

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационных систем и технологий
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Web-приложение «Сайт трейлеров»

Выполнил студент Олексюк Андрей Викторович
(Ф.И.О.)
Руководитель проекта преп.-стаж. Чурак Е.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)
Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)
Консультанты преп.-стаж. Чурак Е.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)
Нормоконтролер преп.-стаж. Чурак Е.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)
Курсовой проект защищен с оценкой _____

Минск 2018

Содержание

Введение.....	3
1. Постановка задачи.....	4
2. Архитектура приложения	5
3. Функциональные особенности приложения.....	9
4. Руководство пользователя	10
5. Тестирование.....	13
Заключение	14
Список литературы	15
Приложение А	16
Приложение Б.....	20
Приложение В.....	22

Введение

В данном курсовом проекте требовалось разработать кроссплатформенное веб-приложение, построенное на REST-архитектуре, на тему «Сайт трейлеров».

В ходе разработки было создано само приложение, были проведены интеграционные тесты, была составлена документация к API приложения и написана пояснительная записка.

Пояснительная записка состоит из введения, пяти разделов, списка литературы, заключения и приложения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается подход, который использовался при разработке, а также список основных классов приложения.

Во втором разделе представлена архитектура приложения и базы данных.

В третьем разделе описываются особенности данного программного средства.

В четвертом разделе представлено руководство пользователя.

В пятом разделе представлены результаты тестирования приложения.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе проектирования и разработки приложения.

1. Постановка задачи

При реализации курсового проекта использовались технологии шаблонизатор Pug для реализации front-end, фреймворк Express на базе Node.js для разработки back-end и MySQL в качестве базы данных.

1.1 Описание архитектуры

Курсовой проект построен на многоуровневой архитектуре (N-layer architecture). Данная архитектура основана на следующих принципах:

1. Проектирование чётко устанавливает разграничение функций между уровнями;
2. Нижние уровни независимы от верхних уровней;
3. Верхние уровни вызывают функции нижних уровней, но при этом взаимодействуют только соседние уровни иерархии.

Для данного курсового проекта были разработаны следующие уровни:

1. Уровень «Модель» - описание таблиц в базе данных;
2. Уровень «Контекст» - описание связей между моделями в базе данных и отображение данных в объекты приложения;
3. Уровень «Сервис» - создание CRUD-интерфейса для работы с таблицами в базе данных, а так же имеются сервисы авторизации, проверяющие роли пользователя;
4. Уровень «Контроллер» - описание классов, методы которых предназначены для создания ответа пользователю по его запросу.
5. Уровень «Авторизация» - назначение пользователю прав и привилегий для работы с объектами базы данных.
6. Уровень «Аутентификация» - проверка подлинности пользователя.

Каждый запрос клиента проходит как минимум через 3 уровня («Аутентификация», «Авторизация» и «Контроллер»), после чего клиент получает ответ. В случае успешной аутентификации и авторизации клиента сервер также может задействовать другие уровни приложения.

2. Архитектура приложения

2.1. Структура базы данных

В базе данных (далее БД) хранятся данные о фильмах, трейлеров, пользователей и комментариев. В приложении используется реляционная БД (MySQL), которая содержит 4 таблицы для хранения данных:

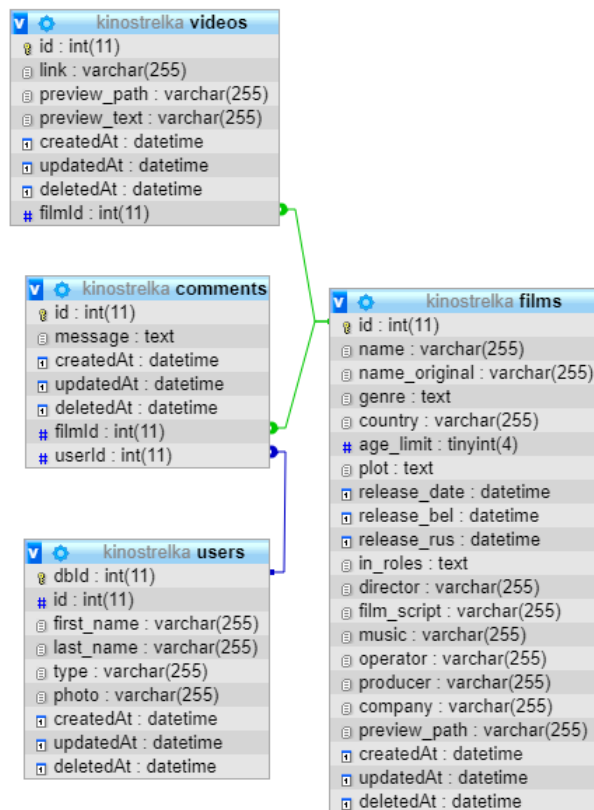


Рисунок 2.1 – Общая схема базы данных

Таблица users – содержит информацию о всех пользователях.

Таблица comments – содержит информацию о всех комментариях.

Таблица videos – содержит данные о каждом видео.

Таблица films – содержит данные о каждом фильме.

2.2. Структура курсового проекта

В решении курсового проекта была использована n-layer архитектура. Представлена на рисунке 2.2. Эта архитектура выбрана так как она идеально подходит для приложений с длительным жизненным циклом и сложной бизнес логикой.

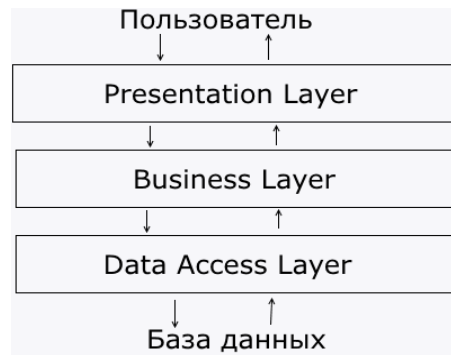


Рисунок 2.2 – Общая схема n-layer архитектуры

Для данного курсового проекта схема приложения выглядит как показано на рисунке 2.3.

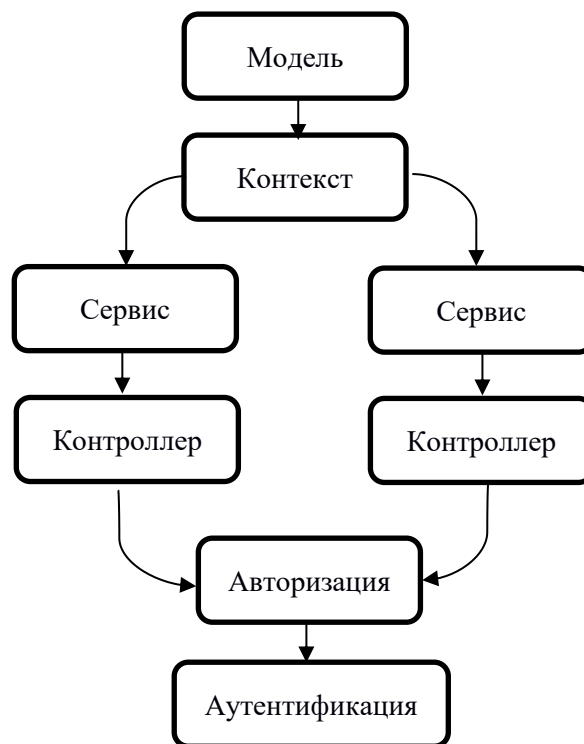


Рисунок 2.3 – Структура проекта

В папках context и models находятся модели, контекст и репозитории для взаимодействия с БД. Более подробное взаимодействие между моделями и контекстом представлено на рисунке 2.4.

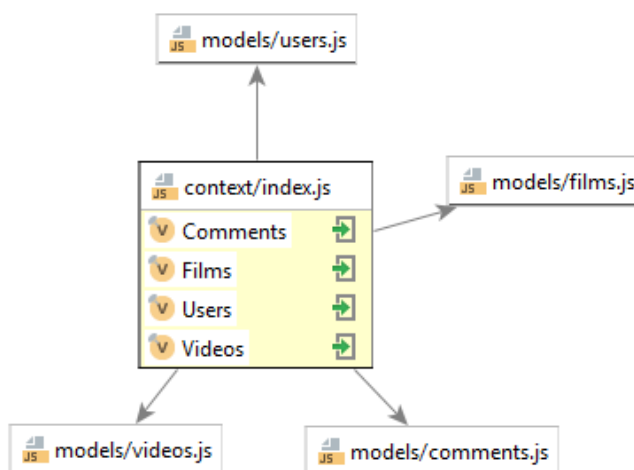


Рисунок 2.4 – UML диаграмма контекста и моделей проекта

В папке `services` представлены классы сервисов, которые используются во всех проектах. UML диаграмма слоя «Сервис» представлена на рисунке 2.5.

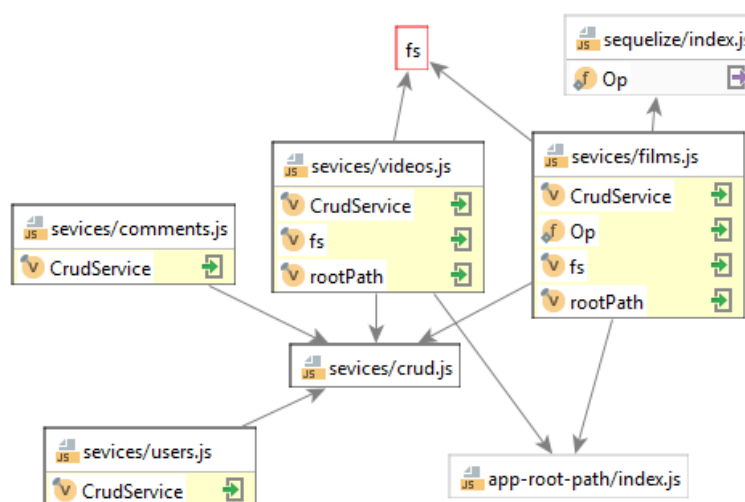


Рисунок 2.5 – UML диаграмма сервисов проекта

В папке `controllers` находятся контроллеры, которые связывают модели и сервисы. При обращении с UI мы вызываем методы контроллеров, а они в свою очередь передают эти данные для обработки в сервисы. При вызове методов контроллеров мы обращаемся к сервисам авторизации для разрешения их выполнения. В случае, если клиент не имел права на выполнение метода, то ему будет возвращено «access denied». UML диаграмма контроллеров представлена на рисунке 2.6.

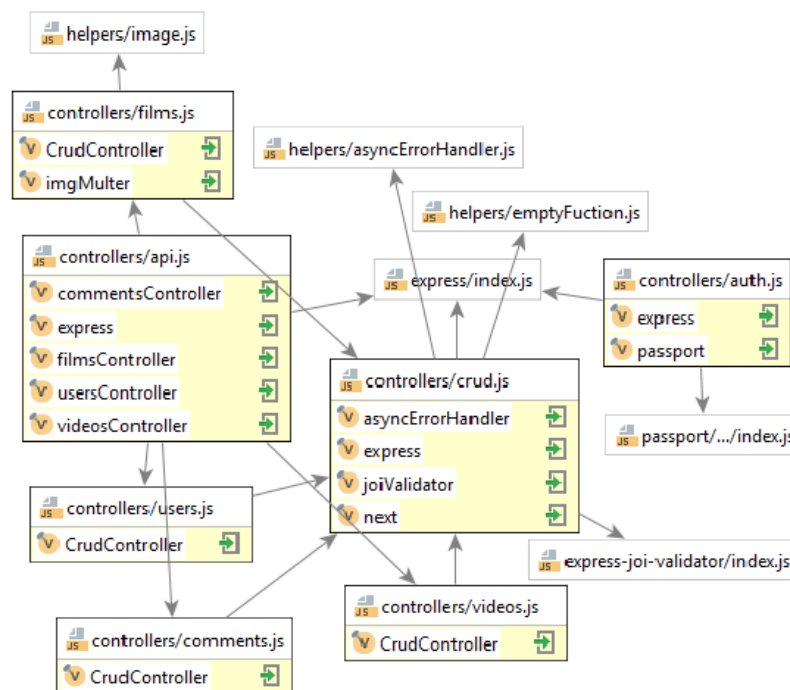


Рисунок 2.6 – UML диаграмма контроллеров проекта

В папке `helpers` располагаются различные модули, к которым могут обращаться все слои приложения и служат для централизации определённой логики в одном месте. Например, функции сохранения изображений на сервере, а так же функция авторизации OAuth 2.0. UML диаграмма вспомогательных функций представлена на рисунке 2.7.

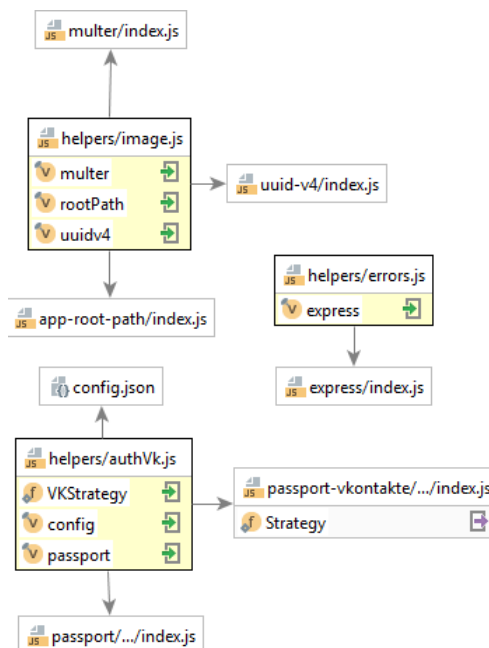


Рисунок 2.7 – UML диаграмма вспомогательных функций

Таким образом в данном программном средстве осуществляется обработка всех данных.

3. Функциональные особенности приложения

В данном программном средстве используется система аутентификации и авторизации.

В приложении реализованы три вида пользователей: анонимный пользователь, аутентифицированный пользователь и администратор.

Анонимный пользователь имеет возможность просматривать список фильмов, трейлеров и комментариев.

Аутентификация в проекте реализована при помощи OAuth 2.0. Это протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению логин и пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу. В данном случае аутентификация происходит через социальную сеть «ВКонтакте».

В отличие от анонимного пользователя аутентифицированный еще имеет право оставлять комментарии.

Администратор в свою очередь имеет доступ к API и может добавлять фильмы и трейлеры к ним. Следует отметить, что администратор может назначаться только системным администратором БД, что ограничивает возможности для злоумышленников.

Все запросы к контроллерам приложения выполнены в стиле REST.

В случае запроса несуществующей страницы пользователь будет переводиться на домашнюю страницу сайта.

Так же, при добавлении фильма на сервер отправляется картинка, которая сохраняется на сервере, для дальнейшего использования ее на сайте.

4. Руководство пользователя

Для получения полного доступа к комментированию, анонимному пользователю предлагается форма аутентификации «Вконтакте» представленная на рисунке 4.1.

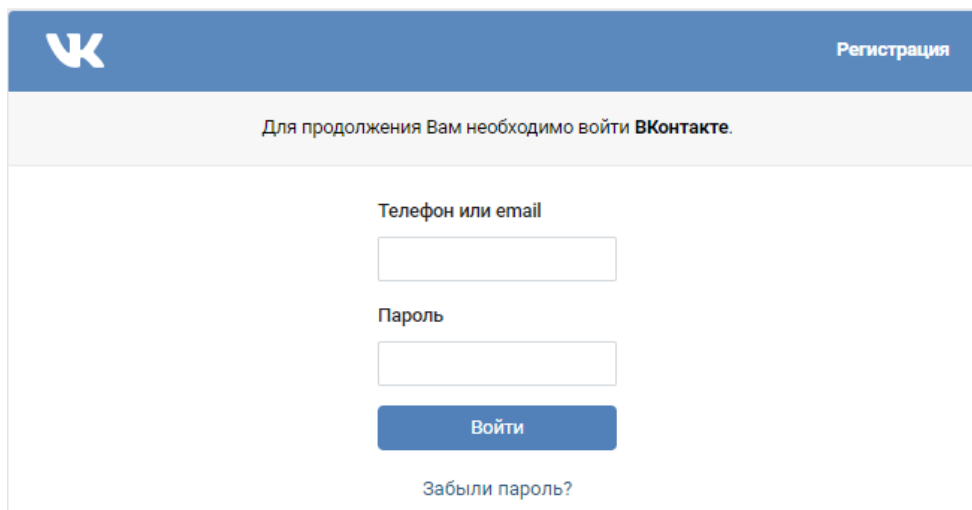


Рисунок 4.1 – форма аутентификации «Вконтакте»

После аутентификации пользователь увидит свои имя и аватар. Пример представлен на рисунке 4.2.

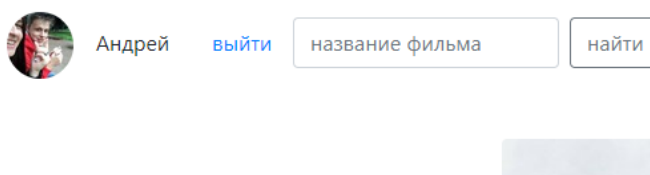


Рисунок 4.2 – Страница аутентифицированного пользователя

После успешной аутентификации пользователь может переходить по фильмам и оставлять комментарии. Пример представлен на рисунке 4.3. При попытке отправить пустой комментарий пользователю выведется соответствующее сообщение на экран. Пример на рисунке 4.4.

отправить

Андрей 5/24/2018, 6:30 PM
wwwww



Андрей 5/24/2018, 11:19 AM
aaaa



Андрей 5/24/2018, 11:19 AM
lalala



Андрей 5/24/2018, 7:08 PM
1212



Андрей 5/24/2018, 7:09 PM
222

Рисунок 4.3 – Комментарии на странице фильма

Подтвердите действие на странице localhost:3000
Введите текст комментария

OK

Рисунок 4.4 – Попытка отправить пустой комментарий

Так же на странице фильмов есть возможность найти необходимый фильм. Пример на рисунке 4.5. Если ничего не нашлось на странице отображается соответствующее сообщение. Пример на рисунке 4.6.



Андрей

выйти

название фильма

найти

Рисунок 4.5 – Поле поиска фильма

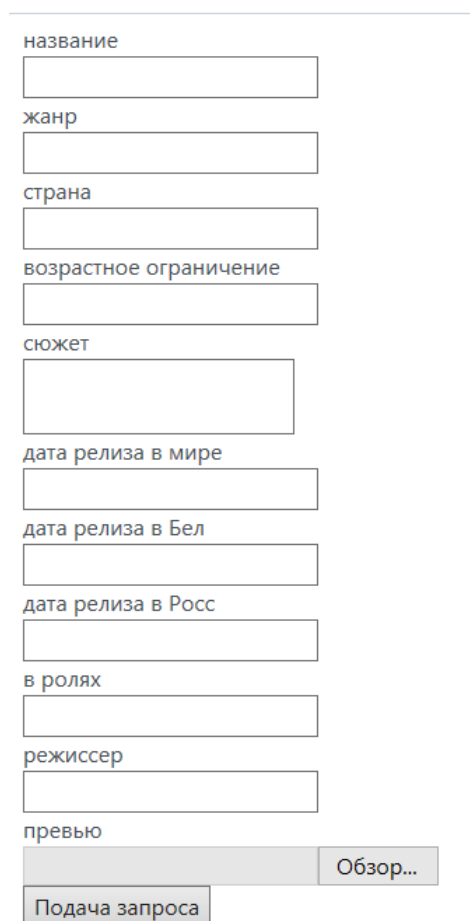
12345

найти

Ничего не найдено

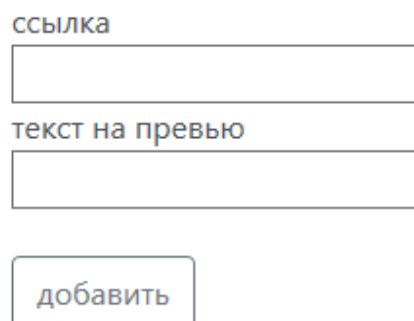
Рисунок 4.6 – Результат неудачного поиска

Если же пользователь является администратором, то на странице отображаются формы для добавления фильмов и трейлеров. Они представлены на рисунках 4.7 и 4.8 соответственно.



A vertical form for adding a movie. It consists of several text input fields, each preceded by a label: 'название' (title), 'жанр' (genre), 'страна' (country), 'возрастное ограничение' (age rating), 'сюжет' (plot), 'дата релиза в мире' (world release date), 'дата релиза в Бел' (Belarus release date), 'дата релиза в Росс' (Russia release date), 'в ролях' (cast), 'режиссер' (director), and 'превью' (preview). The 'превью' field is a larger area with a light gray background. To the right of this area is a button labeled 'Обзор...'. Below the preview area is a button labeled 'Подача запроса'.

Рисунок 4.7 – Форма администратора для добавления фильма



A vertical form for adding a trailer. It consists of two text input fields: the first is labeled 'ссылка' (link) and the second is labeled 'текст на превью' (text on preview). Below these fields is a button labeled 'добавить' (add).

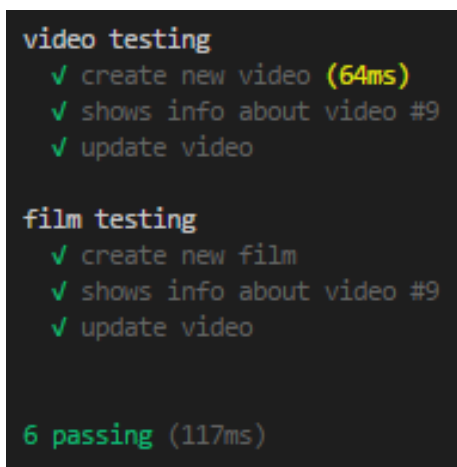
Рисунок 4.8 – Форма администратора для добавления трейлера

И конечно же каждый пользователь на странице фильма может насладиться трейлерами к нему и обсудить свои впечатления в комментариях.

5. Тестирование

На основную логику проекта были разработаны и успешно выполнены интеграционные тесты. Интеграционные тесты дают уверенность, что ваша программа работает как задумано. Такие тесты можно запускать многократно. Успешное выполнение тестов покажет разработчику, что его изменения не сломали ничего, что ломать не планировалось.

Провалившийся тест позволит обнаружить, что в коде сделаны изменения, которые меняют или ломают его поведение. Исследование ошибки, которую выдает провалившийся тест, и сравнение ожидаемого результата с полученным даст возможность понять, где возникла ошибка, будь она в коде или в требованиях. На рисунке 5.1. представлены результаты выполненных тестов.



```
video testing
✓ create new video (64ms)
✓ shows info about video #9
✓ update video

film testing
✓ create new film
✓ shows info about video #9
✓ update video

6 passing (117ms)
```

Рисунок 5.1 – Результаты выполнения тестов

Также в проекте реализована валидация на стороне сервера на случай неочевидного или неправильного поведения пользователя.

В данном курсовом проекте валидируются все поля ввода, на которых в случае некорректного ввода, может произойти ошибка или непредвиденное поведение программы. Валидация приложения – это один из основных гарантов надёжности приложения и всякий программист должен предусмотреть и предотвратить непредвиденное поведение пользователя.

Заключение

Решая поставленную задачу, пришли к удовлетворительному результату. Таким образом, была достигнута цель и создана база данных и приложение «Сайт трейлеров».

Данное программное средство отлично справляется с поставленной задачей. В программе была реализован сервис аутентификации через социальную сеть.

В программе имеется возможность просматривать трейлеры к фильмам, оставлять комментарии, а так же добавлять фильмы и трейлеры.

Данное программное средство использует Pug для реализации front-end, фреймворк Express на базе Node.js для разработки back-end. В качестве базы данных использовалась MySQL – реляционная база данных. На back-end реализована n-layer архитектура.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список литературы

1. Цыганенко, Н. П. Курс лекций по предмету «Программирование серверных кроссплатформенных приложений» - 2018.
2. N-layer архитектура [Электронный ресурс] / Github – Режим доступа : <https://accetone.github.io/cwp/10/#/> - Дата доступа : 16.05.2018.
3. sequelize [Электронный ресурс] / npm – Режим доступа : <https://www.npmjs.com/package/sequelize> - Дата доступа : 16.05.2018.
4. mocha [Электронный ресурс] / npm – Режим доступа : <https://www.npmjs.com/package/mocha> - Дата доступа : 16.05.2018.
5. joi [Электронный ресурс] / npm – Режим доступа : <https://www.npmjs.com/package/joi> - Дата доступа : 16.05.2018.
6. multer [Электронный ресурс] / npm – Режим доступа : <https://www.npmjs.com/package/multer> - Дата доступа : 16.05.2018.
7. bootstrap [Электронный ресурс] / npm – Режим доступа : <https://www.bootstrap-4.ru/docs/4.1/getting-started/> - Дата доступа : 16.05.2018.
8. passport-vkontakte [Электронный ресурс] / npm – Режим доступа : <https://www.npmjs.com/package/passport-vkontakte> - Дата доступа : 16.05.2018.
9. pug [Электронный ресурс] / npm – Режим доступа : <https://pugjs.org/api/getting-started.html> - Дата доступа : 16.05.2018.

Приложение А

Листинг 1. Бизнес-логика приложения

```
class CrudService {
  constructor(repository, errors) {
    this.repository = repository;
    this.errors = errors;

    this.defaults = {
      readChunk: {
        limit: 10,
        page: 1,
        order: "desc",
        orderField: "id"
      }
    };
  }

  async readChunk(
    options = this.defaults.readChunk,
    repository = this.repository,
    filter = null
  ) {
    options = { ...this.defaults.readChunk, ...options };

    let limit = options.limit;
    let offset = (options.page - 1) * options.limit;

    let data = await repository.findAndCountAll({
      where: { ...filter },
      raw: true,
      limit: limit,
      offset: offset,
      order: [[options.orderField, options.order.toUpperCase()]]
    });

    options.pages = Math.ceil(data.count / limit);

    return {
      data: data.rows,
      meta: options,
    }
  }

  async read(id) {
    id = parseInt(id);

    if (isNaN(id)) {
      throw this.errors.invalidId;
    }

    const item = await this.repository.findById(id, { raw: true });

    if (!item) {
```



```

        throw this.errors.notFound;
    }

    return item;
}

async create(data) {
    const item = await this.repository.create(data);

    return await item.get({ plain: true });
}

async update(id, data) {
    id = parseInt(id);
    await this.repository.update(data, { where: { id }, limit: 1 });

    return await this.read(id);
}

async delete(id) {
    id = parseInt(id);
    return await this.repository.destroy({ where: { id } });
}

}

class UsersService extends CrudService {
    constructor(usersRepository, commentsRepository, errors){
        super(usersRepository, errors);
        this.commentsRepository = commentsRepository;
    }
    async readForSpId(id) {
        const user = await this.repository.findOne({where: {id: id}});
        if (!user) {
            return null;
        }
        return user;
    }
}

class VideosService extends CrudService {
    constructor(videosRepository, filmsRepository, errors){
        super(videosRepository, errors);
        this.filmsRepository = filmsRepository;
    }
    async bindVideo(id, filmId) {
        const videoBeforeUpdate = await super.read(id);
        const film = await this.filmsRepository.findById(filmId, {raw: true});

        if(!videoBeforeUpdate || !film) {
            this.errors.notFound;
        }

        return await super.update(id, {

```

```

        ...videoBeforeUpdate,
        filmId: filmId
    });
}
async unbindVideo(id) {
    const videoBeforeUpdate = await super.read(id);
    if(!videoBeforeUpdate) {
        this.errors.notFound;
    }
    return await super.update(id, {
        ...videoBeforeUpdate,
        filmId: null
    })
}
}
}

class FilmsService extends CrudService {
    constructor(filmsRepository, videosRepository, errors){
        super(filmsRepository, errors);
        this.videosRepository = videosRepository;
    }

    async readVideos(id) {
        const film = await super.read(id);

        if(!film) {
            this.errors.notFound;
        }

        return await super.readChunk({}, this.videosRepository, {
            filmId: film.id
        })
    }
    async delete(id) {
        const film = await super.read(id);

        fs.unlink(pathFolder + film.preview_path, (err) => {});

        return await this.repository.destroy({ where: { id } });
    }
    async update(id, data) {
        const film = await super.read(id);

        fs.unlink(pathFolder + film.preview_path, (err) => {});

        await this.repository.update(data, { where: { id }, limit: 1 });

        return await this.read(id);
    }
    async find(name) {
        return await this.repository.findAll({
            where: {
                name:{
                    [Op.like]: `%${name}%`
                }
            }
        })
    }
}

```

```

        });
    }
}

class CommentsService extends CrudService {
    constructor(commentsRepository, usersRepository, filmsRepository, errors){
        super(commentsRepository, errors);
        this.usersRepository = usersRepository;
        this.filmsRepository = filmsRepository;
    }
    async getFilmComments(filmId){
        const film = await this.filmsRepository.findById(filmId);

        if(!film) {
            this.errors.notFound;
        }

        return await this.repository.findAll({
            where: { filmId: film.id },
            include: [
                {model: this.usersRepository, required: true}
            ]
        });
    }
}

```

Приложение Б

Листинг 2. Модели приложения

```
module.exports = (Sequelize, sequelize) => {
  return sequelize.define("comments", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    message: Sequelize.TEXT,
  });
};

module.exports = (Sequelize, sequelize) => {
  return sequelize.define("films", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    name: Sequelize.STRING,
    name_original: Sequelize.STRING,
    genre: Sequelize.TEXT,
    country: Sequelize.STRING,
    age_limit: Sequelize.TINYINT,
    plot: Sequelize.TEXT,
    release_date: Sequelize.DATE,
    release_bel: Sequelize.DATE,
    release_rus: Sequelize.DATE,
    in_roles: Sequelize.TEXT,
    director: Sequelize.STRING,
    film_script: Sequelize.STRING,
    music: Sequelize.STRING,
    operator: Sequelize.STRING,
    producer: Sequelize.STRING,
    company: Sequelize.STRING,
    preview_path: Sequelize.STRING,
  });
};

module.exports = (Sequelize, sequelize) => {
  return sequelize.define("users", {
    dbId: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    id: Sequelize.INTEGER,
    first_name: Sequelize.STRING,
    last_name: Sequelize.STRING,
    type: Sequelize.STRING,
    photo: Sequelize.STRING
  });
};
```

```
};

module.exports = (Sequelize, sequelize) => {
  return sequelize.define("videos", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    link: Sequelize.STRING,
    preview_path: Sequelize.STRING,
    preview_text: Sequelize.STRING,
  });
};
```

Приложение В

Листинг 3. Модель API

```
const express = require("express");

module.exports = (
  videosService,
  filmsService,
  usersService,
  commentsService,

  videosSchema,
  filmsSchema,
  usersSchema,
  commentsSchema
) => {
  const router = express.Router();

  const videosController = require('./videos')(videosService, videosSchema);
  const usersController = require('./users')(usersService, usersSchema);
  const filmsController = require('./films')(filmsService, filmsSchema,
  usersService);
  const commentsController = require('./comments')(commentsService, commentsSchema);

  router.use("/videos", videosController);
  router.use('/films', filmsController);
  router.use('/users', usersController);
  router.use('/comments', commentsController);

  return router;
}
```