

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Серверное приложение для Web-портала
агрегации зрительской и экспертной кинокритики**

Студент _____ Костюк Денис Андреевич _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Шибанов С.В. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В. _____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от ____ № ____)

Заведующий кафедрой _____ Козлов А.Ю. _____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от ____ № ____)

Секретарь ГЭК _____ Попова Н.А. _____
(подпись) (фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов «25»
декабря 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Костюк Денис Андреевич гр. 20ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Серверное приложение для Web-портала
агрегации зрительской и экспертной кинокритики

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Шибанов С.В.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: серверное приложение обрабатывает бизнес-запросы от клиента в рамках
области агрегации зрительской и экспертной кинокритики

Основные функции:

1) Предоставление возможности просмотра информации по фильмам и рецензиям на них

2) Предоставление возможности написания рецензий к фильмам внутри системы

3) Предоставление пользователям возможности оставлять свои оценки и комментарии к
фильмам и рецензиям на них

Язык программирования: Java

Инструментальные средства разработки: Spring Boot, PostgreSQL, Docker, DrawIO, Visual
Paradigm, Microsoft Project 2007, Swagger

4. Перечень подлежащих разработке вопросов _____

1) Анализ предметной области и требований к системе

- 2) Проектирование программных средств _____
- 3) Реализация программных средств _____
- 4) Тестирование программных средств _____
- 5) Оформление пояснительной записки _____
- _____
- _____

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 138 листов, 28 рисунков, 15 таблиц, 21 использованный источник, 8 приложений.

КИНОКРИТИКА, СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, JAVA, SPRING, DOCKER, POSTGRESQL, VISUAL PARADIGM, DRAWIO, SWAGGER

Объект разработки – серверное приложение для web-портала агрегации зрительской и экспертной кинокритики.

Цель работы - изучить предметную область разрабатываемой системы, определить функциональные требования и затраты на разработку, спроектировать систему и базу данных, разработать приложение, провести тестирование.

Технологии разработки – объектно-ориентированное программирование на Java, Spring boot, Docker, PostgreSQL, DrawIO, Microsoft Project 2007, Visual Paradigm, Swagger.

Результаты работы: спроектированы и разработаны база данных и серверное приложение для Web-сервиса агрегации зрительской и экспертной кинокритики.

					ПГУ 09.03.04 – 8BP201.10 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Костюк Д.А.			Серверное приложение для Web-портала агрегации зрительской и экспертной кинокритики Пояснительная записка	Лит.	Лист	Листов
Провер.		Шибанов С.В.					4	138
Реценз.						Группа 20ВП1		
Н. Контр.		Такташкин Д.В						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи на разработку серверного приложения	10
1.1 Анализ предметной области приложения	10
1.1.1 Анализ структурной организации предметной области	10
1.1.2 Анализ свойств информационных объектов предметной области...	11
1.1.3 Анализ процессов ведения данных в предметной области	12
1.2 Сравнительный анализ аналогичных разработок	14
1.2.1 Цели и задачи сравнительного анализа	14
1.2.2 Выбор приложений-аналогов для анализа	14
1.2.3 Обзор приложений-аналогов	15
1.2.4 Результаты сравнительного анализа	19
1.3 Постановление задачи на разработку серверного приложения	21
1.3.1 Цели разработки и назначение приложения	21
1.3.2 Классы пользователей приложения	21
1.3.3 Перечень автоматизируемых функций.....	22
1.4 Планирование и оценка процесса разработки серверного приложения ...	23
1.4.1 Выбор программных средств планирования и оценки процесса разработки.....	23
1.4.2 Планирование процесса разработки приложения	24
1.4.3 Оценка временных и финансовых затрат на разработку	25
2 Проектирование серверного приложения	27
2.1 Моделирование функциональных возможностей приложения	27
2.2 Разработка концептуальной модели данных.....	33

2.3 Обобщенная архитектура веб-портала	35
2.4 Требования к аппаратно-программному обеспечению серверного приложения	37
2.5 Обоснование выбора технологий и средств разработки серверного приложения	38
3 Разработка программного обеспечения серверного приложения	39
3.1 Разработка архитектуры серверного приложения	39
3.2 Разработка физической модели базы данных	41
3.3 Разработка программного обеспечения доступа к базе данных	44
3.4 Разработка программного обеспечения взаимодействия с клиентским приложением	47
3.5 Разработка программного обеспечения бизнес-логики серверного приложения	50
4 Контроль качества программного обеспечения	57
4.1 Планирование процесса контроля качества программного обеспечения .	57
4.1.1 Обоснование выбора модели тестирования	57
4.1.2 Обоснование выбора метрик программного кода	58
4.2 Тестирование программных средств	59
4.2.1 Тестирование API	59
4.2.2 Структурное тестирование	61
4.3 Оценка качества программного кода на основе метрик	63
Заключение	65
Список использованных источников	66
Приложение А (справочное) Глоссарий	69
Приложение Б (рекомендуемое) Спецификации атрибутов таблиц	71
Приложение В (рекомендуемое) Сущностные объекты таблиц	76
Приложение Г (рекомендуемое) SQL-скрипты	87

Приложение Д (рекомендуемое) Классы типа Repository	92
Приложение Е (рекомендуемое) Классы типа Controller.....	96
Приложение Ж (рекомендуемое) Классы типа Service	106
Приложение И (рекомендуемое) Иллюстрационные материалы к тестированию API	116

Введение

Кинокритика – вид словесного творчества, состоящий в анализе и интерпретации фильмов и определении их художественной ценности [1]. Кинокритика, в отличие от киноведения, предназначена для более широкой аудитории.

Жизнь современного человека ускоряется, и люди начинают сильнее ценить своё время. И, чтобы не тратить его на часовые поиски кинокартины или на просмотр кино плохого качества – они обращаются к сайтам с критическими рецензиями на кино. Однако, на рынке существует недостаточное количество сайтов-агрегаторов с подобным видом рецензий, а потому, вопрос создания подобного ресурса является актуальным.

Целью выпускной квалификационной работы является проектирование и разработка серверного приложения для Web-портала агрегации зрительской и экспертной кинокритики. Для решения задач, поставленных в практике, необходимо:

- изучение предметной области кинокритики;
- анализ приложений-аналогов и выделение необходимого функционала системы;
- определение требований к разрабатываемому продукту;
- планирование трудовых, временных и денежных ресурсов;
- выбор архитектуры приложения и средств для его разработки;
- проектирование системы и её базы данных;
- разработка серверного приложения;
- тестирование полученного продукта.

Для разработки модели базы данных используется среда Visual Paradigm. Создание UML диаграмм происходит в DrawIO. Планирование ресурсов выполняется в Microsoft Project 2007. Разработка серверного приложения будет выполняться с использованием языка программирования Java и его фреймворка Spring, средства контейнеризации Docker и системы управления базами данных

PostgreSQL. Тестирование продукта и документирование API проводится с использованием инструмента Swagger.

Все термины, сложные для понимания данной работы, представлены в приложении А.

1 Анализ предметной области и постановка задачи на разработку серверного приложения

1.1 Анализ предметной области приложения

1.1.1 Анализ структурной организации предметной области

Предметной областью данной работы является киноиндустрия и информационный ресурс, связанный с кинокритикой. Перед описанием структуры предметной области приведены основные определения.

Кинематограф – часть человеческой деятельности, которая создаёт движущиеся изображения или объекты. Данная деятельность делится на две основных части: киноискусство и киноиндустрия.

Киноискусство – изобразительное искусство, производство которого осуществляется при использовании специального оборудования и воссоздании реальных, инсценированных или мультипликационных событий. Киноискусство совмещает в себе свойства театра, литературы, музыки и изобразительного искусства.

Киноиндустрия – область промышленности, производящая кинофильмы, мультипликацию и спецэффекты. Большую часть этой сферы деятельности занимает кинопрокат, при помощи которого осуществляется распространение кинопродуктов по разным странам мира.

Кинокритик – это человек, который анализирует и даёт рецензию кинематографическому процессу. В рецензиях кинокритик высказывается открытым образом обо всех достоинствах и недостатках кинопроизведений.

Рецензия – жанр журналистики, а также научной и художественной критики. Рецензия содержит оценку работы, указывает на её достоинства и недостатки, требующие устранения.

Следует выделить основные жанры кинокартин:

– боевик – фильмы данного жанра не имеют сложного сюжета, в большинстве случаев главный герой сталкивается с силами зла, которые в конечном итоге побеждает;

– драма – изображает частную жизнь человека и его взаимодействие с обществом;

– исторический фильм – данный жанр описывает реальные исторические события, стараясь воссоздать атмосферу описываемого времени;

– кинокомедия – жанр, сюжет которого наполнен шутками разного качества;

– мелодрама – произведения данного жанра ставят основной целью раскрыть весь эмоциональный спектр героев;

– триллер – жанр, который создаёт чувство напряжения и переживания у зрителя как атмосферой, так и сюжетом;

– фильм ужасов – данный жанр призван вызвать чувство страха у зрителя.

Любой фильм может быть представителем нескольких жанров одновременно.

1.1.2 Анализ свойств информационных объектов предметной области

Для предметной области кинокритики можно выделить следующие информационные объекты:

Фильм – уникальная кинокартина, имеющая своё название, режиссера, дату выхода, жанры и продолжительность. Именно фильмы являются объектом оценки в рецензиях.

Рецензия – оценочный вид журналистской деятельности. В рамках предметной области рецензии пишутся на конкретный фильм. Рецензия имеет свой заголовок, автора, рецензируемый фильм, текст рецензии и оценку к фильму.

Дополнительные материалы – сопроводительные материалы к рецензии, которые автор добавляет к ней. Это могут быть ссылки на социальные сети рецензента или его места работы, видеоматериалы, фотоматериалы и другие.

Зритель – человек, который смотрит или собирается смотреть определенный фильм. Зритель является аудиторией, для которой и создаются фильмы и рецензии на них.

Рецензент – человек, который профессионально оценивает определенную кинокартину, создавая рецензию на неё. Обычно является сотрудником определенного журнала.

Администратор – человек, который отвечает за верификацию рецензентов и реагирование на поступающие жалобы.

Комментарий – это словесное высказывание, которое пишут зрители к фильму или рецензии, чтобы выразить своё мнение. Такая оценка не является профессиональной и несет субъективный характер.

Оценка – количественный параметр, выставляемый к фильмам и рецензиям зрителями и рецензентами. Оценки используют для определения качества фильма или рецензии.

Верификация – это подтверждение экспертности рецензента. Для этого рецензент представляет определенный верификационный документ: диплом журналиста, выписка с работы в киножурнале, ссылка на канал с кинокритикой с определенным количеством аудитории.

Жалоба – письменное обращение, в котором указывается определенное нарушение, допущенное при написании рецензии или комментария.

1.1.3 Анализ процессов ведения данных в предметной области

В любой предметной области протекают определенные процессы. В области кинокритики создаются фильмы, которые нужно оценить для зрителей. На эти фильмы пишутся рецензии, чтобы дать профессиональную оценку фильму и подсказать, стоит ли потратить своё время на данную картину или нет. Фильмы и рецензии комментируются обычными зрителями. Протекание этих процессов отражено в диаграмме деятельности, представленной на рисунке 1.

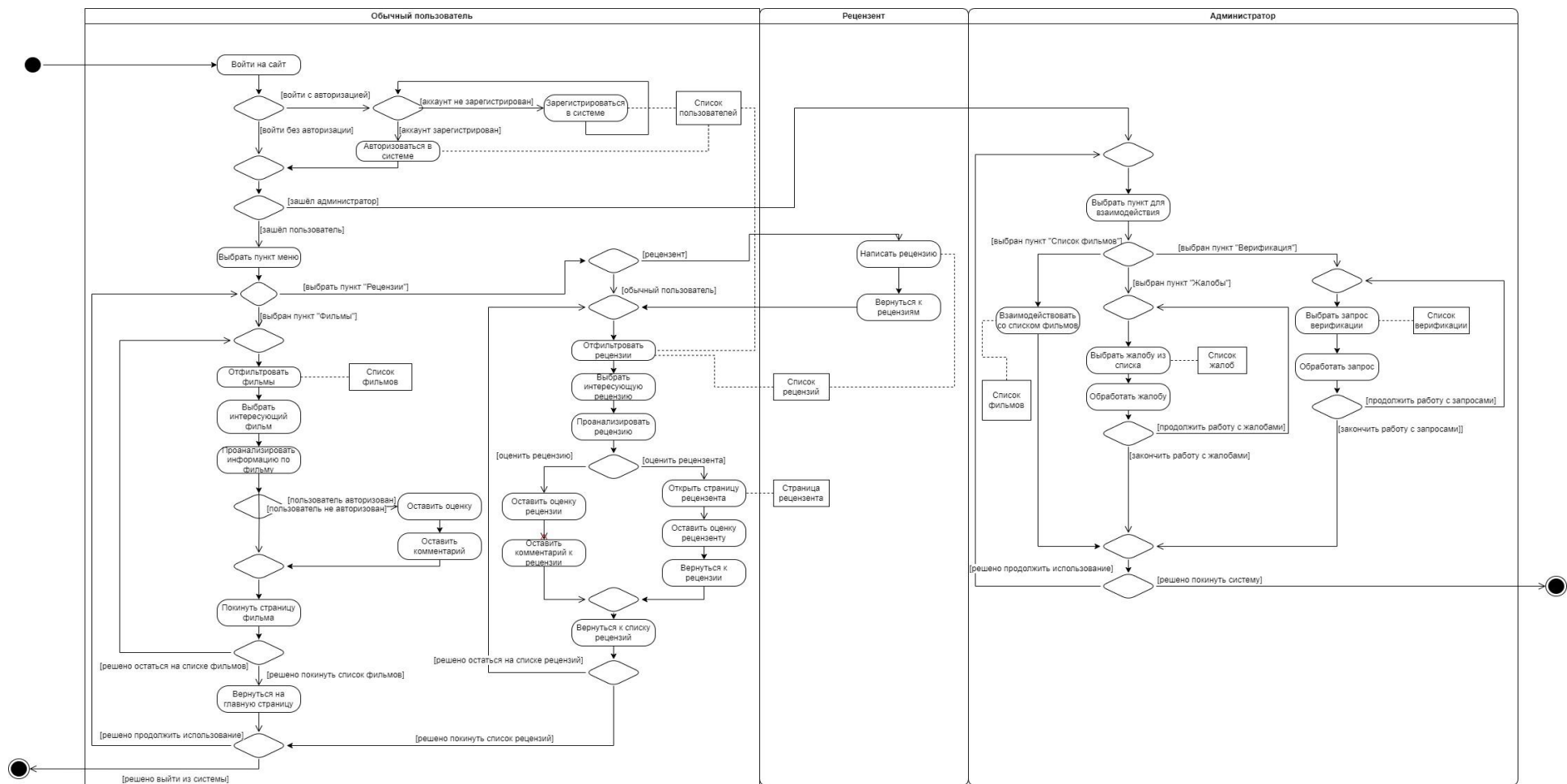


Рисунок 1 – Диаграмма деятельности для процессов, протекающих в области кинокритики

В ходе протекания процессов возникают и используются определенные данные: список фильмов, список рецензий, сами рецензии, список верификации и верификационные документы, комментарии и оценки к фильмам и рецензиям.

1.2 Сравнительный анализ аналогичных разработок

1.2.1 Цели и задачи сравнительного анализа

Перед проектированием приложения необходимо произвести сравнительный анализ аналогичных продуктов. По результатам анализа будут выделены их базовые функциональные возможности, отобраны необходимые функции для добавления их в разрабатываемое приложение, а также некоторые интересные особенности систем. Кроме того, анализ поможет выделить новые функции системы, которых нет в приложениях-аналогах, и добавить их в разрабатываемую систему.

В качестве критериев для сравнительного анализа приложений-аналогов были выбраны:

- функциональная наполненность;
- удобство пользовательского интерфейса;
- поисковые возможности системы;
- полнота рецензий;
- прозрачность оценки;
- доступность для отечественного пользователя.

Выделенные критерии позволят оценить сильные и слабые качества аналогов.

1.2.2 Выбор приложений-аналогов для анализа

Для сравнения были просмотрены следующие Web-сайты с кинокритикой:

- Кинопоиск (Яндекс);

- Retinsky (Разработчик неизвестен);
- Metacritic (Fandom inc);
- Rotten Tomatoes (Fandango);
- Критиканство (КГ-дизайн);
- Rogereberg (Ebert Digital LLC);
- IMBd (Amazon);
- Film Criticism (Michigan Publishing);
- The New York Times (The New York Times Company);
- The Guardian (Guardian News & Media Limited);
- Дзен (Яндекс).

Из них для анализа были отобраны самые популярные ресурсы: Кинопоиск, Metacritic, Rotten Tomatoes и IMBd. Остальные сайты или проигрывают названным по охвату пользователей, либо не являются профильными агрегаторами кинокритики.

1.2.3 Обзор приложений-аналогов

Кинопоиск – единственный из анализируемых продуктов, являющийся отечественной разработкой. Сайт был разработан в 2003 году Виталием Таций и Дмитрием Сухановым и выкуплен компанией «Яндекс». Сервис содержит в себе всю информацию по фильмам, мультфильмам и сериалам, в том числе и кинокритику. Кроме того, Кинопоиск предоставляет услуги онлайн кинотеатра по подписке.

Из плюсов сайта можно выделить: приятный пользовательский интерфейс, большое количество охватываемых кинокартин, внушительный объем информации по каждому фильму, понятное разделение рецензий на профессиональные и любительские, а также на отечественные и мировые, наличие названий у рецензий.

Из минусов можно выделить: невозможность просмотра списка рецензий отдельно от фильмов, невозможность выставить определённую оценку картине (отзыв может быть лишь трёх видов: отрицательный, нейтральный или положительный), функциональная и информационная перегруженность сайта.

Интерфейс Web-ресурса «Кинопоиск» представлен на рисунке 2.

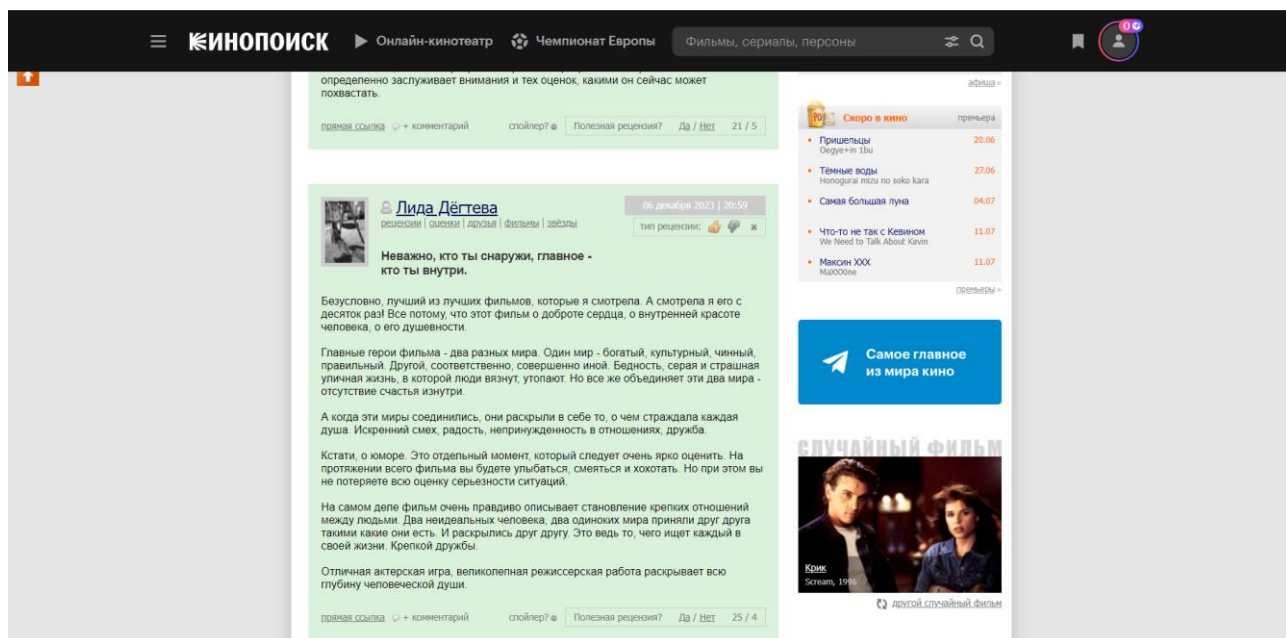


Рисунок 2 – Интерфейс Web-ресурса «Кинопоиск»

Следующим на очереди идёт Metacritic, начавший свою работу в 2001 году. Сайт агрегирует критику по фильмам, сериалам, мультфильмам, телешоу, играм и музыке.

Из плюсов Metacritic можно выделить огромный охват материала. Кроме того, сайт имеет приятный и удобный интерфейс. Имеется возможность просмотра критики у конкретного рецензента. В системе разделяются зрительские и профессиональные рецензии. Возможность выставить конкретную оценку картине тоже является плюсом.

Из минусов можно отметить невозможность просмотра рецензий отдельно от фильмов, отсутствие названий у рецензий, отсутствие собственной платформы для содержания профессиональной кинокритики (на сайте находятся лишь выдержки из рецензии; чтобы просмотреть рецензию

полностью – придётся идти на сайт журнала). Главным же минусом является недоступность данного ресурса для граждан Российской Федерации из-за политики компании-владельца.

Интерфейс сайта «Metacritic» представлен на рисунке 3.

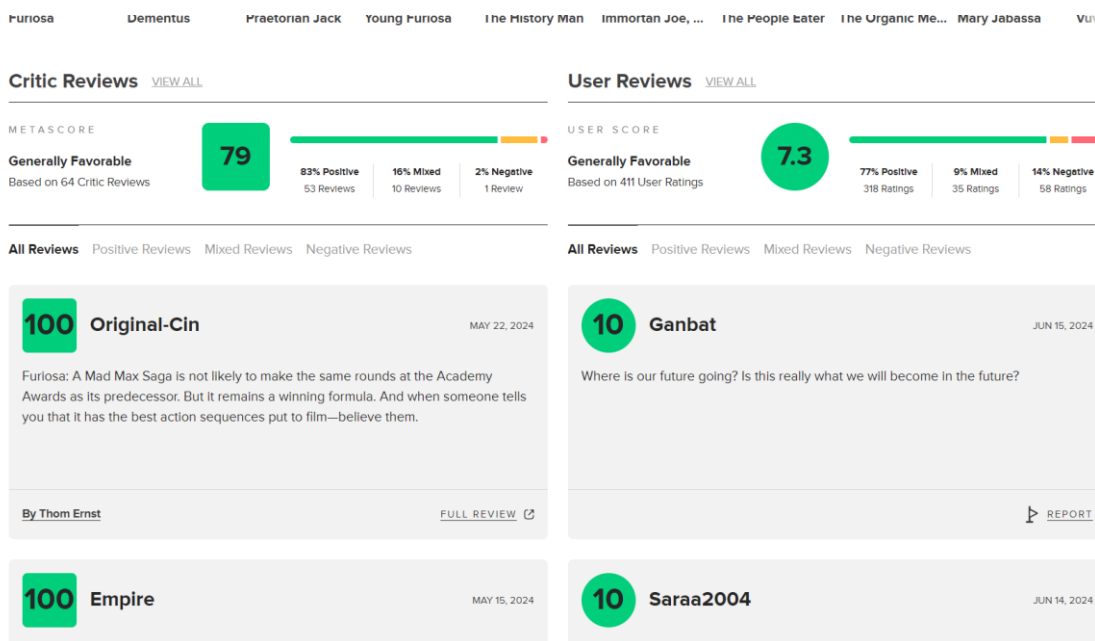


Рисунок 3 – Интерфейс сайта «Metacritic»

Проект Rotten Tomatoes был запущен в 1998 году. Сайт специализируется на агрегации критики по фильмам, сериалам и мультфильмам. Особенностью сайта является его особая метрика оценки картин, называемая «рейтинг свежести». Вместо того чтобы выставлять числовые оценки фильмам, пользователи дают им «процент свежести».

Проект Rotten Tomatoes имеет огромный охват в рамках кинопроизводства, не уступая в этом пункте сайту Metacritic. Интерфейс сайта является хоть и удобным, но визуально устаревшим. На сайте присутствует разделение критики на экспертную и зрительскую. Проект предоставляет большой объем информации по каждому фильму, однако, в отличие от Кинопоиска, рецензии не теряются в нем. Плюсом является и возможность просмотра критики от отдельного рецензента.

Из минусов можно выделить невозможность просмотра рецензий отдельно от фильмов, отсутствие платформы для содержания профессиональной кинокритики, отсутствие названий у рецензий, необъективность модели процентной оценки фильмов, которая может выставить среднему непопулярному фильму довольно высокие оценки свежести при средних рецензиях. Также, у сайта отсутствует российская локализация и возможность написания рецензий на русском языке.

Интерфейс Web-портала «Rotten Tomatoes» представлен на рисунке 4.

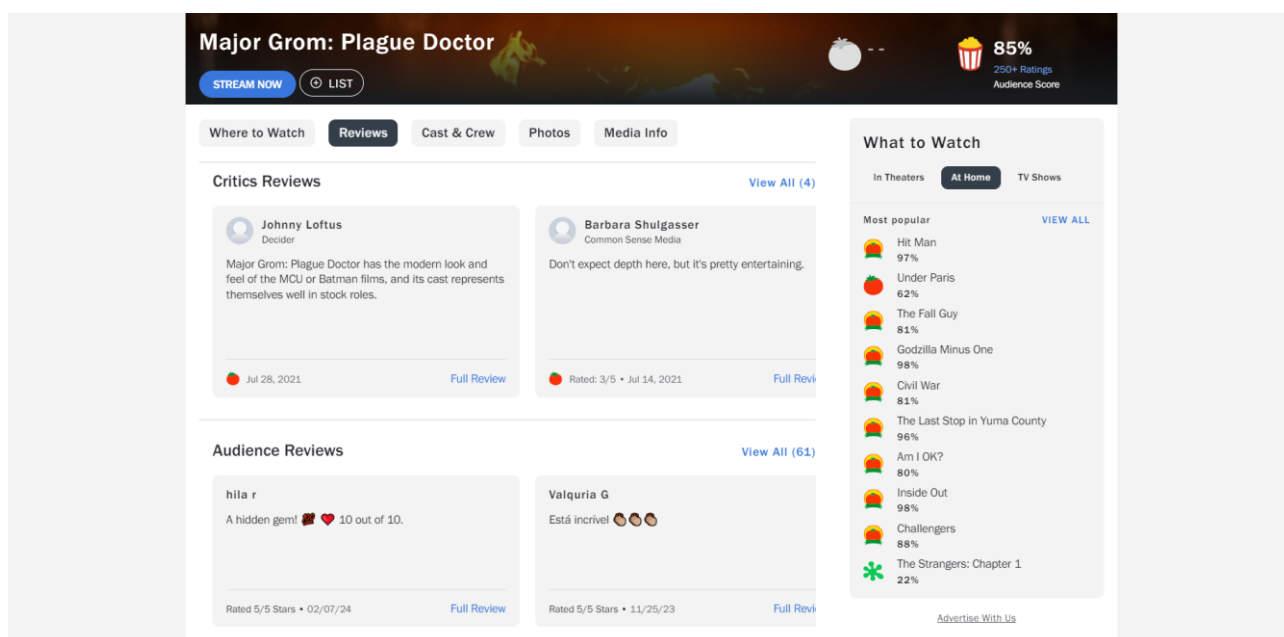


Рисунок 4 – Интерфейс Web-портала «Rotten Tomatoes»

Последним проектом для анализа выступает IMDb. Он зародился как база данных статей о кинофильмах в 1990-м году. В сайт-агрегатор IMDb был переделан в 1993 году. Специализируется на фильмах, сериалах, мультфильмах и телевизионных шоу.

Из плюсов можно выделить приятный интерфейс, огромный охват по кинематографу всех стран, наличие частичной локализации (заголовки фильмов отображаются на русском языке), рецензии имеют названия (но поиск по ним невозможен).

Из минусов выделяются: отсутствие разделения рецензий на профессиональные и любительские, перегруженность страницы информацией по фильму (до рецензии придётся долго добираться), невозможность просмотра рецензий отдельно от фильмов, невозможность поиска рецензий по их заголовкам, невозможность просмотра рецензий у конкретного рецензента, невозможность написания рецензий на русском языке.

Интерфейс сайта «IMDb» представлен на рисунке 5.

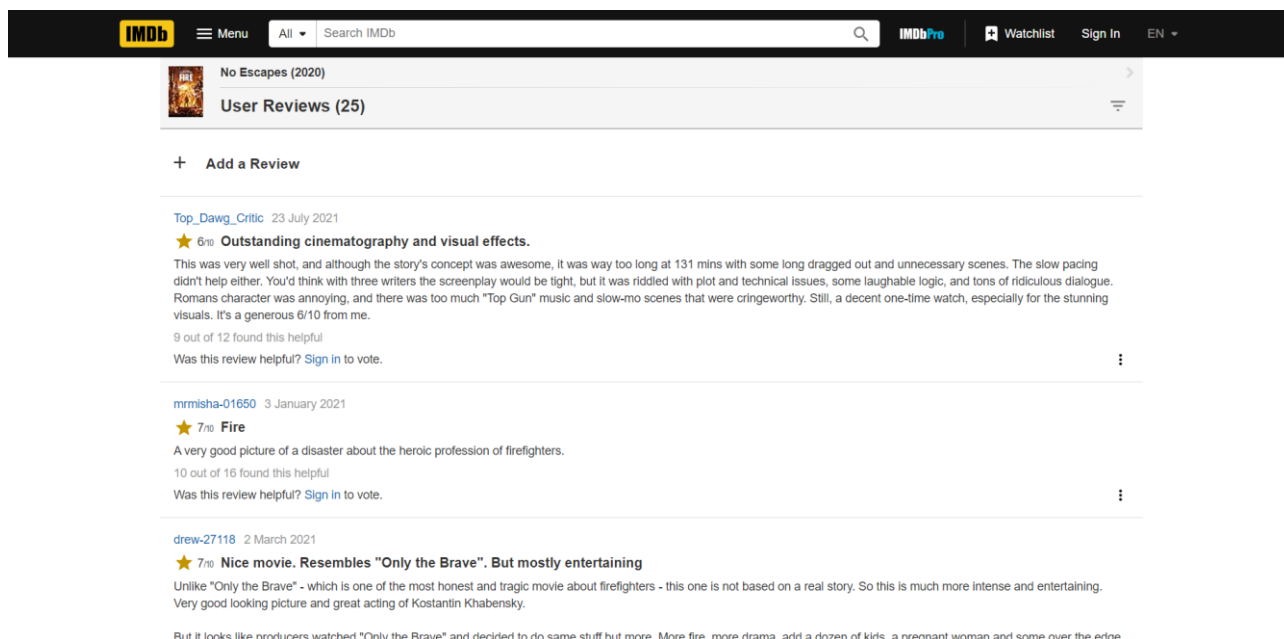


Рисунок 5 – Интерфейс сайта «IMDb»

Таким образом, все рассмотренные аналоги имеют свои конкуретные преимущества, которые можно реализовать в разрабатываемой системе.

1.2.4 Результаты сравнительного анализа

Оценивая сильные и слабые стороны анализируемых систем, была сформирована сравнительная таблица. Для каждого критерия сравнения будет выставлена оценка от 0 до 5. Разрабатываемая система также оценивается по выставленным критериям.

Результат сравнительного анализа продемонстрирован в таблице 1.

Таблица 1 – Сравнительная таблица

Критерий	Разрабатываемая система	Кинопоиск	Metacritic	Rotten Tomatoes	IMDb
Функциональная наполненность	4	5	3	4	4
Удобство интерфейса	4	3	4	5	3
Поисковые возможности системы	4	4	4	4	4
Полнота рецензий	5	5	3	4	4
Прозрачность оценки	5	3	5	4	5
Доступность для отечественного пользователя	5	5	0	2	4

После анализа можно выделить общие функции, присущие всем продуктам-аналогам:

- возможность просмотра рецензий по конкретному фильму;
- возможность выставить оценки фильму;
- возможность оценить и прокомментировать рецензию;
- возможность фильтрации фильмов по жанрам;
- предоставление информации по фильмам.

Кроме того, нужно включить некоторые необходимые возможности, которые предоставляют практически все аналоги:

- разделение рейтингов от профессиональных рецензентов и обычных пользователей;
- верификация профессиональных рецензентов;
- возможность просмотра критики от конкретного рецензента;
- возможность выставить конкретную оценку рецензируемому фильму;
- предоставление платформы для написания рецензии на сайте.

Разрабатываемое приложение будет предоставлять возможность просматривать рецензии без привязки к конкретному фильму, искать рецензии не только по фильму или автору, но и по названию самой рецензии.

Реализуя весь описанный функционал, разрабатываемое приложение выглядит конкурентоспособным для рынка сайтов, агрегирующих

кинокритику.

1.3 Постановление задачи на разработку серверного приложения

1.3.1 Цели разработки и назначение приложения

Система предназначена для агрегации профессиональной кинокритики по кинокартинам различных жанров. Она позволяет создавать, хранить, просматривать и оценивать рецензии на фильмы, мультфильмы и сериалы. Дополнительно, система предоставляет возможность обычным пользователям оценивать как сами фильмы, так и рецензии на них.

Можно выделить следующие цели разработки:

- спроектировать и сконструировать базу данных приложения;
- спроектировать базу данных и серверное приложение;
- сконструировать базу данных и серверное приложение;
- связать серверное приложение с базой данных;
- предоставить конечные точки от серверного приложения для связи с клиентским приложением;
- протестировать полученное приложение и оценить метрики кода.

По итогу разработки необходимо получить приложение, отвечающее всем требованиям.

1.3.2 Классы пользователей приложения

Классы пользователей построены так, что каждый следующий класс имеет все возможности предыдущего вдобавок к своим функциям. Исключением является класс «администратор», который имеет свой собственный набор функций. Категории пользователей информационной системы представлены в таблице 2.

Таблица 2 – Категории пользователей информационной системы

п/п	Категории пользователей	Наименование приложения	Автоматизируемые бизнес-процессы	Классы информационных объектов
1	Неавторизованный пользователь	Просмотр рецензий	Просмотр рецензий	Фильм, рецензент, рецензия
			Просмотр информации о фильме	Фильм
			Просмотр комментариев	Фильм, рецензент, рецензия, комментарий
			Фильтрация фильмов	Список фильмов
			Фильтрация рецензий	Список фильмов, список рецензий, рецензент
2	Обычный пользователь	Просмотр и комментирование рецензий	Создание комментариев	Фильм, рецензент, рецензия
			Оценивание фильмов	Фильм
			Оценивание рецензий	Фильм, рецензия
3	Пользователь-рецензент	Создание и просмотр рецензий	Создание рецензий	Фильм, рецензент, рецензия, список дополнительных материалов
4	Администратор	Приложение администратора	Добавление фильма	Фильм, список фильмов
			Верификация статуса пользователя	Страница пользователя, подтверждающий документ
			Удаление контента	Жалоба, рецензия, комментарий
			Блокировка пользователя	Жалоба, страница пользователя

Разрабатываемая система должна предоставить функционал для всех категорий пользователей системы.

1.3.3 Перечень автоматизируемых функций

Информационная система должна предоставить ряд функций:

- регистрация пользователя на портале;
- авторизация пользователя на портале;
- изменение данных о пользователе в личном кабинете;
- удаление страницы пользователя;
- получение доступа к рецензиям, написанным пользователем;
- прохождение верификации для получения роли «рецензента»;
- просмотр списка фильмов;
- просмотр информации о конкретном фильме;

- просмотр рецензий по конкретному фильму;
- комментирование фильма;
- фильтрация списка фильмов;
- просмотр списка рецензий;
- просмотр содержания конкретной рецензии;
- фильтрация рецензий по определённому критерию;
- оставление комментариев к рецензии;
- написание рецензий на фильм;
- прикрепление сопроводительных материалов к рецензии;
- просмотр отзывов по конкретной картине;
- выставление оценок кинокартине;
- выставление оценок рецензии;
- возможность пожаловаться на комментарий или рецензию.

Система будет хранить: данные о пользователях, список фильмов, написанные рецензии, отзывы пользователей, жалобы от пользователей.

1.4 Планирование и оценка процесса разработки серверного приложения

1.4.1 Выбор программных средств планирования и оценки процесса разработки

Для планирования и оценки процесса разработки программы будет использована диаграмма Ганта.

Диаграмма Ганта - это инструмент управления проектами, который позволяет регулировать работу по нескольким направлениям: планировать и составлять графики проектов, отслеживать зависимости между задачами, делегировать, распределять ресурсы, постоянно держать под контролем процесс выполнения каждой задачи [2]. Кроме того, она позволяет нам оценить временные и денежные затраты на разработку приложения.

Руководители проектов и сотрудники могут просматривать календарь по задачам и отслеживать этапы их выполнения. Планирование задач и отображение их в виде графики позволяет командам соблюдать дедлайны.

Для разработки диаграммы Ганта был выбран Microsoft Project 2007.

1.4.2 Планирование процесса разработки приложения

В ходе планирования проекта была выделена 21 задача. Список задач проекта, их виды, длительность и предшественники приведены на рисунке 6.

1	[-] Разработка серверного приложения	74 дней	Пт 01.09.23	Ср 13.12.23	
2	Начало проекта	0 дней	Пт 01.09.23	Пт 01.09.23	
3	[-] Анализ предметной области приложения	12 дней	Пт 01.09.23	Пн 18.09.23	
4	Изучение предметной области	4 дней	Пт 01.09.23	Ср 06.09.23	2
5	Изучение аналогов	2 дней	Чт 07.09.23	Пт 08.09.23	4
6	Сравнение аналогов	2 дней	Пн 11.09.23	Вт 12.09.23	5
7	Составление требований к системе	1 день	Ср 13.09.23	Ср 13.09.23	6
8	Планирование процесса разработки прилс	3 дней	Чт 14.09.23	Пн 18.09.23	7
9	Конец анализа предметной области прило	0 дней	Пн 18.09.23	Пн 18.09.23	8
10	[-] Проектирование приложения	8 дней	Вт 19.09.23	Чт 28.09.23	
11	Моделирование функциональных возмож	3 дней	Вт 19.09.23	Чт 21.09.23	9
12	Разработка концептуальной модели данн	3 дней	Пт 22.09.23	Вт 26.09.23	11
13	Выбор технологий разработки	1 день	Ср 27.09.23	Ср 27.09.23	12
14	Формирование аппаратно-программных тр	1 день	Чт 28.09.23	Чт 28.09.23	13
15	Конец проектирования приложения	0 дней	Чт 28.09.23	Чт 28.09.23	14
16	[-] Разработка приложения	36 дней	Пт 29.09.23	Пт 17.11.23	
17	Разработка архитектуры серверного прилс	3 дней	Пт 29.09.23	Вт 03.10.23	15
18	Разработка физической модели базы дан	3 дней	Ср 04.10.23	Пт 06.10.23	17
19	Разработка ПО доступа к БД	9 дней	Пн 09.10.23	Чт 19.10.23	18
20	Разработка ПО взаимодействия с клиентс	15 дней	Пн 09.10.23	Пт 27.10.23	18
21	Разработка ПО бизнес-логики приложения	15 дней	Пн 30.10.23	Пт 17.11.23	19,20
22	Конец разработки приложения	0 дней	Пт 17.11.23	Пт 17.11.23	21
23	[-] Тестирование приложения	9 дней	Пн 20.11.23	Чт 30.11.23	
24	Выбор метрик программного кода	1 день	Пн 20.11.23	Пн 20.11.23	22
25	Функциональное тестирование	2 дней	Вт 21.11.23	Ср 22.11.23	24
26	Нефункциональное тестирование	2 дней	Чт 23.11.23	Пт 24.11.23	25
27	UNIT-тестирование	3 дней	Пн 27.11.23	Ср 29.11.23	26
28	Вычисление метрик качества кода	1 день	Чт 30.11.23	Чт 30.11.23	27
29	Конец тестирования приложения	0 дней	Чт 30.11.23	Чт 30.11.23	28
30	Составление глоссария	2 дней	Пт 01.12.23	Пн 04.12.23	29
31	Составление отчёта	7 дней	Вт 05.12.23	Ср 13.12.23	30
32	Конец проекта	0 дней	Ср 13.12.23	Ср 13.12.23	31

Рисунок 6 – Список задач проекта

Кроме того, был выделен ресурс для выполнения всех задач проекта (рисунок 7).

	1	Название ресурса	Тип	Краткое название	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполыз.	Начисление	Базовый календарь
1		Разработчик	Трудовой	P	100%	35 000,00 P/мес	0,00 P/ч	0,00 P	Пропорциональное	Стандартный

Рисунок 7 – Список ресурсов проекта

Представленный трудовой ресурс распределяется на все задачи проекта для их выполнения. Распределение ресурсов по задачам представлено на рисунке 8.

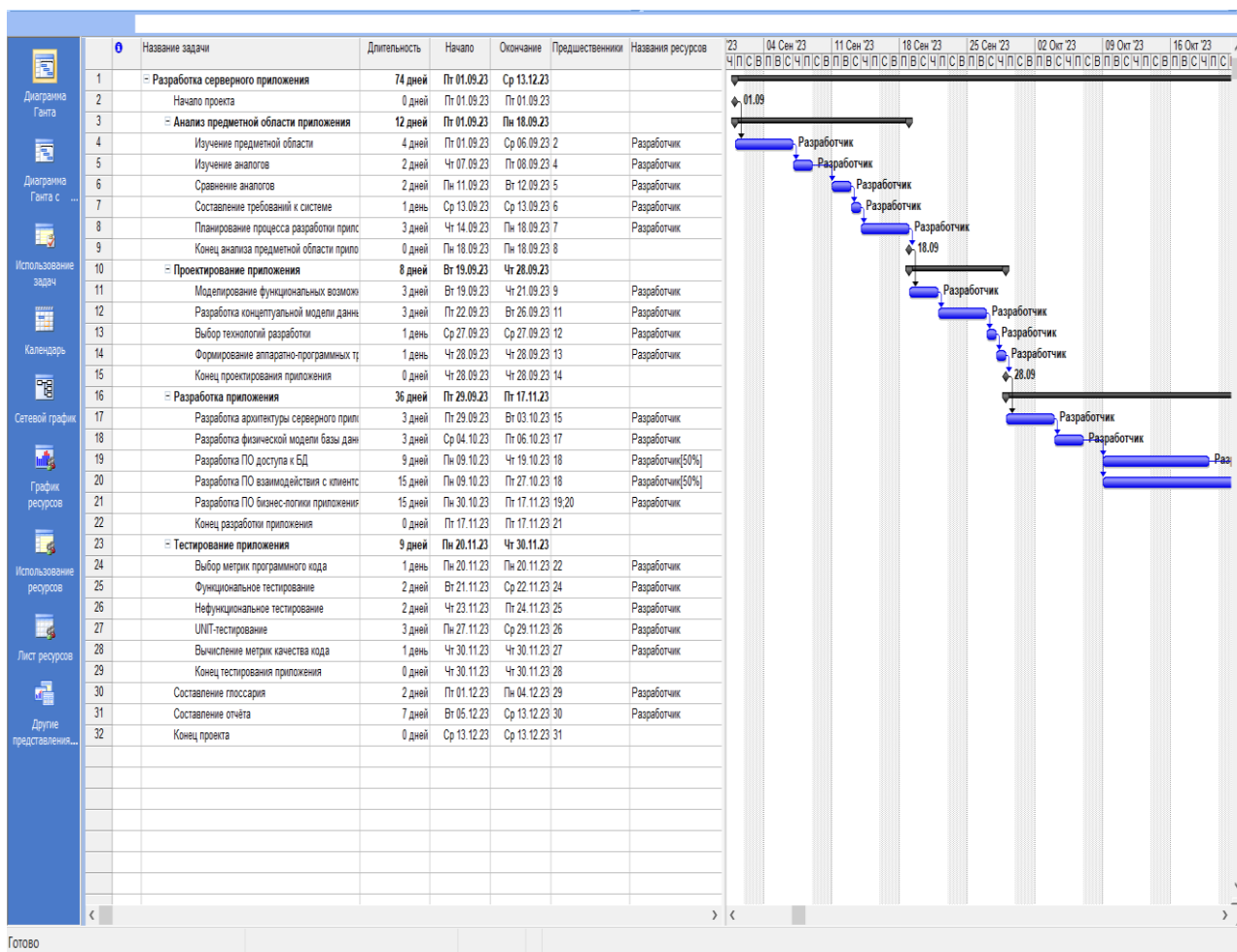


Рисунок 8 – Распределение трудовых ресурсов по задачам

Как видно из рисунка, все задачи на проекте выполняются одним трудовым ресурсом – разработчиком.

1.4.3 Оценка временных и финансовых затрат на разработку

Кроме распределения трудовых ресурсов, в проекте требуется определить временные и денежные ресурсы. Распределение временных и денежных ресурсов по задачам представлено на рисунке 9.

	Название задачи	Длительность	Начало	Окончание	% завершения	Затраты	Трудозатраты
1	[-] Разработка серверной части	74 дней	Пт 01.09.23	Ср 13.12.23	0%	86 975,00 Р	397,6 ч
2	Начало проекта	0 дней	Пт 01.09.23	Пт 01.09.23	0%	0,00 Р	0 ч
3	[-] Анализ предметной области	12 дней	Пт 01.09.23	Пн 18.09.23	0%	14 700,00 Р	67,2 ч
4	Изучение предметной области	4 дней	Пт 01.09.23	Ср 06.09.23	0%	4 900,00 Р	22,4 ч
5	Изучение аналогов	2 дней	Чт 07.09.23	Пт 08.09.23	0%	2 450,00 Р	11,2 ч
6	Сравнение аналогов	2 дней	Пн 11.09.23	Вт 12.09.23	0%	2 450,00 Р	11,2 ч
7	Составление технического задания	1 день	Ср 13.09.23	Ср 13.09.23	0%	1 225,00 Р	5,6 ч
8	Планирование работ	3 дней	Чт 14.09.23	Пн 18.09.23	0%	3 675,00 Р	16,8 ч
9	Конец анализа предметной области	0 дней	Пн 18.09.23	Пн 18.09.23	0%	0,00 Р	0 ч
10	[-] Проектирование архитектуры	8 дней	Вт 19.09.23	Чт 28.09.23	0%	9 800,00 Р	44,8 ч
11	Моделирование архитектуры	3 дней	Вт 19.09.23	Чт 21.09.23	0%	3 675,00 Р	16,8 ч
12	Разработка кода	3 дней	Пт 22.09.23	Вт 26.09.23	0%	3 675,00 Р	16,8 ч
13	Выбор технологий	1 день	Ср 27.09.23	Ср 27.09.23	0%	1 225,00 Р	5,6 ч
14	Формирование требований	1 день	Чт 28.09.23	Чт 28.09.23	0%	1 225,00 Р	5,6 ч
15	Конец проектирования архитектуры	0 дней	Чт 28.09.23	Чт 28.09.23	0%	0,00 Р	0 ч
16	[-] Разработка прикладной части	36 дней	Пт 29.09.23	Пт 17.11.23	0%	40 425,00 Р	184,8 ч
17	Разработка архитектуры прикладной части	3 дней	Пт 29.09.23	Вт 03.10.23	0%	3 675,00 Р	16,8 ч
18	Разработка функционала	3 дней	Ср 04.10.23	Пт 06.10.23	0%	3 675,00 Р	16,8 ч
19	Разработка программного кода	9 дней	Пн 09.10.23	Чт 19.10.23	0%	5 512,50 Р	25,2 ч
20	Разработка программного кода	15 дней	Пн 09.10.23	Пт 27.10.23	0%	9 187,50 Р	42 ч
21	Разработка программного кода	15 дней	Пн 30.10.23	Пт 17.11.23	0%	18 375,00 Р	84 ч
22	Конец разработки прикладной части	0 дней	Пт 17.11.23	Пт 17.11.23	0%	0,00 Р	0 ч
23	[-] Тестирование программного кода	9 дней	Пн 20.11.23	Чт 30.11.23	0%	11 025,00 Р	50,4 ч
24	Выбор метрик	1 день	Пн 20.11.23	Пн 20.11.23	0%	1 225,00 Р	5,6 ч
25	Функциональное тестирование	2 дней	Вт 21.11.23	Ср 22.11.23	0%	2 450,00 Р	11,2 ч
26	Нефункциональное тестирование	2 дней	Чт 23.11.23	Пт 24.11.23	0%	2 450,00 Р	11,2 ч
27	UNIT-тестирование	3 дней	Пн 27.11.23	Ср 29.11.23	0%	3 675,00 Р	16,8 ч
28	Вычисление метрик	1 день	Чт 30.11.23	Чт 30.11.23	0%	1 225,00 Р	5,6 ч
29	Конец тестирования программного кода	0 дней	Чт 30.11.23	Чт 30.11.23	0%	0,00 Р	0 ч
30	Составление glossary	2 дней	Пт 01.12.23	Пн 04.12.23	0%	2 450,00 Р	11,2 ч
31	Составление отчёта	7 дней	Вт 05.12.23	Ср 13.12.23	0%	8 575,00 Р	39,2 ч
32	Конец проекта	0 дней	Ср 13.12.23	Ср 13.12.23	0%	0,00 Р	0 ч

Рисунок 9 – Распределение временных и денежных затрат

Итоговые временные затраты на создание проекта составляют 398 часов, а денежные – 86975 рублей.

2 Проектирование серверного приложения

2.1 Моделирование функциональных возможностей приложения

Для отображения моделирование функциональных возможностей разрабатываемого приложения была разработана диаграмма вариантов использования, представленная на рисунке 10.

На диаграмме представлены функции для трех категорий пользователя: обычный пользователь, рецензент и администратор. В ходе выполнения своих функций они обращаются к определенным информационным объектам. Каждый из основных вариантов использования поделен на подфункции, отражающие будущие функции системы.

Описание вариантов использования представлено в таблице 3.

Таблица 3 – Описание вариантов использования

№ варианта	Название варианта использования	Описание варианта использования
1	Войти на сайт	Пользователь вводит свои данные для авторизации и попадает на главную страницу приложения. Возможно войти на сайт без авторизации, но с ограниченным функционалом
2	Написать рецензию	Рецензент формирует рецензию на фильм, используя предложенную форму
3	Взаимодействовать со списком рецензий	Пользователь выбирает рецензию для просмотра и оценки. Для быстрого поиска пользователь может найти рецензии с определённым названием или к определённому фильму.
4	Взаимодействовать с личным кабинетом	Пользователь просматривает или дополняет информацию о себе в своём личном кабинете. Отправление заявки на получение роли «рецензента» происходит в отдельной вкладке личного кабинета.
5	Взаимодействовать с информацией о фильмах	Пользователь выбирает фильм для просмотра информации о нем, либо рецензий, написанных к этому фильму. Для быстрого поиска пользователь может воспользоваться фильтрами или поисковой строкой.
6	Взаимодействовать со списком фильмов	Администратор производит добавление фильма в список, изменение информации о нем, либо удаляет его из списка.
7	Взаимодействовать со списком жалоб	Администратор взаимодействует со списками жалоб на рецензии или комментарии. Администратор может отказать в жалобе, выдать предупреждение, удалить рецензию/комментарий, либо, в особых случаях, заблокировать пользователя, на которого пришла жалоба.
8	Взаимодействовать со списком верификации	Администратор просматривает список заявок на получение роли «рецензента» и принимает, либо отклоняет их.

Разработанная для будущего приложения диаграмма вариантов использования представлена на рисунке 10.

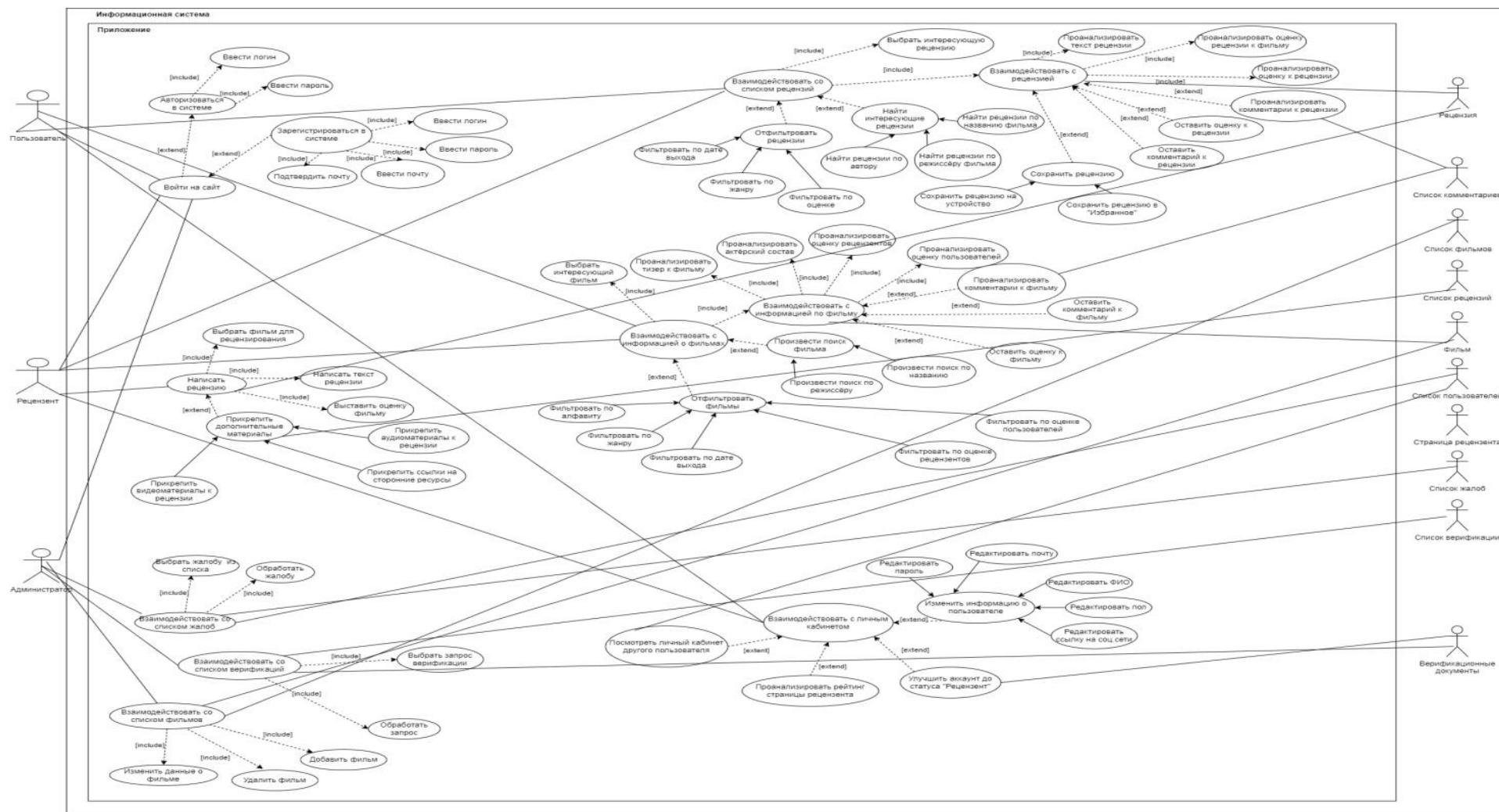


Рисунок 10 – Диаграмма вариантов использования для разрабатываемого приложения

Сценарий исполнения варианта использования «Войти на сайт» представлен на рисунке 11.

Войти на сайт
Id: 1
Описание: Пользователь входит на сайт
Действующие лица: Пользователь
Предусловие: Пользователь открывает сайт
<p>Основной поток:</p> <p>1.Пользователь хочет авторизоваться в системе:</p> <ul style="list-style-type: none"> - Пользователь вводит логин; - Пользователь вводит пароль; <p>Альтернативные потоки:</p> <p>1а. Пользователь хочет зарегистрироваться в системе:</p> <ul style="list-style-type: none"> - Вводит логин; - Вводит пароль; - Вводит почту; - Подтверждает почту; <p>1б. Пользователь заходит без авторизации.</p>
Постусловие: Пользователь находится на главной странице сайта

Рисунок 11 – Сценарий варианта использования «Войти на сайт»

Сценарий исполнения варианта использования «Написать рецензию» представлен на рисунке 12.

Написать рецензию
Id:2
Описание: Рецензент пишет рецензию
Действующие лица: Рецензент
Предусловие: Открыто окно создания рецензии
<p>Основной поток:</p> <ul style="list-style-type: none"> - Рецензент выбирает фильм для рецензирования; - Пишет текст рецензии; - Ставит оценку фильму; <p>Альтернативные потоки:</p> <p>1а. Прикрепляет дополнительные материалы:</p> <ul style="list-style-type: none"> - Прикрепляет видеоматериалы к рецензии; - Прикрепляет ссылки на сторонние ресурсы; - Прикрепляет аудиоматериалы к рецензии;
Постусловие: Добавлена рецензия к списку рецензий

Рисунок 12 – Сценарий варианта использования «Написать рецензию»

Сценарий исполнения варианта использования «Взаимодействовать со списком жалоб» представлен на рисунке 13.

Взаимодействовать со списком жалоб
Id:3
Описание: Администратор обрабатывает жалобы
Действующие лица: Администратор
Предусловие: Администратор хочет обработать жалобу
Основной поток: - Администратор выбирает жалобу из списка; - Обрабатывает жалобу;
Постусловие: Администратор обработал жалобу

Рисунок 13 – Сценарий варианта использования «Взаимодействовать со списком жалоб»

Сценарий исполнения варианта использования «Взаимодействовать со списком фильмов» представлен на рисунке 14.

Взаимодействовать со списком фильмов
Id:4
Описание: Администратор <u>добавляет/удаляет/изменяет</u> фильм
Действующие лица: Администратор
Предусловие: открыта страница с фильмами
Основной поток: 1. Администратор добавляет фильм; Альтернативные потоки: 1а. Администратор удаляет фильм; 1б. Администратор изменяет фильм;
Постусловие: Данные о фильмах изменены/ просмотрены

Рисунок 14 – Сценарий варианта использования «Взаимодействовать со списком фильмов»

Добавление фильма регулируется администратором для обеспечения корректности наполнения списка.

Сценарий исполнения варианта использования «Взаимодействовать со списком верификаций» представлен на рисунке 15.

Взаимодействовать со списком верификаций
Id:5
Описание: Администратор хочет обработать запрос с верификацией
Действующие лица: Администратор
Предусловие: Открыта страница с запросами верификаций
Основной поток: - Администратор выбирает запрос с верификацией; - Обработывает запрос с верификацией;
Постусловие: Администратор обработал запрос с верификацией

Рисунок 15 – Сценарий варианта использования «Взаимодействовать со списком верификаций»

Обработывая запрос на верификацию, администратор либо выдает пользователю роль «рецензента», либо отклоняет её, и системы отправляет уведомление пользователю.

Сценарий исполнения варианта использования «Взаимодействовать с личным кабинетом» представлен на рисунке 16.

Взаимодействовать с личным кабинетом
Id:6
Описание: Пользователь/рецензент хочет взаимодействовать с личным кабинетом
Действующие лица: Пользователь/рецензент
Предусловие: Пользователь/рецензент прошел авторизацию
Основной поток: 1. Пользователь/рецензент изменяет информацию о себе в своем личном кабинете; Альтернативные потоки: 1а. Улучшает аккаунт до статуса «Рецензент»; 1б. Анализирует рейтинг страницы рецензента; 1в. Смотрит личный кабинет <u>другого</u> ;
Постусловие: Данные в личном кабинете просмотрены или отредактированы

Рисунок 16 – Сценарий варианта использования «Взаимодействовать с личным кабинетом»

Сценарий исполнения варианта использования «Взаимодействовать с информацией о фильмах» представлен на рисунке 17.

Взаимодействовать с информацией о фильмах	
Id: 7	
Описание:	Пользователь/рецензент может фильтровать/искать/выбирать фильмы
Действующие лица:	Пользователь/рецензент
Предусловие:	Открыта страница с фильмами
Основной поток:	<ol style="list-style-type: none"> 1. Выбирает интересующий фильм 2. Взаимодействует с информацией по фильму;
Альтернативные потоки:	<ol style="list-style-type: none"> 1а. Фильтрует фильмы; 1б. Производит поиск;
Постусловие:	Переход на страницу с описанием фильма

Рисунок 17 – Сценарий варианта использования «Взаимодействовать с информацией о фильмах»

Сценарий исполнения варианта использования «Взаимодействовать со списком рецензий» представлен на рисунке 18.

Взаимодействовать со списком рецензий	
Id: 8	
Описание:	Пользователь/рецензент может выбирать/фильтровать/искать рецензии
Действующие лица:	Пользователь/рецензент
Предусловие:	Открыта страница с рецензиями
Основной поток:	<ol style="list-style-type: none"> 1. Выбирает интересующую рецензию 2. Взаимодействует с рецензией;
Альтернативные потоки:	<ol style="list-style-type: none"> 1а. Фильтрует рецензии; 1б. Производит поиск;
Постусловие:	Переход на страницу с описанием рецензии

Рисунок 18 – Сценарий варианта использования «Взаимодействовать со списком рецензий»

На рисунках выше представлены сценарии выполнения основных бизнес-процессов системы.

2.2 Разработка концептуальной модели данных

Парадигма объектно-ориентированного программирования повсеместно используется при создании современного программного обеспечения. Модель объектов, заложенная в эту парадигму, способна достаточно точно описывать свойства и возможности сущностей реального мира [3].

Перенести объекты в базу данных напрямую нельзя – обычные базы данных не поддерживают такую объектную структуру. Инструмент объектно-реляционного отображения ORM (Object-Relational Mapping) помогает упростить перевод между двумя парадигмами: объектно-ориентированного программирования и реляционных баз данных [4].

Создается база данных при помощи специальных классов сущностей (Entity). Поля таких классов транслируются в базу данных как атрибуты таблицы, Ограничения обозначаются в специальных аннотациях над полями.

Для проектирования концептуальной модели данных на основе объектного создания базы данных была создана диаграмма классов, представленная на рисунке 19.

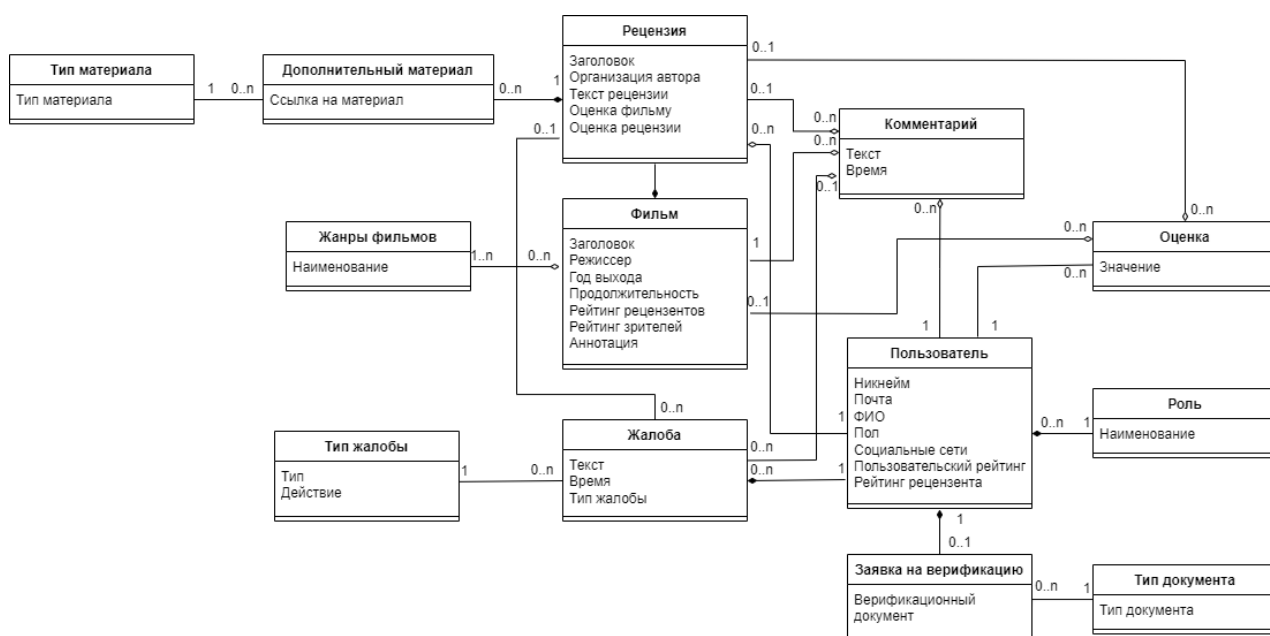


Рисунок 19 – Диаграмма классов для объектной модели данных

Табличное описание классов представлено в таблице 4.

Таблица 4 – Описание классов объектной модели данных

№	Название класса	Описание класса	Поле класса	Описание поля
1	Тип материала	Класс для формирования справочника типов дополнительных материалов	Тип материала	Содержит название дополнительного материала
2	Дополнительный материал	Класс для формирования таблицы с дополнительными материалами к рецензии	Ссылка на материал	Содержит ссылку на дополнительный материал
3	Рецензия	Класс для формирования таблицы рецензий на фильмы	Заголовок	Содержит заголовок рецензии. Заголовки рецензий могут повторяться
			Организация автора	Содержит информацию об организации автора
			Оценка фильму	Содержит выставленную оценку фильму
			Оценка рецензии	Содержит среднюю оценку рецензии от зрителей
4	Фильм	Класс для формирования таблицы с фильмами	Заголовок	Содержит заголовок фильма. Заголовки фильма могут повторяться
			Режиссер	Содержит имя фамилию режиссера, снявшего фильм
			Год выхода	Содержит год выхода картины
			Продолжительность	Содержит продолжительность фильма в минутах
			Рейтинг рецензентов	Содержит среднюю оценку рецензентов на фильм
			Рейтинг зрителей	Содержит среднюю оценку зрителей на фильм
			Аннотация	Содержит текстовую аннотацию к фильму
5	Жанры фильмов	Класс для формирования справочника с жанрами фильмов	Наименование	Содержит название жанра
6	Комментарий	Класс для формирования таблицы с комментариями	Текст	Содержит текстовый комментарий к фильму или рецензии
			Время	Содержит дату и время отправления комментария
7	Оценка	Класс для формирования таблицы оценок к фильмам и рецензиям	Значение	Содержит числовую оценку к фильму или рецензии. Оценка может находиться в диапазоне от 0 до 5
8	Тип жалобы	Класс для формирования справочника	Тип	Содержит тип жалобы: нецензурная лексика, расизм, оскорбление чувств верующих и другое.
			Действие	Содержит возможное наказание за определенное нарушение
9	Жалоба	Класс для формирования таблицы с жалобами на рецензии или комментарии	Текст	Содержит текст жалобы
			Время	Содержит время, в которое жалоба была отправлена

Продолжение таблицы 4

№	Название класса	Описание класса	Поле класса	Описание поля
10	Пользователь	Класс для формирования таблицы с пользователями	Никнейм	Содержит имя пользователя в системе. Имя должно быть уникальным
			Почта	Содержит адрес электронной почты пользователя
			ФИО	Содержит фамилию, имя и отчество (при наличии) пользователя
			Пол	Содержит информацию о гендере пользователя
			Социальные сети	Содержит ссылки на социальные сети пользователя
			Пользовательский рейтинг	Содержит пользовательский рейтинг. Рейтинг повышается ежегодно, а уменьшается из-за принятых жалоб
			Рейтинг рецензента	Содержит рейтинг рецензента, зависящий от рейтинга его рецензий
11	Роль	Класс для формирования справочника с ролями пользователей	Наименование	Содержит наименование роли
12	Заявка на верификацию	Класс для формирования таблицы с заявками на верификацию	Верификационный документ	Содержит ссылку на верификационный документ, подтверждающий экспертность пользователя
13	Тип документа	Класс для формирования справочника с типами верификационных документов	Тип документа	Содержит название типа верификационного документа

Данная модель будет использована для создания физической модели базы данных и дальнейшего её формирования.

2.3 Обобщенная архитектура веб-портала

При проектировании приложения была выбрана клиент-серверная архитектура. Клиент представляет собой Web-приложение или приложение на устройстве и отвечает за верстку и отображение контента в системе. Серверная часть в такой архитектуре отвечает за получение и обработку поступающих запросов от клиента, а также взаимодействие с сервером базы данных. Сервер базы данных используется для хранения данных приложения.

Из плюсов архитектуры можно выделить следующие:

- отсутствие дублирования – весь сайт хранится на машине-сервере, позволяя удалённо обращаться к нему с разных устройств;

- минимальные требования к пользователю – для доступа к сайту требуется лишь его клиентская часть;
- безопасность – данные хранятся в базе данных и пользователи не могут напрямую взаимодействовать с ними;
- удобство модернизации – возможность модернизации системы без необходимости выпускать официальные обновления системы.

Можно выделить и некоторые минусы архитектуры:

- выход из строя сервера или базы данных сделает сервис недоступным сразу для всех пользователей;
- затраты на поддержку – серверу нужен специалист, который будет обслуживать сервер и быстро реагировать на возможные поломки [5].

Обобщённая архитектура проектируемого приложения отражена на диаграмме развертывания, представленной на рисунке 20.

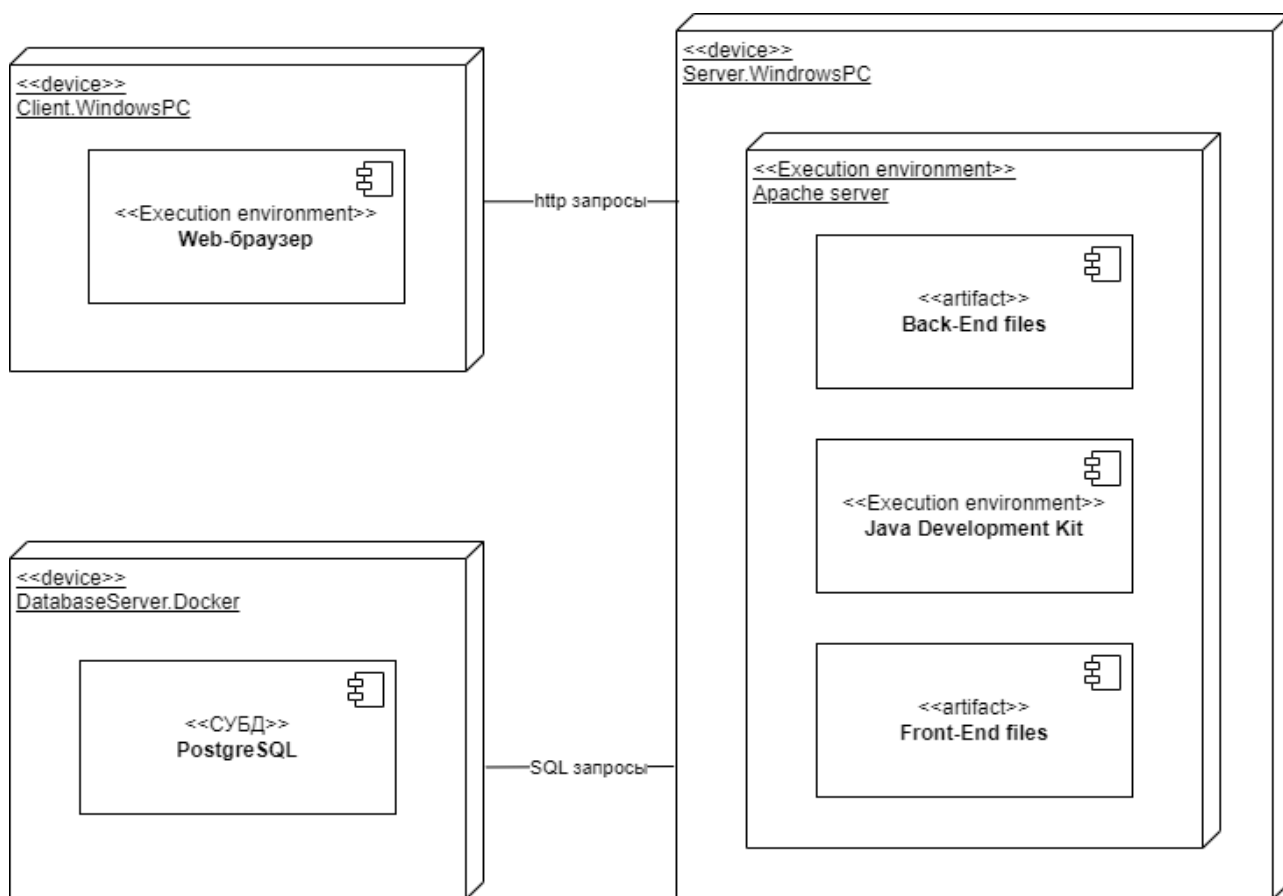


Рисунок 20 – Диаграмма развертывания для разрабатываемой системы

На диаграмме отражены три элемента системы: компьютер клиента с любым web-браузером для обращения к серверу с помощью http запросов, компьютер сервера, хранящий в себе все файлы системы (кроме файлов базы данных) и выполняющий все операции, и docker-контейнер с развёрнутой СУБД PostgreSQL, к которому сервер обращается с SQL запросами.

В свою очередь, компьютер сервера хранит у себя Java Backend файлы, отвечающие за обработку информации и взаимодействие с базой данных и файлами FrontEnd, формирующие страницы для Web-браузера клиента. Кроме того, на машине сервера находится пакет JDK (Java Development Kit), включающий в себя виртуальную машину JVM (Java Virtual Machine) и средства разработки для запуска java файлов.

2.4 Требования к аппаратно-программному обеспечению серверного приложения

Для корректной работы разрабатываемого серверного приложения машина, на которой оно будет разворачиваться, должна соответствовать некоторым требованиям. Минимальные и рекомендуемые требования к аппаратному обеспечению представлены в таблице 5.

Таблица 5 – Требования к аппаратному обеспечению

Характеристика	Минимум	Рекомендуется
Процессор	x64 1.4 ГГц	X64 2 ГГц и выше
Оперативная память	2 Гб	8 Гб и выше
Тип хранилища данных	HDD	SSD
Место на хранилище данных	32 Гб	100 Гб и выше
Пропускная способность сетевого интерфейса	1 Гбит/с	1 Гбит/с

Кроме того, необходимо иметь на машине некоторое программное обеспечение: JDK (Java Development Kit) 8 версии или выше, Docker версии 3.10 или выше с развёрнутым контейнером системы управления базами данных PostgreSQL версии 10 или выше.

2.5 Обоснование выбора технологий и средств разработки серверного приложения

Для разработки серверного приложения был выбран язык программирования Java. Он предоставляет фреймворки Spring и Hibernates для создания масштабируемых, безопасных и эффективных серверных систем. Язык Java лучше конкурентов справляется с тяжелыми вычислительными задачи благодаря своей многопоточной функциональности. Благодаря JVM (Java Virtual Machine), приложения на данном языке можно запускать на любом устройстве. Эти аспекты делают Java самым популярным языком для серверной разработки [6].

В качестве СУБД был выбран PostgreSQL, так как он является надежной, производительной и бесплатной платформой для базы данных сервера. Надежность обеспечивается механизмом транзакций, гарантирующих целостность данных даже в случае сбоев, а также ряда встроенных механизмов безопасности. Производительность достигается благодаря оптимизированной структуре СУБД [7].

Сервер СУБД PostgreSQL разворачивается в контейнере средства контейнеризации Docker. Она позволяет нам разворачивать сервер базы данных на специально созданном и настроенном образе PostgreSQL в контейнере. Контейнеризация позволяет нам разворачивать базу в удобном для нас месте, а также быстро и безболезненно переходить на другую версию СУБД при необходимости.

3 Разработка программного обеспечения серверного приложения

3.1 Разработка архитектуры серверного приложения

В качестве архитектуры серверного приложения была выбрана монолитная архитектура с четырёхуровневой топологией.

Монолитная архитектура – это традиционная модель разработки ПО, в которой одна база кода используется для выполнения нескольких бизнес-функций [8]. При этом, она имеет доступ к общей базе данных.

Многоуровневая, или N-уровневая архитектура – одна из разновидностей монолита. Компоненты в этой архитектуре собраны в логические горизонтальные уровни, каждый из которых выполняет определённую роль и работает со своей логикой [9].

Типовая архитектура четырёхуровневого монолита представлена на рисунке 21.

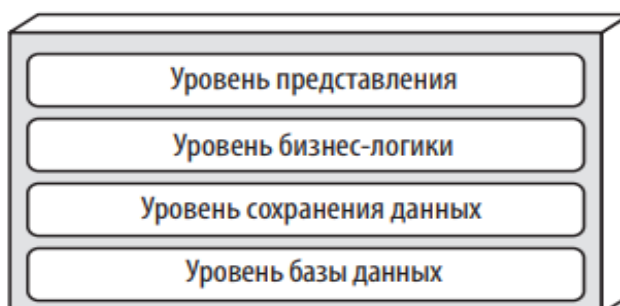


Рисунок 21 – Типовая архитектура четырёхуровневого монолита

Разрабатываемое серверное приложение написано в рамках рекомендаций REST (Representational State Transfer) и предоставляется единообразный интерфейс для HTTP запросов к системе.

Взаимодействие компонентов серверного приложения при типовом HTTP запросе отражено в диаграмме последовательности, представленной на рисунке 22.

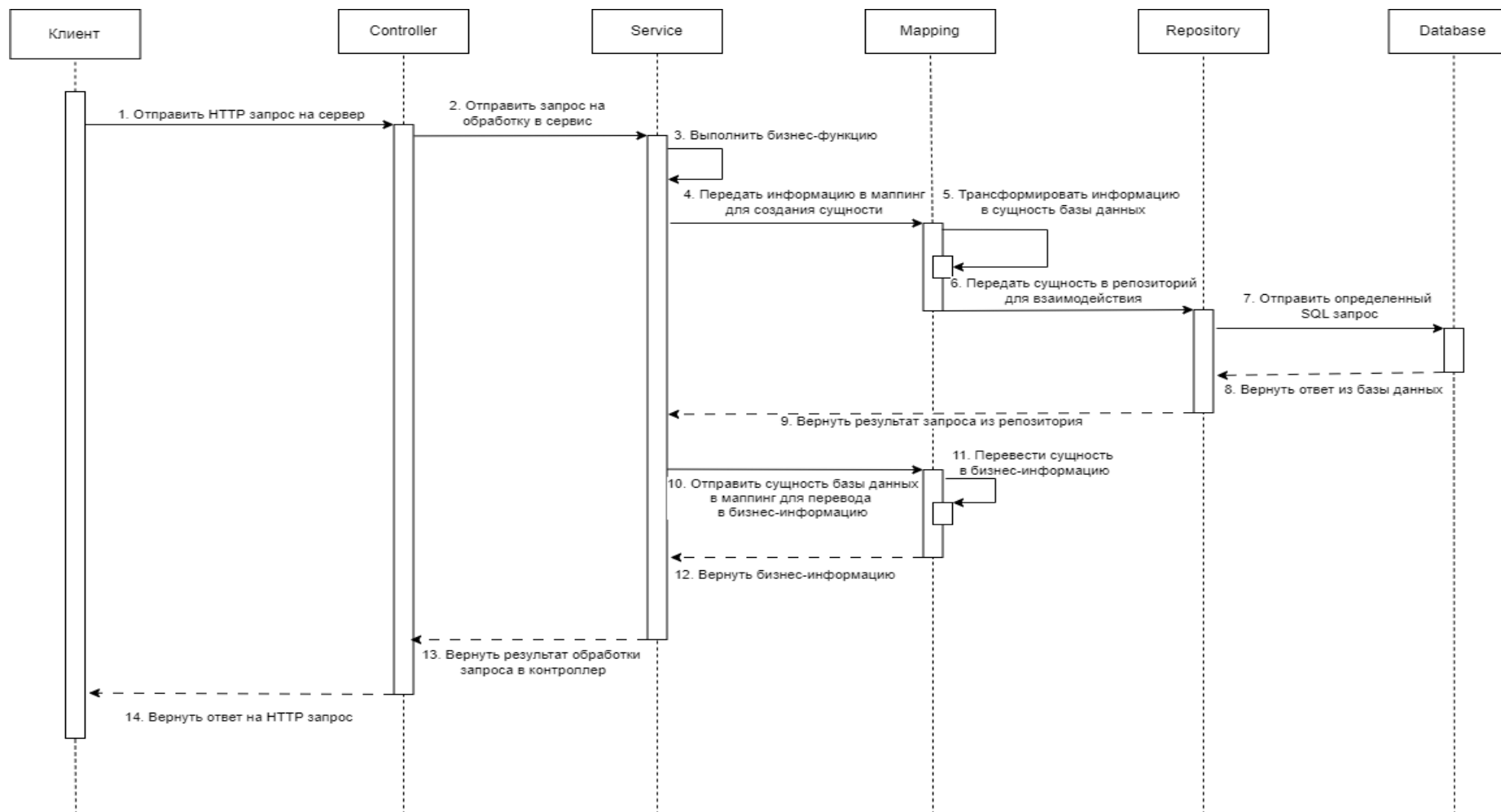


Рисунок 22 – Диаграмма деятельности для процессов, протекающих в области кинокритики

Отправленный клиентом запрос попадает на уровень представления, состоящий из контроллеров (Controller), которые обеспечивают внешние API для HTTP запросов.

Далее тело запроса отправляется в обработку на уровень бизнес-логики, представленный сервисами (Service). Они проводят выполнение некоторых бизнес-функций и, при необходимости, отправляют данные в маппинг или на уровень сохранения данных. Маппинг проводится силами служебных классов Mapping для перевода информации из формата Data Transfer Object в формат Entity.

Уровень сохранения данных покрывают репозитории (JpaRepository), с помощью которых организовано взаимодействие с базой данных. Репозитории часто оперируют объектами типа сущность (Entity) при создании запросов к базе данных и получении ответа от неё.

Уровень базы данных представляют сущности (Entity), функции и триггеры, из которых формируется база данных.

3.2 Разработка физической модели базы данных

Физическая модель данных – это реализация логической модели данных, создаваемая разработчиками баз данных [10]. Она разрабатывается для определенных СУБД, технологий хранения и соединителей данных. Физическая модель – это конечный результат моделирования базы данных, отражающий её фактическую реализацию.

В физической модели данных отражаются те имена атрибутов, что будут в базе данных. Кроме того, на ней отображаются первичные и внешние ключи, уникальные значения, типы данных и ограничения. В физической модели связь «многие-ко-многим» отображается как две связи «один-ко-многим» к промежуточной, искусственно созданной таблице.

При разработке базы данных была сформирована диаграмма «сущность-связь» физической модели данных, представленная на рисунке 23.

Табличное описание сущностей представлено в таблице 6.

Таблица 6 – Описание сущностей

№ п/п	Наименование	Описание	Категории	Количество экземпляров	Режим обновления
1	Review	Список созданных рецензий	Стержневая	1000000	30% за год
2	Addition materials	Список материалов, прикрепляемых к рецензиям	Стержневая	1000000	20% за год
3	Addition material type	Типы прикрепляемых материалов	Справочная	4	Нет
4	Film	Список фильмов, представленных на сайте	Стержневая	600000	15% за год
5	Film genre	Список жанров фильмов	Справочная	19	Нет
6	Rating	Список оценок, выставленных к фильмам и рецензиям	Стержневая	1000000	30% за год
7	Comment	Список комментариев, оставленных к фильмам и рецензиям	Стержневая	1000000	40% за год
8	Report	Список жалоб на комментарии или рецензии	Стержневая	1000000	100% за год
9	Type of reports	Типы жалоб и реакции на них	Справочная	4	Нет
10	User	Список пользователей, зарегистрированных на сайте	Стержневая	1000000	25% за год
11	Role	Виды ролей пользователей	Справочная	3	Нет
12	Verification list	Список заявок пользователей на верификацию	Стержневая	10000	3% за год
13	Document type	Виды подаваемых на верификацию документов	Справочная	5	Нет
14	Film genres	Жанры фильмов, находящихся в системе	Ассоциативная	50000	10% за год

Спецификации атрибутов сущностей представлены в приложении Б.

Сущности в Java – это классы, которые могут быть сопоставлены таблицам базы данных, и эти классы определяют структуру данных и действуют как мост между объектно-ориентированным программированием и реляционными базами данных [11].

Сущностные объекты таблиц, на основе которых формируется база данных, представлены в приложении В.

Скрипты создания функций и триггеров и заполнения таблиц представлены в приложении Г.

3.3 Разработка программного обеспечения доступа к базе данных

Для обеспечения доступа к базе данных были созданы классы Repository. Классы типа Repository предоставляют механизм для хранения, извлечения, обновления и удаления сущностей в базе данных [12]. Они реализуют функционал интерфейса JpaRepository и позволяют взаимодействовать с данными с помощью создаваемых методов, содержащих в своём названии SQL запрос.

Запросы к базе данных в классах Repository строятся с помощью создаваемых методов. Имя метода содержит в себе ключевые слова для формирования запроса, где сочетаются заранее зарезервированные «find», «get», «save», «delete» и передаваемые параметры. Передаваемые объекты определенного типа являются параметрами поиска. Возвращаемое значение содержит либо класс типа Entity, являющийся отражением определенной таблицы, либо определенное значение атрибута. Помимо создаваемых программистом функций, существует и ряд встроенных: getAll, findAll, save, deleteBy.

Для отображения созданных классов Repository была сформирована диаграмма классов, представленная на рисунке 24.

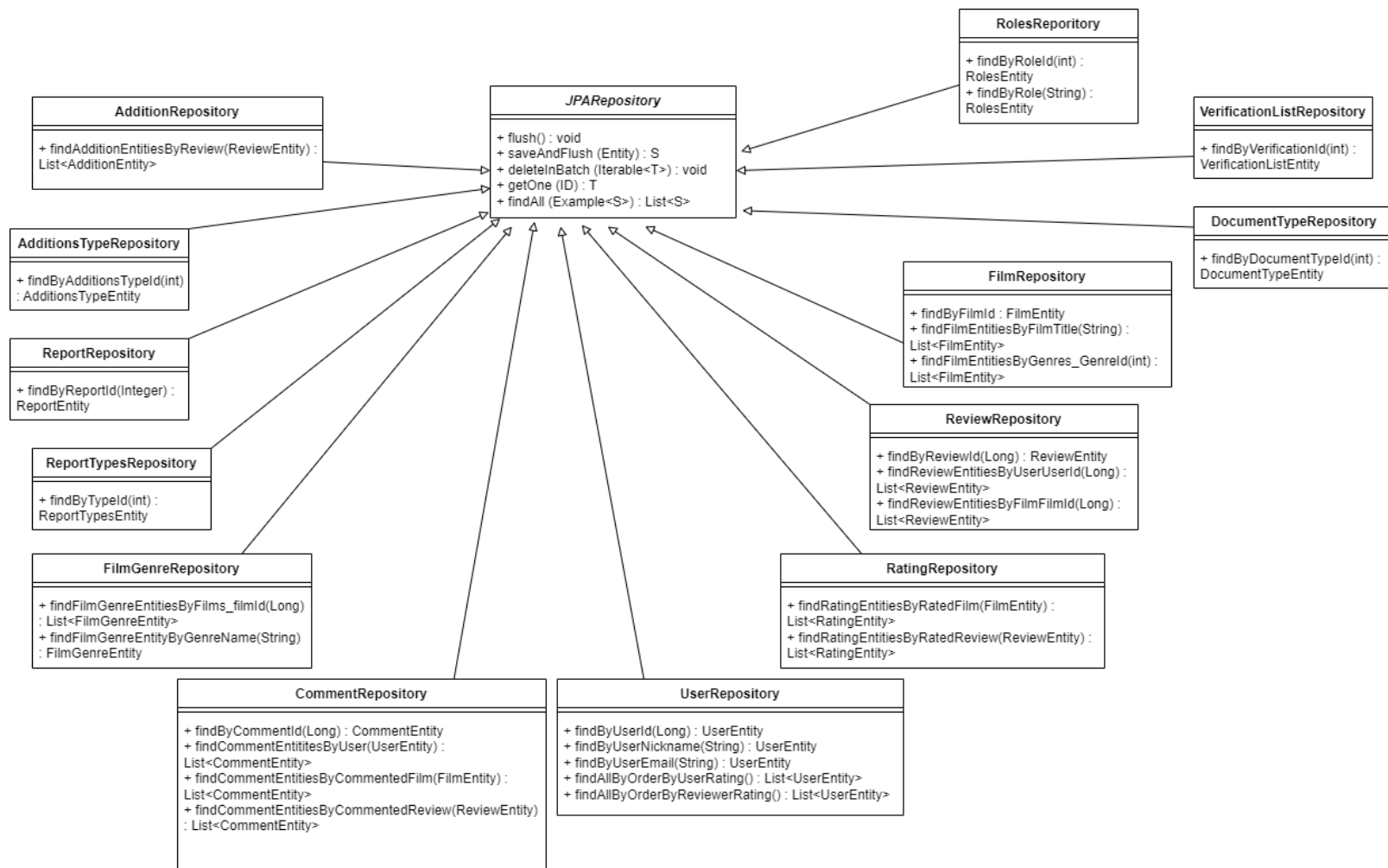


Рисунок 24 – Диаграмма классов для репозитория Repository

Описание классов Repository представлено в таблице 7.

Таблица 7 – Описание классов Repository

№ п/п	Название класса	Описание класса
1	AdditionRepository	Класс для доступа к таблице Addition. Реализован метод получения дополнительных материалов из таблицы.
2	AdditionTypesRepository	Класс для доступа к таблице-справочнику AdditionTypes. Реализован метод получения типа материала
3	CommentRepository	Класс для доступа к таблице Comment. Реализованы методы получения комментария по его id, получение списка комментариев по пользователю, фильму или рецензии
4	DocumentTypeRepository	Класс для доступа к таблице-справочнику DocumentType. Реализован метод получения типа документов
5	FilmGenreRepository	Класс для доступа к таблице FilmGenre. Реализованы метод получения списка всех жанров и метод получения списка жанров фильма по id
6	FilmRepository	Класс для доступа к таблице Film. Реализованы методы получения фильма по его id, получения списка фильмов по заголовку, получения фильмов по определенному жанру
7	RatingRepository	Класс для доступа к таблице Rating. Реализованы методы получения списка оценок по фильму или рецензии
8	ReportRepository	Класс для доступа к таблице Report. Реализован метод получения определенной жалобы по id
9	ReportTypesRepository	Класс для доступа к таблице-справочнику ReportTypes. Реализован метод получения типа жалобы
10	ReviewRepository	Класс для доступа к таблице Review. Реализованы методы получения рецензии по её id, получение списка рецензий определенного пользователя и получение списка рецензий к фильму
11	RolesRepository	Класс для доступа к таблице-справочнику Roles. Реализован метод получения роли
12	UserRepository	Класс для доступа к таблице Users. Реализованы методы получения пользователя по его id, никнейму или почте, а также получение списка пользователей, отсортированного по рейтингу пользователя или рецензента
13	VerificationListRepository	Класс для доступа к таблице VerificationList. Реализован метод получения верификации

Пример созданного репозитория представлен в листинге 1.

Листинг 1 – Класс-репозиторий UserRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.UserEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends JpaRepository<UserEntity, Long> {
    UserEntity findByUserId (Long userId);

    UserEntity findByUserNickname (String nickname);

    UserEntity findByUserEmail (String email);
}
```

```

        List<UserEntity> findAllByOrderByUserRating ();

        List<UserEntity> findAllByOrderByReviewerRating ();
    }

```

Тип возвращаемого значения этой функции является возвращаемой или передаваемой сущностью базы данных. Тело функций в репозитории можно оставить пустым, если не требуется изменять стандартное поведение скриптов SQL.

Код реализации репозитория представлен в приложении Д.

3.4 Разработка программного обеспечения взаимодействия с клиентским приложением

Взаимодействие с клиентским приложением организовано посредством классов Controller и http запросов к ним.

Класс Controller создаётся с аннотацией RestController и позволяет классу обрабатывать запросы, отправляемые клиентом [13]. Они принимают GET, POST, DELETE запросы, перенаправляют необходимые параметры запроса в сервисы для выполнения обработки и возвращают клиенту ответ.

Пример созданного контроллера представлен в листинге 2.

Листинг 2 – Класс-контроллер RatingController.java

```

package com.example.bdcourse.controller;

import com.example.bdcourse.dto.FilmDto;
import com.example.bdcourse.dto.RatingDto;
import com.example.bdcourse.dto.ReviewDto;
import com.example.bdcourse.dto.UserDto;
import com.example.bdcourse.mapping.FilmMapping;
import com.example.bdcourse.mapping.RatingMapping;
import com.example.bdcourse.mapping.ReviewMapping;
import com.example.bdcourse.service.RatingService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

@RestController
@RequiredArgsConstructor

```

```

@RequestMapping("/api/bdcourse/rating")
public class RatingController {
    @Autowired
    private RatingService ratingService;
    @Autowired
    private FilmMapping filmMapping;
    @Autowired
    private ReviewMapping reviewMapping;
    @Autowired
    private RatingMapping ratingMapping;

    @PostMapping("/addrating")
    public ResponseEntity<Integer> addRating(@RequestBody RatingDto
ratingDto) {
        ratingService.addRate(ratingDto);
        return new ResponseEntity<>(HttpStatus.CREATED);
    }

    @GetMapping("/filmrate")
    public int takeFilmRate(@RequestParam("id") Long filmId) {
        String uri =
"http://localhost:8080/api/bdcourse/film/takefilm?id=" + filmId;
        RestTemplate restTemplate = new RestTemplate();
        FilmDto filmDto = restTemplate.getForObject(uri,
FilmDto.class);
        return filmDto != null
            ?
ratingService.calculateFilmRate(filmMapping.mapToFilmEntity(filmDto))
            : 0;
    }

    @GetMapping("/reviewrate")
    public int takeReviewRate(@RequestParam("id") Long reviewId) {
        String uri =
"http://localhost:8080/api/bdcourse/review/takereview?id=" + reviewId;
        RestTemplate restTemplate = new RestTemplate();
        ReviewDto reviewDto = restTemplate.getForObject(uri,
ReviewDto.class);
        return reviewDto != null?
ratingService.calculateReviewRate(reviewMapping.mapToReviewEntity(reviewD
to)) : 0;
    }
}

```

Кроме того, иногда требуется отправить внутренний запрос из одного контроллера в другой. Для этого используется инструмент `RestTemplate`, позволяющий отправлять HTTP запросы со стороны сервера.

Для отображения созданных классов `Controller` была разработана диаграмма классов, представленная на рисунке 25.

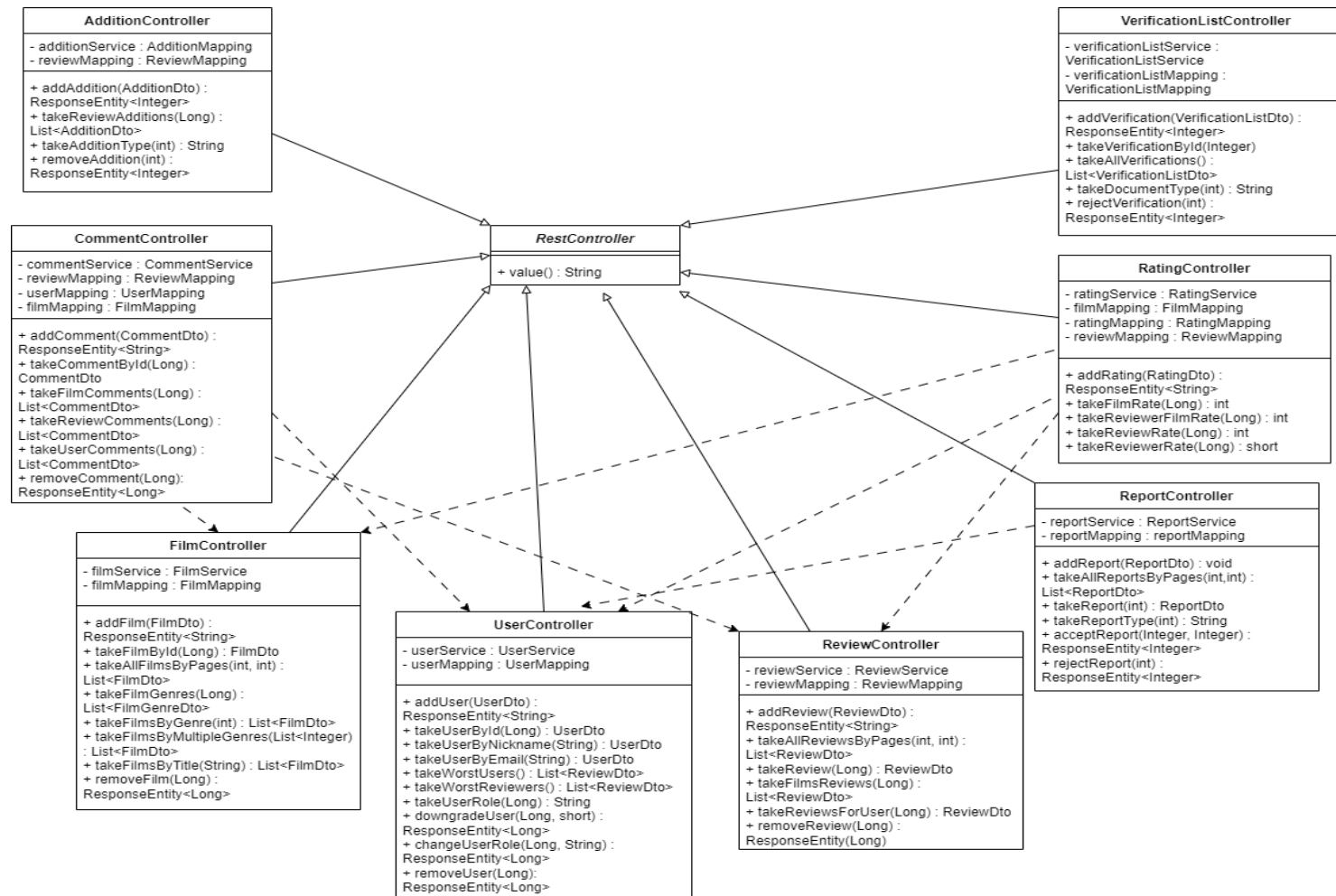


Рисунок 25 – Диаграмма классов для контроллеров Controller

Описание классов Controller представлено в таблице 8.

Таблица 8 – Описание классов Controller

№	Название	Описание класса
1	AdditionController	Контроллер предоставляет API для добавления, получения и удаления дополнительных материалов, а также для получения типа дополнительного материала
2	CommentController	Контроллер предоставляет API для следующих функций: добавить комментарий, получить комментарий по id, получить комментарии к фильму, получить комментарии к рецензии, получить комментарии пользователя, удалить комментарий
3	FilmController	Контроллер предоставляет API для следующих функций: добавить фильм, получить фильм по id, получить список фильмов постранично (первый параметр – номер страницы, второй – количество фильмов на одной странице), получить жанры фильма, получить фильмы по одному или нескольким жанрам, получить фильмы по заголовку, удалить фильм
4	UserController	Контроллер предоставляет API для следующих функций: добавление фильма, получение пользователя по id, никнейму или почте, получить десятку пользователей с наименьшим рейтингом пользователя или рейтингом рецензента, понижение рейтинга пользователя, смена роли пользователя, удаление пользователя
5	ReviewController	Контроллер предоставляет API для следующих функций: добавление рецензии, получение всех рецензий постранично (первый параметр – номер страницы, второй – количество рецензий на одной странице), получение рецензии по id, получение рецензий к фильму, получение рецензий пользователя, удаление рецензии
6	ReportController	Контроллер предоставляет API для следующих функций: добавить жалобу, получить список жалоб постранично (первый параметр – номер страницы, второй – количество жалоб на одной странице), получение жалобы по id, получение типа жалобы, принятие жалобы, отклонение жалобы
7	RatingController	Контроллер предоставляет API для следующих функций: добавить оценку, получить среднюю оценку фильма от зрителей или рецензентов, получить среднюю оценку зрителей к рецензии, получение рейтинга рецензента на основе зрительских оценок на его рецензии
8	VerificationListController	Контроллер предоставляет API для следующих функций: добавление верификации в лист, получение верификации по id, получение всех запросов на верификацию, получение типа верификационного документа, отклонение верификации (принятие проходит через смену роли в контроллере UserController)

Код реализации контроллеров представлен в приложении Е.

3.5 Разработка программного обеспечения бизнес-логики серверного приложения

Уровень бизнес-логики покрывается классами типа Service. Каждый класс обычно отвечает за связанный набор бизнес-процессов, где каждый процесс – метод [14]. Помимо основной логики, сервисы отвечают за взаимодействие между сущностями и их отображением на базу данных, преобразование сущностей в DTO (data transfer object) и обратно. Помечаются

они специальной аннотацией `Service`, для того, чтобы указать приложению на принадлежность класса к уровню бизнес-логики. Пример класса-сервиса представлен в листинге 3.

Листинг 3 – Класс-репозиторий `RatingService.java`

```
package com.example.bdcourse.service;

import com.example.bdcourse.dto.RatingDto;
import com.example.bdcourse.dto.ReviewDto;
import com.example.bdcourse.entity.FilmEntity;
import com.example.bdcourse.entity.RatingEntity;
import com.example.bdcourse.entity.ReviewEntity;
import com.example.bdcourse.mapping.RatingMapping;
import com.example.bdcourse.repository.RatingRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class RatingService {
    //used to map from ratingDto to ratingEntity and reverse
    @Autowired
    private RatingMapping ratingMapping;
    //used to interact with table Rating in database
    @Autowired
    private RatingRepository ratingRepository;

    //adding new rating to anything (user, film, review). Rated film
    and rated review can be empty
    public void addRate(RatingDto ratingDto) {
        RatingDto newRateDto = new RatingDto();
        newRateDto.setRatingValue(ratingDto.getRatingValue());
        newRateDto.setRatedUser(ratingDto.getRatedUser());
        newRateDto.setRatedFilm(ratingDto.getRatedFilm());
        newRateDto.setRatedReview(ratingDto.getRatedReview());

        ratingRepository.save(ratingMapping.mapToRatingEntity(newRateDto));
    }

    //calculate overall film rating by users ratings
    public int calculateFilmRate(FilmEntity film) {
        List<RatingEntity> ratings =
ratingRepository.findRatingEntitiesByRatedFilm(film);
        if (ratings.isEmpty())
            return 0;
        int rateSum = 0;
        for (RatingEntity rate : ratings) {
            rateSum += rate.getRatingValue();
        }
    }
}
```

```

        return rateSum / ratings.size();
    }

    //calculate overall review rating by users ratings
    public int calculateReviewRate(ReviewEntity review) {
        try {
            List<RatingEntity> ratings =
ratingRepository.findRatingEntitiesByRatedReview(review);
            if (ratings.isEmpty())
                return 0;
            int rateSum = 0;
            for (RatingEntity rate : ratings) {
                rateSum += rate.getRatingValue();
            }
            return rateSum / ratings.size();
        } catch (Throwable throwable) {
            return 0;
        }
    }

    //Calculate overall film rating by reviews rating
    public int calculateReviewersFilmRate(List<ReviewDto> dtos){
        int rateSum = 0;
        for (ReviewDto reviewDto : dtos){
            rateSum+=reviewDto.getFilmRate();
        }
        return rateSum/ dtos.size();
    }
}

```

В представленном сервисе реализована логика подсчёта рейтинга для фильмов на основе оценок рецензентов и обычных пользователей, а также подсчёт рейтинга самих рецензий на основе отзывов пользователей. Оценка ресурсу выставляется как среднее арифметическое из всех выставленных оценок. В случае с подсчетом оценки к фильму от рецензентов, в сервис сразу передаётся список рецензий с оценками внутри. Потому, в этом методе не используется репозиторий `RatingRepository`.

В случае с подсчетом средней оценки от пользователей к фильму или рецензии, необходимо в первую очередь получить список их оценок, а уже после проводить подсчет.

Поле `ratingMapping` используется в сервисе для перевода данных из формата DTO в формат Entity и обратно.

Для отображения созданных классов Service была разработана диаграмма классов, представленная на рисунке 26.

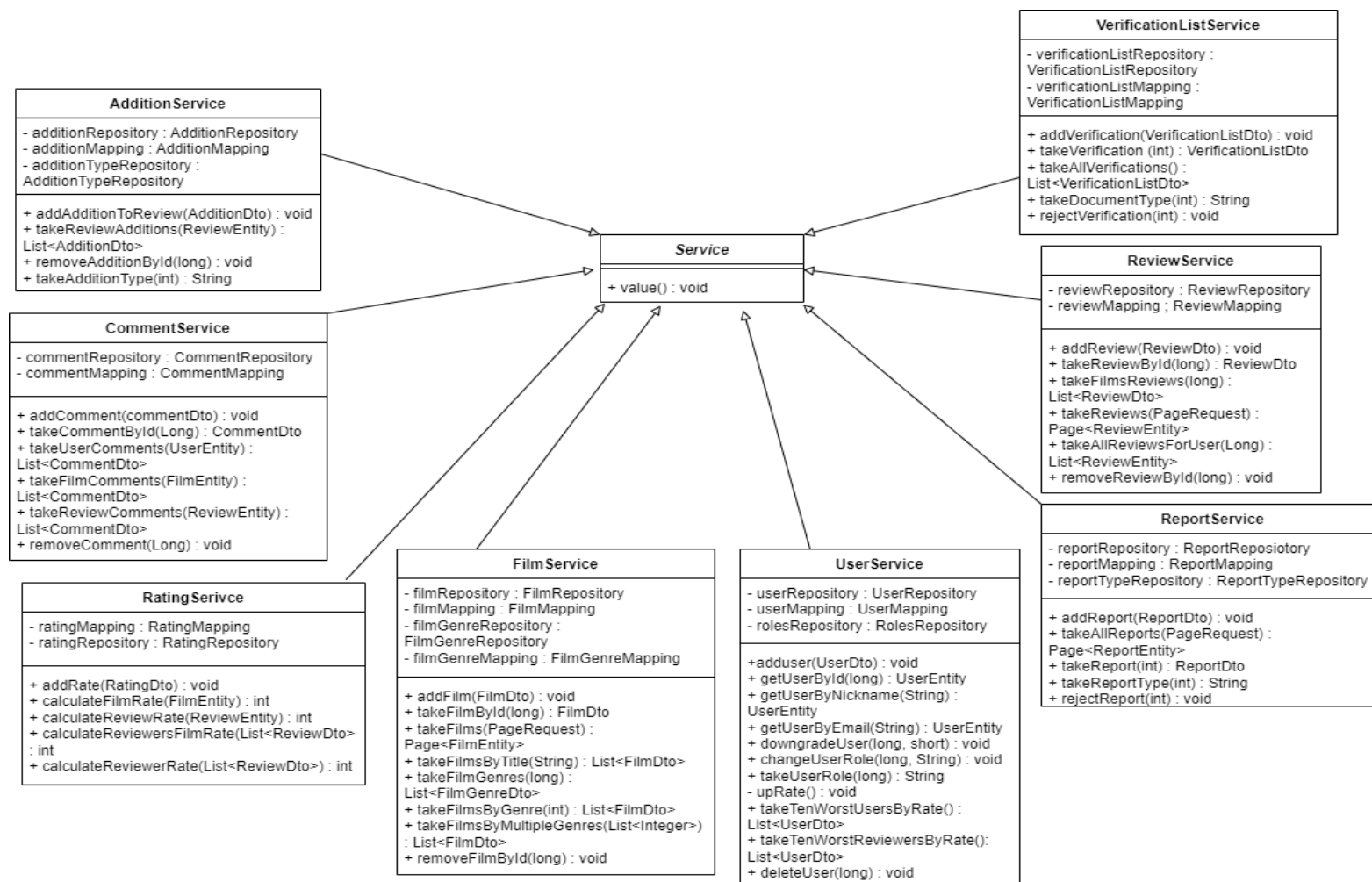


Рисунок 26 – Диаграмма классов для сервисов Service

Описание классов Service представлено в таблице 9.

Таблица 9 – Описание классов Service

№	Название	Описание класса
1	AdditionService	Сервис предоставляет методы для добавления, получения и удаления дополнительных материалов, а также для получения типа дополнительного материала
2	CommentService	Сервис предоставляет методы для следующих функций: добавить комментарий, получить комментарий по id, получить комментарии к фильму, получить комментарии к рецензии, получить комментарии пользователя, удалить комментарий
3	FilmService	Сервис предоставляет методы для следующих функций: добавить фильм, получить фильм по id, получить список фильмов постранично (первый параметр – номер страницы, второй – количество фильмов на одной странице), получить жанры фильма, получить фильмы по одному или нескольким жанрам, получить фильмы по заголовку, удалить фильм
4	UserService	Сервис предоставляет методы для следующих функций: добавление фильма, получение пользователя по id, никнейму или почте, получить десятку пользователей с наименьшим рейтингом пользователя или рейтингом рецензента, понижение рейтинга пользователя, ежегодного повышения рейтинга пользователя, смена роли пользователя, удаление пользователя
5	ReviewService	Сервис предоставляет методы для следующих функций: добавление рецензии, получение всех рецензий постранично (первый параметр – номер страницы, второй – количество рецензий на одной странице), получение рецензии по id, получение рецензий к фильму, получение рецензий пользователя, удаление рецензии
6	ReportService	Сервис предоставляет методы для следующих функций: добавить жалобу, получить список жалоб постранично (первый параметр – номер страницы, второй – количество жалоб на одной странице), получение жалобы по id, получение типа жалобы, принятие жалобы, отклонение жалобы
7	RatingService	Сервис предоставляет методы для следующих функций: добавить оценку, получить среднюю оценку фильма от зрителей или рецензентов, получить среднюю оценку зрителей к рецензии, получение рейтинга рецензента на основе зрительских оценок на его рецензии
8	VerificationListService	Сервис предоставляет методы для следующих функций: добавление верификации в лист, получение верификации по id, получение всех запросов на верификацию, получение типа верификационного документа, отклонение верификации (принятие проходит через смену роли в контроллере UserController)

Представленные на диаграмме сервисы покрывают функционал контроллеров, из-за чего имеют идентичные методы. Исключением является приватный метод `upRate` в классе `UserService`, который является внутрисистемным и увеличивает рейтинг всех пользователей в системе на 10 раз в год. Отсчет таймера начинается с момента первого запуска сервера.

Код реализации сервисов представлен в приложении Ж.

Рассмотрим взаимодействие компонентов системы на примере добавления рецензии на сайт. Для этого была создана диаграмма последовательности, представленная на рисунке 27.

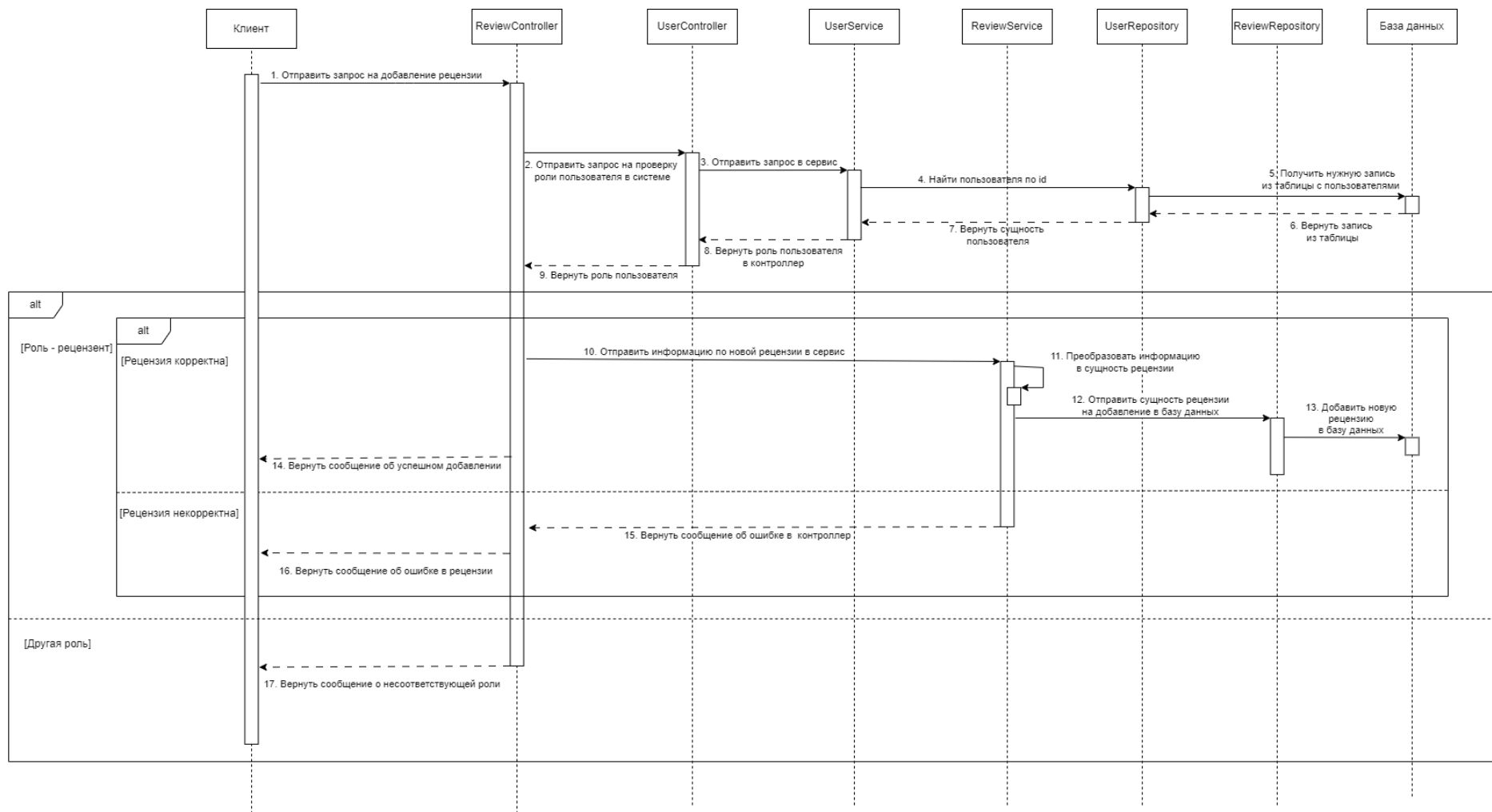


Рисунок 27 – Диаграмма последовательности для процесса добавления рецензии

Клиент отправляет запрос на контроллер `ReviewController` с информацией о добавляемой рецензии. Перед тем, как передать информацию сервису на бизнес-уровень, он отправляет запрос на другой контроллер `UserController` для получения роли пользователя, который пытается добавить рецензию. Запрос опускает вниз по уровням до уровня репозитория `UserRepository`, который получает запись с пользователем из базы данных и формирует из неё сущность пользователя (`UserEntity`). На уровне сервиса пользователя `UserService` из сущности выбирается лишь нужная в запросе информация – ранг пользователя. Полученная роль возвращается в контроллер `ReviewController`, и он, если роль соответствует, отправляет информацию о рецензии на сервис `ReviewService`. Информация о рецензии проверяется в сервисе и конвертируется в сущностный объект рецензии (`ReviewEntity`). Полученная сущность отправляется в репозиторий `ReviewRepository` для добавления новой записи в базу данных. Если всё прошло успешно, сервер возвращает клиенту сообщение об успешном выполнении запроса, иначе – возвращает клиенту сообщение об ошибке и указывает в нем, что именно пошло не так.

4 Контроль качества программного обеспечения

4.1 Планирование процесса контроля качества программного обеспечения

4.1.1 Обоснование выбора модели тестирования

По степени знания системы выделяют тестирование «черного ящика» и тестирование «белого ящика».

При первом виде тестирования взаимодействие возможно лишь с входными и выходными параметрами системы, при этом, содержимое и логика кода сокрыты. Такой подход позволяет сосредоточиться на тестировании взаимодействия программы с внешним миром, не вдаваясь в детали внутренней реализации.

При тестировании «белого ящика» тестирующий имеет доступ к внутренней логике программы. Это позволяет проверить детали реализации программы и выявить возможные ошибки, которые могли бы остаться незамеченными при тестировании «черного ящика» [15].

В качестве тестирования «черным ящиком» было выбрано тестирование API, а именно его вид – тестирование методов. Тестирование API (программного интерфейса приложений) – это ручная или автоматическая проверка обмена данными между двумя модулями программы, разными приложениями, веб-сервисами и серверами [16].

Тестирование методов заключается в том, что каждый метод тестируют по отдельности, чтобы убедиться, что они работают правильно и возвращают ожидаемые результаты [17].

Структурное тестирование является представителем тестирования «белого ящика» и основывается на детальном изучении логики программы и подборе тестов, которые обеспечивают максимально возможное покрытие операторов, логических ветвлений и условий [18].

При построении тестов используют следующие критерии:

- покрытие операторов путем выбора такого набора данных, при котором обеспечивается выполнение всех операторов в программе, по крайней мере один раз;
- покрытие условий путем подбора такого наборов данных, который обеспечивает в узлах ветвления с более чем одним условием принятие каждым условием значения «истина» или «ложь» хотя бы раз;
- комбинаторное покрытие условий путем подбора тестов, обеспечивающих в узлах ветвления с более чем одним условием перебор всех возможных сочетаний значений условий в одном узле ветвления.

4.1.2 Обоснование выбора метрик программного кода

Метрики кода представляют собой набор оценок программного обеспечения, которые дают разработчикам представление о разрабатываемом коде. Используя преимущества метрик кода, разработчики могут понять, какие типы и методы следует переработать или более тщательно протестировать [19].

Метрики программного кода делятся на несколько групп:

- метрики сложности потока управления программы (метрики Холстеда);
- количественные метрики (SLOC-метрика).

SLOC(Source Lines of Code)-метрика – это метрика программного кода, которая используется для измерения его объёма с помощью подсчёта количества его строк [20]. При этом выделяют два вида строк:

- физические строки, включающие в себя все строки кода, в том числе и пустые;
- логические строки, содержащие в себе команды.

Программные показатели Холстеда представляют собой набор показателей, предложенных Морисом Холстедом для оценки сложности программного обеспечения [21].

Метрика Холстеда состоит из четырёх характеристик программы:

- число уникальных операторов программы, включая имена процедур, знаки операций и символы-разделители;
- число уникальных операндов программы;
- общее число операторов в программе;
- общее число операндов в программе.

На основе этих характеристик можно рассчитать следующие оценки:

- словарь программы;
- длина программы;
- объем программы;
- сложность программы.

Следующие оценки позволят оценить сложность программы.

4.2 Тестирование программных средств

4.2.1 Тестирование API

Тестирование API проводилось с использованием средства документирования Swagger. Результаты тестирования основных процессов приведены в таблице 10. Иллюстрационные материалы по тестированию приведены в приложении И.

Таблица 10 – Результаты тестирования API

№ теста	Описание теста	Ожидаемый результат	Фактический результат	Тест пройден/ не пройден
1	Корректное добавление пользователя в систему	Пользователь успешно добавлен в систему	Соответствует ожидаемым результатам (рисунки И.1-И.2)	Тест пройден
2	Некорректное добавление пользователя в систему	Возвращен код 400 – Bad request. Пользователь не добавлен	Соответствует ожидаемым результатам (рисунок И.3)	Тест пройден
3	Получение пользователя по его никнейму	Возвращена информация по пользователю с соответствующим никнеймом	Соответствует ожидаемым результатам (рисунок И.4)	Тест пройден

Продолжение таблицы 10

№ теста	Описание теста	Ожидаемый результат	Фактический результат	Тест пройден/не пройден
4	Изменение роли пользователя	Роль пользователя успешно изменена	Соответствует ожидаемым результатам (рисунок И.5)	Тест пройден
5	Получение роли пользователя	Возвращена роль пользователя в системе	Соответствует ожидаемым результатам (рисунок И.6)	Тест пройден
6	Некорректное добавление фильма	Возвращён код 400 – Bad Request с сообщением об ошибке	Соответствует ожидаемым результатам (рисунок И.7)	Тест пройден
7	Добавление нового фильма	Новый фильм успешно добавлен в систему	Соответствует ожидаемым результатам (рисунок И.8)	Тест пройден
8	Получение списка фильмов	Получен список фильмов, разбитый по страницам	Соответствует ожидаемым результатам (рисунок И.9)	Тест пройден
9	Получение жанров фильма	Получен список жанров для запрошенного фильма	Соответствует ожидаемым результатам (рисунок И.10)	Тест пройден
10	Получение фильмов определённого жанра	Получен список фильмов, которые подходят под определённый жанр	Соответствует ожидаемым результатам (рисунок И.11)	Тест пройден
11	Некорректное добавление рецензии в систему	Получен код 400 – Bad Request с сообщением об ошибке	Соответствует ожидаемым результатам (рисунок И.12)	Тест пройден
12	Корректное добавление рецензии в систему	Рецензия успешно добавлена в систему	Соответствует ожидаемым результатам (рисунок И.13)	Тест пройден
13	Получение списка рецензий	Получен список рецензий, находящихся в системе	Соответствует ожидаемым результатам (рисунок И.14)	Тест пройден
14	Получение списка рецензий на конкретный фильм	Получен список рецензий, написанных к определённому фильму	Соответствует ожидаемым результатам (рисунок И.15)	Тест пройден
15	Получение списка рецензий, написанных конкретным пользователем	Получен список рецензий, написанных конкретным пользователем в системе	Соответствует ожидаемым результатам (рисунок И.16)	Тест пройден
16	Добавление оценки с некорректным значением	Возвращен код 400 – Bad Request с сообщением об ошибке	Соответствует ожидаемым результатам (рисунок И.17)	Тест пройден
17	Добавление оценки с корректным значением	Оценка успешно добавлена в систему	Соответствует ожидаемым результатам (рисунок И.18)	Тест пройден
18	Получение рейтинга рецензента	Получена оценка рецензента на основе оценок его рецензий	Соответствует ожидаемым результатам (рисунок И.19)	Тест пройден
19	Получение рейтинга рецензии	Получена средняя оценка рецензии на основе пользовательских оценок	Соответствует ожидаемым результатам (рисунок И.20)	Тест пройден
20	Получение рейтинга фильма от рецензентов	Получена средняя оценка рецензентов к фильму на основе их рецензий	Соответствует ожидаемым результатам (рисунок И.21)	Тест пройден
21	Получение рейтинга фильма от пользователей	Получена средняя оценка обычных пользователей к фильму	Соответствует ожидаемым результатам (рисунок И.22)	Тест пройден
22	Некорректное добавление комментария	Возвращен код 400 – Bad Request	Соответствует ожидаемым результатам (рисунок И.23)	Тест пройден
23	Корректное добавление комментария	Комментарий успешно добавлен в систему	Соответствует ожидаемым результатам (рисунок И.24)	Тест пройден

Окончание таблицы 10

№ теста	Описание теста	Ожидаемый результат	Фактический результат	Тест пройден/ не пройден
24	Получение комментариев пользователя	Получен список комментариев, написанный определенным пользователем	Соответствует ожидаемым результатам (рисунок И.25)	Тест пройден
25	Получение комментариев к рецензии	Получен список комментариев, написанный к определенной рецензии	Соответствует ожидаемым результатам (рисунок И.26)	Тест пройден
26	Получение комментариев к фильму	Получен список комментариев, написанный к определенному фильму	Соответствует ожидаемым результатам (рисунок И.27)	Тест пройден

Результаты прохождения теста показывают корректное функционирование всех основных API системы.

4.2.2 Структурное тестирование

Проведем тестирование на основе алгоритма «подсчёта рейтинга пользователей к фильму». Блок-схема алгоритма представлена на рисунке 28.

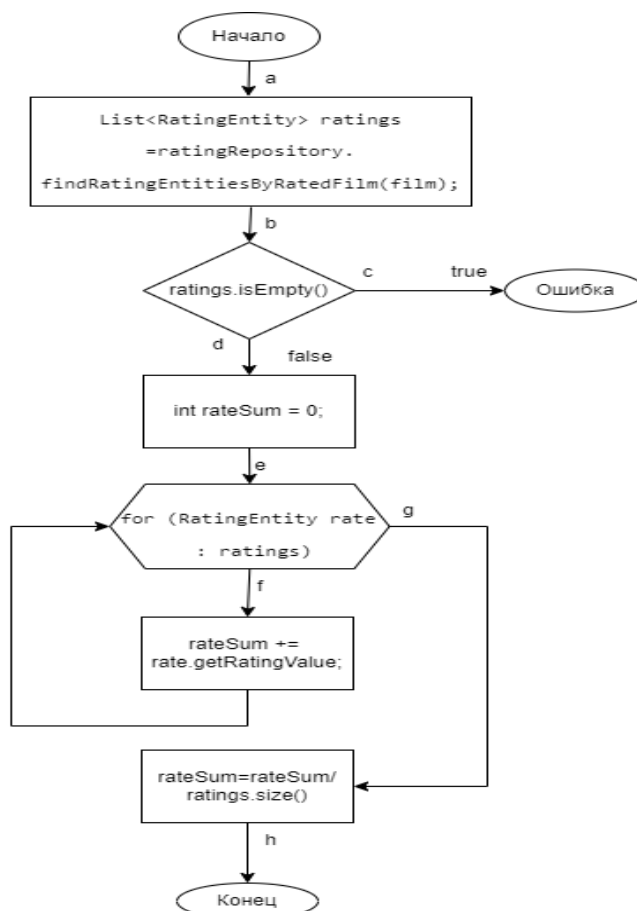


Рисунок 28 – Блок-схема для алгоритма подсчёта оценки фильму

В структурном подходе используются три метода покрытия: покрытие операторов, покрытие операторов и покрытие маршрутов. Метод покрытия операторов подразумевает выполнение каждого оператора хотя бы один раз.

Для покрытия операторов необходимо два теста, представленных в таблице 11.

Таблица 11 – Тесты метода покрытия операторов

Маршрут	Содержание теста	Фактический результат	Тест пройден/не пройден
abc	Запрашивается список оценок к фильму, но система не содержит ни одной оценки к этому фильму	Возвращается 0 – фильм не оценен	Тест пройден
abdefgh	Запрашивается список оценок, система содержит оценки к этому фильму	Посчитана и возвращена средняя оценка фильма	Тест пройден

Все тесты прошли успешно, при этом каждый оператор выполнен, по крайней мере, один раз.

Для метода покрытия условий необходимо, чтобы все возможные результаты каждого условия выполнялись, по крайней мере, один раз.

Для этого будут созданы два теста, представленные в таблице 12.

Таблица 12 – Тесты метода покрытия условий

Маршрут	Входные данные	Фактический результат	Тест пройден/не пройден
abc	ratings.isEmpty==true	Возвращается 0 – фильм не оценен	Тест пройден
abdefgh	ratings.isEmpty==false	Возвращается оценка фильму	Тест пройден

Все тесты прошли успешно, каждое условие выполнилось единожды.

Для покрытия всех маршрутов было составлено два теста, представленных в таблице 13.

Таблица 13 – Тесты метода покрытия маршрутов

Маршрут	Содержание теста	Фактический результат	Тест пройден/не пройден
abc	Запрашивается список оценок к фильму, но система не содержит ни одной оценки к этому фильму	Возвращается 0 – фильм не оценен	Тест пройден
abdefgh	Запрашивается список оценок, система содержит оценки к этому фильму, идет подсчет средней оценки до тех пор, пока не закончится список полученных оценок	Посчитана и возвращена средняя оценка фильма	Тест пройден

Все тесты прошли успешно, были пройдены все маршруты.

4.3 Оценка качества программного кода на основе метрик

Произведем расчет SLOC-метрики для класса ReviewService (приложение E). Результаты расчета представлены в таблице 14.

Таблица 14 – Расчет SLOC-метрики

Физические строки	60
Логические строки	40
Пустые строки	10
Строки комментариев	6

Подсчитаем метрику Холстеда для класса-сервиса RatingService (приложение E). Результат представлен в таблице 15.

Таблица 15 – Операторы и операнды для метрики Холстеда

Операторы	Число вхождений	Операнды	Число вхождений
public	5	rateSum	3
private	2	ratingMapping	1
void	1	ratingRepository	3
int	8	reviewDto	2
size	4	reviewDtos	2
+=	4	ratingDto	7
=	6	newRateDto	6
/	4	ratings	8
<	1	dtos	3
>	1	0	8
	1	10	1
:	4		
()	39		
if	3		
for	4		
return	6		
getFilmRate	1		
getRatingValue	5		
getRatedUser	1		
getRatedFilm	1		
getRatedreview	1		
n1 = 21	N1 = 102	n2 = 11	N2 = 44

На основании представленных характеристик можно рассчитать следующие оценки.

Длина программы вычисляется при помощи формулы:

$$N = N_1 + N_2 = 102 + 44 = 146,$$

где N – длина программы (общее количество операторов и операндов);

N_1 – общее количество операторов;

N_2 – общее количество операндов.

Для вычисления словаря программы применяется формула:

$$n = n_1 + n_2 = 21 + 11 = 32,$$

где n – словарь программы;

n_1 – количество уникальных операторов;

n_2 – количество уникальных операндов.

Метрика Холстеда вычисляется по формуле:

$$V = N * \log_2 n = 146 * \log_2 32 = 730,$$

где V – вычислительная сложность программы по метрике Холстеда;

N – длина программы;

n – словарь программы.

Объем программы по Холстеду определяет объем запрограммированного кода. Размер метрики для одной функции должен находиться в пределах от 20 до 1000.

Объем программы по Холстеду находится в пределе нормы.

Заключение

В результате выполнения выпускной квалификационной работы было реализовано серверное приложение для Web-сервиса агрегации зрительской и экспертной кинокритики.

Была изучена предметная область кинокритики. Проведен анализ аналогов на рынке, выделены необходимые функции системы и возможные её преимущества. Определены требования к разрабатываемой системе. Было выполнено планирование и расчет бюджета проекта. Спроектированы база данных и архитектура приложения, определены требования к системе. На основе этого была разработана база данных и серверное приложение, включающее сущностные объекты базы данных, средства взаимодействия с базой данных, сервисы обработки бизнес-процессов и контроллеры для обработки поступающих запросов от клиента. Проведено тестирование программы, показавшее корректность его работы.

Список использованных источников

1. Загвоздкина Е.Г. Кинокритика. // Большая Российская Энциклопедия [Электронный ресурс]. URL: <https://bigenc.ru/c/kinokritika-f1292c> (дата обращения: 10.05.2024)
2. Крицкая М. Диаграмма Ганта: как с её помощью управляют проектами. // Журнал «Контур» [Электронный ресурс]. URL: <https://kontur.ru/articles/6559> (дата обращения: 16.05.2024)
3. Использование диаграммы классов UML при проектировании и документировании программного обеспечения // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/572234/> (дата обращения: 10.06.2024)
4. Кушнир П. Как фреймворк Hibernate помогает Java-разработчику облегчить работу с базами данных// Яндекс практикум [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/hibernate-java/> (дата обращения: 10.06.2024)
5. Клиент-серверная архитектура // Онлайн-школа Hexlet [Электронный ресурс]. URL: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit (дата обращения: 15.05.2024)
6. Why to choose Java for Backend development. // [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/why-to-choose-java-for-backend-development/> (дата обращения: 15.05.2024)
7. PostgreSQL: почему идёт миграция в эту СУБД. // [Электронный ресурс]. URL: <https://procloud.ru/blog/technologies/postgresql-pochemu-idet-massovaya-migratsiya-v-etu-subd/> (дата обращения: 15.05.2024)
8. В чем разница между монолитной архитектурой и архитектурой микросервисов? // Amazon [Электронный ресурс]. URL: <https://aws.amazon.com/ru/compare/the-difference-between-monolithic-and-microservices-architecture/> (дата обращения: 10.06.2024)
9. Ричардс, М. Фундаментальный подход к программной архитектуре:

паттерны, свойства, проверенные методы / Ричардс М, Форд Н. – СПб.: Питер, 2023. – 448 с.: ил. – (Серия «Для профессионалов»)

10. Что такое моделирование данных? // Microsoft Power Bi [Электронный ресурс]. URL: <https://powerbi.microsoft.com/ru-ru/what-is-data-modeling/> (дата обращения: 10.06.2024)

11. JPA – Entity introduction // GeekForGeeks [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/jpa-entity-introduction/> (дата обращения: 10.06.2024)

12. Spring Repository Annotation with examples // GeekForGeeks [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/spring-repository-annotation-with-example/> (дата обращения: 10.06.2024)

13. Spring – Rest Controller // GeekForGeeks [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/spring-rest-controller/> (дата обращения: 10.06.2024)

14. Слой сервисов // Java. Корпоративные приложения на Spring Boot [Электронный ресурс]. URL: https://ru.hexlet.io/courses/java-spring/lessons/service-layer/theory_unit (дата обращения: 10.06.2024)

15. Какие бывают этапы и виды тестирования: подробный разбор // Онлайн-школа Hexlet [Электронный ресурс]. URL: <https://ru.hexlet.io/blog/posts/vidy-testirovaniya> (дата обращения: 10.06.2024)

16. Тестирование API: виды, методы, инструменты // Онлайн-школа Hexlet [Электронный ресурс]. URL: <https://ru.hexlet.io/blog/posts/testirovanie-api-vidy-metody-instrumenty> (дата обращения: 10.06.2024)

17. Беляков В. Как тестировщики проверяют работу API: принципы, инструменты и реальные примеры // Яндекс практикум [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/kak-testirovat-api/> (дата обращения: 10.06.2024)

18. Лекции по дисциплине «Тестирование информационных систем» // РБПОУ РД «Колледж сферы услуг» [Электронный ресурс]. URL:

https://пкcy.пф/wp-content/uploads/Lektsii-po-distsipline-_Testirovanie-informatsionnyh-sistem_-1.pdf (дата обращения: 10.06.2024)

19. Значение метрик кода // Microsoft learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата обращения: 10.06.2024)

20. Казанцева А. Программный код и его метрики // Habr [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 10.06.2024)

21. Halstead's Software Metrics – Software Engineering // GeekForGeeks [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/> (дата обращения: 10.06.2024)

Приложение А
(справочное)
Глоссарий

Серверное приложение – это приложение, ожидающее запрос от приложения-клиента и отвечает на него, выполняя определенный бизнес-процесс.

Сущность (Entity) – тип класса, позволяющий держать таблицу в качестве объекта и взаимодействовать с ней как с программным объектом.

Репозиторий (Repository, JpaRepository) – тип класса, позволяющий формировать SQL запросы в базу данных с помощью программного кода и получать, изменять или удалять необходимые сущности.

Контроллер (Controller, RestController) – тип класса, отвечающий за обработку поступающих HTTP запросов и выдачу ответов.

HTTP запрос – это сообщение, сформированное определенным образом и отправляемое клиентом, чтобы получить какую-либо услугу со стороны сервера.

Сервис (Service) – тип класса, отвечающий за выполнение бизнес-процессов в системе.

JDK (Java Development Kit) – комплект разработчика на языке java, который включает в себя компилятор java, JVM и базовые библиотеки классов.

JVM (Java Virtual Machine) – виртуальная машина, обеспечивающая выполнение java приложений. Основная задача – конвертация байт-кода java в машинный код для дальнейшего исполнения кода процессором.

API (Application Programming Interface) – это набор способов и правил для взаимодействия между двумя программами

Приложение Б
(рекомендуемое)
Спецификации атрибутов таблиц

Таблица Б.1 – Спецификация атрибутов таблицы User

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_user	Идентификатор	integer	15	PK	Да	0 9	
2	user_nickname	Никнейм пользователя	varchar	50				
3	user_email	Почта пользователя	varchar	2		Да		
4	user_fio	Фино пользователя	varchar	50		Да	А-я	
5	user_gender	Пол пользователя	char	7				
6	user_socials	Социальные сети пользователя	varchar	255				
7	user_rating	Рейтинг пользователя	integer	3				
8	id_role	Идентификатор роли	integer	1	FK			

Таблица Б.2 – Спецификация атрибутов таблицы Role

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_role	Идентификатор	integer	1	PK	Да	0 9	
2	role_name	Название роли	varchar	14				

Таблица Б.3 – Спецификация атрибутов таблицы Document_type

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_document_type	Идентификатор	integer	1	PK	Да	0 9	
2	document_name	Название типа документа	varchar	100				

Таблица Б.4 – Спецификация атрибутов таблицы Rating

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_rate	идентификатор рейтинга	integer	20	PK	Да	0 9	
2	rate_type	тип рейтинга	varchar	20				
3	rate_value	значение	integer	3				
4	id_film	идентификатор фильма	integer	10				
5	id_review	идентификатор рецензии	integer	10				
6	id_user	идентификатор пользователя	integer	15				

Таблица Б.5 – Спецификация атрибутов таблицы Comment

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_comment	идентификатор	integer	10	PK	Да	0 9	
2	comment_text	текст комментария	varchar	200				
3	comment_time	время написания	timestamp					
4	id_film	идентификатор фильма	integer	10				
5	id_user	идентификатор пользователя	integer	15				
6	id_review	идентификатор рецензии	integer	10				

Таблица Б.6 – Спецификация атрибутов таблицы Film

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_film	Идентификатор	integer	10	PK	Да	0 9	
2	film_title	Название роли	varchar	100				
3	date	Год выпуска	integer	4				
4	director	Режиссер	varchar	50				
5	timing	Продолжительность	integer	3				
6	reviewer_rating	Рейтинг рецензентов	integer	1				
7	users_rate	Рейтинг пользователей	integer	1				
8	Annotation	Аннотация на фильм	varchar	2500				

Таблица Б.7 – Спецификация атрибутов таблицы Review

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_review	идентификатор	integer	10	PK	Да	0 9	
2	review_title	название рецензии	varchar	50				
3	author_organization	автор	varchar	50				
4	review_text	текст рецензии	varchar	250000				
5	film_rate	рейтинг фильма	integer	1				
6	review_rate	рейтинг рецензии	integer	1				
7	Addition materials	дополнительные материалы	varchar	255				
8	id_film	идентификатор фильма	integer	10				
9	id_user	идентификатор пользователя	integer	15				

Таблица Б.8 – Спецификация атрибутов таблицы Addition materials

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_type	идентификатор	integer	1	PK	Да	0 9	
2	type	тип дополнительного материала	varchar	14				

Таблица Б.9 – Спецификация атрибутов таблицы Film genre

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_genre	идентификатор	integer	2	PK	Да	0 9	
2	genre_name	название жанра	varchar	50				

Таблица Б.10 – Спецификация атрибутов таблицы Film genres

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id	идентификатор	integer	10	PK	Да	0 9	
2	id_genre	идентификатор жанра	integer	2				
3	id_film	идентификатор фильма	integer	10				

Таблица Б.11 – Спецификация атрибутов таблицы Type of reports

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_type	Идентификатор типа	integer	10	PK	Да	0 9	
2	type	тип жалобы	varchar	20				
3	action	действие	varchar	50				

Таблица Б.12 – Спецификация атрибутов таблицы Report

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_report	идентификатор жалобы	integer	20	PK	Да	0 9	
2	report_text	текст жалобы	varchar	200				
3	report_time	Время написания	timestamp					
4	id_type	идентификатор типа жалобы	integer	20				
5	id_comment	идентификатор комментария	integer	20				
6	id_review	идентификатор рецензии	integer	20				

Таблица Б.13 – Спецификация атрибутов таблицы Addition material type

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_type	идентификатор дополнительного материала	integer	1	РК	Да	0 9	
2	type	тип дополнительного материала	varchar	14				

Таблица Б.14 – Спецификация атрибутов таблицы Verification list

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	id_verification	идентификатор	integer	1	РК	Да	0 9	
2	verification_text	текст	varchar	1000				
3	id_user	идентификатор пользователя	integer	1				
4	id_document_type	Идентификатор типа документа	integer	1				

Приложение В
(рекомендуемое)
Сущностные объекты таблиц

Листинг В.1 – Сущность AdditionEntity.java

```
package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
@Table(name = "addition")
public class AdditionEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "material_id")
    private Long additionMaterialId;
    @Column(name = "material_url", nullable = false)
    private String materialUrl;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "type_id", referencedColumnName = "type_id")
    private AdditionsTypeEntity additionType;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "review_id", referencedColumnName = "review_id")
    private ReviewEntity review;
}
```

Листинг В.2 – Сущность AdditionsTypeEntity.java

```
package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "additionstype")
public class AdditionsTypeEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "type_id")
    private int additionsTypeId;
    @Column(name = "type", unique = true, nullable = false)
    private String additionsTypeName;
    //Additions
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "additionType")
    private Set<AdditionEntity> additions = new HashSet<>();
}
```

Листинг В.3 – Сущность CommentEntity.java

```
package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.sql.Timestamp;
import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "comment")
public class CommentEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "comment_id")
    private Long commentId;
    @Column(name = "comment_text", nullable = false)
    private String commentText;
    @Column(name = "comment_time", nullable = false)
    private Timestamp commentTime;
    //Reports
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "reportedComment")
    Set<ReportEntity> reports = new HashSet<>();
    //Film
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "film_id", referencedColumnName = "film_id")
    private FilmEntity commentedFilm;
    //User
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", referencedColumnName = "user_id")
    private UserEntity user;
    //Review
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "review_id", referencedColumnName = "review_id")
    private ReviewEntity commentedReview;
}
```

Листинг В.4 – Сущность DocumentTypeEntity.java

```
package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
```

```

@Setter
@Table(name = "documenttype")
public class DocumentTypeEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "document_type_id")
    private int documentTypeId;
    @Column(name = "document_name", unique = true, nullable = false)
    private String documentName;
    //Verification
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "documentType")
    @Column(name = "verification_list")
    Set<VerificationListEntity> verificationList = new HashSet<>();
}

```

Листинг В.5 – Сущность FilmEntity.java

```

package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "film")
public class FilmEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "film_id")
    private Long filmId;
    @Column(name = "film_title", nullable = false)
    private String filmTitle;
    @Column(name = "release_date", nullable = false)
    private short releaseDate;
    @Column(name = "director", nullable = false)
    private String director;
    @Column(name = "film_timing", nullable = false)
    private short filmTiming;
    @Column(name = "annotation", nullable = false)
    private String annotation;
    @Column(name = "reviewer_rating")
    private short reviewerRating;
    @Column(name = "users_rating")
    private short usersRating;
    //Reviews
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "film")
    private Set<ReviewEntity> reviews = new HashSet<>();
    //Comments

```

```

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "commentedFilm")
    private Set<CommentEntity> comments = new HashSet<>();
    //Rating
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "ratedFilm")
    private Set<RatingEntity> rates = new HashSet<>();
    //Film genres
    @ManyToMany(fetch = FetchType.EAGER,
        cascade = {
            CascadeType.PERSIST,
            CascadeType.MERGE
        })
    @JoinTable(name = "filmgenre",
        joinColumns = {@JoinColumn(name = "film_id")},
        inverseJoinColumns = {@JoinColumn(name = "genre_id")})
    private Set<FilmGenreEntity> genres = new HashSet<>();

    public void AddGenre(FilmGenreEntity filmGenre) {
        this.genres.add(filmGenre);
        filmGenre.getFilms().add(this);
    }

    public void removeGenre(FilmGenreEntity filmGenre) {
        this.genres.remove(filmGenre);
        filmGenre.getFilms().remove(this);
    }
}

```

Листинг В.6 – Сущность FilmGenreEntity.java

```

package com.example.bdcourse.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "filmgenres")
public class FilmGenreEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "genre_id")
    private int genreId;
    @Column(name = "genre_name", nullable = false, unique = true)
    private String genreName;
    @ManyToMany(fetch = FetchType.LAZY,
        cascade = {
            CascadeType.PERSIST,

```

```

                                CascadeType.MERGE
                                },
                                mappedBy = "genres")
    @JsonIgnore
    private Set<FilmEntity> films = new HashSet<>();
}

```

Листинг В.7 – Сущность RatingEntity.java

```

package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
@Table(name = "rating")
public class RatingEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "rating_id")
    private Long ratingId;
    @Column(name = "rating_value", nullable = false)
    private short ratingValue;
    //User
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", referencedColumnName = "user_id")
    private UserEntity ratedUser;
    //Film
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "film_id", referencedColumnName = "film_id")
    private FilmEntity ratedFilm;
    //Review
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "review_id", referencedColumnName = "review_id")
    private ReviewEntity ratedReview;
}

```

Листинг В.8 – Сущность ReportEntity.java

```

package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.sql.Timestamp;

@Entity
@Getter
@Setter
@Table(name = "report")

```

```

public class ReportEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "report_id")
    private Integer reportId;
    @Column(name = "report_text", nullable = false)
    private String reportText;
    @Column(name = "report_time")
    private Timestamp reportTime;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "type_id", referencedColumnName = "type_id")
    private ReportTypesEntity reportType;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private UserEntity reportedUser;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "comment_id", referencedColumnName =
"comment_id")
    private CommentEntity reportedComment;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "review_id", referencedColumnName = "review_id")
    private ReviewEntity reportedReview;
}

```

Листинг В.9 – Сущность ReportTypesEntity.java

```

package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "typeofreports")
public class ReportTypesEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "type_id")
    private int typeId;
    @Column(name = "type", unique = true)
    private String reportType;
    @Column(name = "action")
    private String reportAction;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "reportType")
    private Set<ReportEntity> reports = new HashSet<>();
}

```

Листинг В.10 – Сущность ReviewEntity.java

```

package com.example.bdcourse.entity;

```

```

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "review")
public class ReviewEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "review_id")
    private Long reviewId;
    @Column(name = "review_title", unique = true, nullable = false)
    private String reviewTitle;
    @Column(name = "author_organization")
    private String authorOrganization;
    @Column(name = "review_text", nullable = false)
    private String reviewText;
    @Column(name = "film_rate", nullable = false)
    private short filmRate;
    @Column(name = "review_rate")
    private short reviewRate;
    //Film
    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "film_id", referencedColumnName = "film_id")
    private FilmEntity film;
    //User
    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", referencedColumnName = "user_id")
    private UserEntity user;
    //Additions
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "review")
    @Column(name = "review_additions")
    private Set<AdditionEntity> reviewAddition = new HashSet<>();
    //Reports
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "reportedReview")
    @Column(name = "review_reports")
    private Set<ReportEntity> reviewReports = new HashSet<>();
    //Comments
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "commentedReview")
    @Column(name = "review_comments")
    private Set<CommentEntity> reviewComments = new HashSet<>();
    //Rating
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "ratedReview")
    @Column(name = "review_rates")
    private Set<RatingEntity> reviewRates = new HashSet<>();

```

```
}
```

Листинг В.11 – Сущность RolesEntity.java

```
package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "roles")
public class RolesEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "roles_id")
    private int roleId;
    @Column(name = "role", unique = true, nullable = false)
    private String role;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "userRole")
    Set<UserEntity> users = new HashSet<>();
}
```

Листинг В.12 – Сущность UserEntity.java

```
package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Getter
@Setter
@Table(name = "users")
public class UserEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long userId;
    @Column(name = "user_nickname", unique = true, nullable = false)
    private String userNickname;
    @Column(name = "user_email", unique = true, nullable = false)
    private String userEmail;
    @Column(name = "user_fio")
    private String userFio;
    @Column(name = "user_gender")
```

```

    private String gender;
    @Column(name = "user_socials")
    private String userSocials;
    @Column(name = "user_rating")
    private short userRating = 50;
    @Column(name = "user_reviewer_rating")
    private short reviewerRating = 0;
    //Reviews
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "user")
    @Column(name = "user_reviews")
    private Set<ReviewEntity> userReviews = new HashSet<>();
    //Comments
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "user")
    @Column(name = "user_comments")
    private Set<CommentEntity> userComments = new HashSet<>();
    //Rates
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "ratedUser")
    @Column(name = "user_rates")
    private Set<RatingEntity> userRates = new HashSet<>();
    //Role
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "roles_id", referencedColumnName = "roles_id")
    private RolesEntity userRole;
    //Reports
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "reportedUser")
    @Column(name = "user_reports")
    private Set<ReportEntity> userReports = new HashSet<>();
}

```

Листинг В.13 – Сущность VerificationListEntity.java

```

package com.example.bdcourse.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
@Table(name = "verification")
public class VerificationListEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "verification_id")
    private int verificationId;
    @Column(name = "verification_text", nullable = false)
    private String verificationText;
    //User
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id", referencedColumnName = "user_id")

```

```
private UserEntity user;
//Document Type
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "document_type_id", referencedColumnName =
"document_type_id")
private DocumentTypeEntity documentType;}
```

Приложение Г
(рекомендуемое)
SQL-скрипты

Листинг Г.1 – Скрипт db_filling.sql

```

INSERT INTO additionstype (type)
VALUES
('text'),
('video'),
('audio'),
('socials');

INSERT INTO filmgenres (genre_name)
VALUES
('Экшен'),
('Фантастика'),
('Хоррор'),
('Боевик');

INSERT INTO roles (role)
VALUES
('user'),
('reviewer'),
('admin');

INSERT INTO documenttype (document_name)
VALUES
('Диплом об окончании высшего образования'),
('Выписка с работы'),
('youtube канал'),
('telegram канал');

INSERT INTO typeofreports (type, action)
VALUES
('Ненормативная лексика', 'предупреждение, понижение рейтинга'),
('Сексуальный контент', 'понижение рейтинга, бан'),
('Расизм', 'понижение рейтинга, бан');

INSERT INTO film (annotation, director, film_timing, film_title,
release_date, reviewer_rating, users_rating)
VALUES
('some annotation1', 'Vilnev', 76, 'Duna', 2021, 0, 0),
('some annotation2', 'Richi', 84, 'Snitch', 2001, 0, 0),
('some annotation3', 'director3', 71, 'Alise in wonderlands', 2011, 0,
0),
('some annotation4', 'director4', 80, 'Harry Potter ', 2008, 0, 0);

INSERT INTO filmgenre (film_id, genre_id)
VALUES
(1,4),
(1,2),
(1,1),
(2,1),
(2,4),
(3,3),
(4,2);

```

```

INSERT INTO users (user_nickname, user_gender, user_email, user_fio,
user_rating, user_reviewer_rating, user_socials, roles_id)
VALUES
('melti','female','someemail6@gmail.com', null, 50, 0, null, 1),
('jorgi','male','someemail1@gmail.com', null, 50, 0, null, 2),
('betti','female','someemail2@gmail.com', null, 50, 0, null, 1),
('john58', 'male', 'someemail3@gmail.com', null, 50, 0, null, 2),
('Alexa', 'female', 'someemail4@gmail.com', null, 50, 0, null, 1);

```

```

INSERT INTO verification (verification_text, document_type_id, user_id)
VALUES
('text1', 1, 1),
('text2', 3, 3);

```

```

INSERT INTO review (author_organization, film_rate, review_rate,
review_text, review_title, film_id, user_id)
VALUES
(null, 4, 0, 'teeext', 'Some good film', 1, 1),
('picabu', 5, 0, 'teeext2', 'Its awesome!', 2, 3),
(null, 1, 0, 'soooo baaad', 'Title', 3, 3);

```

```

INSERT INTO addition (material_url, type_id, review_id)
VALUES
('www.example.com', 1, 1),
('www.google.com', 2, 2),
('www.yandex.ru', 2, 3),
('www.vk.com', 3, 2);

```

```

INSERT INTO comment (comment_text, comment_time, film_id, review_id,
user_id)
VALUES
('nice film','2023-12-16 16:22:15', 1, null, 3),
('booooring film','2023-12-16 16:23:16', 1, null, 2),
('cool','2023-12-16 16:23:33', 1, 1, 1),
('cool x2','2023-12-16 16:24:44', 2, 2, 3);

```

```

INSERT INTO rating (rating_value, film_id, review_id, user_id)
VALUES
(5, 1, null, 1),
(4, 1, null, 2),
(4, 2, 1, 3),
(2, 3, 3, 4);

```

```

INSERT INTO report (report_text, report_time, type_id, comment_id,
review_id, user_id)
VALUES
('noncomfort speach', '2023-12-16 17:14:11', 1, null, 1, 1),
('noncomfort speach2', '2023-12-16 17:14:13', 1, 2, 1, 1),

('noncomfort speach3', '2023-12-16 17:17:14', 1, 2, null, 1);

```

Листинг Г.2 – Скрипт timetrigger.sql

```

CREATE OR REPLACE FUNCTION current_time_adjustment_func()
    Returns trigger AS
$$
BEGIN
    Update COMMENT SET comment_time = CURRENT_TIMESTAMP WHERE
comment_id = NEW.comment_id;
RETURN NULL;
END;
$$
LANGUAGE 'plpgsql';
CREATE OR REPLACE TRIGGER current_time_adjustment
AFTER INSERT
ON comment
FOR EACH ROW
EXECUTE PROCEDURE current_time_adjustment_func();

CREATE OR REPLACE FUNCTION current_report_time_func()
    Returns trigger AS
$$
BEGIN
    Update report SET report_time = CURRENT_TIMESTAMP WHERE report_id =
NEW.report_id;
RETURN NULL;
END;
$$
LANGUAGE 'plpgsql';
CREATE OR REPLACE TRIGGER current_report_time
AFTER INSERT
ON report
FOR EACH ROW
EXECUTE PROCEDURE current_report_time_func();

-----Review-----
CREATE FUNCTION update_review_rate_func()
RETURNS TRIGGER AS $$
DECLARE
total_rating_value INT;
total_ratings_count INT;
BEGIN
    -- Вычисляем сумму значений полей rating_value для заданного
review_id
    SELECT SUM(rating_value), COUNT(*) INTO total_rating_value,
total_ratings_count
    FROM rating
    WHERE review_id = NEW.review_id;

    -- Если есть рейтинги для заданного review_id, то обновляем
значение поля review_rate в таблице review
    IF total_ratings_count > 0 THEN
        UPDATE review SET review_rate = total_rating_value /
total_ratings_count WHERE review_id = NEW.review_id;
    END IF;

```

```
        RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER update_review_rate_trigger  
AFTER INSERT  
ON rating  
FOR EACH ROW  
EXECUTE FUNCTION update_review_rate_func();
```

Приложение Д
(рекомендуемое)
Классы типа Repository

Листинг Д.1 – Репозиторий CommentRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.CommentEntity;
import com.example.bdcourse.entity.FilmEntity;
import com.example.bdcourse.entity.ReviewEntity;
import com.example.bdcourse.entity.UserEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface CommentRepository extends JpaRepository<CommentEntity, Long> {
    CommentEntity findById(Long id);

    List<CommentEntity> findCommentEntitiesByUser(UserEntity user);

    List<CommentEntity> findCommentEntitiesByCommentedFilm(FilmEntity film);

    List<CommentEntity> findCommentEntitiesByCommentedReview(ReviewEntity review);
}
```

Листинг Д.2 – Репозиторий FilmGenreRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.FilmGenreEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface FilmGenreRepository extends JpaRepository<FilmGenreEntity, Short> {
    List<FilmGenreEntity> findFilmGenreEntitiesByFilms_filmId(Long filmId);

    FilmGenreEntity findFilmGenreEntityByGenreName(String genre);
}
```

Листинг Д.3 – Репозиторий FilmRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.FilmEntity;
import com.example.bdcourse.entity.FilmGenreEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
import java.util.List;

@Repository
public interface FilmRepository extends JpaRepository<FilmEntity, Long> {
    FilmEntity findByFilmId(Long Id);

    List<FilmEntity> findFilmEntitiesByFilmTitle(String filmTitle);

    List<FilmEntity> findFilmEntitiesByGenres_GenreId(int genreid);
}
```

Листинг Д.4 – Репозиторий RatingRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.FilmEntity;
import com.example.bdcourse.entity.RatingEntity;
import com.example.bdcourse.entity.ReviewEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface RatingRepository extends JpaRepository<RatingEntity, Long> {
    List<RatingEntity> findRatingEntitiesByRatedFilm(FilmEntity film);

    List<RatingEntity> findRatingEntitiesByRatedReview(ReviewEntity review);
}
```

Листинг Д.5 – Репозиторий ReviewRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.ReviewEntity;
import com.example.bdcourse.entity.UserEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ReviewRepository extends JpaRepository<ReviewEntity, Long> {
    ReviewEntity findByReviewId(Long id);
    List<ReviewEntity> findReviewEntitiesByUser(UserEntity user);

    List<ReviewEntity> findReviewEntitiesByUserUserId(Long userId);

    List<ReviewEntity> findReviewEntitiesByFilmFilmId(Long filmId);
}
```

Листинг Д.6 – Репозиторий UserRepository.java

```
package com.example.bdcourse.repository;

import com.example.bdcourse.entity.UserEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends JpaRepository<UserEntity, Long> {
    UserEntity findById(Long userId);

    UserEntity findByUserNickname(String nickname);

    UserEntity findByUserEmail(String email);

    List<UserEntity> findAllOrderByUserRating();

    List<UserEntity> findAllOrderByReviewerRating();
}
```

Приложение Е
(рекомендуемое)
Классы типа Controller

Листинг Е.1 – Контроллер CommentController.java

```
package com.example.bdcource.controller;

import com.example.bdcource.dto.CommentDto;
import com.example.bdcource.dto.FilmDto;
import com.example.bdcource.dto.ReviewDto;
import com.example.bdcource.dto.UserDto;
import com.example.bdcource.mapping.CommentMapping;
import com.example.bdcource.mapping.FilmMapping;
import com.example.bdcource.mapping.ReviewMapping;
import com.example.bdcource.mapping.UserMapping;
import com.example.bdcource.service.CommentService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/bdcouce/comment")
public class CommentController {
    @Autowired
    private CommentService commentService;
    @Autowired
    private CommentMapping commentMapping;
    @Autowired
    private ReviewMapping reviewMapping;
    @Autowired
    private UserMapping userMapping;
    @Autowired
    private FilmMapping filmMapping;

    @PostMapping("/addcomment")
    public ResponseEntity<String> addComment(@RequestBody CommentDto
commentDto) {
        try {
            commentService.addComment(commentDto);
            return new ResponseEntity<>(HttpStatus.CREATED);
        } catch (IllegalArgumentException iea) {
            return new ResponseEntity<>(iea.getMessage(),
HttpStatus.BAD_REQUEST);
        }
    }

    @GetMapping("/takecomment")
    public CommentDto takeCommentById(@RequestParam("id") Long
commentId) {
        return commentService.takeCommentById(commentId);
    }
}
```

```

        @GetMapping("/takefilmcomments")
        public List<CommentDto> takeFilmComments(@RequestParam("id") Long
filmId) {
            String uri =
"http://localhost:8080/api/bdcource/film/takefilm?id=" + filmId;
            RestTemplate restTemplate = new RestTemplate();
            FilmDto filmDto = restTemplate.getForObject(uri,
FilmDto.class);
            return filmDto != null
                ?
commentService.takeFilmComments(filmMapping.mapToFilmEntity(filmDto))
                : null;
        }

        @GetMapping("/takereviewcomments")
        public List<CommentDto> takeReviewComments(@RequestParam("id") Long
reviewId) {
            String uri =
"http://localhost:8080/api/bdcource/review/takereview?id=" + reviewId;
            RestTemplate restTemplate = new RestTemplate();
            ReviewDto reviewDto = restTemplate.getForObject(uri,
ReviewDto.class);
            return reviewDto != null
                ?
commentService.takeReviewComments(reviewMapping.mapToReviewEntity(reviewD
to))
                : null;
        }

        @GetMapping("/takeusercomments")
        public List<CommentDto> takeUserComments(@RequestParam("id") Long
userId) {
            String uri =
"http://localhost:8080/api/bdcource/user/takeuserbyid?id=" + userId;
            RestTemplate restTemplate = new RestTemplate();
            UserDto userDto = restTemplate.getForObject(uri,
UserDto.class);
            return userDto != null
                ?
commentService.takeUserComments(userMapping.mapToUserEntity(userDto))
                : null;
        }

        @DeleteMapping("/removecomment")
        public ResponseEntity<Long> removeComment(@RequestParam("id") Long
commentId) {
            commentService.removeComment(commentId);
            return new ResponseEntity<>(HttpStatus.OK);
        }
    }
}

```

Листинг Е.2 – Контроллер FilmController.java

```
package com.example.bdcource.controller;
```

```

import com.example.bdcource.dto.FilmDto;
import com.example.bdcource.dto.FilmGenreDto;
import com.example.bdcource.mapping.FilmMapping;
import com.example.bdcource.service.FilmService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/bdcource/film")
public class FilmController {
    @Autowired
    private FilmService filmService;
    @Autowired
    private FilmMapping filmMapping;

    @PostMapping("/addfilm")
    public ResponseEntity<String> addFilm(@RequestBody FilmDto filmDto)
    {
        try {
            filmService.addFilm(filmDto);
            return new ResponseEntity<>(HttpStatus.CREATED);
        } catch (IllegalArgumentException iae) {
            return new ResponseEntity<>(iae.getMessage(),
HttpStatus.BAD_REQUEST);
        }
    }

    @GetMapping("/takefilm")
    public FilmDto takeFilmById(@RequestParam("id") Long filmId) {
        return filmService.takeFilmById(filmId);
    }

    @GetMapping("/takefilms")
    public List<FilmDto> takeAllFilmsByPages(@RequestParam(required =
false, defaultValue = "0") int page,
@RequestParam(required = false, defaultValue = "10") int size) {
        return filmService.takeFilms(PageRequest.of(page,
size)).getContent()
            .stream().map(filmMapping::mapToFilmDto)
            .collect(Collectors.toList());
    }

    @GetMapping("/takefilmgenres")
    public List<FilmGenreDto> takeFilmGenres(@RequestParam("id") Long
filmId) {

```

```

        return filmService.takeFilmGenres(filmId);
    }

    @GetMapping("/takefilmsbygenre")
    public List<FilmDto> takeFilmsByGenre(@RequestParam("id") int
filmGenreId) {
        return filmService.takeFilmsByGenre(filmGenreId);
    }

    @GetMapping("/takefilmsbygenres")
    public List<FilmDto> takeFilmsByMultipleGenres(@RequestBody
List<Integer> filmGenres) {
        return filmService.takeFilmsByMultipleGenres(filmGenres);
    }

    @GetMapping("/takefilmsbytitle")
    public List<FilmDto> takeFilmsByTitle(@RequestParam("title") String
title) {
        return filmService.takeFilmsByTitle(title);
    }

    @DeleteMapping("/removefilm")
    public ResponseEntity<Long> removeFilm(@RequestParam("id") Long
filmId) {
        filmService.removeFilmById(filmId);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}

```

Листинг Е.3 – Контроллер RatingController.java

```

package com.example.bdcourse.controller;

import com.example.bdcourse.dto.FilmDto;
import com.example.bdcourse.dto.RatingDto;
import com.example.bdcourse.dto.ReviewDto;
import com.example.bdcourse.dto.UserDto;
import com.example.bdcourse.mapping.FilmMapping;
import com.example.bdcourse.mapping.RatingMapping;
import com.example.bdcourse.mapping.ReviewMapping;
import com.example.bdcourse.service.RatingService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/bdcourse/rating")
public class RatingController {
    @Autowired

```

```

private RatingService ratingService;
@Autowired
private FilmMapping filmMapping;
@Autowired
private ReviewMapping reviewMapping;
@Autowired
private RatingMapping ratingMapping;

@PostMapping("/addrating")
public ResponseEntity<String> addRating(@RequestBody RatingDto
ratingDto) {
    try {
        ratingService.addRate(ratingDto);
        return new ResponseEntity<>(HttpStatus.CREATED);
    } catch (IllegalArgumentException iae) {
        return new ResponseEntity<>(iae.getMessage(),
HttpStatus.BAD_REQUEST);
    }
}

@GetMapping("/filmrate")
public int takeFilmRate(@RequestParam("id") Long filmId) {
    String uri =
"http://localhost:8080/api/bdcourse/film/takefilm?id=" + filmId;
    RestTemplate restTemplate = new RestTemplate();
    FilmDto filmDto = restTemplate.getForObject(uri,
FilmDto.class);
    return filmDto != null
        ?
ratingService.calculateFilmRate(filmMapping.mapToFilmEntity(filmDto))
        : 0;
}

@GetMapping("/filmreviewerrate")
public int takeReviewerFilmRate(@RequestParam("id") Long filmId) {
    String uri =
"http://localhost:8080/api/bdcourse/review/takereviewbyfilm?id=" +
filmId;
    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<ReviewDto[]> response =
restTemplate.getForEntity(uri, ReviewDto[].class);
    List<ReviewDto> reviewDtos = List.of(response.getBody());
    return ratingService.calculateReviewersFilmRate(reviewDtos);
}

@GetMapping("/reviewrate")
public int takeReviewRate(@RequestParam("id") Long reviewId) {
    String uri =
"http://localhost:8080/api/bdcourse/review/takereview?id=" + reviewId;
    RestTemplate restTemplate = new RestTemplate();
    ReviewDto reviewDto = restTemplate.getForObject(uri,
ReviewDto.class);
    return reviewDto != null
        ?
ratingService.calculateReviewRate(reviewMapping.mapToReviewEntity(reviewD

```

```

to))
        : 0;
    }

    @GetMapping("/takereviewerrate")
    public short takeReviewerRate(@RequestParam("id") Long userId) {
        String uri =
"http://localhost:8080/api/bdcourse/user/takeuserbyid?id=" + userId;
        RestTemplate restTemplate = new RestTemplate();
        UserDto userDto = restTemplate.getForObject(uri,
UserDto.class);
        if (userDto == null)
            return 0;
        uri =
"http://localhost:8080/api/bdcourse/review/takereviewsforuser?userid=" +
userId;
        ResponseEntity<ReviewDto[]> response =
restTemplate.getForEntity(uri, ReviewDto[].class);
        if (response.getBody() == null)
            return 0;
        List<ReviewDto> reviews = List.of(response.getBody());
        return (short) ratingService.calculateReviewerRate(reviews);
    }
}

```

Листинг Е.4 – Контроллер ReviewController.java

```

package com.example.bdcourse.controller;

import com.example.bdcourse.dto.ReviewDto;
import com.example.bdcourse.mapping.ReviewMapping;
import com.example.bdcourse.mapping.UserMapping;
import com.example.bdcourse.service.ReviewService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/bdcourse/review")
public class ReviewController {
    @Autowired
    private ReviewService reviewService;
    @Autowired
    private ReviewMapping reviewMapping;
    @Autowired
    private UserMapping userMapping;
}

```

```

    @PostMapping("/addreview")
    public ResponseEntity<String> addReview(@RequestBody ReviewDto
reviewDto) {
        RestTemplate restTemplate = new RestTemplate();
        if (reviewDto.getUser() == null){
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        String uri =
"http://localhost:8080/api/bdcource/user/takeuserrole?id=" +
reviewDto.getUser();
        String response = restTemplate.getForObject(uri,
String.class);
        if (response == null)
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        if (response.equals("user"))
            return new ResponseEntity<>(HttpStatus.NOT_ACCEPTABLE);
        try {
            reviewService.addReview(reviewDto);
            return new ResponseEntity<>(HttpStatus.CREATED);
        } catch (IllegalArgumentException iae) {
            System.out.println(iae.getMessage());
            return new ResponseEntity<>(iae.getMessage(),
HttpStatus.BAD_REQUEST);
        }

    }

    @GetMapping("/takereviews")
    public List<ReviewDto> takeAllReviewsByPages(@RequestParam(required
= false, defaultValue = "0") int page,

@RequestParam(required = false, defaultValue = "10") int size) {
        return reviewService.takeReviews(PageRequest.of(page,
size)).getContent()
            .stream().map(reviewMapping::mapToReviewDto)
            .collect(Collectors.toList());
    }

    @GetMapping("/takereview")
    public ReviewDto takeReview(@RequestParam("id") Long reviewId) {
        return reviewService.takeReviewById(reviewId);
    }

    @GetMapping("/takereviewbyfilm")
    public List<ReviewDto> takeFilmsReviews(@RequestParam("id") Long
filmId) {
        return reviewService.takeFilmsReviews(filmId);
    }

    @GetMapping("/takereviewsforuser")
    public List<ReviewDto> takeReviewsForUser(@RequestParam("userid")
Long user) {
        return reviewService.takeAllReviewsForUser(user)
            .stream().map(reviewMapping::mapToReviewDto)

```

```

        .collect(Collectors.toList());
    }

    @DeleteMapping("/removereview")
    public ResponseEntity<Long> removeReview(@RequestParam("id") Long
reviewId) {
        reviewService.removeReviewById(reviewId);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}

```

Листинг Е.5 – Контроллер UserController.java

```

package com.example.bdcourse.controller;

import com.example.bdcourse.dto.UserDto;
import com.example.bdcourse.mapping.UserMapping;
import com.example.bdcourse.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/bdcourse/user")
public class UserController {
    @Autowired
    private UserService userService;
    @Autowired
    private UserMapping userMapping;

    @PostMapping("/adduser")
    public ResponseEntity<String> addUser(@RequestBody UserDto userDto)
{
        try {
            userService.addUser(userDto);
            return new ResponseEntity<>(HttpStatus.CREATED);
        } catch (IllegalArgumentException iae) {
            return new ResponseEntity<>(iae.getMessage(),
HttpStatus.BAD_REQUEST);
        }
    }

    @GetMapping("/takeuserbyid")
    public UserDto takeUserById(@RequestParam("id") Long userId) {
        return
userMapping.mapToUserDto(userService.getUserById(userId));
    }
}

```

```

        @GetMapping("/takeuserbynickname")
        public UserDto takeUserByNickname(@RequestParam("nickname") String
nickname) {
            return
userMapping.mapToUserDto(userService.getUserByNickname(nickname));
        }

        @GetMapping("/takeuserbyemail")
        public UserDto takeUserByEmail(@RequestParam("email") String email)
{
            return
userMapping.mapToUserDto(userService.getUserByEmail(email));
        }

        @GetMapping("/takeworstusers")
        public List<UserDto> takeWorstUsers() {
            return userService.takeTenWorstUsersByRate();
        }

        @GetMapping("/takeworstreviewers")
        public List<UserDto> takeWorstReviewers() {
            return userService.takeTenWorstReviewersByRate();
        }

        @GetMapping("/takeuserrole")
        public String takeUserRole(@RequestParam("id") Long userId) {
            return userService.takeUserRole(userId);
        }

        @PutMapping("/downgradeuser")
        public ResponseEntity<Long> downgradeUser(@RequestParam("id") Long
id, @RequestParam("rate") short rate) {
            userService.downgradeUser(id, rate);
            return new ResponseEntity<>(id, HttpStatus.OK);
        }

        @PutMapping("/changeuserrole")
        public ResponseEntity<Long> changeUserRole(@RequestParam("id") Long
userId, @RequestParam("role") String role) {
            userService.changeUserRole(userId, role);
            return new ResponseEntity<>(userId, HttpStatus.ACCEPTED);
        }

        @DeleteMapping("/removeuser")
        public ResponseEntity<Long> removeUser(@RequestParam("id") Long id)
{
            userService.deleteUser(id);
            return new ResponseEntity<>(HttpStatus.OK);
        }
    }

```

Приложение Ж
(рекомендуемое)
Классы типа Service

Листинг Ж.1 – Сервис CommentService.java

```
package com.example.bdcourse.service;

import com.example.bdcourse.dto.CommentDto;
import com.example.bdcourse.entity.FilmEntity;
import com.example.bdcourse.entity.ReviewEntity;
import com.example.bdcourse.entity.UserEntity;
import com.example.bdcourse.mapping.CommentMapping;
import com.example.bdcourse.repository.CommentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class CommentService {
    @Autowired
    private CommentRepository commentRepository;
    @Autowired
    private CommentMapping commentMapping;

    public void addComment(CommentDto commentDto) {
        if (commentDto.getCommentText() == null) {
            throw new IllegalArgumentException("Comment can't be
empty");
        }

        commentRepository.save(commentMapping.mapToCommentEntity(commentDto
));
    }

    public CommentDto takeCommentById(Long commentId) {
        return
commentMapping.mapToCommentDto(commentRepository.findById(commentId));
    }

    public List<CommentDto> takeUserComments(UserEntity user) {
        return commentRepository.findCommentEntitiesByUser(user)
            .stream().map(commentMapping::mapToCommentDto)
            .collect(Collectors.toList());
    }

    public List<CommentDto> takeFilmComments(FilmEntity film) {
        return
commentRepository.findCommentEntitiesByCommentedFilm(film)
            .stream().map(commentMapping::mapToCommentDto)
            .collect(Collectors.toList());
    }
}
```

```

        public List<CommentDto> takeReviewComments(ReviewEntity review) {
            return
commentRepository.findCommentEntitiesByCommentedReview(review)
                    .stream().map(commentMapping::mapToCommentDto)
                    .collect(Collectors.toList());
        }

        public void removeComment(Long commentId) {
            commentRepository.deleteById(commentId);
        }
    }
}

```

Листинг Ж.2 – Сервис FilmService.java

```

package com.example.bdcourse.service;

import com.example.bdcourse.dto.FilmDto;
import com.example.bdcourse.dto.FilmGenreDto;
import com.example.bdcourse.entity.FilmEntity;
import com.example.bdcourse.mapping.FilmGenreMapping;
import com.example.bdcourse.mapping.FilmMapping;
import com.example.bdcourse.repository.FilmGenreRepository;
import com.example.bdcourse.repository.FilmRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class FilmService {
    @Autowired
    private FilmRepository filmRepository;
    @Autowired
    private FilmMapping filmMapping;
    @Autowired
    private FilmGenreRepository filmGenreRepository;
    @Autowired
    private FilmGenreMapping filmGenreMapping;

    public void addFilm(FilmDto filmDto) {
        if (filmDto.getFilmTitle() == null)
            throw new IllegalArgumentException("Film title can't be
empty");
        if (filmDto.getDirector() == null)
            throw new IllegalArgumentException("Film director can't
be empty");

        filmRepository.save(filmMapping.mapToFilmEntity(filmDto));
    }
}

```

```

    public FilmDto takeFilmById(long filmId) {
        return
        filmMapping.mapToFilmDto(filmRepository.findByFilmId(filmId));
    }

    public Page<FilmEntity> takeFilms(PageRequest pageRequest) {
        return filmRepository.findAll(pageRequest);
    }

    public List<FilmDto> takeFilmsByTitle(String name) {
        return filmRepository.findFilmEntitiesByFilmTitle(name)
            .stream().map(filmMapping::mapToFilmDto)
            .collect(Collectors.toList());
    }

    public void removeFilmById(long filmId) {
        filmRepository.deleteById(filmId);
    }

    public List<FilmGenreDto> takeFilmGenres(long filmId) {
        return
        filmGenreRepository.findFilmGenreEntitiesByFilms_filmId(filmId)
            .stream().map(filmGenreMapping::mapToFilmGenreDto)
            .collect(Collectors.toList());
    }

    public List<FilmDto> takeFilmsByGenre(int filmGenreId) {
        return
        filmRepository.findFilmEntitiesByGenres_GenreId(filmGenreId)
            .stream().map(filmMapping::mapToFilmDto)
            .collect(Collectors.toList());
    }

    public List<FilmDto> takeFilmsByMultipleGenres(List<Integer>
        filmGenreIds) {
        List<FilmDto> films = new ArrayList<>();
        for (int genreId : filmGenreIds) {
            List<FilmDto> temporarylist =
            filmRepository.findFilmEntitiesByGenres_GenreId(genreId)
                .stream().map(filmMapping::mapToFilmDto)
                .toList();
            films.addAll(temporarylist);
        }
        return films;
    }
}

```

Листинг Ж.3 – Сервис RatingService.java

```

package com.example.bdcourse.service;

import com.example.bdcourse.dto.RatingDto;
import com.example.bdcourse.dto.ReviewDto;
import com.example.bdcourse.entity.FilmEntity;

```

```

import com.example.bdcourse.entity.RatingEntity;
import com.example.bdcourse.entity.ReviewEntity;
import com.example.bdcourse.mapping.RatingMapping;
import com.example.bdcourse.repository.RatingRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@Service
@RequiredArgsConstructor
public class RatingService {
    //used to map from ratingDto to ratingEntity and reverse
    @Autowired
    private RatingMapping ratingMapping;
    //used to interact with table Rating in database
    @Autowired
    private RatingRepository ratingRepository;

    //adding new rating to anything (user, film, review). Rated film
    and rated review can be empty
    public void addRate(RatingDto ratingDto) {
        if (ratingDto.getRatingValue() < 0 ||
ratingDto.getRatingValue() > 10) {
            throw new IllegalArgumentException("Wrong rating value -
" + ratingDto.getRatingValue());
        }
        RatingDto newRateDto = new RatingDto();
        newRateDto.setRatingValue(ratingDto.getRatingValue());
        newRateDto.setRatedUser(ratingDto.getRatedUser());
        newRateDto.setRatedFilm(ratingDto.getRatedFilm());
        newRateDto.setRatedReview(ratingDto.getRatedReview());

        ratingRepository.save(ratingMapping.mapToRatingEntity(newRateDto));
    }

    //calculate overall film rating by users ratings
    public int calculateFilmRate(FilmEntity film) {
        List<RatingEntity> ratings =
ratingRepository.findRatingEntitiesByRatedFilm(film);
        if (ratings.isEmpty())
            return 0;
        int rateSum = 0;
        for (RatingEntity rate : ratings) {
            rateSum += rate.getRatingValue();
        }
        return rateSum / ratings.size();
    }

    //calculate overall review rating by users ratings
    public int calculateReviewRate(ReviewEntity review) {
        try {
            List<RatingEntity> ratings =

```

```

ratingRepository.findRatingEntitiesByRatedReview(review);
    if (ratings.isEmpty())
        return 0;
    int rateSum = 0;
    for (RatingEntity rate : ratings) {
        rateSum += rate.getRatingValue();
    }
    return rateSum / ratings.size();
} catch (Throwable throwable) {
    return 0;
}
}

//Calculate overall film rating by reviews rating
public int calculateReviewersFilmRate(List<ReviewDto> dtos) {
    int rateSum = 0;
    for (ReviewDto reviewDto : dtos) {
        rateSum += reviewDto.getFilmRate();
    }
    return rateSum / dtos.size();
}

//Calculate reviewer rating
public int calculateReviewerRate(List<ReviewDto> reviewDtos) {
    RestTemplate restTemplate = new RestTemplate();
    String url;
    int rating = 0;
    for (ReviewDto review : reviewDtos) {
        url =
"http://localhost:8080/api/bdcourse/rating/reviewrate?id=" +
review.getReviewId();
        rating += restTemplate.getForObject(url, Integer.class);
    }
    return (rating / reviewDtos.size());
}
}

```

Листинг Ж.4 – Сервис ReviewService.java

```

package com.example.bdcourse.service;

import com.example.bdcourse.dto.ReviewDto;
import com.example.bdcourse.entity.ReviewEntity;
import com.example.bdcourse.mapping.ReviewMapping;
import com.example.bdcourse.repository.ReviewRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service

```

```

@RequiredArgsConstructor
public class ReviewService {
    @Autowired
    private ReviewRepository reviewRepository;
    @Autowired
    private ReviewMapping reviewMapping;

    public void addReview(ReviewDto reviewDto) {
        if (reviewDto.getReviewText() == null)
            throw new IllegalArgumentException("Your review text
can't be empty");
        if (reviewDto.getReviewTitle() == null)
            throw new IllegalArgumentException("Your review title
can't be empty");

        reviewRepository.save(reviewMapping.mapToReviewEntity(reviewDto));
    }

    public void removeReviewById(long reviewId) {
        reviewRepository.deleteById(reviewId);
    }

    public ReviewDto takeReviewById(long reviewId) {
        return
reviewMapping.mapToReviewDto(reviewRepository.findByReviewId(reviewId));
    }

    public List<ReviewDto> takeFilmsReviews(long filmId) {
        return reviewRepository.findReviewEntitiesByFilmFilmId(filmId)
            .stream().map(reviewMapping::mapToReviewDto)
            .collect(Collectors.toList());
    }

    public Page<ReviewEntity> takeReviews(PageRequest pageRequest) {
        return reviewRepository.findAll(pageRequest);
    }

    // public List<ReviewEntity> takeAllReviewsForUser(UserEntity
userEntity) {
    //     return reviewRepository.findReviewEntitiesByUser(userEntity);
    // }

    public List<ReviewEntity> takeAllReviewsForUser(Long userId) {
        return
reviewRepository.findReviewEntitiesByUserUserId(userId);
    }
}

```

Листинг Ж.5 – Сервис UserService.java

```

package com.example.bdcourse.service;

import com.example.bdcourse.dto.UserDto;
import com.example.bdcourse.entity.UserEntity;

```

```

import com.example.bdcourse.mapping.UserMapping;
import com.example.bdcourse.repository.RolesRepository;
import com.example.bdcourse.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
public class UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private UserMapping userMapping;
    @Autowired
    private RolesRepository rolesRepository;

    public void addUser(UserDto userDto) {
        if (userDto.getUserNickname() == null) {
            throw new IllegalArgumentException("Nickname is empty");
        }
        if (userDto.getUserEmail() == null) {
            throw new IllegalArgumentException("Email is empty");
        }
        if
(userRepository.findByUserNickname(userDto.getUserNickname()) != null) {
            throw new IllegalArgumentException("Nickname already
taken");
        }
        if (userDto.getUserRating() == 0) {
            userDto.setUserRating((short) 50);
        }
        if (userDto.getUserRole() == null)
            userDto.setUserRole("user");
        userRepository.save(userMapping.mapToUserEntity(userDto));
    }

    public UserEntity getUserById(long userId) {
        return userRepository.findById(userId);
    }

    public UserEntity getUserByNickname(String nick) {
        return userRepository.findByUserNickname(nick);
    }

    public UserEntity getUserByEmail(String email) {
        return userRepository.findByUserEmail(email);
    }

    public void deleteUser(long id) {
        userRepository.deleteById(id);
    }

```

```

    }

    public void downgradeUser(long userId, short rateValue) {
        UserEntity user = userRepository.findById(userId);
        short newRateValue = (short) (user.getUserRating() -
rateValue);
        if (newRateValue > 0)
            user.setUserRating(newRateValue);
        else
            user.setUserRating((short) 0);
        userRepository.save(user);
    }

    public void changeUserRole(long userId, String role) {
        UserEntity user = userRepository.findById(userId);
        user.setUserRole(rolesRepository.findByRole(role));
        userRepository.save(user);
    }

    public String takeUserRole(long userId) {
        UserEntity user = userRepository.findById(userId);
        return user.getUserRole().getRole();
    }

    public short calculateReviewerRating(List<Integer> reviewRatings) {
        short rateSum = 0;
        for (Integer rates : reviewRatings) {
            rateSum += rates;
        }
        return (short) (rateSum / reviewRatings.size());
    }

    @Scheduled(cron = "@yearly")
    public void upRate() {
        List<UserEntity> users = userRepository.findAll();
        for (UserEntity user : users) {
            user.setUserRating((short) (user.getUserRating() + 10));
        }
    }

    public List<UserDto> takeTenWorstUsersByRate() {
        List<UserEntity> users =
userRepository.findAllByOrderByUserRating();
        List<UserDto> worstUsers = new ArrayList<>();
        for (int i = 0; i < 10; ++i) {
            if (i == users.size())
                break;
            worstUsers.add(userMapping.mapToUserDto(users.get(i)));
        }
        return worstUsers;
    }

    public List<UserDto> takeTenWorstReviewersByRate() {
        List<UserEntity> users =
userRepository.findAllByOrderByReviewerRating();

```

```
List<UserDto> worstReviewers = new ArrayList<>();
for (int i = 0; i < 5; ++i) {
    if (i == users.size())
        break;

    worstReviewers.add(userMapping.mapToUserDto(users.get(i)));
}
return worstReviewers;
}
```

Приложение И
(рекомендуемое)

Иллюстрационные материалы к тестированию API

POST /api/bdcourse/user/adduser

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "userId": 0,
  "userNickname": "string",
  "userEmail": "string",
  "userRole": null,
  "userRating": 0,
  "reviewerRating": 0
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcourse/user/adduser' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "userId": 0,
    "userNickname": "string",
    "userEmail": "string",
    "userRole": null,
    "userRating": 0,
    "reviewerRating": 0
  }'
```

Request URL

http://localhost:8080/api/bdcourse/user/adduser

Server response

Code Details

201 Undocumented

Response headers

```
connection: keep-alive
content-length: 0
date: Mon, 10 Jun 2024 11:09:29 GMT
```

Рисунок И.1 – Корректное добавление пользователя: процесс добавления

```
filmcritic=# select * from users;
```

user_id	user_gender	user_reviewer_rating	user_email	user_fio	user_nickname	user_rating	user_socials	roles_id
2	female	0	someemail2@gmail.com		betti	50		1
3	male	0	someemail3@gmail.com		john58	50		2
4	female	0	someemail4@gmail.com		Alexa	50		1
1	male	0	someemail1@gmail.com		jongi	47		2
5		0	robert@email.com		robert	50		1
11		0	string		string	50		1

(6 rows)

Рисунок И.2 – Корректное добавление пользователя: проверка добавления

POST /api/bdcourse/user/adduser

Parameters Cancel Reset

No parameters

Request body required application/json

```
{  "userId": 0,  "userNickname": null,  "userEmail": null,  "userRole": "string",  "userRating": 0,  "reviewerRating": 0}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
'http://localhost:8080/api/bdcourse/user/adduser' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
  "userId": 0,
  "userNickname": null,
  "userEmail": null,
  "userRole": "string",
  "userRating": 0,
  "reviewerRating": 0
}'
```

Request URL

`http://localhost:8080/api/bdcourse/user/adduser`

Server response

Code	Details
400	Error: response status is 400

Response body

Nickname is empty

Рисунок И.3 – Некорректное добавление пользователя

GET /api/bdcourse/user/takeuserbynickname

Parameters

Name	Description
nickname * required string (query)	string

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/user/takeuserbynickname?nickname=string' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/user/takeuserbynickname?nickname=string
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "userId": 11, "userNickname": "string", "userEmail": "string", "userRole": "user", "userRating": 50, "reviewerRating": 0 }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:15:53 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Рисунок И.4 – Получение пользователя по никнейму

PUT /api/bdcourse/user/changeuserrole

Parameters

Name	Description
id * required integer(\$int64) (query)	11
role * required string (query)	reviewer

Execute Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:8080/api/bdcourse/user/changeuserrole?id=11&role=reviewer' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/user/changeuserrole?id=11&role=reviewer
```

Server response

Code	Details
202 Undocumented	<p>Response body</p> <pre>11</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:16:45 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Рисунок И.5 – Изменение роли пользователя

GET

/api/bdcourse/user/takeuserrole

^

Parameters

Cancel

Name	Description
id * required	
integer(\$int64)	11
(query)	

ExecuteClear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/user/takeuserrole?id=11' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/user/takeuserrole?id=11
```

Server response

Code	Details
200	<div><div>Response body</div><div>reviewer</div><div>Download</div></div> <div><div>Response headers</div><div>connection: keep-alive content-length: 8 content-type: text/plain; charset=UTF-8 date: Mon, 10 Jun 2024 11:18:27 GMT keep-alive: timeout=60</div></div>

Рисунок И.6 – Получение роли пользователя

No parameters

Request body required application/json

```
{
  "filmId": 0,
  "filmTitle": null,
  "releaseDate": 0,
  "director": null,
  "filmTiming": 0,
  "annotation": "string",
  "reviewerRating": 0,
  "usersRating": 0,
  "filmGenres": [
    "string"
  ]
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdsource/film/addfilm' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "filmId": 0,
    "filmTitle": null,
    "releaseDate": 0,
    "director": null,
    "filmTiming": 0,
    "annotation": "string",
    "reviewerRating": 0,
    "usersRating": 0,
    "filmGenres": [
      "string"
    ]
  }'
```

Request URL

```
http://localhost:8080/api/bdsource/film/addfilm
```

Server response

Code Details

400

Undocumented

Error: response status is 400

Response body

Film title can't be empty

Рисунок И.7 – Некорректное добавление фильма

POST /api/bdcourse/film/addfilm

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "filmId": 0,
  "filmTitle": "string",
  "releaseDate": 0,
  "director": "string",
  "filmTiming": 0,
  "annotation": "string",
  "reviewerRating": 0,
  "usersRating": 0,
  "filmGenres": [
    "Xoppop",
    "Boemka"
  ]
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
'http://localhost:8080/api/bdcourse/film/addfilm' \
-H 'accept: */*' \
-H 'Content-type: application/json' \
-d '{
  "filmId": 0,
  "filmTitle": "string",
  "releaseDate": 0,
  "director": "string",
  "filmTiming": 0,
  "annotation": "string",
  "reviewerRating": 0,
  "usersRating": 0,
  "filmGenres": [
    "Xoppop",
    "Boemka"
  ]
}'
```

Request URL

`http://localhost:8080/api/bdcourse/film/addfilm`

Server response

Code Details

Рисунок И.8 – Корректное добавление фильма

GET

/api/bdcourse/film/takefilms

⌵

Parameters

Cancel

Name	Description
page	
integer(\$int32)	0
(query)	
size	
integer(\$int32)	10
(query)	

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/api/bdcourse/film/takefilms?page=0&size=10' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/film/takefilms?page=0&size=10
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{ "filmId": 5, "filmTitle": "Гладиатор", "releaseDate": 2000, "director": "Ридли Скотт", "filmRating": 104, "annotation": "string", "reviewerRating": 0, "usersRating": 0, "filmGenres": [] }, { "filmId": 9, "filmTitle": "string", "releaseDate": 0, "director": "string", "filmRating": 0, "annotation": "string", "reviewerRating": 0, "usersRating": 0, "filmGenres": ["Drama", "Horror"] }</pre></div></div>

Рисунок И.9 – Получение списка фильмов

GET

/api/bdcourse/film/takefilmgenres

Parameters

Cancel

Name	Description
id * required integer(\$int64) (query)	<input type="text" value="9"/>

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/film/takefilmgenres?id=9' \
  -H 'accept: */*'
```

Request URL

http://localhost:8080/api/bdcourse/film/takefilmgenres?id=9

Server response

Code	Details
200	<div><div>Response body</div><div><pre>[{ "genreId": 3, "genreName": "Xopop" }, { "genreId": 4, "genreName": "Goesak" }]</pre></div><div><div>Download</div></div></div> <div><div>Response headers</div><div><pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:42:06 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre></div></div>

Рисунок И.10 – Получение жанров фильма

GET /api/bdcourse/film/takefilmsbygenre

Parameters

Name	Description
id * required	
integer(\$int32)	3
(query)	

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/film/takefilmsbygenre?id=3' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/film/takefilmsbygenre?id=3
```

Server response

Code Details

200

Response body

```
{
  {
    "fileId": 3,
    "fileTitle": "Alice in wonderlands",
    "releaseDate": 2011,
    "director": "director3",
    "fileRating": 7,
    "annotation": "some annotation3",
    "reviewerRating": 0,
    "usersRating": 0,
    "filmGenres": [
      "Xoppop"
    ]
  },
  {
    "fileId": 9,
    "fileTitle": "string",
    "releaseDate": 0,
    "director": "string",
    "fileRating": 0,
    "annotation": "string",
    "reviewerRating": 0,
    "usersRating": 0,
    "filmGenres": [
      "Xoppop",
      "Боевик"
    ]
  }
}
```

Download

Рисунок И.11 – Получение фильмов определенного жанра

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "reviewId": 0,
  "reviewTitle": null,
  "authorOrganization": "string",
  "reviewText": null,
  "filmRate": 0,
  "reviewRate": 0,
  "film": 0,
  "user": 11
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcourse/review/addreview' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "reviewId": 0,
    "reviewTitle": null,
    "authorOrganization": "string",
    "reviewText": null,
    "filmRate": 0,
    "reviewRate": 0,
    "film": 0,
    "user": 11
  }'
```

Request URL

http://localhost:8080/api/bdcourse/review/addreview

Server response

Code

Details

400

Undocumented Error: response status is 400

Response body

Your review text can't be empty

Рисунок И.12 – Некорректное добавление рецензии

POST

/api/bdcourse/review/addreview

^

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "reviewId": 0,
  "reviewTitle": "string",
  "authorOrganization": "string",
  "reviewText": "string",
  "filmRate": 5,
  "reviewRate": 0,
  "film": 5,
  "user": 11
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcourse/review/addreview' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "reviewId": 0,
    "reviewTitle": "string",
    "authorOrganization": "string",
    "reviewText": "string",
    "filmRate": 5,
    "reviewRate": 0,
    "film": 9,
    "user": 11
  }'
```

Request URL

http://localhost:8080/api/bdcourse/review/addreview

Server response

Code

Details

201

Undocumented

Response body

0

Рисунок И.13 – Корректное добавление рецензии

GET /api/bdcourse/review/takereviews

Parameters

Name	Description
page	integer(\$int32) (query)
size	integer(\$int32) (query)

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/review/takereviews?page=0&size=10' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/review/takereviews?page=0&size=10
```

Server response

Code Details

200

Response body

```
{
  "reviews": [
    {
      "reviewId": 5,
      "reviewTitle": "Интересное произведение Дени Вильмёва",
      "filmRate": 5,
      "reviewRate": 0,
      "film": 1,
      "user": 1
    },
    {
      "reviewId": 6,
      "reviewTitle": "Some good film",
      "authorOrganization": null,
      "reviewText": "teeext",
      "filmRate": 4,
      "reviewRate": 4,
      "film": 1,
      "user": 1
    },
    {
      "reviewId": 7,
      "reviewTitle": "string",
      "authorOrganization": "string",
      "reviewText": "string",
      "filmRate": 5,
      "reviewRate": 3,
      "film": 0,
      "user": 11
    }
  ]
}
```

Download

Рисунок И.14 – Получение списка рецензий

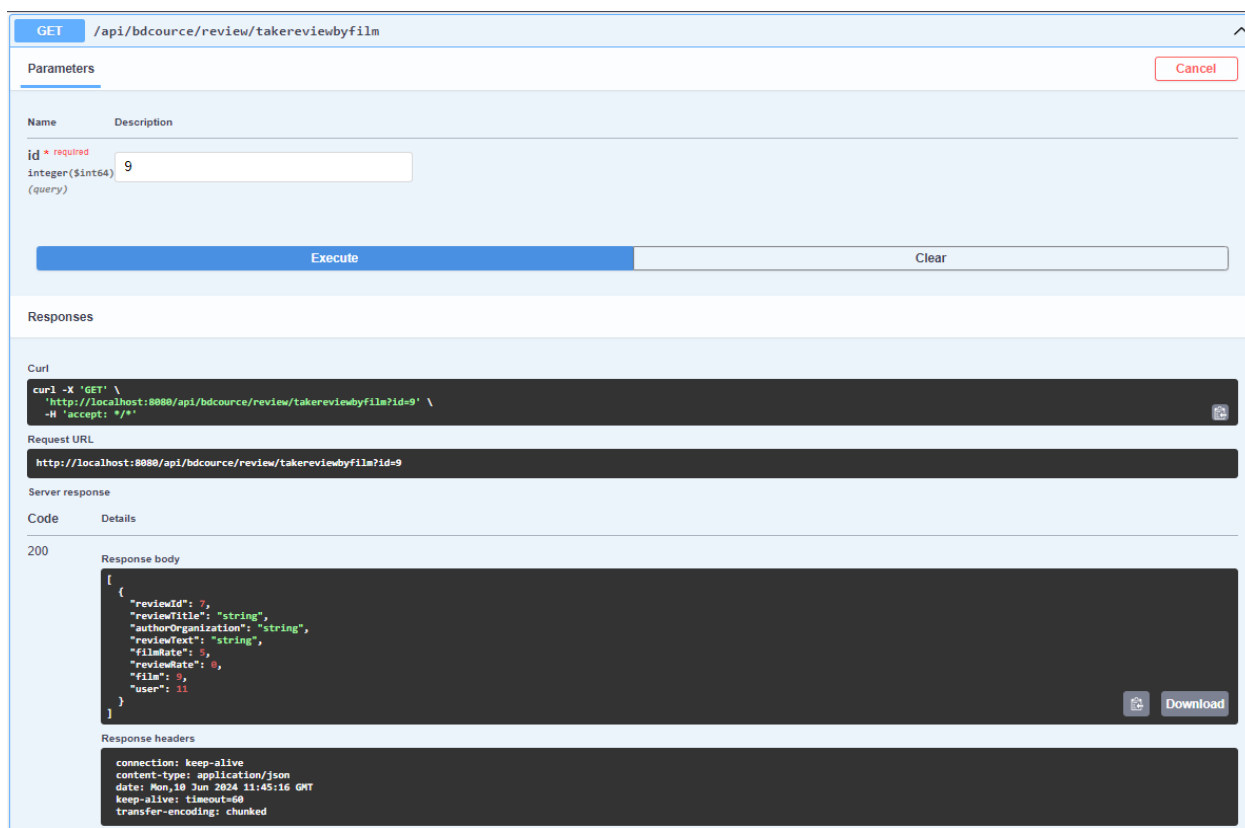


Рисунок И.15 – Получение списка рецензий к конкретному фильму

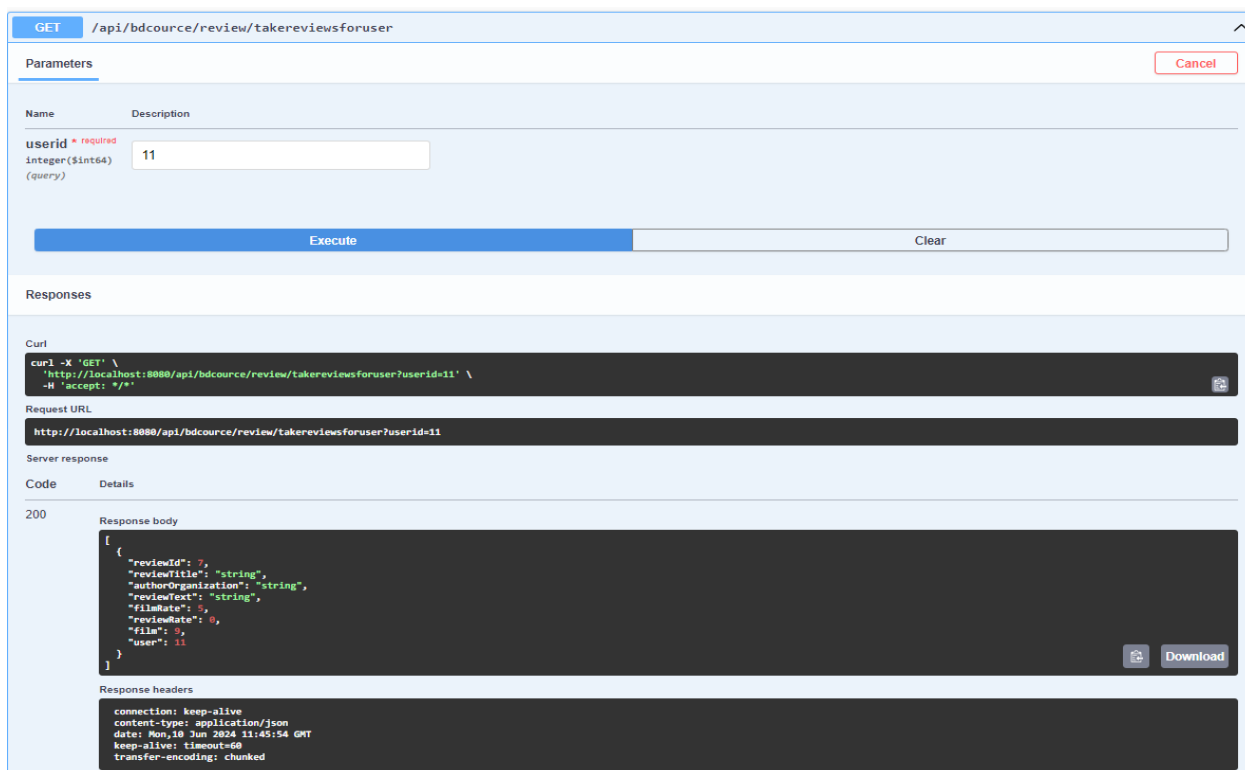


Рисунок И.16 – Получение списка рецензий конкретного пользователя

POST /api/bdcourse/rating/addrating

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "ratingId": 0,
  "ratingValue": -5,
  "ratedFilm": 9,
  "ratedUser": 11,
  "ratedReview": 0
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcourse/rating/addrating' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "ratingId": 0,
    "ratingValue": -5,
    "ratedFilm": 9,
    "ratedUser": 11,
    "ratedReview": 0
  }'
```

Request URL

http://localhost:8080/api/bdcourse/rating/addrating

Server response

Code	Details
400	Error: response status is 400

Response body

```
Wrong rating value - -5
```

Рисунок И.17 – Добавление оценки с некорректным значением

POST /api/bdcourse/rating/addrating

Parameters Cancel Reset

No parameters

Request body ^{required} application/json

```
{
  "ratingId": 0,
  "ratingValue": 5,
  "ratedFile": 9,
  "ratedUser": 11,
  "ratedReview": 7
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcourse/rating/addrating' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "ratingId": 0,
    "ratingValue": 5,
    "ratedFile": 9,
    "ratedUser": 11,
    "ratedReview": 7
  }'
```

Request URL

http://localhost:8080/api/bdcourse/rating/addrating

Server response

Code	Details
201	Response headers

Undocumented

```
connection: keep-alive
content-length: 0
date: Mon, 10 Jun 2024 11:48:33 GMT
keep-alive: timeout=60
```

Рисунок И.18 – Добавление оценки с корректным значением

GET /api/bdcourse/rating/takereviewerrate

Parameters

Name	Description
id * required integer(\$int64) (query)	11

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/api/bdcourse/rating/takereviewerrate?id=11' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/rating/takereviewerrate?id=11
```

Server response

Code	Details
200	<p>Response body</p> <p>3</p> <p>Download</p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:58:17 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Рисунок И.19 – Получение рейтинга рецензента

GET /api/bdcourse/rating/reviewrate

Parameters

Name	Description
id * required integer(\$int64) (query)	7

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/api/bdcourse/rating/reviewrate?id=7' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcourse/rating/reviewrate?id=7
```

Server response

Code	Details
200	<p>Response body</p> <p>3</p> <p>Download</p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:51:00 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Рисунок И.20 – Получение рейтинга рецензии

GET /api/bdcourse/rating/filmreviewerrate

Parameters

Cancel

Name	Description
id * required integer(\$int64) (query)	9

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/rating/filmreviewerrate?id=9' \
  -H 'accept: */*'
```

Request URL

http://localhost:8080/api/bdcourse/rating/filmreviewerrate?id=9

Server response

Code	Details
200	<p>Response body</p> <p>5</p> <p>Download</p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:51:51 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Рисунок И.21 – Получение рейтинга фильма от рецензентов

GET /api/bdcourse/rating/filmrate

Parameters

Cancel

Name	Description
id * required integer(\$int64) (query)	9

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcourse/rating/filmrate?id=9' \
  -H 'accept: */*'
```

Request URL

http://localhost:8080/api/bdcourse/rating/filmrate?id=9

Server response

Code	Details
200	<p>Response body</p> <p>3</p> <p>Download</p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:52:24 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Рисунок И.22 – Получение рейтинга фильма от пользователей

POST

/api/bdcouce/comment/addcomment

^

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "commentId": 0,
  "commentText": null,
  "commentTime": "2024-06-10T14:59:34.109Z",
  "commentedFilm": 7,
  "user": 11,
  "commentedReview": 0
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcouce/comment/addcomment' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "commentId": 0,
    "commentText": null,
    "commentTime": "2024-06-10T14:59:34.109Z",
    "commentedFilm": 7,
    "user": 11,
    "commentedReview": 0
  }'
```

Request URL

```
http://localhost:8080/api/bdcouce/comment/addcomment
```

Server response

Code	Details
400	Error: response status is 400

Response body

```
Comment can't be empty
```

Рисунок И.23 – Некорректное добавление комментария

POST

/api/bdcouce/comment/addcomment

^

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "commentId": 0,
  "commentText": "string",
  "commentTime": "2024-06-10T11:54:13.920Z",
  "commentedFile": 9,
  "commentedReview": 7,
  "user": 3,
  "commentedReview": 7
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/bdcouce/comment/addcomment' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "commentId": 0,
    "commentText": "string",
    "commentTime": "2024-06-10T11:54:13.920Z",
    "commentedFile": 9,
    "commentedReview": 7,
    "user": 3,
    "commentedReview": 7
  }'
```

Request URL

http://localhost:8080/api/bdcouce/comment/addcomment

Server response

Code

Details

201

undocumented

Response headers

connection: keep-alive
content-length: 0

Рисунок И.24 – Корректное добавление комментария

GET /api/bdcouce/comment/takeusercomments

Parameters

Name	Description
id * required integer(\$int64) (query)	3

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcouce/comment/takeusercomments?id=3' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcouce/comment/takeusercomments?id=3
```

Server response

Code	Details
200	<p>Response body</p> <pre>[{ "commentId": 1, "commentText": "nice file", "commentTime": "2023-12-16T13:22:15.000+00:00", "commentedFile": 1, "user": 3, "commentedReview": null }, { "commentId": 4, "commentText": "cool x2", "commentTime": "2023-12-16T13:24:44.000+00:00", "commentedFile": 2, "user": 3, "commentedReview": 2 }, { "commentId": 637, "commentText": "string", "commentTime": "2024-06-10T11:54:34.975+00:00", "commentedFile": 3, "user": 3, "commentedReview": 7 }]</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json</pre>

Download

Рисунок И.25 – Получение комментариев пользователя

The screenshot displays a REST client interface for a GET request to the endpoint `/api/bdcouce/comment/takereviewcomments`. The request parameters include a required `id` (integer) with the value `7`. The interface shows the executed request, the resulting 200 status code, and the JSON response body. The response body contains a single comment object with the following fields: `commentId` (637), `commentText` (string), `commentTime` (2024-06-10T11:54:34.975+00:00), `commentedFile` (9), `user` (3), and `commentedReview` (7). The response headers indicate a `content-type` of `application/json` and a `transfer-encoding` of `chunked`.

Parameters

Name	Description
<code>id</code> * required integer(\$int64) (query)	7

Execute **Clear**

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcouce/comment/takereviewcomments?id=7' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcouce/comment/takereviewcomments?id=7
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "commentId": 637, "commentText": "string", "commentTime": "2024-06-10T11:54:34.975+00:00", "commentedFile": 9, "user": 3, "commentedReview": 7 }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Mon, 10 Jun 2024 11:55:47 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	OK	No links

Рисунок И.26 – Получение комментариев к рецензии

GET

/api/bdcouce/comment/takefilmcomments

^

Parameters

Cancel

Name	Description
id * required	
integer(\$int64)	9
(query)	

ExecuteClear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/bdcouce/comment/takefilmcomments?id=9' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/bdcouce/comment/takefilmcomments?id=9
```

Server response

Code	Details
200	<div>Response body</div> <div><pre>{ "commentId": 637, "commentText": "string", "commentTime": "2024-06-10T11:54:34.975+00:00", "commentedFilm": 9, "user": 3, "commentedReview": 7 }</pre></div> <div>Download</div>

Response headers

```
connection: keep-alive
content-type: application/json
date: Mon, 10 Jun 2024 11:56:27 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Рисунок И.27 – Получение комментариев к фильму