МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Аниме-отзовик

Студен гр. 1303 _____ Чернуха В.В.

Студент гр. 1303 _____ Петров А.С.

Студентка гр. 1303 _____ Виноградова Е.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студен Чернуха В.В.

Студен Петров А.С.

Студентка Виноградова Е.А.

Группа 1303

Тема: Аниме-отзовик

Исходные данные:

Необходимо реализовать веб-приложение для аниме-отзовика с использованием СУБД MongoDB.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 05.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студент	_____	Чернуха В.В.
Студент	_____	Петров А.С.
Студентка	_____	Виноградова Е.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках ИДЗ разработано веб-приложение, представляющее собой каталог аниме. Приложение включает функционал для просмотра информации о тайтлах, добавление, редактирования и удаление отзывов, добавление десятибалльной оценки. Реализована система фильтрации и поиска аниме по различным критериям.

Для разработки использованы технологии React, ASP.NET, СУБД MongoDB.

Исходный код: https://github.com/moevm/nosql2h24-anime

SUMMARY

A web application has been developed as part of the IDZ, which is an anime catalog. The application includes functionality for viewing information about titles, adding, editing and deleting reviews, adding a ten-point rating. A system for filtering and searching anime by various criteria has been implemented.

React, ASP.NET, and MongoDB DBMS technologies were used for development.

Source code: https://github.com/moevm/nosql2h24-anime

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы

С ростом популярности японской анимации в мире возникает необходимость в удобных инструментах для поиска, фильтрации и быстрого получения нужной информации о каком-либо аниме. Существующие решения часто не предлагают удобных механизмов для фильтрации, поиска нужных аниме.

1.2. Постановка задачи

Задача проекта заключается в разработке производительного веб-приложения, которое позволяет пользователям:

- Находить аниме с подробным описанием, отзывами пользователей.
- Фильтровать аниме по различным параметрам;
- Оставлять отзывы и оценки;
- Вести свой профиль

1.3. Предлагаемое решение

Для реализации веб-приложения были выбраны технологии React, ASP.NET и MongoDB для хранения данных. Приложение поддерживает фильтрацию, поиск и публикацию отзывов и оценок.

1.4. Качественные требования к решению

Решение должно быть удобным, производительным, надёжным и легко расширяемым.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

Ниже представлены макеты страниц приложения.

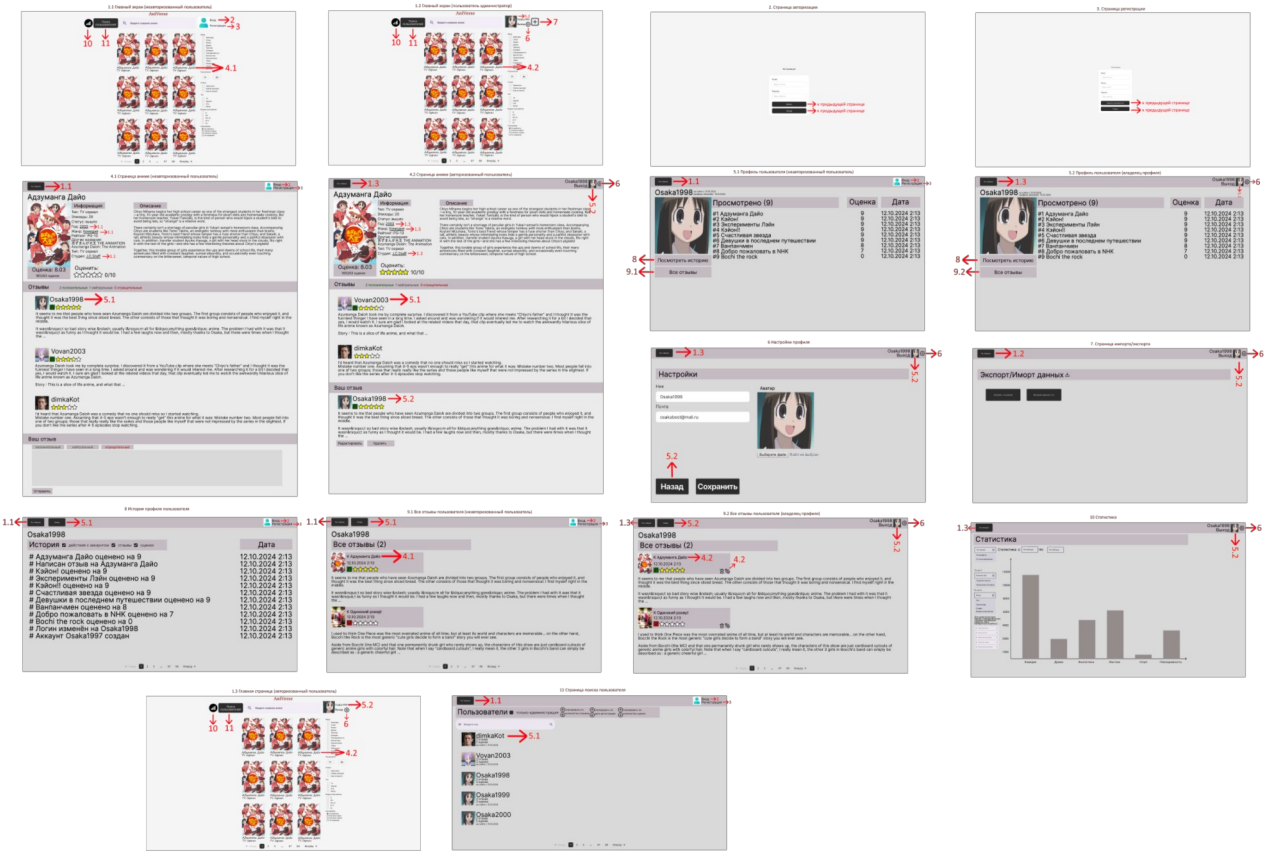


Рисунок 1.Макеты приложения

2.2. Сценарий использования для импорта данных

1. Сценарий использования: Импорт данных

Цель:

Обновить базу данных.

Действующее лицо:

Администратор

Основной сценарий:

1. Администратор попадает на главную страницу.
2. Администратор нажимает кнопку «Импорт/Экспорт».
3. Система отображает страницу загрузки/выгрузки файлов.

4. Администратор выбирает файл с разрешением .json, из которого хочет считать данные.

5. Администратор нажимает кнопку "Загрузить".

6. Система начинает процесс импорта данных.

7. Система сохраняет данные в базе данных.

Альтернативные сценарии:

7а. Неверный формат или имена файлов.

7а.1 Система выводит сообщение об ошибке и предлагает повторить попытку.

Результат:

База данных обновлена.

2. Сценарий использования: Добавление отзыва

Цель:

Пользователь добавляет новый отзыв.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь попадает на главную страницу.

2. Пользователь переходит на страницу аниме.

3. Пользователь вводит отзыв в поле для отзыва.

4. Пользователь нажимает кнопку «Отравить».

5. Система сохраняет данные и обновляет страницу.

Альтернативные сценарии:

4а. Пользователь не проставил оценку

4а.1 Система выводит сообщение об ошибке и предлагает проставить оценку аниме.

4а. Пользователь не заполнил поел ввода.

4а.1 Поля ввода подсвечиваются красным, под полем ввод написано сообщение «Это обязательное поле».

Результат:

Отзыв добавлен в базу данных и отображается на странице аниме.

3. Сценарий использования: Добавление оценки

Цель:

Пользователь добавляет новую оценку.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь попадает на главную страницу.
2. Пользователь переходит на страницу аниме.
3. Пользователь выбивает свою оценку от 1 до 10.
4. Пользователь нажимает кнопку «Отравить».
5. Система сохраняет данные и обновляет страницу.

Альтернативные сценарии:

Результат:

Оценка добавлена в базу данных и отображается на странице аниме.

2.3. Сценарий использования для представления данных

Для просмотра данных существует пять сценариев использования: поиск аниме, поиск пользователя, просмотр страницы аниме, просмотр профиля и просмотр отзывов пользователя.

1. Сценарий использования: Поиск аниме

Цель:

Пользователь просматривает каталог и применяет фильтры для поиска аниме.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь открывает главную страницу приложения.
2. На экране отображается список аниме и фильтры.
3. Пользователь отмечает параметры фильтрации, вводит в поле поиска название аниме.

4. Система обновляет список аниме на основе выбранных фильтров.

5. Пользователь просматривает результаты и выбирает нужное аниме.

Результат:

Пользователь находит интересующее аниме.

2. Сценарий использования: Поиск пользователя

Цель:

Пользователь ищет другого пользователя.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь переходит на страницу поиска пользователей через кнопку « поиск пользователей»

2. Пользователь отмечает параметры фильтрации, вводит в поле поиска логин пользователя.

3. Система обновляет список пользователей на основе выбранных фильтров.

4. Пользователь просматривает результаты и выбирает нужного пользователя.

Результат:

Пользователь находит интересующего его пользователя.

3. Сценарий использования: Просмотр страницы аниме

Цель:

Пользователь просматривает информацию об аниме.

Действующее лицо:

Пользователь

Основной сценарий:

1. Вызывается сценарий «Поиск аниме».

2. Пользователь переходит кликом на страницу нужного ему аниме.

3. Система отображает страницу с информацией и отзывами об аниме.

Результат:

Пользователь просматривает информацию об аниме.

4. Сценарий использования: Просмотр профиля

Цель:

Пользователь просматривает профиль пользователя.

Действующее лицо:

Пользователь

Основной сценарий:

1. Вызывается сценарий «Поиск пользователя».
2. Пользователь переходит кликом на страницу нужного ему пользователя.
3. Система отображает страницу с информацией о пользователе.

Результат:

Пользователь просматривает информацию о пользователе.

5. Сценарий использования: Просмотр отзывов пользователя

Цель:

Пользователь просматривает отзывы пользователя.

Действующее лицо:

Пользователь

Основной сценарий:

1. Вызывается сценарий «Просмотр профиля».
2. Пользователь переходит через кнопку «Отзывы» на страницу отзывов пользователя.
3. Система отображает страницу с информацией о отзывах пользователя.

Результат:

Пользователь просматривает информацию о отзывах пользователя.

2.4. Сценарий использования для анализа данных

Сценарий использования: Подсчет статистики в системе

Цель:

Посмотреть статистику данных в системе.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь попадает на главную страницу.
2. Пользователь нажимает кнопку «Статистика».
3. Пользователь выбирает временной промежуток для построения статистики.
4. Пользователь выбирает критерий, по которым построить статистику (например, аниме, пользователи, средняя оценка и т.д.).
5. Система формирует график со статистикой.
6. Система отображает график пользователю.

Альтернативные сценарии:

Результат:

Пользователь смотрит статистику, построенную на основе выбранных им данных.

2.5. Сценарий использования для экспорта данных

Сценарий использования: Экспорт данных

Цель:

Экспортировать данные в JSON файл.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь попадает на главную страницу.
2. Пользователь нажимает кнопку «Импорт/Экспорт».
3. На странице импорта/экспорта пользователь нажимает кнопку «Загрузить данные».
4. Файл автоматически загружается на устройство пользователя.

Альтернативные сценарии:

Результат:

Данные экспортированы на устройство пользователя.

2.6. Вывод

Для нашего решения пользователи по большей части будут просматривать данные, так как оставление отзывов и комментариев более редкие по сравнению с просмотром информации об аниме, пользователях.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Графическое представление модели

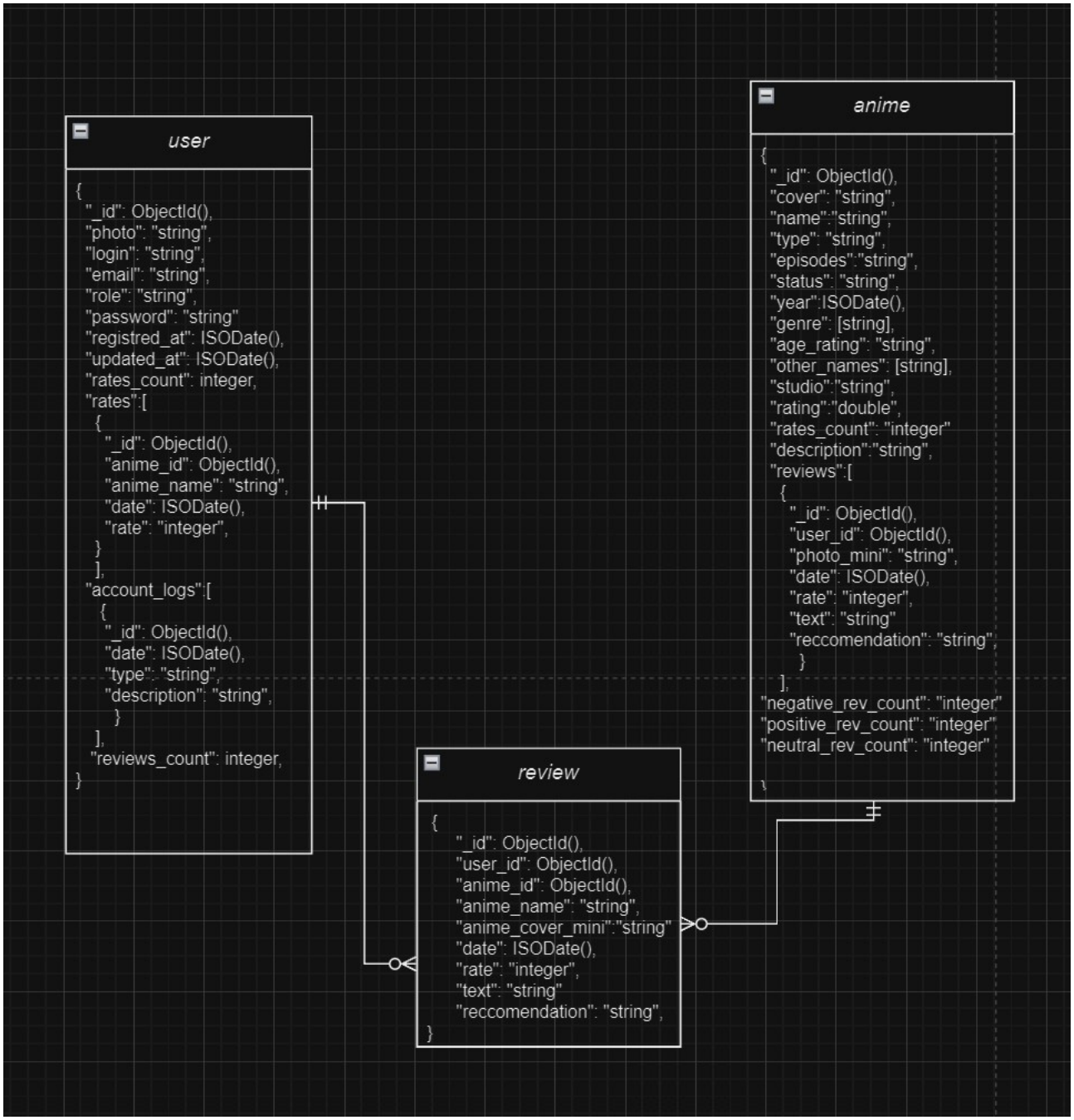


Рисунок 2 - графическое представление модели

Описание назначений коллекций, типов данных и сущностей

Описание коллекций приведено на json схемах ниже.

```
{
  "_id": ObjectId(), //id пользователя - 12 байт
  "photo": "string", // URL фотографии пользователя - в среднем 60 байт
  "login": "string", // Логин пользователя - в среднем 10 байт
  "email": "string", // Почта пользователя - в среднем 30 байт
  "role": "string", // Роль (админ, просто юзер) пользователя - 5 байт
  "password": "string", // Пароль пользователя - 256 байт
  "registred_at": ISODate(), // Дата регистрации - 8 байт
  "updated_at": ISODate(), // Дата обновления профиля - 8 байт
  "rates_count": integer, // Количество оценок - 4 байта
  "rates":[ // Оценки пользователя - в среднем 10 оценок ~ 610 байт
    {
      "_id": ObjectId(), // id оценки - 12 байт
      "anime_id": ObjectId(), // id аниме - 12 байт
      "anime_name": "string", // название аниме - в среднем 25 байт
      "date": ISODate(), // дата оценки - 8 байт
      "rate": "integer", // оценка - 4 байта
    }
    ],
  "account_logs":[ // Изменения аккаунта - в среднем 5 изменений ~ 300 байт
     {
      "_id": ObjectId(), // id изменения - 12 байт
      "date": ISODate(), // дата изменения - 8 байт
      "type": "string", // тип изменения (создание, изменение ника...) - в среднем 10 байт
      "description": "string",    // описание - в среднем 30 байт
     }
    ],
   "reviews_count": integer, // число отзывов пользователя - 4 байта
}
```

Рисунок 3 — схема коллекции user

```
{
  "_id": ObjectId(), // id аниме - 12 байт
  "cover": "string", // url фото обложки - в среднем 60 байт
  "name":"string", // название - в среднем 25 байт
  "type": "string",   // тип (сериал, фильм...)   - в среднем 7 байт
  "episodes":"string", // количество эпизодов  - 2 байта
  "status": "string",     // статус (вышло, выходит..) - в среднем 6 байт
  "year":ISODate(), // год выхода - 8 байт
  "genre": "[string]",   // жанр - в среднем 10 байт
  "age_rating": "string", // возрастной рейтинг - в среднем 3 байта
  "other_names": [string],   //  другие названия - в среднем по 2 названия, в среднем ~ 50 байт
  "studio":"string", // студия - в среднем 15 байт
  "rating":"double", // рейтинг - 8 байт
  "rates_count": "integer" // количество оценок - 4 байта
  "description":"string", // описание - в среднем 1000 байт
  "reviews":[ // отзывы - в среднем 10 отзывов ~ 9040 байт
    {
      "_id": ObjectId(),       //  id отзыва - 12 байт
      "user_id": ObjectId(),  // id пользователя - 12 байт
      "photo_mini": "string",   // url маленького фото пользователя - в среднем 60 байт
      "date": ISODate(),         //  дата написания - 8 байт
      "rate": "integer",        //  оценка - 4 байта
      "text": "string"         // текст отзыва - в среднем 800 байт
      "reccomendation": "string",  //  положительный/отрицательный/нейтральный отзыв - в среднем 8 байт
        }
    ]
   "negative_rev_count": "integer" // кол-во отрицательных отзывов
   "positive_rev_count": "integer"// кол-во положительных отзывов
   "neutral_rev_count": "integer"// кол-во нейтральных отзывов
}
```

Рисунок 4 — схема коллекции anime

```
 {
      "_id": ObjectId(), // id отзыва - 12 байт
      "user_id": ObjectId(),  // id пользователя - 12 байт
      "anime_id": ObjectId(), // id аниме - 12 байт
      "anime_name": "string",  // название аниме - в среднем 25 байт
      "anime_cover_mini":"string", // url маленькой обложки аниме - в среднем 60 байт
      "date": ISODate(),  // дата написания - 8 байт
      "rate": "integer", //  оценка - 8 байт
      "text": "string", //  текст отзыва - в среднем 800 байт
      "reccomendation": "string",  //  положительный/отрицательный/нейтральный отзыв - в среднем 8 байт
}
```

Рисунок 5 — схема коллекции review

Оценка объема информации, хранимой в модели

Средний размер одного документа коллекции:

users - 1307 байт

Общий объем для всех пользователей: u*1307 байт, где u - количество пользователей
anime - 10262 байт

Общий объем для всех пользователей: a*10262 байт, где a - количество аниме
review - 945 байт

Общий объем для всех пользователей: r*945байт, где r - количество отзывов

Если выражать через количество пользователей, то получится следующая формула:

V(u) = (1307 + 945 * 10) * u + (904 * 10 * u + 1222)= 19797 * u + 1222

где u - количество пользователей, принимается в расчёт что каждый оставляет 10 отзывов

Примеры запросов для совершения сценариев использования

1. Поиск аниме с фильтрами

```
db.anime.find({genre: "Комедия", name: "Адзуманга", type: "TV сериал"}).limit(9)
```

2. Аниме с самыми мощными (по количеству) рецензиями

```
db.anime.aggregate([{$project:{ count: { $size:"$reviews" }}}, { $sort: { count: -1 } }])
```

3. Выставление оценки:

```
db.anime.aggregate([
   {
     $match: {_id: "233ca3fb4a4fgc7fa9f5ea23"}
   },
{
$set: {rates_count: {$add: ["$rates_count", 1]}}
},
   {
     $set: {
             rating: {$divide: [ { $add: [ { $multiply: ["$rating", "$rates_count"] }, 10] } ,"$rates_count" ]}
     }
   },

   {
     $merge: {
```

```
        into: "anime",
        whenMatched: "merge"
     }
    }
])
db.user.updateOne({_id: "233ca3fb4a4fgc7fa9f5ea21"}, {$push :
{rates:       {       _id:       ObjectId(),       anime_id:
"233ca3fb4a4fgc7fa9f5ea23",anime_name:    "Anime",date:    new
Date(), rate: 7,} }, $inc: {rates_count: 1} })
```

4. Изменение оценки:

```
db.anime.aggregate([
    {
      $match: {_id: "233ca3fb4a4fgc7fa9f5ea23"}
    },


    {
      $set: {
                  rating:  {$divide:  [  {  $subtract:  [{$add:
[   {   $multiply:   ["$rating",   "$rates_count"]   },   10]},
7] } ,"$rates_count" ]}
      }
    },

    {
      $merge: {
        into: "anime",
        whenMatched: "merge"
     }
    }
])
db.user.updateOne({_id:            "233ca3fb4a4fgc7fa9f5ea21",
"rates._id":  "233ca3fb4a4fgc7fa9f5ea29"},  {$set  :  {rates.
$.rate: 7, rates.$.date: new Date()} } })
```

5. Добавление отзыва:

```
db.review.insertOne( {
      user_id: "233ca3fb4a4fgc7fa9f5ea21",
      anime_id:  "233ca3fb4a4fgc7fa9f5ea23",
      anime_name: "Anime",
      anime_cover_mini:"https://anime.com/anime_mini.jpg",
      date: new Date(),
      rate: 7,
      text: "good"
     reccomendation: "positive"
```

```
})
db.anime.updateOne({_id: "233ca3fb4a4fgc7fa9f5ea23"},
{$push    :    {reviews:    {    _id:    ObjectId(),    user_id:
"233ca3fb4a4fgc7fa9f5ea21",
      photo_mini:"https://anime.com/me_mini.jpg",
      date: new Date(),
      rate: 7,
      text: "good",
     reccomendation: "positive"} },$inc: { positive_rev_count:
1}  })
db.user.updateOne({_id:"233ca3fb4a4fgc7fa9f5ea21"},{$inc:
{reviews_count: 1}} )
```

6. Просмотр отзывов пользователя:

```
db.review.find({user_id:
"233ca3fb4a4fgc7fa9f5ea21"}).limit(2)
```

7. Просмотр пользователей по увеличению количества отзывов:

```
db.user.find().sort({reviews_count: 1}).limit(5)
```

8. Редактирование отзыва:

```
db.anime.updateOne({_id:           "233ca3fb4a4fgc7fa9f5ea23",
"reviews._id": "233ca3fb4a4fgc7fa9f5ea27"}, {$set : {reviews.
$.text:  "norm"  }  ,$currentDate:  {  updated_at:  true,  date:
true } })
db.review.updateOne({_id:         "233ca3fb4a4fgc7fa9f5ea27"},
{$set  :  {text:  "norm"  }  ,$currentDate:  {  updated_at:  true,
date: true } })
```

9. Удаление отзыва:

```
db.anime.deleteOne({_id:"233ca3fb4a4fgc7fa9f5ea23",
"rewirws._id": "671aba5d6cf54cffac86b024"})
db.anime.updateOne({_id:"233ca3fb4a4fgc7fa9f5ea23"},    {$inc:
{positive_rev_count: -1}})
db.review.deleteOne({_id: "671aba5d6cf54cffac86b024"})
```

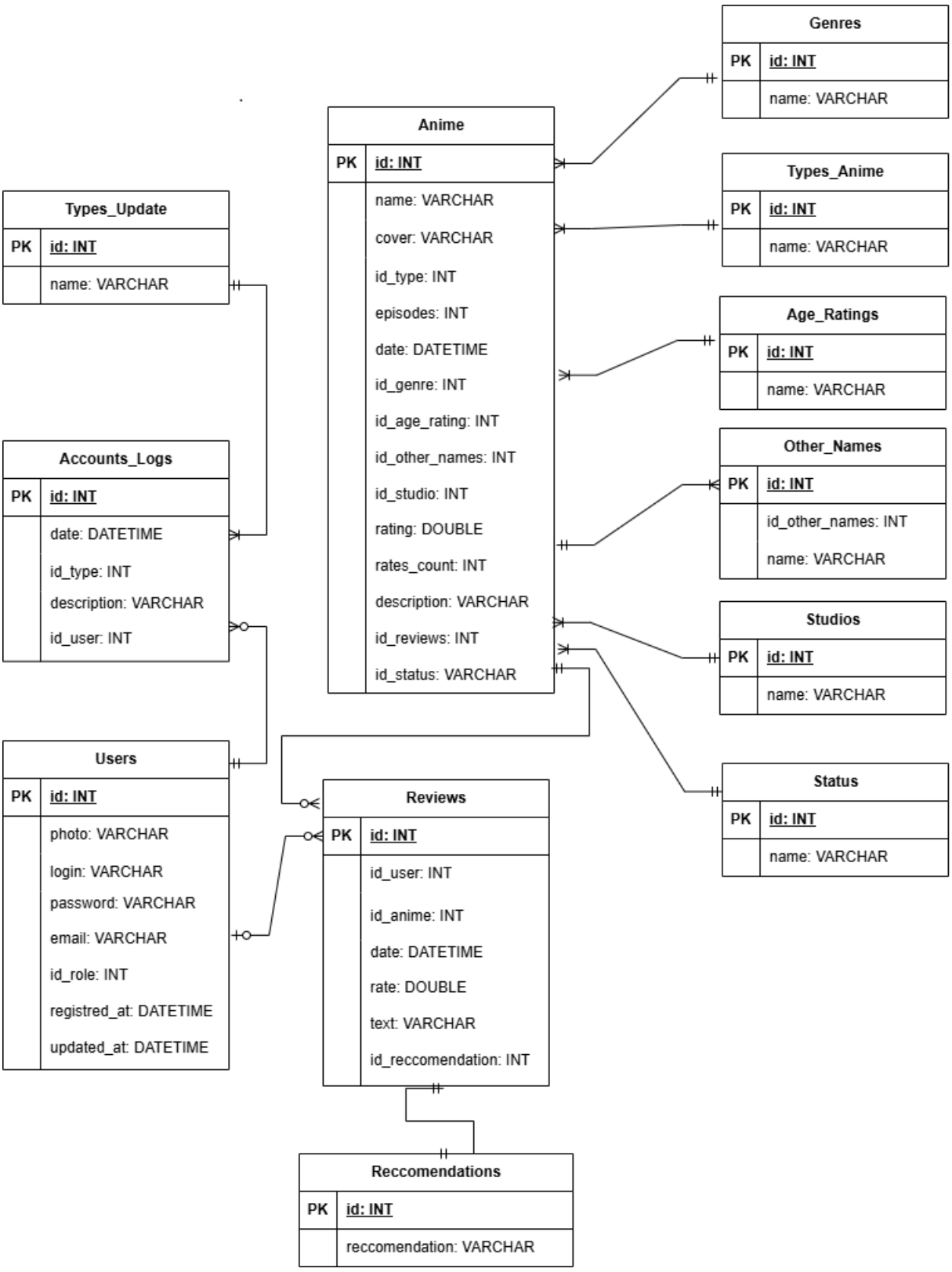
11. Вычисление поля rating

```
db.anime.aggregate([
  {
          $match: {
              _id: "233ca3fb4a4fgc7fa9f5ea23",
          }
```

```
    },
    {
        $lookup: {
            from: "user",
            localField: "_id",
            foreignField: "rates.anime_id",
            as: "rates"
        }
    },
    {
      $project: {
       "rates": "$rates.rates",
     }
   },
   { $unwind: "$rates" },
   { $unwind: "$rates" },
   {
    $match: {
        "rates.anime_id": "233ca3fb4a4fgc7fa9f5ea23",
    }
  },
  {
    $group:
      {
        _id: "233ca3fb4a4fgc7fa9f5ea23",
        sumrate: { $sum: "$rates.rate" },
        count: {$sum: 1}
      }
  },
  { $set: {rating: {$divide: ["$sumrate", "$count"] } } },
    {
      $project: {
       "rating": "$rating",
     }
   },
    {
      $merge: {
        into: "anime",
        whenMatched: "merge"
     }
    }
])
```

3.2. Аналог модели данных для SQL СУБД

Графическое представление модели



Описание коллекций

Anime

Назначение: хранение информации об аниме.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

cover: VARCHAR(N + 1)

id_type: INT(4)

episodes: INT(4)

date: DATETIME(8)

id_genre: INT(4)

id_age_rating: INT(4)

id_other_names: INT(4)

id_studio: INT(4)

rating: DOUBLE(8)

rates_count: INT(4)

description: VARCHAR(N + 1)

id_reviews: INT(4)

Genres

Назначение: хранение информации о жанрах аниме.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

Types_Anime:

Назначение: хранение информации о типах аниме.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

Age_Ratings

Назначение: хранение информации о возрастных рейтингах аниме.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

Other_Names

Назначение: хранение информации о других названиях аниме.

Поля:

id: INT(4)

id_other_names: INT(4)

name: VARCHAR(N + 1)

Studios

Назначение коллекции: хранение информации о студиях-производителях аниме.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

Status

Назначение коллекции: хранение информации о статусе, в котором сейчас находится данное аниме.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

Users

Назначение: хранение информации о зарегистрированных пользователях приложения

Поля:

id: INT(4)

photo: VARCHAR(N + 1)

login: VARCHAR(N + 1)

password: VARCHAR(N + 1)

email: VARCHAR(N + 1)

id_role: INT(4)

registred_at: DATETIME(8)

updated_at: DATETIME(8)

Account_Logs

Назначение: хранение логов действий пользователя со своим профилем в приложении.

Поля:

id: INT(4)

date: DATETIME(8)

id_type: INT(4)

description: VARCHAR(N + 1)

id_user: INT(4)

Types_Update

Назначение: хранение типов действий пользователя со своим профилем.

Поля:

id: INT(4)

name: VARCHAR(N + 1)

Reviews

Назначение: хранение информации об отзывах, оставленных пользователями на аниме.

Поля:

id: INT(4)

id_user: INT(4)

id_anime: INT(4)

date: DATETIME(8)

rate: DOUBLE(8)

text: VARCHAR(N + 1)

id_reccomendation: INT(4)

Reccomendations

Назначение: хранение отзывов на аниме, оставленных пользователями.

Поля:

id: INT(4)

reccomendation: VARCHAR(N + 1)

Оценка объема информации, хранимой в модели

Предположительно коллекции:

Genres, Types_Anime, Age_Ratings, Other_Names, Studios, Types_Update будут иметь около 20 записей;

коллекция Anime будет иметь около 10000 записей;

коллекция Accounts_Logs будет иметь 1000 записей на каждого пользователя;

Коллекции Reviews и Reccomendations будут иметь по 10 записей на каждого пользователя.

При количестве пользователей U имеем следующий объем занятого места:

V(U) = 10000 * 355 + 20 * 105 + 20 * 105 + 20 * 105 + 10 * 105 + 10 * 109 + U * 428 + 1000 * U * 117 + 20 * 105 + 10 * 10000 * U * 133 + 10 * 10000 * U * 105.

После приведения подобных слагаемых получим следующее выражение:

V(U) = 23917428 * U + 3560540 (байт)

Выразим полученный результат в Мбайт:

V(U) = 22.8 * U + 3.4 (Мбайт)

Примеры запросов

1. Получение страницы с аниме:

```
SELECT Anime.name, Anime.cover, Anime.episodes, Anime.date,
Anime.rating, Anime.rates_count, Anime.description,
Genres.name, Types_Anime.name, Age_Ratings.name,
Other_Names.name, Studios.name, Reviews.date, Reviews.rate,
Reviews,text, Reccomendations.reccomendation, User.photo,
User.login FROM Anime
INNER JOIN Genres
ON Anime.id_genre = Genres.id
INNER JOIN Types_Anime
ON Anime.id_type = Types_Anime_id
INNER JOIN Age_Ratings
ON Anime.id_age_rating = Age_Ratings.id
INNER JOIN Other_Names
ON Anime.id_other_names = Other_Names.id_other_names
INNER JOIN Studios
ON Anime.id_studio = Studios.id
INNER JOIN Reviews
ON Anime.id_reviews = Reviews.id_anime
INNER JOIN Users
ON Reviews.id_user = User.id
INNER JOIN Reccomendations
ON Reviews.id_reccomendation = Reccomendations.reccomendation
WHERE name = "anime_name";;
```

2. Профиль пользователя:

```
SELECT photo, login, registred_at, updated_at, Reviews.text,
Reviews.date, Anime.name, Anime.rating
FROM Users
INNER JOIN Reviews
ON Users.id = Reviews.id_user
INNER JOIN Anime
ON Reviews.id_anime = Anime.id
WHERE login = "Osaka1998";
```

3. Изменение профиля пользователя:

```
UPDATE Users SET login = "Makaka_228", password = "key",
photo = "url1"
WHERE login = "Osaka1998";
```

4. История профиля пользователя:

```
SELECT login, Types_Update.name, Accounts_Logs.date,
Accounts_Logs.description
FROM Users
INNER JOIN Accounts_Logs
ON Users.id = Accounts_Logs.id_user
INNER JOIN Types_Update
ON Accounts_Logs.id_type = Types_Update.id
WHERE login = "Osaka1998";
```

5. Все отзывы пользователя:

```
SELECT Anime.name, Reviews.date, Reviews.rate, Reviews.text
FROM Users
INNER JOIN Reviews
ON Users.id = Reviews.id_user
INNER JOIN Anime
ON Reviews.id_anime = Anime.id
WHERE login = "Osaka1998";
```

6.Аниме с самыми мощными (по количеству) рецензиями:

```
SELECT Anime.name, Anime.cover, Anime.episodes, Anime.date,
Anime.rating, Anime.rates_count, Anime.description,
Genres.name, Types_Anime.name, Age_Ratings.name,
Other_Names.name, Studios.name FROM Anime
INNER JOIN Genres
ON Anime.id_genre = Genres.id
INNER JOIN Types_Anime
ON Anime.id_type = Types_Anime_id
INNER JOIN Age_Ratings
ON Anime.id_age_rating = Age_Ratings.id
INNER JOIN Other_Names
ON Anime.id_other_names = Other_Names.id_other_names
INNER JOIN Studios
ON Anime.id_studio = Studios.id
INNER JOIN Reviews
ON Anime.id_reviews = Reviews.id_anime
GROUP BY Anime.id
ORDER BY COUNT(*) DESC
LIMIT 20;
```

7. Подсчет и заполнение вычисляемого поля rating:

```
UPDATE Anime
SET rating = (
    SELECT AVG(rating)
```

```
    FROM Reviews
    WHERE Reviews.id_anime = Anime.id
);
```

8. Подсчет и заполнение вычисляемого поля reviews_count:

```
UPDATE Users
SET reviews_count = (
    SELECT COUNT(*)
    FROM Reviews
    WHERE Reviews.id_user = Users.id
);
```

9. Подсчет и заполнение вычисляемого поля rates_count:

```
UPDATE Anime
SET review_count = (
    SELECT COUNT(*)
    FROM Reviews
    WHERE Reviews.id_anime = Anime.id
);
```

10. Аниме, у которых самые полярные рецензии (много как положительных, так и отрицательных рецензий):

```
SELECT Anime.name, Anime.cover, Anime.episodes, Anime.date,
Anime.rating,      Anime.rates_count,      Anime.description,
Genres.name,        Types_Anime.name,        Age_Ratings.name,
Other_Names.name, Studios.name FROM Anime
INNER JOIN Genres
ON Anime.id_genre = Genres.id
INNER JOIN Types_Anime
ON Anime.id_type = Types_Anime_id
INNER JOIN Age_Ratings
ON Anime.id_age_rating = Age_Ratings.id
INNER JOIN Other_Names
ON Anime.id_other_names = Other_Names.id_other_names
INNER JOIN Studios
ON Anime.id_studio = Studios.id
INNER JOIN Reviews
ON Anime.id_reviews = Reviews.id_anime
GROUP BY Anime.id
HAVING AVG(Reviews.rate) >= 3 AND AVG(Reviews.rate) <= 7
ORDER BY COUNT(*) DESC
LIMIT 20;
```

3.3. Сравнение моделей

Удельный объем информации

Основываясь на расчётах объемов информации в пункте "Оценка удельного объема информации" для нереляционных и реляционных БД, можно сделать следующее сравнение.

Разница общих объемов нереляционной и реляционной БД соответственно:

V(U) = (19797 * U + 1210) - (23917428 * U + 3560540) = -2399761 * U - 3560330

Разница чистых объемов нереляционной и реляционной БД соответственно:

V_clean(U) = (10757 * U + 1330) - (4088 * U + 240000) = 6669 * U - 238670

В реляционных БД данные занимают больше объема. Однако, при расчете "чистых" объемов реляционные БД оказываются более "легкими".

Запросы по отдельным юзкейсам

Количество запросов для каждой модели одинаково и равно 1.

Количество задействованных коллекций

Количество задействованных коллекций в нереляционной модели равно трем(users, anime, review). В реляционной же БД это количество будет значительно больше, поскольку каждая подколлекция выражается в дополнительную сущность (Anime, Genres, Types_Anime, Age_Ratings, Other_Names, Studios, Status, Users, Account_Logs, Types_Update, Reviews, Reccomendations).

Вывод

В ходе сравнения реляционных и нереляционных баз данных можно выделить несколько ключевых моментов.

1. Удельный объем информации: Реляционные БД демонстрируют больший общий объем, однако при рассмотрении "чистых" объемов, то есть объема информации без учета вспомогательной структуры, оказывается, что реляционные модели имеют меньший объем.

2. Запросы по отдельным юзкейсам: Нереляционная БД использует меньшее количество коллекций, что приводит к более простой модели данных и меньшему числу необходимых запросов. Также реляционная БД требует гораздо большего количества сущностей, что может усложнить процесс запросов и увеличивает траты ресурсов.

Таким образом, в данном проекте нереляционная модель данных подходит лучше поскольку выигрывает по простоте реализации, а также обеспечивают гибкость и простоту при работе с менее структурированными данными.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1 Краткое описание

Весь код приложения разделен на две части: back-end и front-end.

Back-end, база данных и front-end существуют в отдельных контейнерах и запускаются через docker-compose.

Back-end реализован с использованием фреймворка ASP.NET и служит элементом, обеспечивающим через API получение необходимых данных из БД. Сам Back-end разделён на контроллеры и сервисы, контроллеры служат как непосредственно API, сервисы управляют запросами к БД для конкретных сущностей. Так же там реализована необходимая бизнес-логика, параметры авторизации, аутентификации.

Front-end обращается к API back-end части приложения и отображает полученные данные. При разработке использовались компоненты React, что позволяет использовать их повторно. Практически все страницы имеют элементы настройки — чекбоксы, поля ввода, списки. Стиль страниц выдержан в ностальгическом стиле ранних 2000-ых.

4.2 Использованные технологии

БД: MongoDB.

Back-end: ASP.NET.

Front-end: React.

4.3 Схема экранов приложения

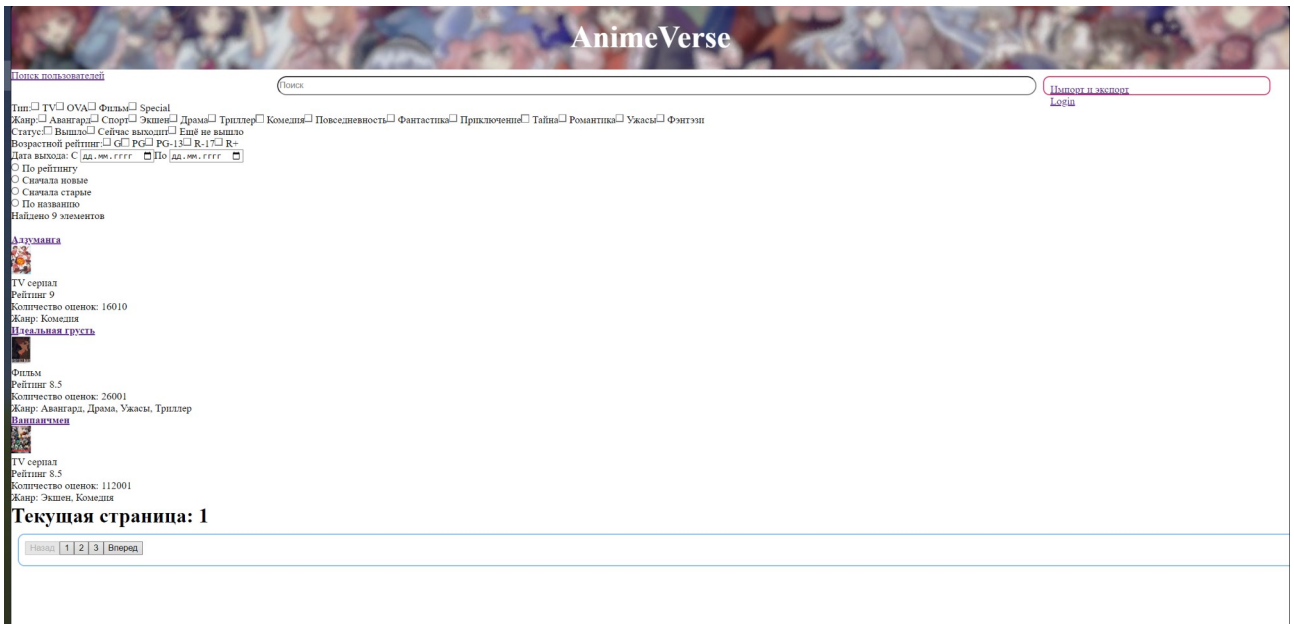


Рисунок 6— главный экран

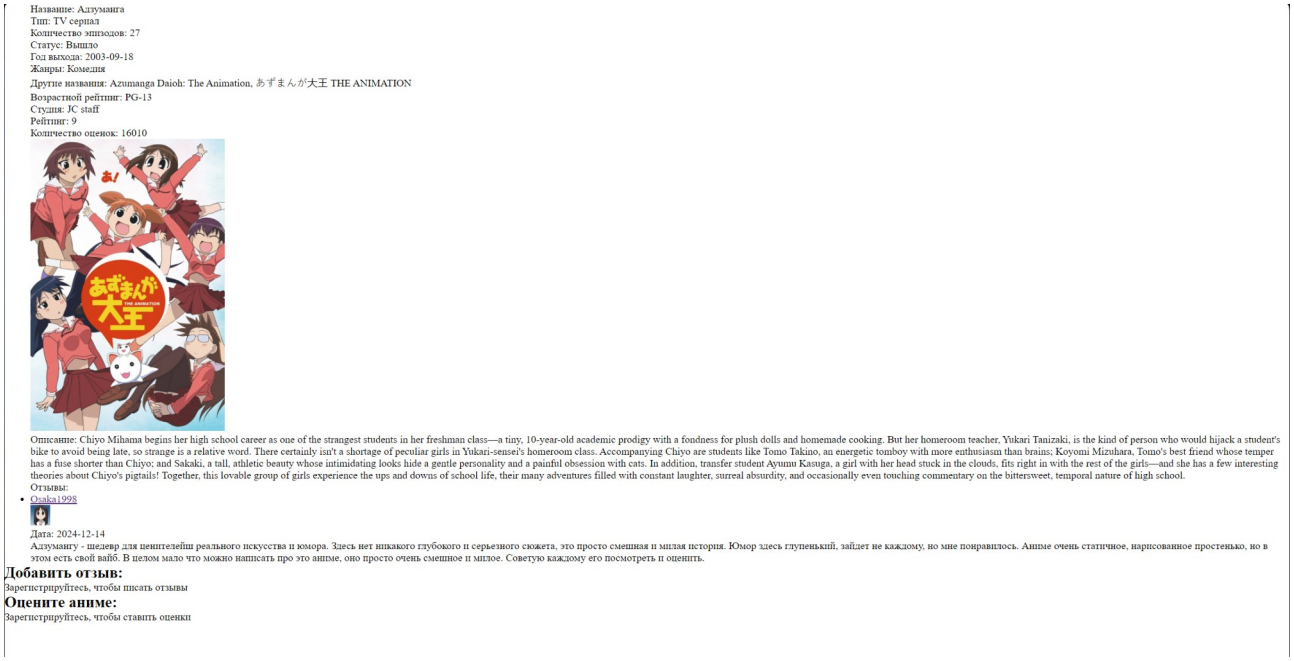


Рисунок 7 — страница аниме

5. ВЫВОДЫ

5.1 Достигнутые результаты

В ходе работы было разработано приложение "Aniverse", представляющее собой каталог аниме, которое позволяет пользователям просматривать информацию об аниме, добавлять, удалять и редактировать отзывы и ставить оценки. Приложение ориентировано на предоставление пользователям эффективного инструмента для поиска и оценки аниме.

Приложение предоставляет гибкие возможности для поиска и фильтрации аниме, пользователей. Для удобства бэкапа доступны функции импорта и экспорта в JSON.

5.2 Недостатки и пути для улучшения

Необходимо привести стиль текущих страниц приложения к виду, обозначенному на макетах.

Необходимо доделать аутентификацию и авторизацию.

5.3 Будущее развитие решения

В будущем планируется внедрение авторизации с разграничением прав доступа. Также будет добавлена функция добавления в друзья пользователей и формирование личных списков.

6. ПРИЛОЖЕНИЯ

6.1 Документация по сборке и развертыванию приложения.

1. Склонировать репозиторий с проектом (ссылка указана в списке литературы) и перейти в директорию проекта.
2. Собрать контейнеры приложения командой: *docker-compose build --no-cache*.
3. Запустить контейнеры командой: *docker-compose up*.
4. Открыть приложение в браузере по адресу 127.0.0.1:8000 или нажав на порт контейнера frontend в приложении Docker Desktop.

6.2 Инструкция для пользователя.

1. Массовый импорт-экспорт данных.

При массовом импорте-экспорте используется 1 файл формата .json. При массовом экспорте этот файл будет скачан на компьютер пользователя. При массовом экспорте необходимо проверить, правильная ли структура выбранного json файла, иначе система выведет ошибку.

2. Добавление новой оценки.

Для того, чтобы добавить новую оценку, необходимо на странице аниме в выпадающем списке выбрать десятибалльную оценку. После этого нажать кнопку «отправить».

3. Оставление отзыва на аниме.

Для того, чтобы оставить отзыв на аниме, необходимо пролистать страницу с информацией об аниме в самый низ, до блока с полем ввода и ввести туда свой комментарий. После его ввода нужно нажать кнопку «Отправить». Если поле не будет заполнено, приложение сообщит об этом пользователю.

4. Редактирование профиля

Для редактирования профиля необходимо нажать соответствующую кнопку в своём профиле, после этого откроется страница редактирования.

После изменения необходимых полей нужно нажать кнопку «Сохранить изменения»

7. ЛИТЕРАТУРА

1. Ссылка на GitHub. - [Электронный ресурс]. - URL: https://github.com/moevm/nosql2h24-anime.
2. React – [Электронный ресурс]. – URL: https://react.dev (дата обращения: 22.12.2024).
3. MongoDB. – [Электронный ресурс]. – URL: https://www.mongodb.com/ (дата обращения: 22.12.2024).
4. ASP.NET – [Электронный ресурс]. – URL: https://www.asp.net/?I (дата обращения: 22.12.2024).