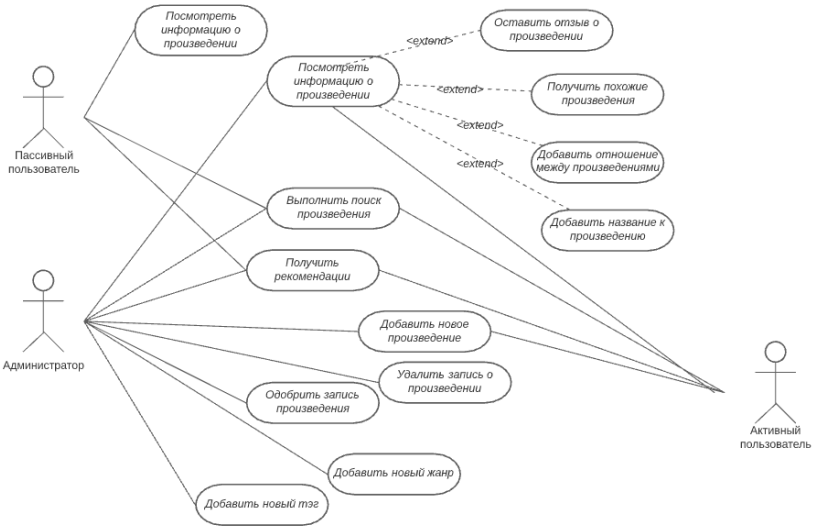


Рис.2.1. Диаграмма прецедентов для произведений

## 2.3. Архитектура проекта и выбор средств разработки

Структура разрабатываемой информационной системы состоит из трех частей:

- База данных;
- Серверная часть;
- Клиентская часть.

Для реализации каждой из частей необходимо подобрать подходящие инструменты.


Рис.2.2. Диаграмма прецедентов для участников

### 2.3.1. Система управления базами данных

База данных будет хранить всю информацию, необходимую для функционирования платформы, в том числе, сведения о произведениях и участниках, различные связи между отношениями. В данном случае одним из определяющих факторов выбора СУБД стало наличие опыта работы с ней, что позволяет эффективно вести работу с самого старта. В качестве используемой СУБД было решено использовать PostgreSQL, подходящий для хранения и анализа большого количества данных.

### 2.3.2. Серверная часть

Для реализации серверной части было решено выбрать Node.js с фреймворком Express.js. В данном случае важным преимуществами Node.js являются неблокирующая обработка I/O, асинхронная обра-

ботка запросов, что помогает быстрее обрабатывать различные запросы от клиентов, в каждом из которых требуется обращаться в базу данных. При этом Node.js не сильно подходит для сложных вычислений, что немного ограничивает список алгоритмов, которые можно использовать для построения автоматических рекомендаций.

В том числе в выборе Node.js сыграл роль и тот факт, что написание обеих частей на одном языке, а именно JavaScript, упрощает работу над проектом, позволяя не переключаться между написанием на разных языках.

Здесь же стоит указать, что для взаимодействия с базой данных было принято решение использовать ORM, так как запросы, которые будут выполняться, возможно без особенных проблем реализовать через функциональность ORM, а удобство написания кода повысится, поскольку будет устранена необходимость часто переключаться с написания javascript кода на SQL-запросы. В качестве ORM-библиотеки была взята популярная js библиотека Sequelize.

### 2.3.3. Клиентская часть

При выборе фреймворка для создания веб-клиента одним из факторов являлась простота освоения, так как у выполняющего работу студента отсутствует большой опыт веб-технологиях. При выборе также учитывалось, что на стороне клиента большая часть кода будет выполнять простые прием и отправление запросов, то есть не понадобится писать какие-либо сложные функции. В итоге выбор пал на Vue.js как на достаточно популярное и доступное в освоении решение [12], позволяющее создавать одностраничные приложения, что позволяет получить высокую скорость работы, адаптированность под мультиплатформенность и хорошую отзывчивость.

# 3. АВТОМАТИЧЕСКАЯ СИСТЕМА РЕКОМЕНДАЦИЙ

Пользователь может искать не только произведение, о котором ему что-либо уже известно, но и что-то новое. В этом случае платформа предлагает рекомендации, которые могут быть сформированы на основе как характеристик произведения, так и профиля пользователя. В случае с рекомендациями на основе произведения в качестве рекомендуемых выбираются такие произведения, у которых совпадает больше всего тэгов или авторов с просматриваемым. В случае же с рекомендациями на основе профиля пользователя будет использована коллаборативная фильтрация. В этом случае выдаются не просто наиболее подходящие произведения, но и предсказанная оценка пользователя для них. При этом должна позволять система выбирать алгоритмы, на основе которых проводится коллаборативная фильтрация, так как в зависимости от числа пользователей и произведений на платформе, а также приемлимого времени расчета рекомендаций разные алгоритмы становятся более выгодными к использованию.

## 3.1. Рекомендации на основе профиля пользователя. Коллаборативная фильтрация

Коллаборативная фильтрация это один из методов построения прогнозов, в рамках данной работы использующийся для генерации рекомендаций. Поиск подходящих произведений для пользователя происходит на основе оценок других пользователей, близких к нему по вкусам. Близость по вкусам оценивается на основе ранее оцененных пользователями произведений. Данный метод генерации рекомендаций считается наиболее популярной и распространенной из техник в рекомендательных системах [7].

Для эффективного использования коллаборативной фильтрации необходимо выполнить три условия [11]:

1. Необходимо участие достаточно большого количества человек
2. Должен существовать простой способ выражения интересов пользователя
3. Алгоритмы должны уметь определять пользователей с похожими вкусами

Так как предполагается, что реализуемая в рамках данной работы платформа будет отвечает всем трем условиям, коллаборативная фильтрация является вполне применимым методом прогнозирования оценок. Так как платформа позволяет участвовать неограниченному числу пользователей, первое условие можно считать выполненным. На платформе присутствует возможность оценить произведение, то есть существует четкое представление интересов пользователя - множество оценок произведений пользователя. Выполнение третьего условия будет достигаться путем рассмотрения различных существующих алгоритмов расчета сходства пользователей и поддержкой их в рамках проекта.

Коллаборативную фильтрацию можно поделить на 2 подхода:

- Основанный на соседстве(memory-based). Исторически первый и более простой подход. При данном подходе для пользователя подбирается группа так называемых критиков, чьи вкусы алгоритм счел похожими, после чего на основе их оценок генерируются рекомендации. Подход можно разбить на три шага:
    1. Расчет сходства пользователей, обычно на основе разницы в оценках произведений, оцененных обоими пользователями;
    2. Выбор нескольких пользователей с наибольшим сходством;
    3. Расчет предсказанных оценок пользователя для произведений, не оцененных им.

Преимуществами подхода являются относительная простота реализации, небольшие требования к ресурсам, а также возможность легко учитывать новые данные, однако его эффективность понижается при большой разреженности данных;

- Основанный на модели(model-based). Более сложный подход , основанный на измерении параметров для статистических моделей пользователей. При разработке моделей часто используются интеллектуальный анализ данных и алгоритмы машинного обучения. Данный подход немного лучше предыдущего осуществляет предсказание оценки, а также справляется с проблемой малого количества данных, но при этом требует большее количество ресурсов: времени и памяти.

Помимо двух основных также выделяют гибридный подход, сочетающий в себе оба вышеупомянутых.

В рамках данной работы будет реализован подход, основанный на соседстве в силу его большей простоты и меньшей трудоемкости, при достаточно неплохой эффективности. Стоит отметить, что возможно также и использование сходства не пользователей, но произведений для предсказания оценок в данном подходе, однако в данной работе такой вариант рассматриваться не будет, а для произведения будут генерироваться лишь простые рекомендации на основе совпадающих тэгов или участников.

Важно отметить, что у данного подхода имеется ряд проблем [6]:

- Проблема разреженности данных. Обычно пользователи оставляют оценки относительно небольшому количеству продуктов, в то время как самих продуктов огромное количество, что приводит к разреженности данных, уменьшающей точность сгенерированных рекомендаций;

- Проблема холодного старта. При начале работы платформы

пользователи еще не успевают поставить достаточно большое количество оценок а значит, системе приходится работать с еще меньшим количеством данных.

В том числе для борьбы с вышеуказанными проблемами в этом подходе подвергаться значительному изменению могут две переменных: алгоритм расчета сходства пользователей и алгоритм предсказания оценок для неоцененных произведений. Далее будут представлены алгоритмы, выбранные для реализации в рамках проектируемой информационной платформы.

## 3.2. Алгоритмы расчета сходства пользователей

Для реализации в рамках проектируемой системы выбраны несколько популярных алгоритмов, а также алгоритм, призванный исправить улучшить работу системы в ситуациях, когда у двух пользователей, для которых рассчитывается сходство, мало произведений, которые они оба оценили, что приводит к понижению точности расчета. Далее кратко будет изложена основная информация о выбранных алгоритмах и их основные характеристики.

### 3.2.1. Расчет сходства пользователей на основе коэффициента корреляции Пирсона

В данном алгоритме [2] при расчете сходства пользователей используется коэффициент корреляции Пирсона. При этом рассматривается не просто зависимость оценки одного пользователя от изменения оценки другого, а зависимость разницы между оценкой пользователя конкретному произведению и средней выставленной им оценкой. Таким образом учитывается тот факт, что пользователи могут иметь разные стандарты: например, кто-то обычно не ставит оценки выше 7, и для него 8 уже выдающееся произведение, а для кого-то 8 нижний порог оценки. Формула для расчета коэффициента:

$$S(x,y)^{PCC} = \frac{\sum_{i \epsilon I_{xy}} (r_{x,i} - \overline{r_x})(r_{y,i} - \overline{r_y})}{\sqrt{\sum_{i \epsilon I_{xy}} (r_{x,i} - \overline{r_x})^2} \sqrt{\sum_{i \epsilon I_{xy}} (r_{y,i} - \overline{r_y})^2}},$$

где $I_{xy}$ - список произведений, оцененных обоими пользователями, $r_{x,i}$ - оценка пользователя $x$ для произведения $i$, $\overline{r_x}$ - средняя оценка, выставленная пользователем $x$.

Основной проблемой этого метода является то, что он достаточно плохо работает при малом числе произведений [14], оцененных обоими пользователями, что приводит к слишком высоким или слишком низким коэффициентам похожести, ведущим к ухудшению качества рекомендаций.

### 3.2.2. Расчет сходства пользователей на основе ограниченный коэффициента корреляции Пирсона

В данном алгоритме [9] отличием от предыдущего является то, что вместо среднего значения оценки пользователя используется медианное значение шкалы оценок. В остальном идея этого алгоритма совпадает с предыдущим.

$$S(x,y)^{CPCC} = \frac{\sum_{i \epsilon I_{xy}} (r_{x,i} - \overline{r_x})(r_{y,i} - r_{med})}{\sqrt{\sum_{i \epsilon I_{xy}} (r_{x,i} - r_{med})^2} \sqrt{\sum_{i \epsilon I_{xy}} (r_{y,i} - r_{med})^2}},$$

где $r_{med}$ - медианное значение шкалы оценок.

Данный метод решает проблему предыдущего метода с выявлением высокого сходства несмотря на различие в оценках, однако все еще страдает от проблемы малого количества совместно оцененных произведений [14].

### 3.2.3. Метод косинуса

В данном алгоритме похожесть пользователей считается как косинусное расстояние [8]. В данном методе пара пользователей определяются как векторы, состоящие из оценок пользователя для произведений, оцененных ими обоими, а их схожесть определяется как косинусное расстояние:

$$S(x,y)^{Cosine} = \frac{\sum_{i \epsilon I_{xy}} (r_{x,i})(r_{y,i})}{\sqrt{\sum_{i \epsilon I_{xy}} (r_{x,i})^2} \sqrt{\sum_{i \epsilon I_{xy}} (r_{y,i})^2}}$$

Данный алгоритм также страдает от проблемы малого количества совместно оцененных произведений, помимо чего может выдавать высокую похожесть, несмотря на значительные отличия в оценке [14].

### 3.2.4. JMSD

JMSD, или Jaccard and mean square difference - алгоритм, являющийся модификацией алгоритма Жаккара путем добавления к нему компоненты среднеквадратичного отклонения [1]. Мера Жаккара - бинарная мера сходства, в данном случае считающаяся как отношение количества произведений, оцененных обоими пользователями, к количеству произведений, оцененных этими пользователями в принципе:

$S(x,y)^{Jaccard} = \frac{|I_x \cap I_y|}{|I_x \cup I_y|}$,

где $|I_x \cap I_y|$ - количество произведений, оцененных обоими пользователями, а $|I_x \cup I_y|$ - количество произведений, оцененных этими пользователями в принципе.

Сам по себе метод Жаккара не учитывает значения оценок для произведений, что можно исправить, добавив компоненту, рассчитываемую на основе разницы оценок, выставленных пользователями, для произведений из $I_x \cap I_y$:

$S(x,y)^{MSD} = 1 - \frac{\sum_{i \epsilon I_{xy}} (r_{x,i} - r_{y,i})^2}{|I_{xy}|}$

Таким образом данный метод учитывает и пропорцию совместно оцененных произведений, и разницу в оценках между двумя пользователями:

$S(x,y)^{JMSD} = S(x,y)^{MSD} * S(x,y)^{Jaccard}$,

Этот метод однако страдает от использования только локальной информации при подсчете похожести, а также использования не всех оценок пары пользователей, а лишь для тех произведений, которые оценили оба пользователя [14].

### 3.2.5. NUSCCF

NUSCCF(Neighbor Users by Subspace Clustering on Collaborative Filtering) - метод поиска соседних пользователей, основанный на кластеризации подпространств [5]. В данной работе будет использована немного упрощенная его версия.

1. Построение матрицы оценок;
2. Создание пользовательских списков произведений по категориям "заинтересован "не заинтересован"и "безразличен";
3. Построение подпространств произведений - наборов произведений, являющихся общими между списками разных пользователей. Например, если у одного пользователя есть список интересных произведений $(i_1, i_2, i_4)$, а у второго $(i_1, i_2, i_3)$ подпространством интересных произведений здесь будет $(i_1, i_2)$;
4. Устранение избыточности;
5. Построение деревьев соседей с помощью подпространств;
6. Расчет похожести пользователей.

Деревья соседей имеют два уровня: прямые соседи пользователя и прямые соседи прямых соседей пользователя. Это позволяет считать похожесть и для пользователей, не имеющих общих оценок. Для прямых соседей похожесть считается с помощью PCC, а для непрямых используются следующие формулы:

$sim(x, y) = \frac{\sum_p \omega * L}{\sum_p \omega}$,

где p - комбинация w и L между прямыми соседями $u_x$ и $u_a$ и прямыми соседями $u_y$ и $u_a$, при этом пользователь $u_a$ является промежуточным между $u_x$ и $u_y$.

$L_{ij} = \frac{\alpha^* |I_i \cap I_j|}{|I_i \cup I_j|}$

$L$ является модификацией метода Джаккарда с добавлением коэффициента $\alpha^*$ для усиления эффекта совместно оцененных произведений.

$w_{ij} = 1 - \frac{\sum_{x \epsilon I_{ij}} (r_{i,x} - r_{j,x})^2}{\beta^* |I_{ij}|}$

$w$ является модификацией MSD с добавлением коэффициента $\beta^*$ для усиления эффекта совместно оцененных произведений.

### 3.3. Алгоритмы предсказания оценок

После вычисления похожести пользователей необходимо отобрать определенное количество наиболее близких по вкусам к определенному пользователю и на основе их оценок составить список рекомендаций или предсказать оценки для неоцененных произведений. Далее представлены используемые для этого в работе алгоритмы.

#### 3.3.1. Взвешенная сумма

Простой метод, учитывающий оценки похожих пользователей и коэффициент их похожести, где рейтинг предсказанный рейтинг :

$P_{x,i} = \frac{\sum_{y \epsilon G_{x,i}} (r_{y,i} * s(x,y))}{\sum_{y \epsilon G_{x,i}} (s(x,y))}$,

где $x$ - пользователь, для которого создаются рекомендации, а G - набор соседей пользователя.

#### 3.3.2. Скорректированная взвешенная сумма

Модификация взвешенной суммы с добавлением средних оценок пользователей. Считается по формуле:

$P_{x,i} = \overline{r_x} + \frac{\sum_{y \epsilon G_{x,i}} ((r_{y,i} - \overline{r_y}) * s(x,y))}{\sum_{y \epsilon G_{x,i}} (s(x,y))}$,

где $\overline{r_x}$ - средняя оценка, выставленная пользователем по всем произведениям, оцененных им.

Данный метод позволяет учитывать тот факт, что некоторые пользователи постоянно ставят высокие оценки, а некоторые наобо-

рот низкие, считая оценку не напрямую, а через модификацию средней оценки, выставленной пользователем, таким образом предсказывая лишь то, насколько произведение будет отличаться от нее.

У этого и предыдущего алгоритмов также добавлена возможность корректировать предсказание оценки произведений, у которых есть связанные произведения: если пользователь оценил связанное с неоцененным произведение, то его предсказанная оценка будет модифицирована с учетом отличия его оценок для связанных произведений от его средней оценки:

$P^{corr}_{x,i} = P_{x,i} + \frac{\sum_{j \epsilon I_{svyaz}} (r_{x,j} - \overline{r_x})}{\alpha r_{max} * |I_{svyaz}|}$,

где $I_{svyaz}$ - список оцененных пользователем произведений, связанных с произведением $i$, $r_{max}$ - максимальное значение шкалы оценок, используемое для пропорционального уменьшения значения итогового значения, чтобы сильно не влиять на изначальную предсказанную оценку, а $\alpha$ - коэффициент, изменяющийся для увеличения или уменьшения влияния связанных произведений. В работе используется $\alpha = 1$.

Таким образом оценки связанных произведений будут немного влиять на предсказанную оценку, при этом сильно не изменяя результат предсказания системы.

### 3.3.3. TOPSIS

Данный алгоритм основан на методе TOPSIS, считая наиболее правильное для рекомендации произведение по расстоянию от идеального и отрицательно идеального решений [13]. Шаги данного алгоритма:

1. Построить матрицы решений, где альтернативами являются произведения, критериями - соседи пользователя;

2. Нормализовать матрицу решений;

3. Добавить к матрице решений веса, то есть умножить элементы на коэффициент схожести, рассчитанный одним из методов из предыдущей секции;

4. Определить идеальное и отрицательно идеальное решения;

5. Подсчитать дистанции от идеальных решений для каждого решения;

6. Подсчитать относительную близость к идеальному решению по формуле: $C_j^* = \frac{S_j^{'}}{(S_j^* + S^{'j})}$, где $S_j^*$ - расстояние до идеального решения, а $S_j^{'}$ - до негативно идеального

7. Ранжировать список альтернатив на основе относительной близости к идеальному решению. Лучшие решения будут иметь наибольшую относительную близость.

# 4. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ

Полный код серверной части можно найти на GitHub по ссылке `https://github.com/Viatus/vkr_backend`.

## 4.1. База данных

Вся работа с базой данных происходит через Sequelize ORM, в том числе, и ее создание с помощью файлов миграции - файлах с описанием таблиц и их полей, которые с помощью команды $db : migrate$ можно перенести в базу данных. На рис.4.1 показана схема базы данных.

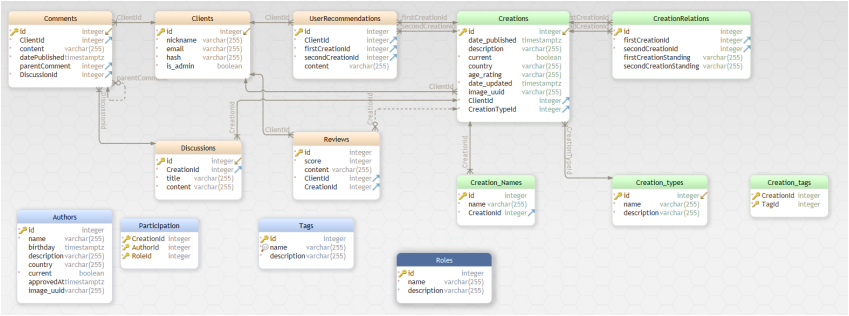


Рис.4.1. Схема базы данных

Были определены следующие сущности:

- Client - схема, описывающая пользователя: либо обычного, либо администратора. Разделение происходит за счет значения поля is_admin. Содержит имя пользователя, адрес электронной почты и хэшированный пароль;
- Creation - схема, описывающая произведение. Содержит столбец current, определяющий, является ли запись о произведении

одобренной администратором, либо находится на рассмотрении. Содержит ссылку на жанр, краткое описание, необязательные столбцы: дата публикации, страна, возрастной рейтинг;

- Authors - схема, описывающая людей, принимавших участие в создании произведения. Эта таблица также содержит столбец current, определяющий, является ли запись об участнике одобренной. Содержит такие поля как имя участника, его краткое описание, необязательно: дата рождения, страна;
- Review - схема, описывающая отзыв о произведении. Содержит оценку и, необязательно, текст;
- Tag - схема, представляющая тэг произведения, нужный для добавления структурированной информации о произведении. Помимо названия содержит описание для пояснения;
- Role - схема, представляющая роль участника в создании произведения;
- UserRecommendation - схема, описывающая двусторонние рекомендации, составленные пользователями вручную для определенного произведения. Помимо ссылок на произведения содержит поле content - текстовое содержание рекомендации;
- Discussion - схема, описывающая обсуждения - его тему, а также заглавный комментарий;
- Comment - схема, описывающая комментарии - их содержание, дату и, если есть, предыдущий комментарий в цепочке ответов;
- CreationRelation - схема, описывающая взаимоотношения между различными произведениями. Помимо ссылок на произведения содержит поля firstCreationStanding и secondCreationStanding, содержащие текстовое описание отношение произведения. Например эти поля могут содержать "сиквел"и "приквел";

- Creation_Type - схема, описывающая жанр произведения;
- Creation_tags - соединительная таблица, отвечающая за принадлежность определенного тэга к определенному произведению;
- Participation - соединительная таблица, описывающая участие участника в каком-либо произведении: содержит идентификатор произведения и автора, а также его роль;
- Creation_Names- схема, представляющая записи о различных названиях одного и того же произведения, необходимые в связи с существованием различных вариантов перевода названий.

Для каждой таблицы создан класс, в котором указаны поля таблицы, а также ее связи с другими таблицами. Рассмотрим на примере таблицы произведений(листинг 4.1). В функции associate описываются связи таблицы, например, так как запись произведения создается пользователем присутствует строка *this.belongsTo(models.Clients);*, а в *Creation.init* описаны поля таблицы и их типы.

Листинг 4.1. Модель проивзедения

```
1  'use strict';
2  const {
3    Model
4  } = require('sequelize');
5  module.exports = (sequelize, DataTypes) => {
6    class Creation extends Model {
7      static associate(models) {
8        this.belongsTo(models.Clients);
9        this.belongsTo(models.Creation_types);
10       this.hasMany(models.Creation_Names);
11       this.belongsToMany(models.Tags, { through: 'Creation_tags' });
12       this.belongsToMany(models.Authors, { through: 'Participation'
             });
13       this.belongsToMany(models.Roles, { through: 'Participation' })
             ;
14       this.hasMany(models.Reviews, { as: "creation_reviews" });
15     }
16   };
```

```
17    Creation.init({
18      CreationTypeId: DataTypes.INTEGER,
19      date_published: DataTypes.DATE,
20      description: DataTypes.STRING,
21      current: DataTypes.BOOLEAN,
22      country: DataTypes.STRING,
23      age_rating: DataTypes.STRING,
24      ClientId: DataTypes.INTEGER,
25      date_updated: DataTypes.DATE,
26      image_uuid: DataTypes.STRING
27    }, {
28      sequelize,
29      modelName: 'Creations',
30    });
31    return Creation;
32  };
```

Работа с таблицами происходит, в основном, через функции *create*() для создания новых записей, *save*() для изменения и *findOne*() и *findAll*() для поиска. В качестве примера можно привести функцию регистрации пользователя(листинг 4.2). В данной функции сначала используется метод *findOne*() для проверки существования пользователя с переданным в запросе аргументом, а затем, если такого пользователя еще не существует хорошо, переданный им пароль хэшируется и создается новая запись пользователя через *create*().

Листинг 4.2. Создание нового пользователя

```
1  const registerClient = async (req, res) => {
2      const { nickname, email, password } = req.body;
3      models.Clients.findOne({ where: { email: email } }).then(async (
           result) => {
4          if (result) {
5              return res.status(StatusCodes.BAD_REQUEST).json({ err: '
                   user already exists' });
6          }
7          const hashedPassword = await bcrypt.hash(password,
               saltRounds);
8
9          try {
10             const client = await models.Clients.create({ nickname:
```

```
                nickname, email: email, hash: hashedPassword,
                is_admin: false });
            return res.status(StatusCodes.CREATED).json({
                client,
            });
        } catch (error) {
            return res.status(StatusCodes.INTERNAL_SERVER_ERROR).
                json({ error: error.message })
        }
    }).catch((err) => {
        return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({
            error: err.message })
    });
}
```

### 4.2. Веб-сервер

Веб-сервер имеет REST-архитектуру.

Основной интерес в проекте представляют файлы двух типов: **Controller.js*(например, *creationController*) и **Route.js*(например, *creationRoute.js*). В файлах первого типа определяются функции API, а во втором указывается путь доступа к ним на сервере. Всего присутствуют 5 групп файлов, разбитых на основе сущностей, с которыми они работают:

- *clientController.js* и *clientRoute.js*;
- *creationController.js* и *creationRoute.js*;
- *authorController.js* и *authorRoute.js*;
- *reviewController.js* и *reviewRoute.js*;
- *discussionController.js* и *discussionRoute.js*.

#### 4.2.1. Контроллер пользователей

В данной секции описываются функции входа и регистрации пользователей, а также подтверждения их подлинности. Для начала

рассмотрим функции из файла *clientController.js*.

*registerClient(req, res)*

Аргументы в теле запроса:

- email - адрес электронной почты;
- password - пароль;
- nickname - имя пользователя.

Метод пытается создать новую запись в таблице клиентов на основе адреса электронной почты, который может использоваться для подтверждения регистрации, никнейма и пароля, который предварительно хэшируется с помощью библиотеки *bcrypt*, что является типовым решением для поставленной задачи. При этом адрес электронной почты должен быть уникальным. Путь на сервере: *'/register'*.

*loginClient()*

Аргументы в теле запроса:

- email - адрес электронной почты;
- password - пароль.

На основе адреса электронной почты и пароля производится попытка авторизации пользователя. В случае успеха в ответ отправляется токен пользователя, сформированный на основе почтового адреса, идентификатора и уровня привилегий, то есть, информацию о том, является ли пользователь администратором. Проверка происходит путем сравнения хэша присланного пароля и хэша, записанного для паролья в базе данных для присланного адреса почты. Путь на сервере: *'/login'*.

Помимо основных функций, являющихся конечными точками, существует одна функция, которая будет вызываться перед выполнением каждого из запросов пользователя после того, как он пройдёт

процедуру авторизации, для расшифровки переданного в заголовке запроса токена. Она выделена в отдельный файл *verifyAuth.js*.

*verifyToken(req, res, next)*

Аргументы в заголовке запроса:

- authorization - токен.

Для создания и проверки токенов используется библиотека *jsonwebtoken*. Эта функция добавляет к телу запроса данные о пользователе для дальнейшей работы с ними.

### 4.2.2. Контроллер произведений

В данном пути находятся функции, отвечающие за операции над произведениями, тэгами, жанрами, связями между произведениями, а также рекомендациями.

Запись о произведении можно добавить с помощью функции *addCreationRecord()*.

Аргументы в теле запроса:

- name - название произведения;
- description - краткое описание произведения;
- date_published - дата публикации произведения;
- country - страна произведения;
- cover - картинка, служащая обложкой произведения;
- creation_type - жанр произведения;
- age_rating - возрастной рейтинг произведения.

Обязательными являются имя, описание, дата публикации и жанр произведения. Путь на сервере: *'/creations'*, операция: *post*.

функции создания доступны и для следующих сущностей, связанных с произведениями и их классификацией. Реализация их является похожей за исключением аргументов запроса:

- Жанр с обязательными аргументами:
  - Название;
  - Описание.
- Тэг с обязательными аргументами:
  - Название;
  - Описание.
- Связанные произведения с обязательными аргументами:
  - Идентификатор первого произведения;
  - Идентификатор второго произведения;
  - Текстовое описание отношения для первого произведения;
  - Текстовое описание отношения для второго произведения.
- Пользовательская рекомендация с обязательными аргументами:
  - Идентификатор первого произведения;
  - Идентификатор второго произведения;
  - Текстовое описание рекомендации.

Для произведения можно также добавить альтернативное название с помощью функции *addNameToCreation()* по пути *'/creation-names/:id'*.

Администратор может одобрить произведение, попутно изменив некоторые его поля - описание, дату публикации, возрастной рейтинг, страну - использовав функцию *approveCreation()* по пути *'/creations-unapproved/:id'*.

Для произведения можно получить связанные произведения с помощью функции *getCreationRelationsForCreation* по пути *'/user-reccomendations/:id'*.

### 4.2.3. Работа с рекомендациями

С помощью функций *getSimilarCreationsOnTagsById()* и *getSimilarCreationsOnAuthorsById()* по путям *'/creations-similar/:id'* и *'/creations-similar-by-author/:id'* соответственно можно получить произведения, с наибольшим количеством совпадающих с заданным тэгов или участников.

Помимо этого можно получить ручные рекомендации от пользователей на основе произведения с помощью функции *getUserRecommendationsForCreation* по пути *'/user-reccomendations/:id'*.

С помощью функции *getRecommendationsForUser()* по пути *'/user-profile-recs'* можно получить рекомендации, основанные на портрете конкретного пользователя. В свою очередь сами рекомендации генерируются с использованием функций *calculateDistances()* и *calculateRecommendedCreations()*.

Перед вызовом этих функций происходит преобразование данных из базы данных в список пользователей с оценками произведениям(листинг 4.3).

Листинг 4.3. Преобразование данных для расчета схожести

```
1     models.Creations.findAll({ where: { current: true }, attributes:
          ['id'] }).then((creations) => {
2         models.Clients.findAll({ include: [{ model: models.Reviews,
              attributes: ['CreationId', 'score'] }], attributes: ['
              id'] }).then((clients) => {
3             var matrix = {};
4             for (cl of clients) {
5                 var clientReviews = {};
6                 for (rev of cl.Reviews) {
7                     let crId = rev.dataValues.CreationId;
8                     let score = rev.dataValues.score;
```

```
9                       clientReviews[crId] = score;
10                  }
11                  matrix[cl.dataValues.id] = clientReviews;
12              }
13              matrixGlobal = matrix;
14              distancesGlobal = calculateDistances(matrix,
                     currentDistanceMethod);
```

*calculateDistances()*, фрагмент которой показан в листинге 4.4, отвечает за расчет похожести пользователей. Среди ее аргументов:

- dataset - матрица пользователей и оценок;
- methodCode - код метода, который будет использоваться для расчета похожести. В самой функции реализованы методы из главы 3.

Листинг 4.4. Расчет схожести на основе одного из методов

```
1     if (methodCode == 5) {
2         for (firstUser in dataset) {
3             distances[firstUser][firstUser] = -1;
4             for (secondUser in dataset) {
5                 if (distances[firstUser][secondUser] === undefined)
                      {
6                     var upperSum = 0;
7                     var firstLowerSum = 0;
8                     var secondLowerSum = 0;
9                     for (creation in dataset[firstUser]) {
10                        if (dataset[secondUser][creation] !==
                               undefined) {
11                            upperSum += dataset[firstUser][creation]
                                   * dataset[secondUser][creation];
12                            firstLowerSum += Math.pow(dataset[
                                   firstUser][creation], 2);
13                            secondLowerSum += Math.pow(dataset[
                                   secondUser][creation], 2);
14                        }
15                    }
16                    let distanceBetweenUsers = upperSum / (Math.sqrt
                          (firstLowerSum) * Math.sqrt(secondLowerSum)
                          );
17                    distances[firstUser][secondUser] =
                          distanceBetweenUsers;
```

```
18                      distances[secondUser][firstUser] =
                            distanceBetweenUsers;
19                  }
20              }
21          }
22      }
```

Данная функция производит расчет схожести пользователей методом, зависящим от кода, переданного ей во входных аргументах, а затем возвращает результат.

Результатом работы этой функции является матрица расстояний между пользователями, которую далее можно использовать для формирования рекомендаций.

Именно этим и занимается функция *calculateRecommendedCreations()*, листинг 4.5, - она выбирает наиболее подходящих соседей, рассчитывает оценки для произведений, неоцененных пользователем и формирует список рекомендуемых произведений для конкретного пользователя. Среди ее аргументов:

- distances - матрица расстояний между пользователями;
- dataset - матрица пользователей и оценок;
- userId - идентификатор пользователя, для которого генерируются рекомендации;
- numberOfCritics - число соседей, на основе которого будут строиться рекомендации;
- numberOfRecs - число выдаваемых функцией рекомендованных произведений;
- boundaryRating - пограничный рейтинг, ниже которого произведения будут считать неинтересными для пользователя;

- methodCode - код метода, используемого для формирования списка рекомендаций. В самой функции реализованы методы из главы 3;
- isUsingRelated - переменная, отвечающая за то, будет ли рейтинг модифицирован с учетом связанных произведений.

Листинг 4.5. Выбор произведений для рекомендации

```
1     let critics = [];
2     for (var user in distances[userId]) {
3         if (user != userId) {
4             if (distances[userId][user] !== undefined) {
5                 if (distances[userId][user] >= 0) {
6                     critics.push([user, distances[userId][user]]);
7                 }
8             }
9         }
10    }
11    critics.sort(function (a, b) {
12        return b[1] - a[1];
13    });
14    critics = critics.slice(0, numberOfCritics);
15    //console.log(critics);
16
17    let recCandidates = [];
18    for (critic of critics) {
19        for (creation of Object.keys(dataset[critic[0]])) {
20            if (!recCandidates.includes(creation) && !Object.keys(
                  dataset[userId]).includes(creation)) {
21                recCandidates.push(creation);
22            }
23        }
24    }
25    //console.log(recCandidates);
26
27    let ratings = [];
28    var avgRatings = {};
29    var avgSquareRatings = {};
30    var userAvgRating;
31    if (methodCode == 3 || methodCode == 4) {
32        for (critic of critics) {
33            var ratingSum = 0;
34            for (creation in dataset[critic[0]]) {
```

```
35                  if (methodCode == 3) {
36                      ratingSum += dataset[critic[0]][creation];
37                  } else {
38                      ratingSum += Math.pow(dataset[critic[0]][
                            creation], 2);
39                  }
40              }
41              avgRatings[critic[0]] = ratingSum / Object.keys(dataset[
                    critic[0]]).length;
42              avgSquareRatings[critic[0]] = Math.sqrt(ratingSum);
43          }
44          var ratingSum = 0;
45          for (creation in dataset[userId]) {
46              ratingSum += dataset[userId][creation];
47          }
48          userAvgRating = ratingSum / Object.keys(dataset[userId]).
                length;
49      }
50      if (methodCode == 1 || methodCode == 2 || methodCode == 3) {
51          for (candidate of recCandidates) {
52              //Average method
53              if (methodCode == 1) {
54                  var totalRating = 0;
55                  var criticsAmount = 0;
56                  for (critic of critics) {
57                      if (dataset[critic[0]][candidate] !== undefined)
                            {
58                          totalRating += dataset[critic[0]][candidate
                                ];
59                          criticsAmount++;
60                      }
61                  }
62                  if (totalRating / criticsAmount > boundaryRating) {
63                      ratings.push([candidate, totalRating /
                            criticsAmount]);
64                  }
65              }
66              if (methodCode == 2) {
67                  var simSum = 0;
68                  var totalRating = 0;
69                  for (critic of critics) {
70                      if (dataset[critic[0]][candidate] !== undefined)
                            {
71                          simSum += critic[1];
72                          totalRating += critic[1] * dataset[critic
                                [0]][candidate];
```

```
73                      }
74                  }
75                  //Чтобы не рекомендовать то, на что шансов нет. Можн
                        о выставить как порог
76                  if (totalRating / simSum > boundaryRating) {
77                      ratings.push([candidate, totalRating / simSum]);
78                  }
79              }
80              if (methodCode == 3) {
81                  var topSum = 0;
82                  var botSum = 0;
83                  for (critic of critics) {
84                      if (dataset[critic[0]][candidate] !== undefined)
                            {
85                          topSum += critic[1] * (dataset[critic[0]][
                                candidate] - avgRatings[critic[0]]);
86                          botSum += critic[1];
87                      }
88                  }
89
90                  let totalRating = userAvgRating + topSum / botSum;
91                  //console.log(totalRating);
92                  if (totalRating > boundaryRating) {
93                      ratings.push([candidate, totalRating]);
94                  }
95              }
96          }
97      }
```

Результатом данной функции является список произведений с, в зависимости от выбранного метода, предсказанной оценкой.

Так как подсчет расстояний между пользователями весьма трудоемкий процесс, и их полное переформирование требуется при каждом внесении новой информации в систему, было принято решение генерировать матрицы расстояния между пользователями и предсказанных оценок не при вызове функции а через интервал времени, заданный в конфигурационном файле .env. Вынесение генерации предсказанных оценок также позволяет без дополнительных вычислений в момент обращения получать их для каждого произведения, помогая пользователю понять, будет ли произведение интересно ему даже без генерации

полного списка рекомендаций.

Методы расчета изначально задаются в конфигурационном файле, а затем метод расчета расстояния можно менять через функцию *changeCalculateDistanceMethod()* по пути *'/rec-method'*.

### 4.2.4. Контроллер участников

В данном пути находятся функции, поддерживающие работу с участниками произведений, ролей, а также участия в создании произведений.

*addAuthor()*

Аргументы в теле запроса:

- name - имя участника;
- description - краткое описание участника;
- birthday - день рождения участника;
- country - страна участника;
- cover - фото участника.

Метод добавляет новую запись об участнике, обязательными параметрами являются имя и описание. Путь на сервере: *'/authors'*.

Аналогичный метод создан и для ролей, где обязательными параметрами являются название и описание, а путь на сервере - *'/roles'*.

С помощью методов *getAuthors()* и *getRoles()* можно получить все записи участников или ролей. Причем для участников можно производить поиск по имени по жесткому совпадению.

Также для участников с помощью функции *getAuthorById()* можно получить подробную информацию по идентификатору. Путь на сервере: *'/authors/:id'*.

Участнику можно добавить роль в произведении с помощью функции *addAuthorRoleInCreation()*, обязательными параметрами для

которой являются идентификаторы участника, произведения и роли. Путь на сервере: *'/author-role'*, метод: *post*.

Далее участие можно получать либо для произведения, либо для участника: с помощью функции *getInvolvedInCreation()* по пути *'creation-role/:id'* можно получить всех участников, участвовавших в создании произведения, а с помощью функции *getAuthorsRoles()* по пути *'author-role/:id'* можно получить все произведения, в которых участвовал автор с его ролями.

### 4.2.5. Контроллер отзывов

Рассматриваемый контроллер поддерживает функции, связанные с добавлением и получением рецензий, а также с подсчетом рейтингов произведений и составлением списков лучших.

*addReview()*

Аргументы в теле запроса:

- score - оценка произведения;
- content - содержание отзыва.

Аргументы в параметрах запроса:

- id - идентификатор произведения, для которого добавляется отзыв.

Метод добавляет новую рецензию к существующему произведению. Обязательными параметрами являются оценка и идентификатор произведения. Перед вставкой новой записи в таблицу происходит проверка, не существует ли уже рецензия на данное произведение от данного пользователя, иначе добавление произведено не будет. Путь на сервере: *'/reviews'*.

*getAllReviews()*

Метод выдает все отзывы, содержание которых не является пустым для вывода на странице произведения. Пустые отзывы все еще учитываются при расчете средней оценки произведения.

*getReviewById()*

Аргументы в параметрах запроса:

- id - идентификатор отзыва.

Метод выдает информацию об отзыве, включая информацию о пользователе и произведении из отызва, по его идентификатору. Путь на сервере: *'/reviews/:id'*.

*getAverageRatingForCreation()*

Аргументы в параметрах запроса:

- id - идентификатор произведения.

Метод выдает средний рейтинг произведения по его идентификатору. Путь на сервере: *'/rating/:id'*.

*getReviewsForCreation()*

Аргументы в параметрах запроса:

- id - идентификатор произведения.

Метод выдает все отзывы о произведении по его идентификатору. Путь на сервере: *'/reviews-creation/:id'*.

*getReviewsByUser()*

Метод выдает все отзывы о произведении по идентификатору пользователя. Путь на сервере: *'/reviews-by-user/:id'*.

*getTopCreations()*

Аргументы в параметрах запроса:

- genres - жанры, среди которых стоит искать произведения;
- tags - тэги, среди которых стоит искать произведения;

- string - строка, которая должна содержать в названии произведения;
- limit - количество произведений в выдаче;
- page - номер страницы списка выдачи.

Метод выдает отфильтрованный список произведений, сортированный по их среднему рейтингу. Путь на сервере: *'/ranking'*.

### 4.2.6. Контроллер обсуждений

Данный контроллер поддерживает функции, связанные с созданием и выдачей обсуждений и комментариев.

*addDiscussion()*

Аргументы в теле запроса:

- CreationId - идентификатор произведения;
- title - Заголовок обсуждения;
- content - содержание обсуждения.

Метод добавляет новое обсуждение к существующему произведению. Обязательными параметрами являются заголовок обсуждения, идентификатор произведения и содержание обсуждения. Путь на сервере: *'/discussions'*.

Аналогично, функция *addComment()* позволяет добавить комментарий.

С помощью функции *getCreationDiscussions* размещенной по пути *'/creation-discussions/:id'* можно получить все обсуждения для конкретного произведения, а с помощью функции *getCommentsForDiscussion* по пути *'/discussion-comments/:id'* можно получить все комментарии для обсуждения.

# 5. РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ

Полный код клиентской части находится на GitHub по ссылке: `https://github.com/Viatus/vkr_frontend`.

## 5.1. Реализация пользовательского интерфейса

Элементы пользовательского интерфейса в проекте делятся на 2 вида: views и components. Первые представляют из себя полноценные страницы, выведенные на определенный путь на сервере, а вторые являются небольшими параметризируемыми элементами, используемыми для отрисовки отдельных элементов, используемых в составе страниц. Для каждого элемента имеется своя разметка, а также свой скрипт, где описываются его основные функции, с помощью которых, в том числе, происходит взаимодействие с веб-сервером. Рассмотрим все страницы проекта, их view и переходы между ними.

### 5.1.1. Страница входа

Страница, отвечающая за вход на сайт. В случае успешного входа перебрасывает пользователя на страницу профиля. Внешний вид показан на рис. 5.1. С помощью кнопки "Зарегистрироваться"можно перейти на страницу регистрации, показанную на рис. 5.2, на которой также есть обратный переход. Стоит отметить, что после регистрации пользователь все равно должен произвести вход в аккаунт.

### 5.1.2. Страница профиля

После входа пользователь попадает на страницу своего профиля, где представлены добавленные им произведения - одобренные и еще только ожидающие одобрения - отзывы, которые он добавил к произведениям, а также персональный список рекомендаций с персонали-

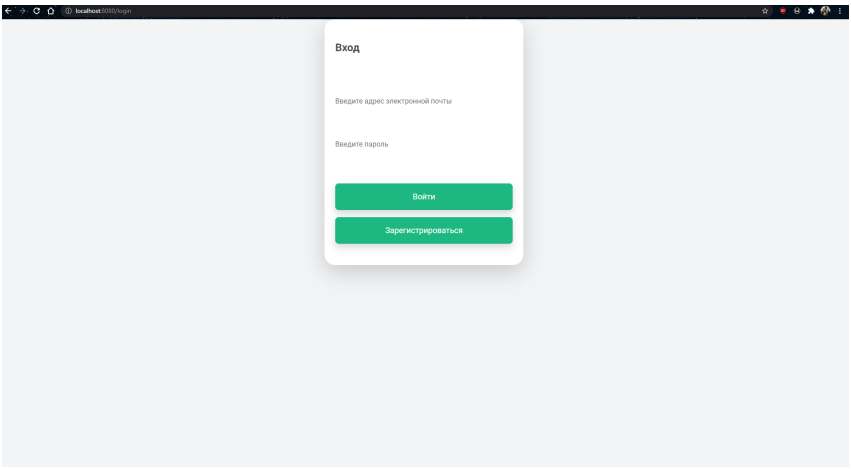


Рис.5.1. Страница входа

зированными для него оценками. Внешний вид страницы показан на рис. 5.3.

### 5.1.3. Страницы добавления

Существуют отдельные страницы добавления произведения и участника, однако при этом участника можно добавить и прямо при добавлении записи произведения. На рис.5.4 показана страница добавления произведения, а на рис.5.5 страница добавления участника. Все обязательные поля должны быть заполнены, в противном случае, возникнет уведомление о невыполнении условий добавления и данные понадобиться исправить. В случае, если при добавлении записи не была прикреплена картинка, при выводе информации на отображение будет использоваться стандартный заменитель.

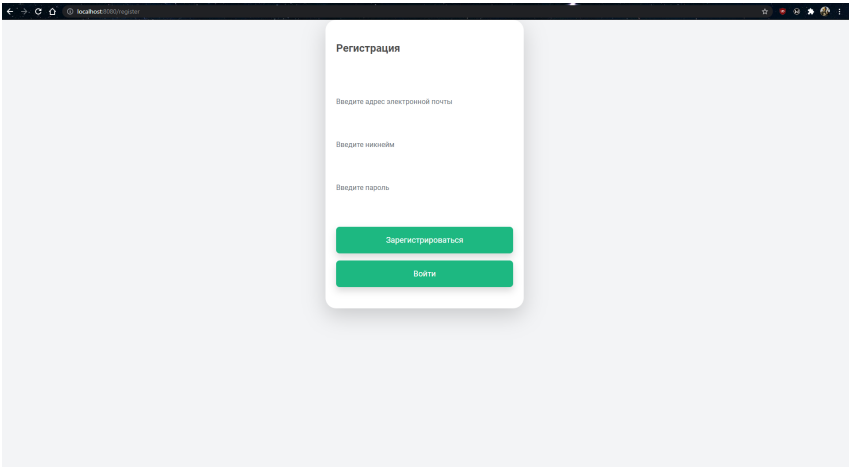


Рис.5.2. Страница регистрации

### 5.1.4. Страница администратора

На странице администратора присутствуют следующие возможности:

- Добавление нового жанра;
- Добавление нового тэга;
- Добавление новой роли;
- Просмотр всех записей произведений, ожидающих одобрения;
- Просмотр всех записей участников, ожидающих одобрения.

На данную страницу попасть может только пользователь с привилегиями администратора, иначе он будет перенаправлен на страницу профиля. Списки ожидающих одобрения записей позволяют переходить на страницы конкретных произведений, ожидающих одобрения. Внешний вид страницы показан на рис. 5.6.

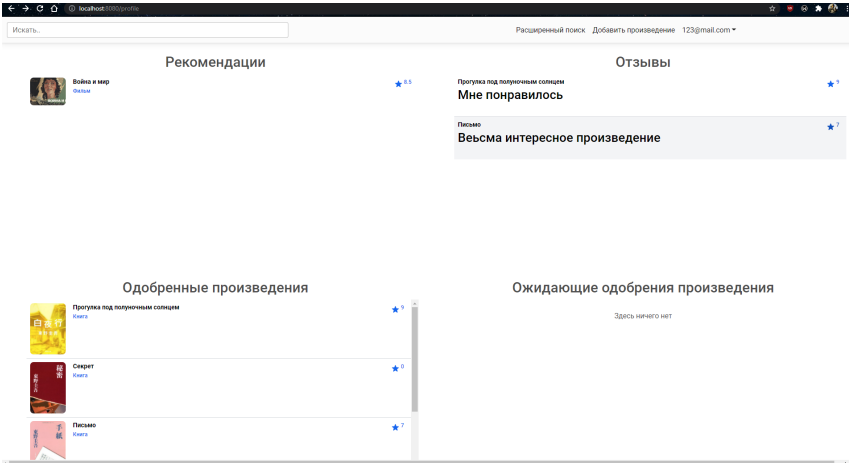


Рис.5.3. Страница профиля

### 5.1.5. Страницы ожидающих одобрения

Страницы ожидающих одобрения записей похожи на страницы создания записей с некоторыми полями заблокированными для редактирования. На такой странице администратор может либо одобрить запись так, как есть, либо внести небольшие изменения в данные и одобрить запись, либо отклонить ее, удалив неодобренную запись. После выполнения одного из двух действий администратор возвращается на экран с ожадиющими одобрения произведениями. Внешний вид страницы для произведения показан на рис.5.7.

### 5.1.6. Страницы с информацией о произведениях и участниках

На информационной странице участника, представленной на рис.5.8, отображается основная информация о нем, а произведения, в которых он принимал участие.

В свою очередь, на странице произведения, помимо кратких све-

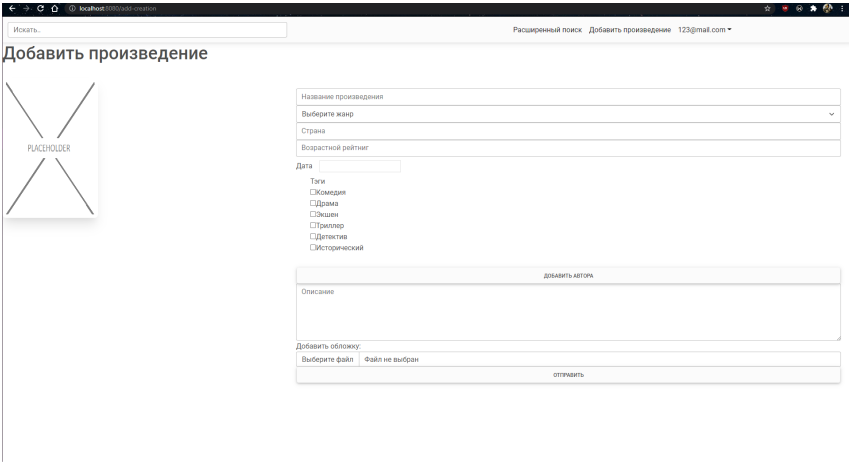


Рис.5.4. Страница добавления произведения

дений о нем, отображается следующее:

- Альтернативные названия произведения;
- Связанные с ним произведения;
- Похожие на основе тэгов произведения;
- Похожие на основе авторов произведения;
- Участники, участвовавшие в создании произведения;
- Рекомендации к произведению от пользователей;
- Список обсуждений, связанных с произведением;
- Отзывы о произведении.

Внешний вид страницы в уменьшенном масштабе показан на рис.5.9.

При этом на странице произведения любой пользователь может:

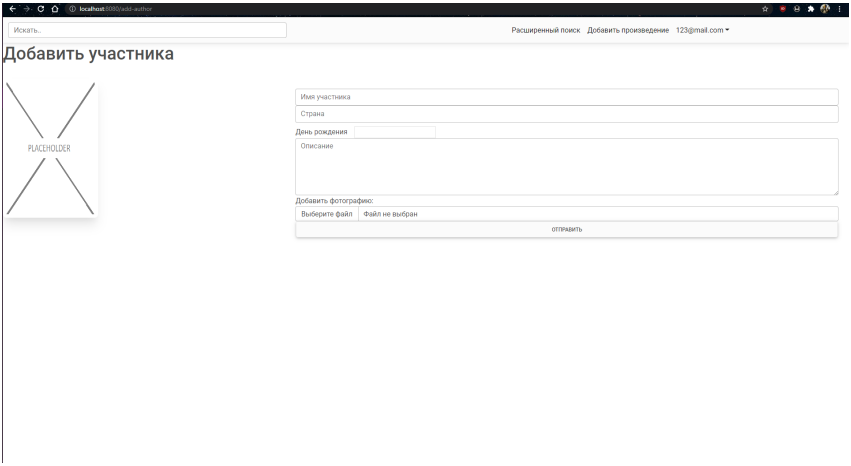


Рис.5.5. Страница добавления участника

- Добавить альтернативное название произведения;
- Добавить участника из уже существующих;
- Оставить отзыв о произведении;
- Создать новое обсуждение;
- Оставить ручную рекомендацию к произведению.

Для всех вышеперечисленных действий используются модальные окна, пример одного такого для добавления ручной рекомендации к произведению показан на рис.5.10.

### 5.1.7. Страница обсуждения

На странице обсуждения отображаются ранее оставленные комментарии, а также имеется возможность оставить новый. Внешний вид страницы показан на рис.5.11.

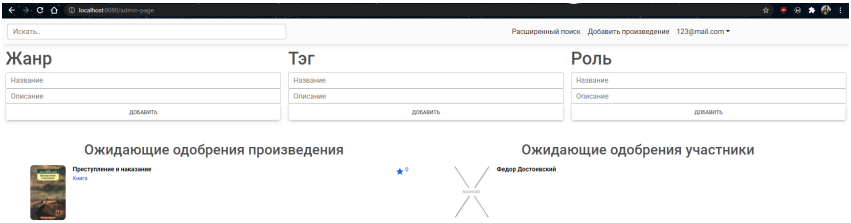


Рис.5.6. Страница администратора

### 5.1.8. Страница поиска

На данной странице можно выполнять расширенный поиск с выбором отдельных жанров и тэгов, размеров страницы выдачи, а также способом сортировки. Внешний вид страницы показан на рис.5.12.

## 5.2. Реализация взаимодействия с API

Для взаимодействия с API разработанного веб-сервиса используется библиотека axios. Данные подгружаются из API по мере необходимости: либо при переходе на страницу, либо после определенных действий пользователя, причем загрузка данных происходит асинхронно, соответственно, можно продолжать работу с приложением, пока оно загружает дополнительные данные. На примере страницы произведения рассмотрим время загрузки данных.

В первую очередь при создании view вызываются различные функции, загружающие множество данных, показанные в листинге 5.1.

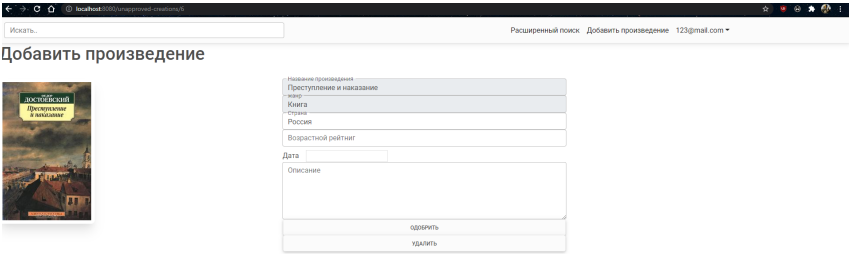


Рис.5.7. Страница ожидающего одобрения произведения

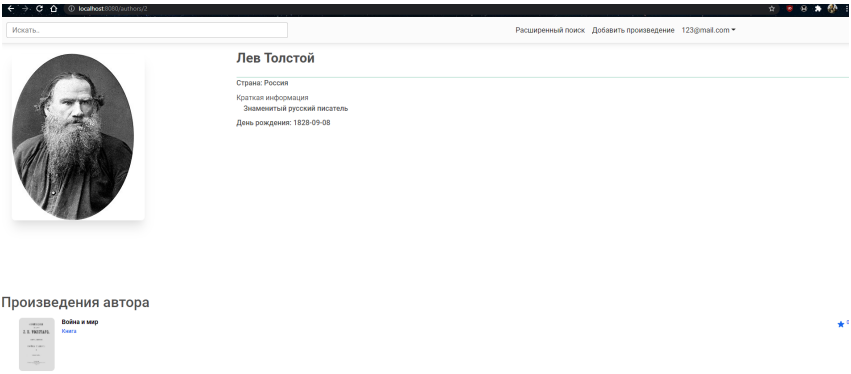


Рис.5.8. Страница участника

Листинг 5.1. Вызов функций сбора данных

```
async created() {
  this.fetchCreationInfo();
  this.fetchCreationTags();
  this.fetchSimilar();
  this.fetchinvolved();
```

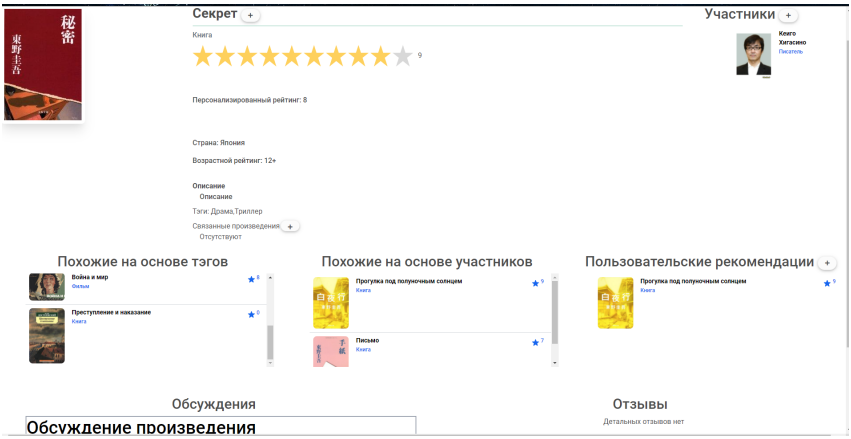


Рис.5.9. Страница произведений

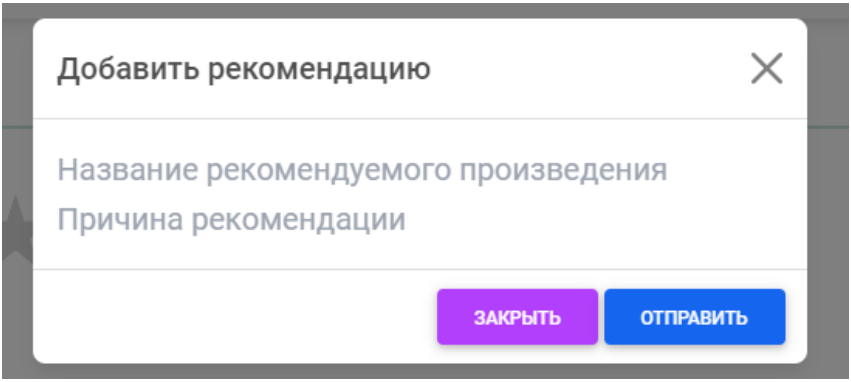


Рис.5.10. Страница произведений


Рис.5.11. Страница обсуждения

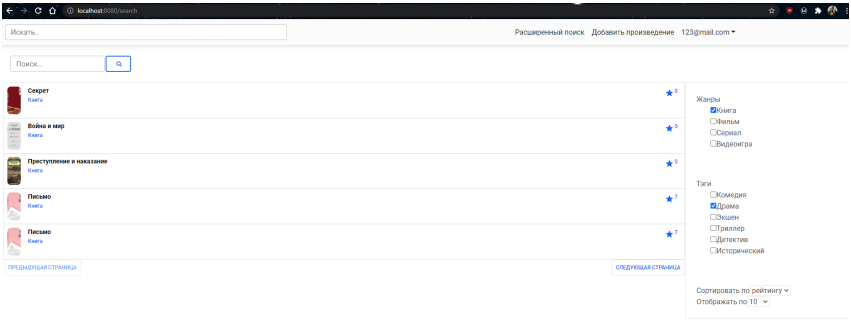


Рис.5.12. Страница расширенного поиска

```
6       this.fetchRating();
7       this.fetchSimilarByAuthors();
8       this.fetchRelatedCreations();
9       this.fetchDiscussions();
10      this.fetchReviews();
11      this.fetchUserRecs();
```

Рассмотрим, например, $fetchSimilar()$, показанную в листинге 5.2. В функции с помощью библиотеки axios производится get-запрос по пути /creations-similar, после чего его результат записывается в переменную.

Листинг 5.2. Сбор похожих произведений

```
1       fetchSimilar() {
2         const fetchedId = this.$route.params.id;
3         axios
4           .get(`${APIURL}/creations-similar/${this.$route.params.id}`,
                )
5           .then((result) => {
6             if (this.$route.params.id !== fetchedId) return;
7             if (result.data.result !== undefined) {
8               this.similar = result.data.result;
```

```
9           }
10        })
11        .catch((error) => {
12          this.$notify({
13            title: "Произошла ошибка",
14            text: error.response.data.error,
15            type: "error",
16          });
17        });
18    },
```

Рассмотрим также другой тип запроса - post-запрос - на примере добавления отзыва(листинг 5.3). В этом случае вся необходимая для оставления отзыва информация собирается в тело запроса, а в заголовок кладется токен, идентифицирующий пользователя. После получения ответа происходит обновление рейтинга произведения.

Листинг 5.3. Сбор похожих произведений

```
1     sendReview() {
2       const fetchedId = this.$route.params.id;
3       const token = localStorage.getItem("token");
4       axios
5         .post(
6           `${APIURL}/reviews/${this.$route.params.id}`,
7           { score: this.choosenRating, content: this.reviewText },
8           { headers: { authorization: token } }
9         )
10        .then(() => {
11          if (this.$route.params.id !== fetchedId) return;
12          this.fetchRating();
13          this.showReviewModal = false;
14          //Оповестить пользователя что все классно
15        })
16        .catch((error) => {
17          this.$notify({
18            title: "Произошла ошибка",
19            text: error.response.data.error,
20            type: "error",
21          });
22        });
```

Весь обмен данными с API происходит похожим образом, отличаются лишь пути и аргументы в теле запроса.

Полный код страницы произведения представлен в Приложении 1.

# 6. ТЕСТИРОВАНИЕ

## 6.1. Тестирование автоматической системы рекомендаций

### 6.1.1. Набор данных и метод тестирования

Для тестирования алгоритмов системы рекомендаций использовался набор данных MovieLens 100K Dataset [4], в котором содержится около 100000 оценок 1700 пользователей для 1000 произведений. В наборе представлено 5 разбиений на базовую и тестовую выборки. В ходе тестирования различные алгоритмы выполнялись на этих 5 разбиениях, после чего их показатели усреднялись. Стоит отметить, что так как никаких данных о связях произведений нет, проверка проводилась для алгоритмов без учета связей.

### 6.1.2. Оценочные метрики

При оценке рекомендательных систем исследователи используют оценочные метрики для их сравнения [3].Эти метрики можно разбить на две группы:

- Точность предсказания;
- Точность классификации.

Точность предсказания определяет, насколько хорошо система предсказывает оценки пользователя. Для определения точности используются две метрики: MAE - средняя абсолютная ошибка, то есть средняя разница между реальной оценкой и оценкой, сгенерированной системой, а также RMSE - среднеквадратичная ошибка модели.

$MAE = \frac{\sum_{i=1}^{MAX} |r_i - \hat{r}_i|}{MAX}$,

где $r_i$ и $\hat{r}_i$ - это настоящий рейтинг из тестового набора и предсказанный системой рейтинг. $MAX$ - число предсказаний. Метрика $RMSE$:

$RMSE = \sqrt{\frac{1}{MAX}\sum_{i=1}^{MAX}(r_i - \hat{r}_i)^2}$

Точность классификации измеряет качество рекомендаций системы. Данная группа оценивает не предсказанные оценки для произведений, а списки рекомендаций. Прежде чем определить метрики этой группы, необходимо определить несколько терминов:

- TP(true positive) - число произведений из тестовой выборки, интересных для пользователя и попавших в список рекомендаций системы;
- FN(false negative) - число произведений из тестовой выборки, интересных для пользователя и не попавших в список рекомендаций системы;
- TN(true negative) - число произведений из тестовой выборки, неинтересных для пользователя и не попавших в список рекомендаций системы;
- FP(false positive) - число произведений из тестовой выборки, неинтересных для пользователя и попавших в список рекомендаций системы.

В данном случае для того, чтобы произведение считалось интересным пользователю, его оценка должна быть больше 2. Данное число было выбрано на основе среднего представления людей об оценке по выбранной шкале - было решено, что понравившееся произведение пользователь оценит не ниже, чем "удовлетворительно"В качестве метрик используются Recall - доля рекомендованных произведений, оказавшихся в тестовом наборе, Precision - отношение количества верно порекомендованных произведений к действительно интересным пользователю произведениям, и F-measure, совмещающую в себе предыдущие две:

$Precision = \frac{TP}{TP+FP}$

$Recall = \frac{TP}{TP+FN}$
$F - measure = \frac{2*(Precision*Recall)}{(Precision+Recall)}$

Каждый из алгоритмов был запущен с 10 соседними пользователями и 10 рекомендациями.

### 6.1.3. Результаты тестирования

Перед приведением результатов тестирования стоит отметить, что для Recall, Precision и F-measure чем ближе значение к единице, тем лучше. Для MAE и RMSE же чем значение меньше, тем лучше. Так как метод TOPSIS выдает ненормированную оценку, для него MAE и RMSE не считаются.

На рис.6.1 приведены значения Recall для разных комбинаций методов.

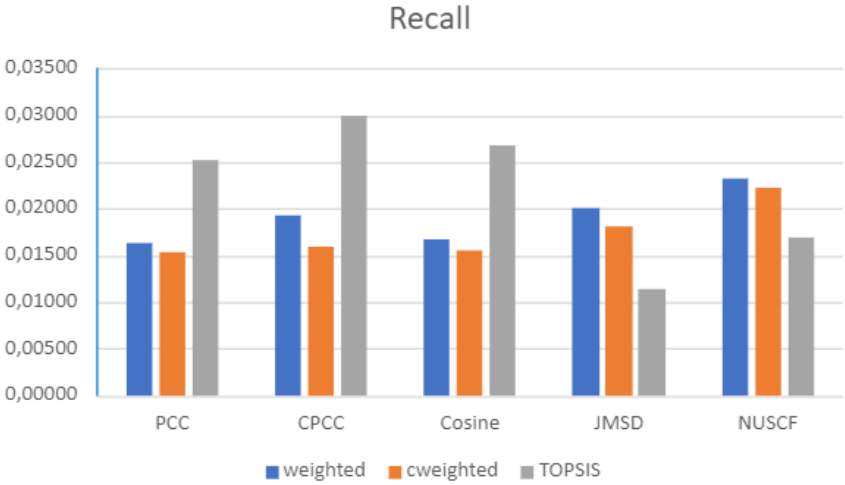


Рис.6.1. График Recall

На рис.6.2 приведены значения Precision для разных комбинаций методов.

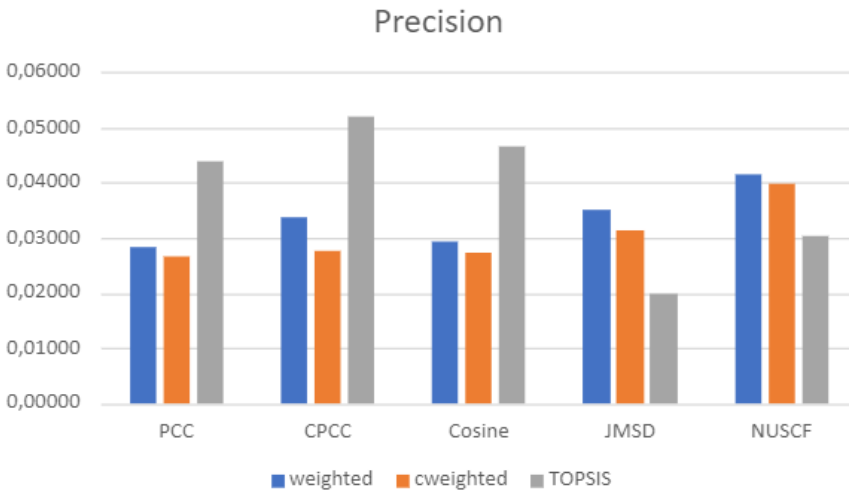


Рис.6.2. График Precision

На рис.6.3 приведены значения F-measure для разных комбинаций методов.

На рис.6.4 приведены значения MAE для разных комбинаций методов.

На рис.6.5 приведены значения RMSE для разных комбинаций методов.

По графикам можно сказать,что с точки зрения Recall, Precision и F-measure наиболее эффективной в текущей реализации является комбинация CPCC и TOPSIS методов. При этом со всеми методами кроме JMSD и NUSCF TOPSIS дает значительный прирост в эффективности работы системы, однако в обмен на это теряется возможность показывать пользователю предсказанную оценку. Это означает, что для генерации списка рекомендаций для пользователя стоит использовать комбинацию CPCC и TOPSIS, а для предсказания оценок использовать CPCC и любой из двух методов генерации рекоменда-

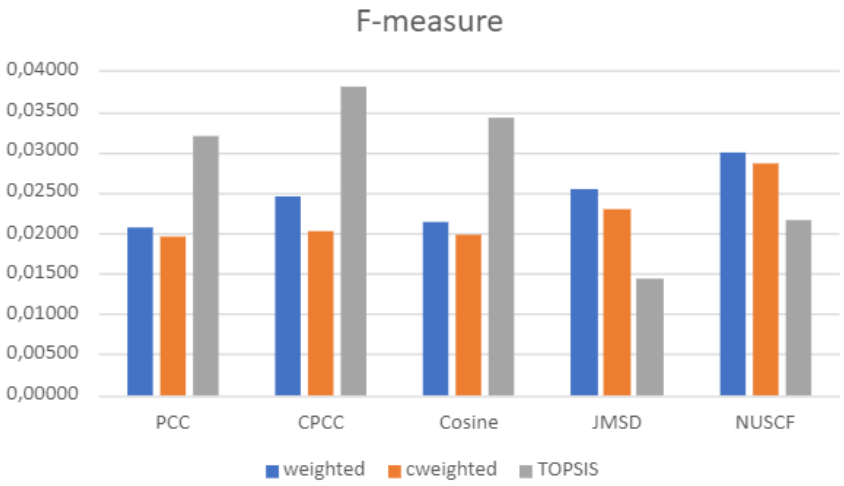


Рис.6.3. График F-measure

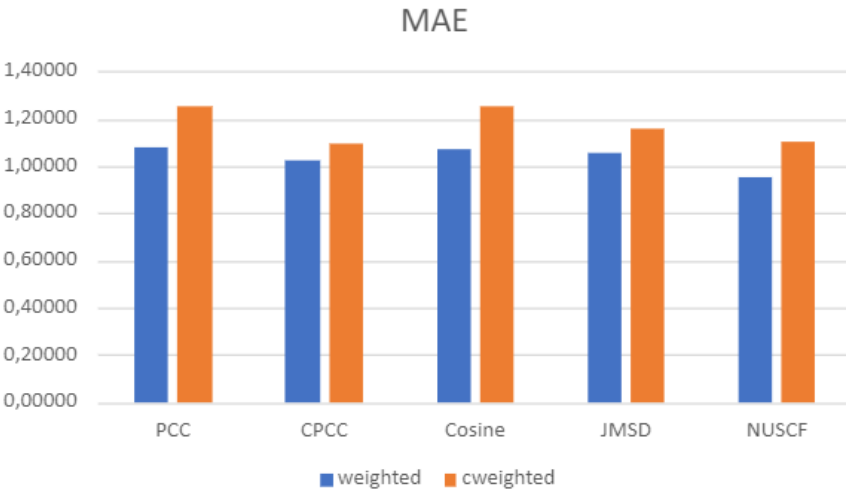


Рис.6.4. График MAE

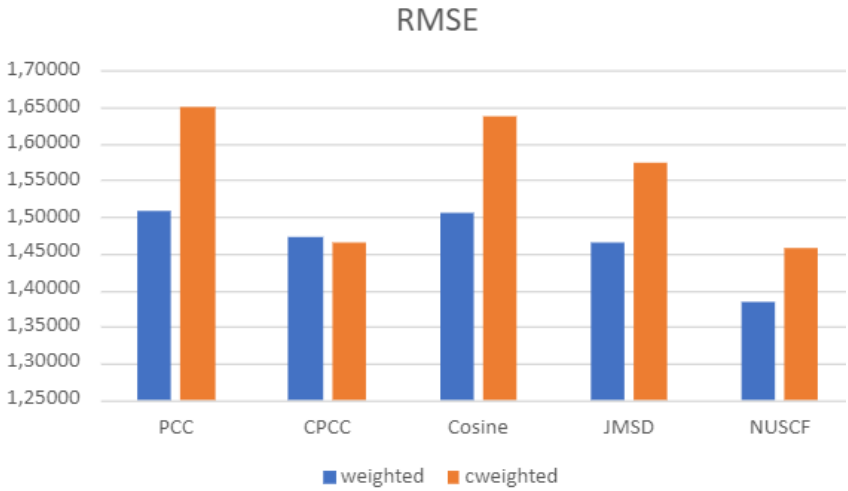


Рис.6.5. График RMSE

ций, либо NUSCF и взвешенный метод, ибо они показали наименьшие значения MAE и RMSE. При этом стоить отметить, что NUSCF занимает больше времени, так что хорошей идеей будет использование его в начала жизненного цикла платформы при меньшем количестве оценок, а затем сменить его на CPCC.

### 6.2. Тестирование платформы

Тестирование платформы было проведено вручную. В первую очередь проект был протестирован на соответствие требованиям(рис.6.6).

Тестирование интерфейса также было проведено вручную. В первую очередь внимание обращалось на корректное поведение переходов и заполнение форм: проводились попытки отправлять формы с незаполненными важными полями, а также на корректную работу авторизации, доступных только администратору. Тестирование интер-

| | Реализовано | Подробно |
|---|---|---|
| Выдача информации, поиск | Да | Поиск доступен как на отдельной странице, так и в заголовке сайта, а вся необходимая информация отображается на страницах произведения или участника |
| Система оценивания | Да | При нажатии на рейтинг произведения на его странице открывается форма, после заполнения которой можно оставить отзыв с оценкой от 1 до 10 и словесным пояснением |
| Система обсуждений | Да | На странице произведения отображается список обсуждений и имеется кнопка создать новое, также при добавлении записи о произведении для него автоматически генерируется общее обсуждение |
| Кроссжанровая система рекомендаций | Да | Ручная система рекомендаций присутствует на странице произведения: любой пользователь может оставить свою рекомендацию. На странице профиля же присутствует список с автоматическими рекомендациями |
| Система связей | Да | На странице произведения можно связать его с каким-либо другим, указав их отношение друг к другу |

Рис.6.6. Выполнение требований

фейса проводилось одновременно с интеграционным тестированием всего проекта на разных компьютерах и браузерах, в итоге ошибок выявлено не было.

# ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была разработана кроссжанровая информационная платформа, позволяющая просматривать информацию, оценивать, искать обсуждения, а также получать различные рекомендации.

Платформа была создана так, чтобы она соответствовала критериям, выведенным в главе 1 на основе анализа существующих информационных платформ, работающих с разными жанрами, а именно: поддержка различных жанров произведений, возможность оценивания, наличие обсуждений, рекомендации между жанрами, навигация между связанными произведениями разных жанров. Одной из ключевых частей системы является рекомендательная система, работающая в двух режимах: ручной и автоматический. Для поддержки работы автоматического режима были реализованы несколько методов подсчета расстояния между пользователями и нахождения рекомендаций для подхода, основанного на профиле пользователя, а также модификация для учета связанных произведений.

Для реализации системы была выбрана трехуровневая структура, состоящая из базы данных PostgreSQL, сервера на платформе Node.js и клиента, использующего фреймворк Vue.js. Обмен данными между клиентом и сервером ведется с помощью RESTful API. Была протестирована как разработанная система, так и отдельно эффективность реализаций методов генерации рекомендаций.

В качестве дальнейшего развития проекта можно добавить рекомендации, основанные на расчете расстояний между произведениями, или же использовать не memory-based алгоритмы, а model-based. Также в перспективе можно добавлять интеграции с внешними сервисами. В проекте реализована интеграция с сервисом Google Books для заполнения информации о книгах по номеру ISBN, можно интегрировать и другие сервисы для заполнения информации о произведениях

других жанров. Ещё одним направлением для дальнейшего развития является улучшение дизайна проекта.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Bobadilla Jesús, Serradilla Francisco, Bernal Jesus. A new collaborative filtering metric that improves the behavior of recommender systems // Knowledge-Based Systems. — 2010. — Vol. 23, no. 6. — P. 520–528.
2. Ekstrand Michael D, Riedl John T, Konstan Joseph A. Collaborative filtering recommender systems. — Now Publishers Inc, 2011.
3. Evaluating Collaborative Filtering Recommender Systems / Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, John T. Riedl // ACM Trans. Inf. Syst. — 2004. — . — Vol. 22, no. 1. — P. 5–53. — URL: https://doi.org/10.1145/963770.963772.
4. Harper F. Maxwell, Konstan Joseph A. The MovieLens Datasets: History and Context // ACM Trans. Interact. Intell. Syst. — 2015. — . — Vol. 5, no. 4. — 19 p. — URL: https://doi.org/10.1145/2827872.
5. Koohi Hamidreza, Kiani Kourosh. A new method to find neighbor users that improves the performance of Collaborative Filtering // Expert Systems with Applications. — 2017. — Vol. 83. — P. 30–39. — URL: https://www.sciencedirect.com/science/article/pii/S0957417417302713.
6. Levinas Claudio Adrian. An analysis of memory based collaborative filtering recommender systems with improvement proposals. — 2014.
7. Ricci Francesco, Rokach Lior, Shapira Bracha. Introduction to Recommender Systems Handbook // Recommender Systems Handbook / Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor. — Boston, MA : Springer US, 2011. — P. 1–35. — ISBN: 978-0-387-85820-3. — URL: https://doi.org/10.1007/978-0-387-85820-3_1.
8. Salton Gerard, McGill Michael. (1986). Introduction to modern information retrieval.

9. Shardanand Upendra, Maes Pattie. Social information filtering: Algorithms for automating “word of mouth” // Proceedings of the SIGCHI conference on Human factors in computing systems. — 1995. — P. 210–217.
10. Sikorsky Pavel. Исследование страницы персоны Кинопоиска. — 2016. — URL: https://medium.com/@IngMaeSing/kinopoisk-person-page-research-4f33e5b4d318.
11. Terveen Loren, Hill Will. Beyond Recommender Systems: Helping People Help Each Other. — 2001. — 02.
12. Wohlgethan Eric. SupportingWeb Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js : Ph. D. thesis / Eric Wohlgethan ; Hochschule für Angewandte Wissenschaften Hamburg. — 2018.
13. An improved memory-based collaborative filtering method based on the TOPSIS technique / Hael Al-bashiri, Mansoor Abdullateef Abdulgabber, Awanis Romli, Hasan Kahtan // PLOS ONE. — 2018. — 10. — Vol. 13, no. 10. — P. 1–26. — URL: https://doi.org/10.1371/journal.pone.0204434.
14. A new similarity measure using Bhattacharyya coefficient for collaborative filtering in sparse data / Bidyut Kr. Patra, Raimo Launonen, Ville Ollikainen, Sukumar Nandi // Knowledge-Based Systems. — 2015. — Vol. 82. — P. 163–177. — URL: https://www.sciencedirect.com/science/article/pii/S0950705115000830.

# ПРИЛОЖЕНИЕ 1
# ЛИСТИНГИ

Листинг П1.1. Страница произведения

```
<template>
  <MDBModal
    id="reviewModal"
    tabindex="-1"
    labelledby="reviewModalLabel"
    class="w-100"
    v-model="showReviewModal"
  >
    <MDBModalHeader>
      <MDBModalTitle id="reviewModalLabel"> Оставить отзыв </
          MDBModalTitle>
    </MDBModalHeader>
    <MDBModalBody
      ><star-rating
        v-model:rating="choosenRating"
        :increment="1"
        :max-rating="10"
        v-bind:star-size="40"
      />
      <textarea
        placeholder="Оставьте детальный отзыв"
        id="reviewText"
        v-model="reviewText"
        class="
          pt-3
          pb-2
          block
          w-full
          px-0
          mt-0
          bg-transparent
          border-2
          appearance-none
          focus:outline-none
          focus:ring-0
          focus:border-black
          border-gray-200
        "
      />
```

```
39      </MDBModalBody>
40      <MDBModalFooter>
41        <MDBBtn color="secondary" @click="showReviewModal = false"
42          >Закрыть</MDBBtn
43        >
44        <MDBBtn color="primary" @click="sendReview()">Отправить</
              MDBBtn>
45      </MDBModalFooter>
46    </MDBModal>
47    <MDBModal
48      id="discussionModal"
49      tabindex="-1"
50      labelledby="discussionModalLabel"
51      class="w-100"
52      v-model="showNewDiscussionModal"
53    >
54      <MDBModalHeader>
55        <MDBModalTitle id="discussionModalLabel">
56          Добавить обсуждение
57        </MDBModalTitle>
58      </MDBModalHeader>
59      <MDBModalBody>
60        <input
61          type="text"
62          placeholder="Название обсуждения"
63          v-model="discussionTitle"
64        />
65        <textarea
66          placeholder="Первый комментарий"
67          v-model="discussionContent"
68          class="
69            pt-3
70            pb-2
71            block
72            w-full
73            px-0
74            mt-0
75            bg-transparent
76            border-2
77            appearance-none
78            focus:outline-none
79            focus:ring-0
80            focus:border-black
81            border-gray-200
82          "
83        />
```

```
84      </MDBModalBody>
85      <MDBModalFooter>
86        <MDBBtn color="secondary" @click="showNewDiscussionModal =
              false"
87          >Закрыть</MDBBtn
88        >
89        <MDBBtn color="primary" @click="addDiscussion()">Отправить</
              MDBBtn>
90      </MDBModalFooter>
91    </MDBModal>
92
93    <MDBModal
94      id="relationModal"
95      tabindex="-1"
96      labelledby="relationModalLabel"
97      class="w-100"
98      v-model="shoNewRealtionModal"
99    >
100     <MDBModalHeader>
101       <MDBModalTitle id="relationModalLabel">
102         Добавить связанное произведение
103       </MDBModalTitle>
104     </MDBModalHeader>
105     <MDBModalBody>
106       <div class="flex flex-col">
107         <input
108           type="text"
109           v-model="relationModalTextSearch"
110           placeholder="Название связанного произведения"
111           @input="searchTextChanged()"
112           @focus="setSearchOptions(true)"
113           @blur="setSearchOptions(false)"
114         />
115         <input
116           type="text"
117           placeholder="Положение этого произведения"
118           v-model="thisCreationStanding"
119         />
120         <input
121           placeholder="Положение связанного произведения"
122           v-model="otherCreationStanding"
123           type="text"
124         />
125       </div>
126       <transition
127         enter-active-class="transition ease-out duration-100"
```

```
128           enter-class="transform opacity-0 scale-95"
129           enter-to-class="transform opacity-100 scale-100"
130           leave-active-class="transition ease-in duration-75"
131           leave-class="transform opacity-100 scale-100"
132           leave-to-class="transform opacity-0 scale-95"
133         >
134           <div
135             v-if="searchOptionsShow"
136             class="
137               origin-bottom-left
138               absolute
139               left-0
140               bottom-8
141               mt-2
142               w-5/6
143               rounded-md
144               shadow-lg
145               text-sm
146               overflow-hidden
147               border
148               z-20
149               bg-white
150               p-0
151             "
152           >
153             <ul
154               v-if="realtionModalCreations.length != 0"
155               class="divide-y divide-gray-300 overflow-auto h-full w-
                      full p-0"
156             >
157               <li
158                 v-for="creation in realtionModalCreations"
159                 :key="creation.id"
160                 class="hover:bg-gray-100"
161               >
162                 <button
163                   v-if="creation.id != $route.params.id"
164                   @click="
165                     otherCreationId = creation.id;
166                     relationModalTextSearch = creation.Creation_Names
                            [0].name;
167                   "
168                 >
169                   {{ creation.Creation_Names[0].name }}
170                 </button>
171               </li>
```

```
172           </ul>
173           <label v-if="realtionModalCreations.length == 0">
174             Ничего не найдено
175           </label>
176         </div>
177       </transition>
178     </MDBModalBody>
179     <MDBModalFooter>
180       <MDBBtn color="secondary" @click="shoNewRealtionModal = false"
181         >Закрыть</MDBBtn
182       >
183       <MDBBtn color="primary" @click="addRelatedCreation()">Отправит
              ь</MDBBtn>
184     </MDBModalFooter>
185   </MDBModal>
186   <MDBModal
187     id="addAuthorModal"
188     tabindex="-1"
189     labelledby="addAuthorModalLabel"
190     class="w-100"
191     v-model="showAddAuthorModal"
192   >
193     <MDBModalHeader>
194       <MDBModalTitle id="addAuthorModalLabel">
195         Добавить участника
196       </MDBModalTitle>
197     </MDBModalHeader>
198     <MDBModalBody>
199       <input
200         type="text"
201         v-model="authorModalTextSearch"
202         placeholder="Имя участника"
203         @input="authorSearchTextChanged()"
204         @focus="setAuthorSearchOptions(true)"
205         @blur="setAuthorSearchOptions(false)"
206       />
207       <select v-model="newAuthorRole" class="form-select">
208         <option>Выберите роль</option>
209         <option v-for="role in roles" :value="role.name" :key="role.
              id">
210           {{ role.name }}
211         </option>
212       </select>
213       <transition
214         enter-active-class="transition ease-out duration-100"
215         enter-class="transform opacity-0 scale-95"
```

```
216         enter-to-class="transform opacity-100 scale-100"
217         leave-active-class="transition ease-in duration-75"
218         leave-class="transform opacity-100 scale-100"
219         leave-to-class="transform opacity-0 scale-95"
220       >
221         <div
222           v-if="authorSearchOptionsShow"
223           class="
224             origin-bottom-left
225             absolute
226             left-0
227             bottom-2
228             mt-2
229             w-5/6
230             rounded-md
231             shadow-lg
232             text-sm
233             overflow-hidden
234             border
235             z-20
236             bg-white
237             p-0
238           "
239         >
240           <ul
241             v-if="modalAuthors.length != 0"
242             class="divide-y divide-gray-300 overflow-auto h-full"
243           >
244             <li
245               v-for="author in modalAuthors"
246               :key="author.id"
247               class="hover:bg-gray-100"
248             >
249               <button
250                 @click="
251                   newAuthorId = author.id;
252                   authorModalTextSearch = author.name;
253                 "
254               >
255                 {{ author.name }}
256               </button>
257             </li>
258           </ul>
259           <label v-if="modalAuthors.length == 0"> Ничего не найдено
                 </label>
260         </div>
```

```
261       </transition>
262     </MDBModalBody>
263     <MDBModalFooter>
264       <MDBBtn color="secondary" @click="showAddAuthorModal = false"
265         >Закрыть</MDBBtn
266       >
267       <MDBBtn color="primary" @click="addAuthor()">Отправить</MDBBtn
              >
268     </MDBModalFooter>
269   </MDBModal>
270
271   <MDBModal
272     id="newNameModal"
273     tabindex="-1"
274     labelledby="newNameLabel"
275     class="w-100"
276     v-model="showNewNameModal"
277   >
278     <MDBModalHeader>
279       <MDBModalTitle id="newNameLabel">
280         Добавить альтернативное название
281       </MDBModalTitle>
282     </MDBModalHeader>
283     <MDBModalBody>
284       <input
285         type="text"
286         placeholder="Название произведения"
287         v-model="newName"
288       />
289     </MDBModalBody>
290     <MDBModalFooter>
291       <MDBBtn color="secondary" @click="showNewNameModal = false"
292         >Закрыть</MDBBtn
293       >
294       <MDBBtn color="primary" @click="addNewName()">Отправить</
              MDBBtn>
295     </MDBModalFooter>
296   </MDBModal>
297
298   <MDBModal
299     id="newRecModal"
300     tabindex="-1"
301     labelledby="newRecLabel"
302     class="w-100"
303     v-model="showNewRecModal"
304   >
```

```
305     <MDBModalHeader>
306       <MDBModalTitle id="newRecLabel"> Добавить рекомендацию </
              MDBModalTitle>
307     </MDBModalHeader>
308     <MDBModalBody>
309       <div class="flex flex-col">
310         <input
311           type="text"
312           v-model="relationModalTextSearch"
313           placeholder="Название рекомендуемого произведения"
314           @input="searchTextChanged()"
315           @focus="setSearchOptions(true)"
316           @blur="setSearchOptions(false)"
317         />
318         <input
319           placeholder="Причина рекомендации"
320           v-model="userRecContent"
321           type="text"
322         />
323       </div>
324       <transition
325         enter-active-class="transition ease-out duration-100"
326         enter-class="transform opacity-0 scale-95"
327         enter-to-class="transform opacity-100 scale-100"
328         leave-active-class="transition ease-in duration-75"
329         leave-class="transform opacity-100 scale-100"
330         leave-to-class="transform opacity-0 scale-95"
331       >
332         <div
333           v-if="searchOptionsShow"
334           class="
335             origin-bottom-left
336             absolute
337             left-0
338             bottom-2
339             mt-2
340             w-5/6
341             rounded-md
342             shadow-lg
343             text-sm
344             overflow-hidden
345             border
346             z-20
347             bg-white
348             p-0
349           "
```

```
350         >
351           <ul
352             v-if="realtionModalCreations.length != 0"
353             class="divide-y divide-gray-300 overflow-auto h-full w-
                    full p-0"
354           >
355             <li
356               v-for="creation in realtionModalCreations"
357               :key="creation.id"
358               class="hover:bg-gray-100"
359             >
360               <button
361                 v-if="creation.id != $route.params.id"
362                 @click="
363                   userRecSecondCreationId = creation.id;
364                   relationModalTextSearch = creation.Creation_Names
                          [0].name;
365                 "
366               >
367                 {{ creation.Creation_Names[0].name }}
368               </button>
369             </li>
370           </ul>
371           <label v-if="realtionModalCreations.length == 0">
372             Ничего не найдено
373           </label>
374         </div>
375       </transition>
376     </MDBModalBody>
377     <MDBModalFooter>
378       <MDBBtn color="secondary" @click="showNewRecModal = false"
379         >Закрыть</MDBBtn
380       >
381       <MDBBtn color="primary" @click="addNewRec()">Отправить</MDBBtn
            >
382     </MDBModalFooter>
383   </MDBModal>
384
385   <div class="min-h-screen">
386     <custom-header />
387     <div v-if="loading">
388       <label> Загрузка </label>
389     </div>
390     <div v-if="!loading" class="grid grid-cols-5 grid-rows-4 pt-4">
391       <img
392         v-if="image"
```

```
393           :src="image"
394           class="
395             row-span-3
396             rounded-none
397             lg:rounded-lg
398             shadow-xl
399             hidden
400             lg:block
401             p-2
402             col-sapn-1 col-start-1
403           "
404         />
405         <div
406           class="pl-12 col-span-3 col-start-2 row-span-4 flex-col
                align-middle"
407         >
408           <div class="flex flex-row">
409             <h1 class="text-3xl font-bold pt-8 lg:pt-0">
410               {{ info.Creation_Names[0].name }}
411             </h1>
412             <MDBBtn
413               tag="a"
414               color="light"
415               rounded
416               @click="showNewNameModal = true"
417               size="sm"
418               class="mb-2 ms-2 mt-2"
419             >
420               <MDBIcon iconStyle="fas" icon="plus"></MDBIcon>
421             </MDBBtn>
422           </div>
423           <div
424             class="mx-auto lg:mx-0 w-100 border-b-2 border-green-600
                  opacity-25"
425           ></div>
426           <h1 class="pt-2 text-gray-500 text-base">{{ info.genre }}</
                h1>
427           <star-rating
428             v-model:rating="rating"
429             :increment="0.01"
430             :max-rating="10"
431             :read-only="true"
432             @click="showReviewModal = true"
433           />
434           <h1 v-if="info.country" id="country" class="text-base pt
                -16">
```

```
435           Страна: {{ info.country }}
436         </h1>
437         <h1 v-if="info.age_rating" id="country" class="text-base pt
                -2">
438           Возрастной рейтинг: {{ info.age_rating }}
439         </h1>
440         <div v-if="info.Creation_Names.length > 1">
441           Альтернативные названия:
442           {{
443             info.Creation_Names.slice(1)
444               .map((a) => a.name)
445               .join()
446           }}
447         </div>
448         <div class="pt-6">
449           <label for="description" class="text-base font-bold">Описа
                ние</label>
450           <h1 id="description" class="text-base pl-4">
451             {{ info.description }}
452           </h1>
453         </div>
454         <div v-if="tags.length > 1">
455           Тэги:
456           {{ tags.map((a) => a.name).join() }}
457         </div>
458         <div class="pt-2">
459           <div class="flex flex-row h-6">
460             Связанные произведения
461             <MDBBtn
462               tag="a"
463               color="light"
464               rounded
465               @click="shoNewRealtionModal = true"
466               size="sm"
467             >
468               <MDBIcon iconStyle="fas" icon="plus"></MDBIcon>
469             </MDBBtn>
470           </div>
471           <ul v-if="relatedCreations.length != 0" class="pl-4">
472             <li
473               v-for="relatedCreation in relatedCreations"
474               :key="relatedCreation.id"
475             >
476               <router-link
477                 v-if="relatedCreation.firstCreationId == $route.
                        params.id"
```

```
478                 :to="{ path: `/creations/${relatedCreation.
                        secondCreationId}` }"
479                 >{{
480                   relatedCreation.secondCreationNames
481                     ? relatedCreation.secondCreationNames[0].name
482                     : relatedCreation.secondCreationId
483                 }}
484                 :
485                 {{ relatedCreation.secondCreationStanding }}
486               </router-link>
487               <router-link
488                 v-if="relatedCreation.secondCreationId == $route.
                        params.id"
489                 :to="{ path: `/creations/${relatedCreation.
                        firstCreationId}` }"
490                 >{{
491                   relatedCreation.firstCreationNames
492                     ? relatedCreation.firstCreationNames[0].name
493                     : relatedCreation.firstCreationId
494                 }}
495                 :
496                 {{ relatedCreation.firstCreationStanding }}
497               </router-link>
498             </li>
499           </ul>
500           <label v-if="relatedCreations.length == 0" class="pl-4"
501             >Отсутствуют</label
502           >
503         </div>
504       </div>
505
506       <div
507         class="pl-12 col-span-1 col-start-5 row-span-4 flex-col
                align-middle"
508       >
509         <div class="flex flex-row">
510           <h2>Участники</h2>
511           <MDBBtn
512             tag="a"
513             color="light"
514             rounded
515             @click="
516               showAddAuthorModal = true;
517               fetchRoles();
518             "
519             size="sm"
```

```
520             class="m-2"
521           >
522             <MDBIcon iconStyle="fas" icon="plus"></MDBIcon>
523           </MDBBtn>
524         </div>
525         <ul
526           v-if="involvment"
527           class="divide-y divide-gray-300 overflow-auto h-full pr-4"
528         >
529           <li v-for="inv in involvment" :key="inv.id">
530             <ul
531               v-if="involvment"
532               class="divide-y divide-gray-300 overflow-auto h-full
                      pr-4"
533             >
534               <li
535                 v-for="author in inv.Authors"
536                 :key="author.id"
537                 class="hover:bg-gray-100"
538               >
539                 <mini-author-info
540                   :author_id="author.id"
541                   :img_height="80"
542                   :img_width="80"
543                   :role="inv.name"
544                   :isApproved="true"
545                 />
546               </li>
547             </ul>
548           </li>
549         </ul>
550         <label v-if="involvment === null" class="pl-4">Никто не указ
              ан</label>
551       </div>
552     </div>
553     <div class="grid grid-cols-3 grid-rows-1">
554       <div
555         class="h-80 col-start-1 col-span-1 flex flex-col align-items
              -center p-4"
556       >
557         <h2>Похожие на основе тэгов</h2>
558         <ul
559           v-if="similar.length > 0"
560           class="divide-y divide-gray-300 overflow-auto h-full w-
                full"
561         >
```

```
562           <li
563             v-for="creation in similar"
564             :key="creation.id"
565             class="hover:bg-gray-100"
566           >
567             <mini-creation-info
568               :creation_id="creation.id"
569               :img_height="80"
570               :img_width="80"
571               :isApproved="true"
572             />
573           </li>
574         </ul>
575         <label v-if="similar.length == 0" class="pl-4"
576           >Такие произведения не найдены</label
577         >
578       </div>
579       <div
580         class="h-80 col-start-2 col-span-1 flex flex-col align-items
              -center p-4"
581       >
582         <h2>Похожие на основе участников</h2>
583         <ul
584           v-if="similarByAuthor.length > 0"
585           class="divide-y divide-gray-300 overflow-auto h-full w-
                full"
586         >
587           <li
588             v-for="creation in similarByAuthor"
589             :key="creation.id"
590             class="hover:bg-gray-100"
591           >
592             <mini-creation-info
593               :creation_id="creation.id"
594               :img_height="80"
595               :img_width="80"
596               :isApproved="true"
597             />
598           </li>
599         </ul>
600         <label v-if="similarByAuthor.length == 0" class="pl-4"
601           >Такие произведения не найдены</label
602         >
603       </div>
604       <div
```

```
605         class="h-80 col-start-3 col-span-1 flex flex-col align-items
                -center p-4"
606       >
607         <div class="flex flex-row">
608           <h2>Пользовательские рекомендации</h2>
609           <MDBBtn
610             tag="a"
611             color="light"
612             rounded
613             @click="showNewRecModal = true"
614             size="sm"
615             class="mb-2 ms-2 mt-2"
616           >
617             <MDBIcon iconStyle="fas" icon="plus"></MDBIcon>
618           </MDBBtn>
619         </div>
620         <ul
621           v-if="userRecs.length > 0"
622           class="divide-y divide-gray-300 overflow-auto h-full w-
                full"
623         >
624           <li
625             v-for="userRec in userRecs"
626             :key="userRec.id"
627             class="hover:bg-gray-100"
628           >
629             <mini-creation-info
630               :creation_id="
631                 userRec.firstCreationId == $route.params.id
632                   ? userRec.secondCreationId
633                   : userRec.firstCreationId
634               "
635               :img_height="80"
636               :img_width="80"
637               :isApproved="true"
638             />
639           </li>
640         </ul>
641         <label v-if="userRecs.length == 0" class="pl-4"
642           >Такие произведения не найдены</label
643         >
644       </div>
645     </div>
646     <div class="grid grid-cols-2 grid-rows-1">
647       <div
```

```
648           class="h-80 col-start-1 col-span-1 flex flex-col align-items
                  -center p-4"
649         >
650           <h2>Обсуждения</h2>
651           <ul
652             v-if="discussions.length > 0"
653             class="divide-y divide-gray-300 overflow-auto h-full"
654           >
655             <li
656               v-for="discussion in discussions"
657               :key="discussion.id"
658               class="hover:bg-gray-100 border-2 border-gray-400"
659             >
660               <router-link :to="{ path: `/discussion-page/${discussion
                      .id}` }">
661                 <h1 class="text-black">{{ discussion.title }}</h1>
662                 <div
663                   class="
664                     mx-auto
665                     lg:mx-0
666                     w-100
667                     border-b-2 border-gray-400
668                     opacity-25
669                   "
670                 ></div>
671                 <h2 class="text-black">{{ discussion.content }}</h2>
672               </router-link>
673             </li>
674           </ul>
675           <label v-if="discussions.length == 0" class="pl-4"
676             >Обсуждений нет</label
677           >
678         </div>
679         <div
680           class="h-80 col-start-2 col-span-1 flex flex-col align-items
                  -center p-4"
681         >
682           <h2>Отзывы</h2>
683           <ul
684             v-if="reviews.length != 0"
685             class="divide-y divide-gray-300 overflow-auto h-full w-
                    full"
686           >
687             <li
688               v-for="review in reviews"
689               :key="review.id"
```

```
690             class="hover:bg-gray-100 border-2 border-gray-400"
691           >
692             <review-block :review_id="review.id" :isOnUserPage="
                    false" />
693           </li>
694         </ul>
695         <label v-if="reviews.length == 0" class="pl-4"
696           >Детальных отзывов нет</label
697         >
698       </div>
699     </div>
700     <MDBBtn color="light" v-if="isAdmin" @Click="deleteCreation"
701       >Удалить произведение</MDBBtn
702     >
703     <MDBBtn color="light" @Click="showNewDiscussionModal = true"
704       >Добавить обсуждение</MDBBtn
705     >
706   </div>
707 </template>
708 <script>
709 import axios from "axios";
710 import { APIURL } from "../constants";
711 import CustomHeader from "../components/CustomHeader";
712 import StarRating from "vue-star-rating";
713 import MiniCreationInfo from "../components/MiniCreationInfo";
714 import MiniAuthorInfo from "../components/MiniAuthorInfo";
715 import Review from "../components/Review.vue";
716 import {
717   MDBModal,
718   MDBModalHeader,
719   MDBModalTitle,
720   MDBModalBody,
721   MDBModalFooter,
722   MDBBtn,
723   MDBIcon,
724 } from "mdb-vue-ui-kit";
725
726 export default {
727   components: {
728     "custom-header": CustomHeader,
729     "mini-creation-info": MiniCreationInfo,
730     "mini-author-info": MiniAuthorInfo,
731     "review-block": Review,
732     StarRating,
733     MDBModal,
734     MDBModalHeader,
```

```
735       MDBModalTitle,
736       MDBModalBody,
737       MDBModalFooter,
738       MDBBtn,
739       MDBIcon,
740     },
741     data() {
742       return {
743         info: null,
744         similar: [],
745         similarByAuthor: [],
746         isAdmin: false,
747         image: require("@/assets/placeholder.png"),
748         tags: [],
749         involvment: null,
750         loading: true,
751         rating: null,
752         showReviewModal: false,
753         choosenRating: 0,
754         reviewText: "",
755         discussionContent: "",
756         discussionTitle: "",
757         showNewDiscussionModal: false,
758         relatedCreations: [],
759         thisCreationStanding: "",
760         otherCreationStanding: "",
761         otherCreationId: null,
762         shoNewRealtionModal: false,
763         relationModalTextSearch: "",
764         realtionModalCreations: [],
765         searchOptionsShow: false,
766         showAddAuthorModal: false,
767         roles: [],
768         newAuthorRole: null,
769         authorSearchOptionsShow: false,
770         authorModalTextSearch: "",
771         modalAuthors: [],
772         newAuthorId: null,
773         showNewNameModal: false,
774         newName: "",
775         discussions: [],
776         reviews: [],
777         userRecs: [],
778         showNewRecModal: false,
779         userRecSecondCreationId: null,
780         userRecContent: "",
```

```
781     };
782   },
783   async created() {
784     this.fetchCreationInfo();
785     this.fetchCreationTags();
786     this.fetchSimilar();
787     this.fetchinvolved();
788     this.fetchRating();
789     this.fetchSimilarByAuthors();
790     this.fetchRelatedCreations();
791     this.fetchDiscussions();
792     this.fetchReviews();
793     this.fetchUserRecs();
794     if (localStorage.getItem("is_admin") == "true") {
795       this.isAdmin = true;
796     }
797   },
798   methods: {
799     fetchCreationInfo() {
800       const fetchedId = this.$route.params.id;
801       axios
802         .get(`${APIURL}/creations/${this.$route.params.id}`)
803         .then((result) => {
804           if (this.$route.params.id !== fetchedId) return;
805           if (result.data.result !== undefined) {
806             this.info = result.data.result;
807             if (this.info.image !== undefined) {
808               this.image = "" + this.info.
                      image;
809             }
810           }
811           axios
812             .get(`${APIURL}/genres`)
813             .then((result) => {
814               if (this.$route.params.id !== fetchedId) return;
815               if (result.data.result !== undefined) {
816                 var genre = null;
817                 for (genre of result.data.result) {
818                   if (genre.id == this.info.CreationTypeId) {
819                     this.info.genre = genre.name;
820                     break;
821                   }
822                 }
823                 this.loading = false;
824               }
825             })
```

```
826             .catch((error) => {
827               this.$notify({
828                 title: "Произошла ошибка",
829                 text: error.response.data.error,
830                 type: "error",
831               });
832             });
833         })
834         .catch((error) => {
835           this.$notify({
836             title: "Произошла ошибка",
837             text: error.response.data.error,
838             type: "error",
839           });
840         });
841     },
842     fetchSimilar() {
843       const fetchedId = this.$route.params.id;
844       axios
845         .get(`${APIURL}/creations-similar/${this.$route.params.id}`,
                )
846         .then((result) => {
847           if (this.$route.params.id !== fetchedId) return;
848           if (result.data.result !== undefined) {
849             this.similar = result.data.result;
850           }
851         })
852         .catch((error) => {
853           this.$notify({
854             title: "Произошла ошибка",
855             text: error.response.data.error,
856             type: "error",
857           });
858         });
859     },
860     deleteCreation() {
861       const token = localStorage.getItem("token");
862       axios
863         .delete(
864           `${APIURL}/creations/${this.$route.params.id}`,
865           {
866             new_record_id: this.$route.params.id,
867           },
868           { headers: { authorization: token } }
869         )
870         .then(() => {
```

```
871           this.$router.push("/main-page");
872         })
873         .catch((error) => {
874           this.$notify({
875             title: "Произошла ошибка",
876             text: error.response.data.error,
877             type: "error",
878           });
879         });
880     },
881     fetchCreationTags() {
882       const fetchedId = this.$route.params.id;
883       axios
884         .get(`${APIURL}/creation-tags/${this.$route.params.id}`)
885         .then((result) => {
886           if (this.$route.params.id !== fetchedId) return;
887           this.tags = result.data.result;
888         })
889         .catch((error) => {
890           this.$notify({
891             title: "Произошла ошибка",
892             text: error.response.data.error,
893             type: "error",
894           });
895         });
896     },
897     fetchinvolved() {
898       const fetchedId = this.$route.params.id;
899       axios
900         .get(`${APIURL}/creation-role/${this.$route.params.id}`)
901         .then((result) => {
902           if (this.$route.params.id !== fetchedId) return;
903           this.involvment = result.data.result;
904         })
905         .catch((error) => {
906           this.$notify({
907             title: "Произошла ошибка",
908             text: error.response.data.error,
909             type: "error",
910           });
911         });
912     },
913     fetchRating() {
914       const fetchedId = this.$route.params.id;
915       axios
916         .get(`${APIURL}/rating/${this.$route.params.id}`)
```

```
917         .then((result) => {
918           if (this.$route.params.id !== fetchedId) return;
919           this.rating = result.data.average;
920         })
921         .catch((error) => {
922           this.$notify({
923             title: "Произошла ошибка",
924             text: error.response.data.error,
925             type: "error",
926           });
927         });
928     },
929     fetchSimilarByAuthors() {
930       const fetchedId = this.$route.params.id;
931       axios
932         .get(`${APIURL}/creations-similar-by-author/${this.$route.
              params.id}`)
933         .then((result) => {
934           if (this.$route.params.id !== fetchedId) return;
935           if (result.data.result !== undefined) {
936             this.similarByAuthor = result.data.result;
937           }
938         })
939         .catch((error) => {
940           this.$notify({
941             title: "Произошла ошибка",
942             text: error.response.data.error,
943             type: "error",
944           });
945         });
946     },
947     sendReview() {
948       const fetchedId = this.$route.params.id;
949       const token = localStorage.getItem("token");
950       axios
951         .post(
952           `${APIURL}/reviews/${this.$route.params.id}`,
953           { score: this.choosenRating, content: this.reviewText },
954           { headers: { authorization: token } }
955         )
956         .then(() => {
957           if (this.$route.params.id !== fetchedId) return;
958           this.fetchRating();
959           this.showReviewModal = false;
960           //Оповестить пользователя что все классно
961         })
```

```
962         .catch((error) => {
963           this.$notify({
964             title: "Произошла ошибка",
965             text: error.response.data.error,
966             type: "error",
967           });
968         });
969     },
970     addDiscussion() {
971       const fetchedId = this.$route.params.id;
972       const token = localStorage.getItem("token");
973       axios
974         .post(
975           `${APIURL}/discussions`,
976           {
977             CreationId: fetchedId,
978             content: this.discussionContent,
979             title: this.discussionTitle,
980           },
981           { headers: { authorization: token } }
982         )
983         .then(() => {
984           if (this.$route.params.id !== fetchedId) return;
985           this.showNewDiscussionModal = false;
986           //Оповестить пользователя что все классно
987         })
988         .catch((error) => {
989           this.$notify({
990             title: "Произошла ошибка",
991             text: error.response.data.error,
992             type: "error",
993           });
994         });
995       this.showNewDiscussionModal = false;
996     },
997     fetchRelatedCreations() {
998       const fetchedId = this.$route.params.id;
999       axios
1000        .get(`${APIURL}/creation-relations/${this.$route.params.id
              }`)
1001        .then((result) => {
1002          if (this.$route.params.id !== fetchedId) return;
1003          this.relatedCreations = result.data.result;
1004          for (var relatedCreation of this.relatedCreations) {
1005            axios
```

```
1006              .get(`${APIURL}/creations/${relatedCreation.
                      firstCreationId}`)
1007              .then((creationInfo) => {
1008                relatedCreation.firstCreationNames =
1009                  creationInfo.data.result.Creation_Names;
1010              })
1011              .catch((error) => {
1012                this.$notify({
1013                  title: "Произошла ошибка",
1014                  text: error.response.data.error,
1015                  type: "error",
1016                });
1017              });
1018            axios
1019              .get(`${APIURL}/creations/${relatedCreation.
                      secondCreationId}`)
1020              .then((creationInfo) => {
1021                relatedCreation.secondCreationNames =
1022                  creationInfo.data.result.Creation_Names;
1023              })
1024              .catch((error) => {
1025                this.$notify({
1026                  title: "Произошла ошибка",
1027                  text: error.response.data.error,
1028                  type: "error",
1029                });
1030              });
1031          }
1032        })
1033        .catch((error) => {
1034          this.$notify({
1035            title: "Произошла ошибка",
1036            text: error.response.data.error,
1037            type: "error",
1038          });
1039        });
1040    },
1041    addRelatedCreation() {
1042      if (this.otherCreationId === null) {
1043        alert("Не указано связанное произведение");
1044        return;
1045      }
1046      const fetchedId = this.$route.params.id;
1047      const token = localStorage.getItem("token");
1048      axios
1049        .post(
```

```
1050          `${APIURL}/creation-relations`,
1051          {
1052            firstCreationId: this.$route.params.id,
1053            secondCreationId: this.otherCreationId,
1054            firstCreationStanding: this.thisCreationStanding,
1055            secondCreationStanding: this.otherCreationStanding,
1056          },
1057          { headers: { authorization: token } }
1058        )
1059        .then(() => {
1060          if (this.$route.params.id !== fetchedId) return;
1061          this.shoNewRealtionModal = false;
1062          //Оповестить пользователя что все классно
1063        })
1064        .catch((error) => {
1065          this.$notify({
1066            title: "Произошла ошибка",
1067            text: error.response.data.error,
1068            type: "error",
1069          });
1070        });
1071      this.shoNewRealtionModal = false;
1072    },
1073    searchTextChanged() {
1074      if (this.relationModalTextSearch == "") {
1075        this.relationModalTextSearch = null;
1076        return;
1077      }
1078      const fetchedId = this.$route.params.id;
1079      axios
1080        .get(`${APIURL}/creations-search`, {
1081          params: { string: this.relationModalTextSearch },
1082        })
1083        .then((result) => {
1084          if (this.$route.params.id !== fetchedId) return;
1085          if (result.data.result !== undefined) {
1086            this.realtionModalCreations = result.data.result;
1087          }
1088        })
1089        .catch((error) => {
1090          this.$notify({
1091            title: "Произошла ошибка",
1092            text: error.response.data.error,
1093            type: "error",
1094          });
1095        });
```

```
1096      },
1097      setSearchOptions(flag) {
1098        this.searchOptionsShow = flag;
1099      },
1100      fetchRoles() {
1101        const fetchedId = this.$route.params.id;
1102        axios
1103          .get(`${APIURL}/roles`)
1104          .then((result) => {
1105            if (this.$route.params.id !== fetchedId) return;
1106            if (result.data.result !== undefined) {
1107              this.roles = result.data.result;
1108            }
1109          })
1110          .catch((error) => {
1111            this.$notify({
1112              title: "Произошла ошибка",
1113              text: error.response.data.error,
1114              type: "error",
1115            });
1116          });
1117      },
1118      setAuthorSearchOptions(flag) {
1119        this.authorSearchOptionsShow = flag;
1120      },
1121      authorSearchTextChanged() {
1122        if (this.authorModalTextSearch == "") {
1123          this.authorModalTextSearch = null;
1124          return;
1125        }
1126        const fetchedId = this.$route.params.id;
1127        axios
1128          .get(`${APIURL}/authors`, {
1129            params: { string: this.authorModalTextSearch },
1130          })
1131          .then((result) => {
1132            if (this.$route.params.id !== fetchedId) return;
1133            if (result.data.result !== undefined) {
1134              this.modalAuthors = result.data.result;
1135            }
1136          })
1137          .catch((error) => {
1138            this.$notify({
1139              title: "Произошла ошибка",
1140              text: error.response.data.error,
1141              type: "error",
```

```
1142            });
1143          });
1144      },
1145      addAuthor() {
1146        if (this.authorModalTextSearch == "") {
1147          return;
1148        }
1149        const token = localStorage.getItem("token");
1150        var roleId;
1151        for (var role of this.roles) {
1152          if (role.name == this.newAuthorRole) {
1153            roleId = role.id;
1154          }
1155        }
1156
1157        axios
1158          .post(
1159            `${APIURL}/author-role`,
1160            {
1161              creation_id: this.$route.params.id,
1162              author_id: this.newAuthorId,
1163              role_id: roleId,
1164            },
1165            {
1166              headers: {
1167                authorization: token,
1168              },
1169            }
1170          )
1171          .then(() => {
1172            this.$notify({
1173              title: "Успех",
1174              text: "Участник добавлен",
1175              type: "success",
1176            });
1177          })
1178          .catch((error) => {
1179            this.$notify({
1180              title: "Произошла ошибка",
1181              text: error.response.data.error,
1182              type: "error",
1183            });
1184          });
1185        this.showAddAuthorModal = false;
1186      },
1187      addNewName() {
```

```
1188        if (this.newName == "") {
1189          return;
1190        }
1191        const token = localStorage.getItem("token");
1192        axios
1193          .post(
1194            `${APIURL}/creation-names/${this.$route.params.id}`,
1195            {
1196              name: this.newName,
1197            },
1198            {
1199              headers: {
1200                authorization: token,
1201              },
1202            }
1203          )
1204          .then(() => {})
1205          .catch((error) => {
1206            this.$notify({
1207              title: "Произошла ошибка",
1208              text: error.response.data.error,
1209              type: "error",
1210            });
1211          });
1212        this.showNewNameModal = false;
1213      },
1214      fetchDiscussions() {
1215        const fetchedId = this.$route.params.id;
1216        axios
1217          .get(`${APIURL}/creation-discussions/${this.$route.params.id
                 }`)
1218          .then((result) => {
1219            if (this.$route.params.id !== fetchedId) return;
1220            this.discussions = result.data.result;
1221          })
1222          .catch((error) => {
1223            this.$notify({
1224              title: "Произошла ошибка",
1225              text: error.response.data.error,
1226              type: "error",
1227            });
1228          });
1229      },
1230      fetchReviews() {
1231        const fetchedId = this.$route.params.id;
1232        axios
```

```
1233          .get(`${APIURL}/reviews-creation/${this.$route.params.id}`)
1234          .then((result) => {
1235            if (this.$route.params.id !== fetchedId) return;
1236            this.reviews = result.data.result;
1237          })
1238          .catch((error) => {
1239            this.$notify({
1240              title: "Произошла ошибка",
1241              text: error.response.data.error,
1242              type: "error",
1243            });
1244          });
1245      },
1246      fetchUserRecs() {
1247        const fetchedId = this.$route.params.id;
1248        axios
1249          .get(`${APIURL}/user-reccomendations/${this.$route.params.id
                 }`)
1250          .then((result) => {
1251            if (this.$route.params.id !== fetchedId) return;
1252            this.userRecs = result.data.result;
1253          })
1254          .catch((error) => {
1255            this.$notify({
1256              title: "Произошла ошибка",
1257              text: error.response.data.error,
1258              type: "error",
1259            });
1260          });
1261      },
1262      addNewRec() {
1263        if (this.userRecSecondCreationId === null) {
1264          alert("Не указано рекомендуемое произведение");
1265          return;
1266        }
1267        const fetchedId = this.$route.params.id;
1268        const token = localStorage.getItem("token");
1269        axios
1270          .post(
1271            `${APIURL}/user-reccomendations`,
1272            {
1273              firstCreationId: this.$route.params.id,
1274              secondCreationId: this.userRecSecondCreationId,
1275              content: this.userRecContent,
1276            },
1277            { headers: { authorization: token } }
```

```
        )
        .then(() => {
          if (this.$route.params.id !== fetchedId) return;
          this.showNewRecModal = false;
          //Оповестить пользователя что все классно
        })
        .catch((error) => {
          this.$notify({
            title: "Произошла ошибка",
            text: error.response.data.error,
            type: "error",
          });
        });
      this.showNewRecModal = false;
    },
  },
};
</script>

<style scoped>
.modal {
  width: 700px;
  padding: 30px;
  box-sizing: border-box;
  background-color: #fff;
  font-size: 20px;
  text-align: center;
}
</style>
```