

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

**Клиентское приложение для web-портала
агрегации зрительской и экспертной
кинокритики**

Студент _____ Кубанычбекова Айпери Кубанычбековна
(подпись, дата) (ФИО полностью)

Руководитель _____ Козлов А.Ю.
(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 Кубанычбекова Айпери Кубанычбековна гр. 20ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Клиентское приложение для web-портала агрегации зрительской и экспертной кинокритики

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 2024г.

2. Руководитель работы _____

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

1) Назначение: Клиентское приложение для web-портала агрегации зрительской и экспертной кинокритики

2) Среда выполнения: Windows 10

Языковые средства разработки: JavaScript

Технологии разработки: Компонентный подход

3) Основные функции разрабатываемого программного обеспечения:

а) регистрация и авторизация пользователя на портале;

б) взаимодействие с личным кабинетом;

в) взаимодействие с окном просмотра рецензий;

г) взаимодействие с окном создания рецензии;

д) взаимодействие с окном отзывов пользователей;

4. Перечень подлежащих разработке вопросов:

- | |
|---|
| <i>1) Анализ предметной области и постановка задачи</i> |
| <i>2) Проектирование клиентского приложения</i> |
| <i>3) Разработка клиентского приложения</i> |
| <i>4) Тестирование</i> |

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 103 страницы, 39 рисунков, 13 таблиц, 1 приложение и 5 использованных источников.

WEB-ПРИЛОЖЕНИЕ, ПАТТЕРН ПРОЕКТИРОВАНИЯ MVC, REACT, КИНОКРИТИКА, АГРЕГАЦИЯ КИНОКРИТИКИ, ИНФОРМАЦИОННАЯ СИСТЕМА, REST.

Объект исследования – агрегация зрительской и экспертной киноcritики.

Цель выпускной квалификационной работы – разработать клиентское приложение для web-портала зрительской и экспертной киноcritики.

Средства разработки – библиотека React, язык программирования Javascript, case-средство Visual Paradigm, для управления состоянием приложения Redux-toolkit, для стилизации ui-компонентов Tailwindcss, среда разработки Visual Studio Code.

Среда исполнения – Windows 10, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы было разработано клиентское приложение для web-портала агрегации зрительской и экспертной киноcritики.

					ПГУ 09.03.04 – 8BP201.11 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Клиентское приложение для web-портала агрегации зрительской и экспертной киноcritики Пояснительная записка	Лит.	Лист	Листов
Разраб.		Кубанычбекова						
Провер.		Козлов А.Ю.					4	103
Реценз.						Группа 20ВП1		
Н. Контр.		Такташкин Д.В						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку клиентского приложения	9
1.1 Анализ предметной области приложения	9
1.2 Сравнительный анализ аналогичных разработок.....	12
1.3 Постановка задачи на разработку клиентского приложения	14
1.3.1 Цели разработки и назначения приложения	14
1.3.2 Классы пользователей приложения	14
1.3.3 Перечень автоматизируемых функций.....	17
1.4 Планирование и функционально-стоимостный анализ разработки приложения.....	18
2 Проектирование клиентского приложения	24
2.1 Моделирование функциональных возможностей приложения	24
2.2 Разработка концептуальной модели данных	30
2.3 Проектирование пользовательского интерфейса	37
2.4 Выбор технологий и средств разработки клиентского приложения.....	47
3 Разработка клиентского приложения.....	49
3.1 Разработка архитектуры клиентского приложения на основе шаблона MVC	49
3.2 Разработка пользовательского интерфейса клиентского приложения	52
3.3 Разработка средств взаимодействия с серверным приложением	53
3.4 Разработка бизнес-логики клиентского приложения.....	54
3.4.1 Управление состоянием	54
3.4.2 Навигация	57
3.4.3 Валидация полей данных	59
4 Тестирование клиентского приложения	62
4.1 Функциональное тестирование	62
4.1.1 Определение классов эквивалентности	63

4.1.2 Метод попарного тестирования.....	64
4.2 Структурное тестирование.....	67
4.2.1 Покрытие операторов, решений и условий.....	68
4.3 Расчет метрик	70
4.3.1 Метрика Холстеда.....	71
4.3.2 Метрика Чепина	73
Заключение	75
Список использованных источников	76
Приложение А (рекомендуемое) Листинг клиентского приложения.....	78

Введение

Изучив историю развития кинокритики в России, можно заключить, что в настоящее время этот жанр на пике своей популярности. Отзывы о фильмах можно найти в различных изданиях — от специализированных журналов до популярных интернет ресурсов. Любой желающий может высказать свое мнение о кино. Популярность данного жанра объясняется тем, что кино, будучи одним из самых популярных и массовых искусств прошлого века, обладает языком, доступным широкой аудитории, и огромным потенциалом для отражения реальности. Таким образом, важность данного направления связана с его популярностью и необходимостью дальнейшего изучения.

Несмотря на широкое распространение жанра кинокритики, многие исследователи утверждают, что он находится на упадке в современной России. Филолог и журналист Михаил Ратгауз выделяет следующие причины снижения качества материалов в жанре кинокритики:

- сдвиг в постиндустриальном обществе, где оформление становится более значимым, чем сама суть;
- огромное количество новых фильмов, что приводит к необходимости суммирования информации и быстрого определения, стоит ли фильм отметить в своем списке или нет;
- интенсивное развитие брокерской индустрии, которая ставит под вопрос уникальность кинокритических взглядов;
- происходящие изменения в самом мире кино, где кинематограф больше не выступает в роли прогрессивного искусства, а скорее следует за тенденциями, копируя уже существующие формы и темы.

Появление интернет-блогов привело к уменьшению влияния профессиональных рецензентов, а развитие коммерческого кино и его стремление к лидерству в прокате негативно повлияли на качество обзоров. Теперь основной целью рецензента стало убедить зрителя посмотреть

определенный фильм. Настоящую критику можно найти только в специализированных изданиях.

Объектом исследования является клиентское приложение для web-портала агрегации зрительской и экспертной кинокритики.

Целью выпускной квалификационной работы является:

- анализ предметной области и постановка задачи на разработку клиентского приложения;
- сравнительный анализ аналогичных разработок;
- проектирование пользовательского интерфейса;
- реализация клиентского приложения;
- тестирование.

В ходе выполнения выпускной квалификационной работы поставленные задачи были выполнены.

1 Анализ предметной области и постановка задачи на разработку клиентского приложения

1.1 Анализ предметной области приложения

Предметная область данной работы охватывает сферу кино и веб-приложение, связанное с анализом и оценкой фильмов. Прежде чем провести детальное рассмотрение структуры этой темы, необходимо определить ключевые понятия данной сферы.

Кинематография – отрасль культуры и промышленности, осуществляющая производство фильмов и показ их зрителю; один из самых массовых видов искусства.

Кинодраматургия – литературно-кинематографический вид творчества; производением кинодраматургии является сценарий – литературно-художественная основа фильма.

Киноискусство – род искусства, произведения которого создаются с помощью киносъемки реальных, специально инсценированных или воссозданных средствами мультипликации событий

Фильм (от англ. film – пленка) (кинофильм), совокупность фотографических изображений (кадров), последовательно расположенных на киноплёнке, связанных одним сюжетом и предназначенных для воспроизведения на экране; произведение киноискусства; фильмокопия.

Рецензия – это жанр в журналистике или научной и художественной критике. Эта работа содержит обзор произведения, его оценку и отзыв автора рецензии [1].

Кинокритик – это специалист, анализирующий кинематографический процесс и рецензирующий кинофильмы. Отражая свои впечатления в рецензиях на фильмы, кинокритик высказывает своё мнение о достоинствах и недостатках данного произведения [2].

Кинокритика толкует и оценивает произведения киноискусства с различных точек зрения: с потребительской – рецензирование и оценивание, с

публицистической – связь с насущными проблемами общественной и духовной жизни, с точки зрения киноведения – как объект искусства, определение его особенностей и основных характеристик.

Веб-портал агрегации зрительской и клиентской кинокритики – онлайн платформа, которая собирает, систематизирует и представляет критические оценки по кинокартинам различных жанров от профессиональных экспертов.

Данная платформа дает возможность создавать, хранить и изучать рецензии на фильмы, а также оценивать их качество.

Для того, чтобы стать рецензентом в системе пользователю необходимо предоставить один из следующих документов:

- диплом об окончании высшего образования в области журналистики или кинопроизводстве;
- выписка с работы в специализированном ресурсе;
- другой документ, подтверждающий факт работы в специализированном ресурсе.

Веб-приложение для web – портала агрегации зрительской и экспертной кинокритики создано для удовлетворения потребности в анализе и оценке фильмов, собирая отзывы от профессиональных кинокритиков и зрителей. Данная платформа не только позволяет пользователям делиться своим мнением о кино, но и предоставляет ценную информацию для кинематографического рынка.

Такая платформа может стать не только площадкой для обмена мнениями о кино, но и инструментом для анализа кинорынка. Агрегированные данные о рейтингах, рецензиях и предпочтениях зрителей могут быть полезны как для самих кинокритиков, так и для кинопрокатчиков, продюсеров и киностудий, помогая им оценивать потенциал фильмов и принимать решения о их продвижении.

На рисунке 1 представлен сценарий выполнения бизнес-процессов системы в виде диаграммы деятельности.

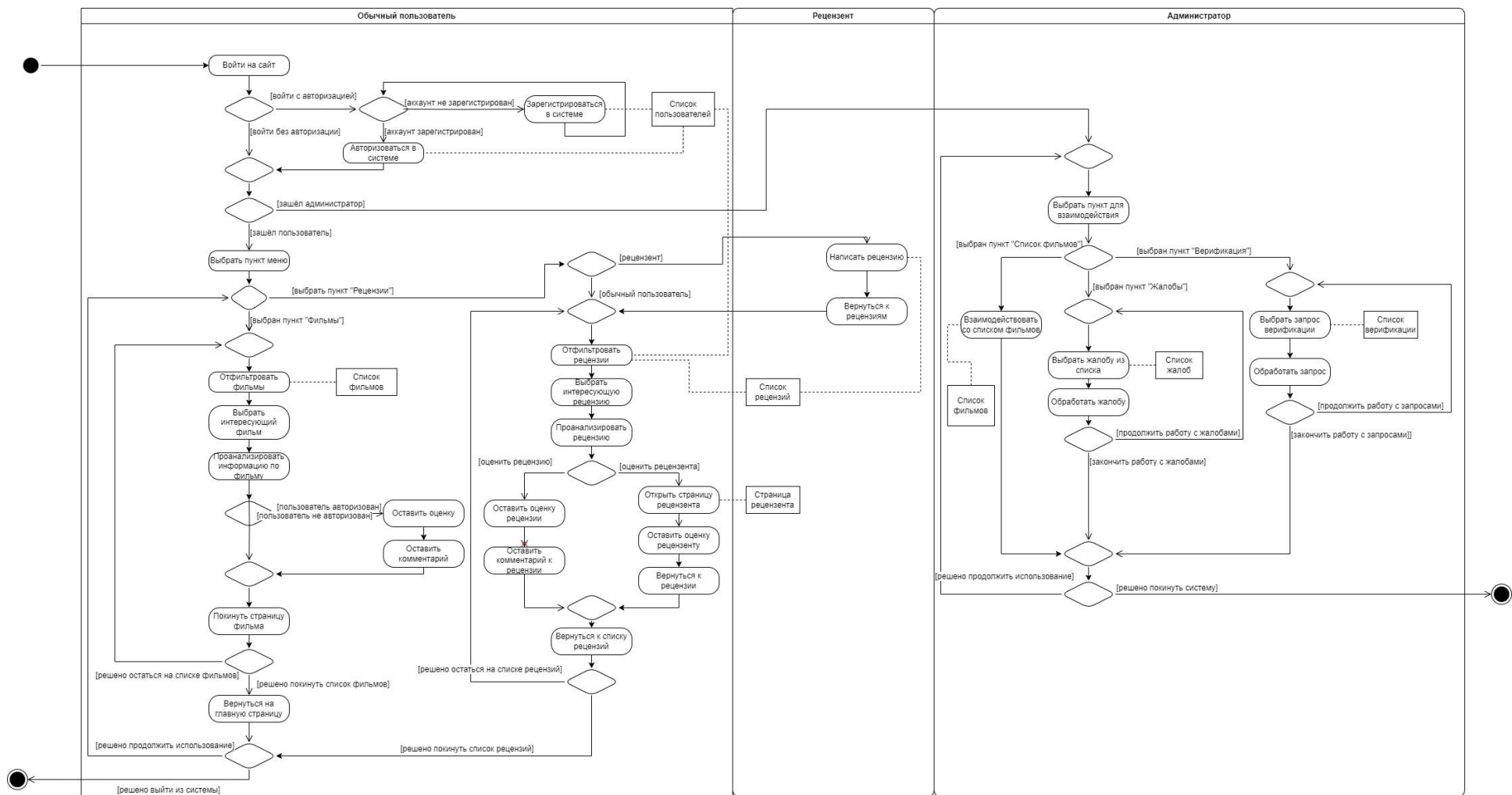


Рисунок 1 – Сценарий выполнения бизнес-процессов в системе

Изучение предметной области и информационных задач представляет собой ключевой этап в процессе разработки информационной системы. Этот этап необходим для понимания потребностей пользователей и создания основы для функциональной модели системы.

1.2 Сравнительный анализ аналогичных разработок

Анализ платформ – агрегаторов рецензий даст возможность выявить самые популярные программные решения, а также оценить их широкое применение и текущую потребительскую привлекательность.

Для того, чтобы сравнивать платформы-агрегаторы рецензий, нужно определить, по каким показателям они могут различаться:

- сообщество;
- функции и удобство использования;
- прозрачность и достоверность;
- полнота и релевантность;
- надежность и авторитет;
- доступность и ценообразование;
- дополнительные особенности.

По итогу были проанализированы и сравнены следующие платформы:

- Кинопоиск;
- Кинокритик;
- Metacritic;
- Rotten Tomatoes.

Среди представленных платформ-агрегаторов встречаются как и бесплатные, так и платные. Информация о сравнительном анализе представлена в таблице 1.

Таблица 1 – Сравнительный анализ

Критерий	Кинопоиск	Кинокритик	Metacritic	Rotten Tomatoes
Сообщество	русскоязычн ое	русскоязычн ое	англоязычное	англоязычное
Прозрачност ь и достовернос ть	не ясно, верифициро ваны ли отзывы	не верифициров аны	требуется проверки СМИ или экспертов	верифицирует критиков, аудиторию
Полнота и релевантнос ть	большое количество отзывов	меньшее количество обзоров, ориентирова нных на экспертов	меньшее количество обзоров, но высокое качество	большой объем отзывов от критиков и аудитории
Функции и удобство использован ия	ограниченн ые фильтры и функции сортировки	удобный интерфейс с фильтрами и функциями сортировки	расширенные функции и надежная навигация	интуитивно понятная навигация и различные варианты просмотра
Надежность и авторитет	популярная и широко используема я платформа	менее известная, но уважаемая критиками	высокоавторит етная и признанная аудиторией	высокоавторит етная и пользуется доверием зрителей
Доступность и ценообразов ание	бесплатная	бесплатная	платные подписки для доступа к дополнительны м функциям	бесплатная для пользователей, платная для критиков
Дополнител ьные особенности	интеграция с социальным и сетями	анализ настроений и подкасты	анализ настроений, сопоставление рейтингов	консенсус критиков, оценка аудитории

Проведя сравнительный анализ аналогичных платформ-агрегаторов, можно сделать вывод, что на русскоязычном рынке отсутствует платформа-агрегатор рецензий, которая специализировалась бы исключительно на фильмах и отзывах от экспертных рецензентов, а также проводила бы верификацию рецензентов.

Кинопоиск, одна из самых популярных платформ для русскоязычных зрителей, предлагает большое количество отзывов, но не все они от экспертов. Кинокритик, известный своими экспертными обзорами, имеет ограниченное количество отзывов по сравнению с более крупными платформами.

Отсутствие такой платформы создает пробел для русскоязычных зрителей, которые ищут надежные и авторитетные экспертные обзоры фильмов. Собранные данные в ходе исследования, станут основой при проектировании и разработки платформы-агрегатора рецензий.

1.3 Постановка задачи на разработку клиентского приложения

1.3.1 Цели разработки и назначения приложения

Разрабатываемое клиентское приложение, предназначенное для сбора профессиональных рецензий на фильмы из проверенных источников, должно предоставить пользователям удобные средства для работы с рецензиями.

Основные цели разработки:

- агрегация сборов: сбор и организация профессиональных обзоров фильмов из проверенных источников;
- фильтрация контента: узкая специализация на фильмах, исключение других форм контента, таких как мультфильмы, книги и игры;
- современный интерфейс: разработка современного интерфейса, который ориентирован на удобство пользователей, интерфейс должен быть простым в навигации и обеспечивать быстрый доступ к необходимой информации.

После завершения основной фазы разработки и запуска клиентского приложения, необходимо обеспечивать его стабильную работу и внедрять новые возможности для улучшения пользовательского опыта.

1.3.2 Классы пользователей приложения

Реализуемая система будет полезна для следующих категорий пользователей:

- кинолюбители и энтузиасты: пользователи, которые хотят быть в курсе последних фильмов и их критического приема;
- кинокритики и обозреватели: пользователи, которые ищут удобный способ доступа к профессиональным рецензиям на фильмы для своей работы или личного интереса;
- исследователи и студенты кино: пользователи, которые нуждаются в надежных источниках критических обзоров для своих исследований или учебных заданий;
- создатели фильмов и кинематографисты: пользователи, которые хотят получить обратную связь и критику своих фильмов от профессиональных критиков;
- зрители, которые хотят принять обоснованное решение о просмотре фильма: Пользователи, которые ищут профессиональные мнения, чтобы решить, стоит ли смотреть фильм или нет.

Исходя из различных потребностей категорий пользователей, реализуемая система будет поддерживать четыре класса пользователей, функциональные возможности которых представлены в таблице 2.

Таблица 2 – Классы пользователей информационной системы

№ п/п	Классы пользователей	Наименование приложения	Автоматизируемые бизнес-процессы	Классы информационных объектов
1	Неавторизованный пользователь	Просмотр рецензий	Просмотр рецензий	Фильм, рецензент, рецензия
			Просмотр комментариев	Фильм, рецензент, рецензия, комментарий
			Фильтрация фильмов	Список фильмов
			Фильтрация рецензий	Список фильмов, список рецензий, рецензент
2	Обычный пользователь	Просмотр и комментирование рецензии	Просмотр рецензий	Фильм, рецензия, рецензент
			Просмотр комментариев	Фильм, рецензент, рецензия, комментарий
			Создание комментариев	Фильм, рецензент, рецензия
			Оценивание фильмов	Фильм
			Оценивание рецензий	Фильм, рецензия

Продолжение таблицы 2

№ п/п	Классы пользователей	Наименование приложения	Автоматизируемые бизнес-процессы	Классы информационных объектов
3	Обычный пользователь	Просмотр и комментирование рецензии	Фильтрация фильмов	Список фильмов
			Фильтрация рецензий	Список фильмов, список рецензий, рецензент
4	Пользователь-рецензент	Создание и просмотр рецензий	Просмотр рецензий	Фильм, рецензент, рецензия
			Просмотр комментариев	Фильм, рецензент, рецензия, комментарий
			Создание комментариев	Фильм, рецензент, рецензия
			Оценивание фильмов	Фильм
			Оценивание рецензий	Фильм, рецензент, рецензия
			Фильтрация фильмов	Список фильмов
			Фильтрация рецензий	Список фильмов, список рецензий, рецензент
			Создание рецензий	Фильм, рецензент, рецензия, список дополнительных материалов
5	Администратор	Приложение администратора	Добавление фильма	Фильм, список фильмов
			Верификация статуса пользователя	Пользователь, верификационный документ
			Удаление контента	
			Блокировка пользователя	Пользователь

Система предлагает различные уровни доступа, позволяющие каждому пользователю получать необходимую информацию и функциональность. Неавторизованный пользователь имеет базовый доступ к просмотру рецензий и фильтров, обычный пользователь может комментировать и оценивать, а пользователь-рецензент обладает полным функционалом для написания и редактирования рецензий. Администратор отвечает за управление системой, добавляя фильмы, верифицируя пользователей и контролируя контент. Такая

четкая система разделения ролей обеспечивает эффективную работу системы и гарантирует безопасность данных и взаимодействий пользователей.

1.3.3 Перечень автоматизируемых функций

Информационная система должна предоставить следующие функции:

- регистрация и авторизация пользователя на портале;
- взаимодействие с личным кабинетом (изменение данных о пользователе, удаление пользователя, получение доступа к отзывам и рецензиям пользователя, повышение уровня пользователя до «профессионального» путем верификации);
- взаимодействие с окном просмотра рецензий (просмотр списка рекомендуемых рецензий, просмотр списка рецензий, просмотр содержания конкретной рецензии, фильтрация рецензий по типу рецензии/размеру рецензии/имени критика в системе/ФИО критика/жанру кинокартины/названию кинокартины, оставление отзыва/оценки к рецензии, сохранение понравившейся рецензии в хранилище);
- взаимодействие с окном создания рецензии (выбор типа рецензии, выбор кинокартины для рецензирования, написание текста рецензии, прикрепление сопроводительных материалов, выставление оценки произведению);
- взаимодействие с окном отзывов пользователей (просмотр отзывов по конкретной картине, выставление оценок кинокартине, возможность пожаловаться на отзыв другого пользователя);
- взаимодействие с окном сохраненных рецензий (просмотр хранимых рецензий, удаление хранимых рецензий, выгрузка рецензий на диск пользователя).

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

Система предназначена для агрегации профессиональной кинокритики по кинокартинам различных жанров. Она позволяет создавать, хранить, просматривать и оценивать рецензии на фильмы.

1.4 Планирование и функционально-стоимостный анализ разработки приложения

В начальной стадии процесса разработки программного обеспечения планирование играет критически важную роль. Тщательное планирование гарантирует своевременное завершение проекта и удовлетворительное качество конечного продукта. Основной целью планирования является создание подробного расписания работ в рамках проекта и оценка приблизительных затрат на его реализацию. Кроме того, оно позволяет определить необходимые ресурсы и составить реалистичный график выполнения задач.

В таблице 3 перечислены задачи, необходимые для завершения проекта. Для построения графика была выбрана рабочая неделя, состоящая из пяти дней.

Таблица 3 – Список задачи проекта

№	Задача	Длительность	Предшественник
1	Начало проекта	-	-
2	Анализ предметной области и постановка задачи на разработку клиентского приложения	14 дней	
3	Анализ предметной области приложения	1 день	1
4	Сравнительный анализ аналогичных разработок	2 дня	3
5	Постановка задачи на разработку клиентского приложения	5 дней	
6	Цели разработки и назначения приложения	3 дня	4
7	Классы пользователей приложения	1 день	6
8	Перечень автоматизируемых функций	1 день	7
9	Планирование и функционально-стоимостный анализ разработки приложения	6 дней	
10	Выбор нотации и программных средств для планирования и оценки процесса разработки приложения	1 день	8
11	Планирование процесса разработки приложения	4 дня	10
12	Оценка временных и финансовых затрат на разработку	1 день	11
13	Проектирование клиентского приложения	10 дней	
14	Моделирование функциональных возможностей приложения	2 дня	12
15	Разработка концептуальной модели данных	1 день	14
16	Проектирование пользовательского интерфейса	4 дня	15
17	Выбор технологий и средств разработки клиентского приложения	3 дня	16

Продолжение таблицы 3

№	Задача	Длительность	Предшественник
18	Разработка клиентского приложения	12 дней	
19	Разработка архитектуры клиентского приложения на основе шаблона mvс	1.5 дня	17
20	Разработка пользовательского интерфейса клиентского приложения	2.5 дня	19
21	Разработка средств взаимодействия с серверным приложением	3 дня	20
22	Разработка бизнес-логики клиентского приложения	5 дней	
23	Управление состоянием	2 дня	21
24	Навигация	2 дня	23
25	Валидация полей данных	1 день	24
26	Тестирование	5 дней	
27	Функциональное тестирование	2 дня	
38	Определение классов эквивалентности	1 день	25
29	Метод попарного тестирования	1 день	28
30	Структурное тестирование	1 день	
31	Покрытие операторов, решений и условий	1 день	29
32	Расчет метрик	2 дня	
33	Метрика Холстеда	1 день	31
34	Метрика Чепина	1 день	33
35	Сдача проекта	1 день	34
36	Оформление и сбор по документации	5 дней	35
37	Ввод эксплуатации	1 день	26
38	Конец проекта	-	37

Следующим шагом является определение необходимых ресурсов и их распределение по задачам проекта. Таблица 4 содержит список ресурсов, требующихся для реализации проекта.

Таблица 4 – Список ресурсов

Название ресурса	Тип ресурса	Стандартная вставка	Сверхурочная вставка
Аналитик	Работа	280руб./час	380руб./час
Программист	Работа	300руб./час	400руб./час
Дизайнер	Работа	200руб./час	300руб./час
Тестировщик	Работа	150руб./час	250руб./час
Интернет	Материал	-	-
Ноутбук	Материал	80000руб	-

После определения необходимых ресурсов следующим шагом является их распределение по задачам проекта. Это гарантирует, что ресурсы используются эффективно и что каждая задача имеет соответствующие средства для

своевременного завершения. Таблица 5 предоставляет подробную информацию о распределении ресурсов, указывая, какие ресурсы назначены для каждой задачи, а также их количество и продолжительность использования.

Таблица 5 – Распределение ресурсов проекта по задачам

№	Задача	Ресурсы
1	Начало проекта	-
2	Анализ предметной области и постановка задачи на разработку клиентского приложения	-
3	Анализ предметной области приложения	Аналитик
4	Сравнительный анализ аналогичных разработок	Аналитик
5	Постановка задачи на разработку клиентского приложения	-
6	Цели разработки и назначения приложения	Аналитик
7	Классы пользователей приложения	Аналитик
8	Перечень автоматизируемых функций	Аналитик, Программист
9	Планирование и функционально-стоимостный анализ разработки приложения	-
10	Выбор нотации и программных средств для планирования и оценки процесса разработки приложения	Аналитик
11	Планирование процесса разработки приложения	Аналитик
12	Оценка временных и финансовых затрат на разработку	Аналитик
13	Проектирование клиентского приложения	-
14	Моделирование функциональных возможностей приложения	Программист
15	Разработка концептуальной модели данных	Аналитик
16	Проектирование пользовательского интерфейса	Программист, дизайнер
17	Выбор технологий и средств разработки клиентского приложения	Программист
18	Разработка клиентского приложения	-
19	Разработка архитектуры клиентского приложения на основе шаблона mvc	Аналитик, программист
20	Разработка пользовательского интерфейса клиентского приложения	Программист, дизайнер
21	Разработка средств взаимодействия с серверным приложением	Программист Программист
22	Разработка бизнес-логики клиентского приложения	-
23	Управление состоянием	Программист
24	Навигация	Программист
25	Валидация полей данных	Программист
26	Тестирование	-
27	Функциональное тестирование	-
38	Определение классов эквивалентности	Тестировщик
29	Метод попарного тестирования	Тестировщик
30	Структурное тестирование	-
31	Покрытие операторов, решений и условий	Тестировщик
32	Расчет метрик	-
№	Задача	Ресурсы
33	Метрика Холстеда	Тестировщик
34	Метрика Чепина	Тестировщик
35	Сдача проекта	Программист
36	Оформление и сбор по документации	Программист
37	Ввод эксплуатации	Программист
38	Конец проекта	-

Для определения продолжительности задач, построения графика и расчета общих затрат по проекту было решено использовать приложение для управления проектами ProjectLibre.

ProjectLibre – это бесплатное и открытое программное обеспечение для управления проектами, которое позволяет создавать планы, распределять ресурсы по задачам, отслеживать прогресс и анализировать рабочую нагрузку. Оно также генерирует диаграммы критического пути. Расписания можно составлять с учетом используемых ресурсов, последовательность которых визуализируется в диаграмме Ганта.

Данное программное обеспечение позволяет эффективно управлять проектами, отслеживать ход работ и вносить необходимые корректировки. Использование ProjectLibre обеспечивает прозрачность и контроль над всеми этапами реализации проекта. Оно также обладает широкими возможностями для анализа и отчетности, что способствует принятию обоснованных управленческих решений. Простой и интуитивно понятный интерфейс программы делает ее доступной для пользователей с различным уровнем подготовки. Применение ProjectLibre в управлении проектами позволяет повысить эффективность и оптимизировать использование ресурсов.

Следующие программы являются аналогами этой программы:

- Microsoft Project;
- Trello;
- Jira;
- Asana.

В ProjectLibre были введены списки задач и ресурсов с их распределением, после чего программа сгенерировала диаграмму Ганта – визуальное представление графика работ с использованием соответствующих им ресурсов.

Дата начала работ была установлена на 8 апреля. Согласно полученной диаграмме, представленной на рисунке 2, окончание работ по проекту запланировано на 12 июня.

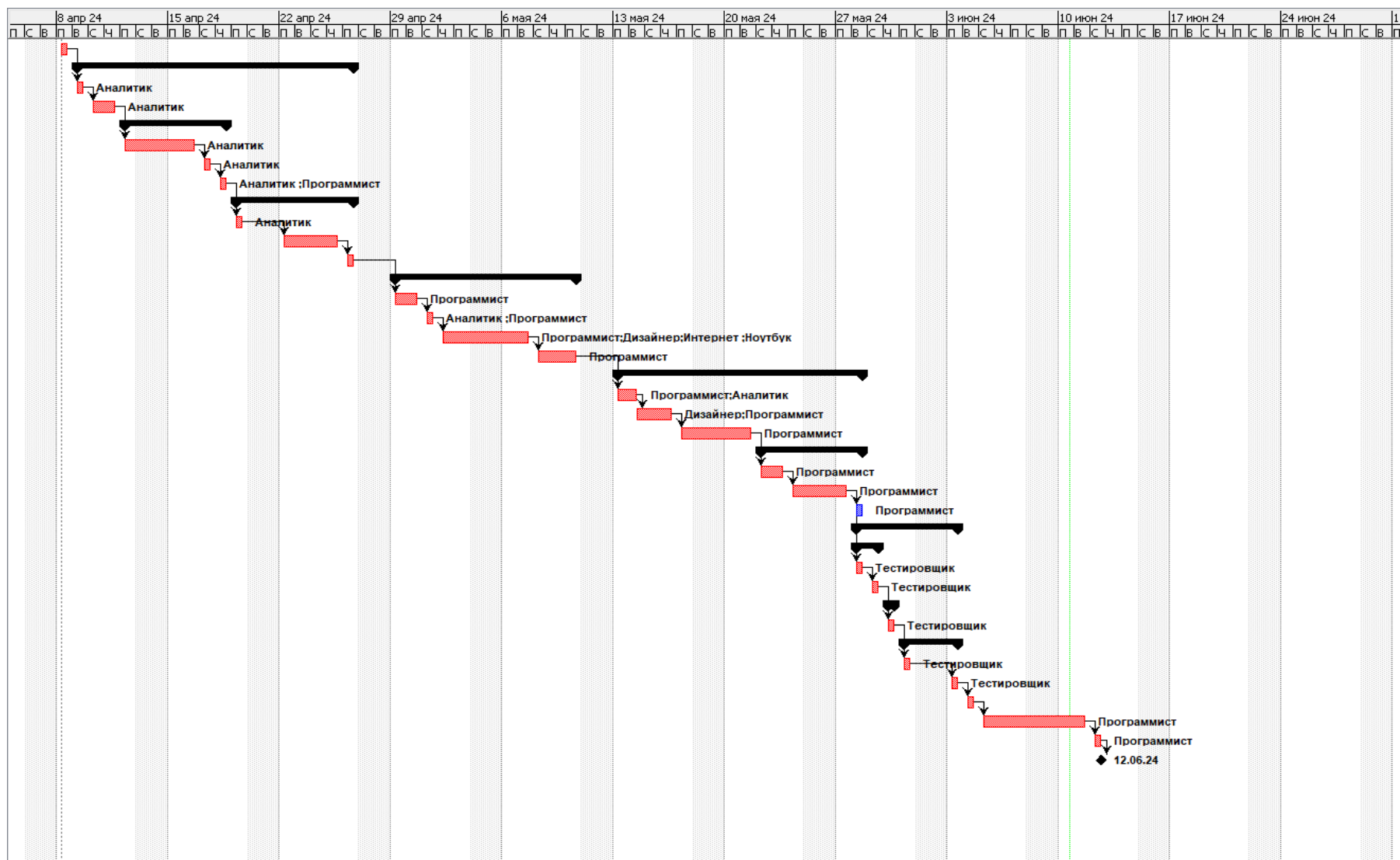


Рисунок 2 – Диаграмма Ганта

Далее, программа сгенерировала отчеты по общим затратам на проект (рисунок 3).

Dates			
Start	08.04.24 8:00	Finish	12.06.24 17:00
Baseline Start		Baseline Finish	
Actual Start		Actual Finish	

Duration			
Scheduled	48 дней	Remaining	48 дней
Baseline	0 дней	Actual	0 дней
		Percent Complete	0%

Work			
Scheduled	472 часов	Remaining	472 часов
Baseline	0 часов	Actual	0 часов

Costs			
Scheduled	359760 руб.	Remaining	359760 руб.
Baseline	0 руб.	Actual	0 руб.
		Variance	0 руб.

Рисунок 3 – Отчет по общим затратам на проект

Согласно отчету, общая занятость трудовых ресурсов составит 472 часа в период с 08.04.2023 по 12.06.2023, учитывая пятидневную рабочую неделю. На основании отчета о бюджете проекта можно сделать вывод, что для разработки клиентского приложения потребуется 359760 рублей.

2 Проектирование клиентского приложения

2.1 Моделирование функциональных возможностей приложения

В этом разделе рассмотрены функциональные возможности приложения и разработана модель, которая будет служить основой для его разработки. Модель функциональных возможностей описывает различные функции и задачи, которые приложение должно выполнять, а также их взаимосвязи и зависимости.

Цели моделирования функциональных возможностей:

- понимание полного набора функций, которые должно предоставлять приложение;
- определение взаимосвязей и зависимостей между различными функциями;
- создание основы для проектирования и разработки приложения;
- упрощение коммуникации между заинтересованными сторонами о возможностях приложения.

Существует несколько подходов к моделированию функциональных возможностей, включая:

- диаграммы вариантов использования: Описание функций приложения с точки зрения пользователей и их целей;
- диаграммы потоков данных: Визуальное представление потока данных через приложение и табличное представление функций приложения и их взаимосвязей.

Для моделирования функциональных возможностей приложения был выбран подход – диаграмма вариантов использования [3]. Диаграмма вариантов использования для приложения представлена на рисунке 4.

Данная диаграмма наглядно отображает основные функции приложения и взаимодействия пользователей с ними. Диаграмма служит основой для дальнейшего проектирования архитектуры системы и разработки ее компонентов.

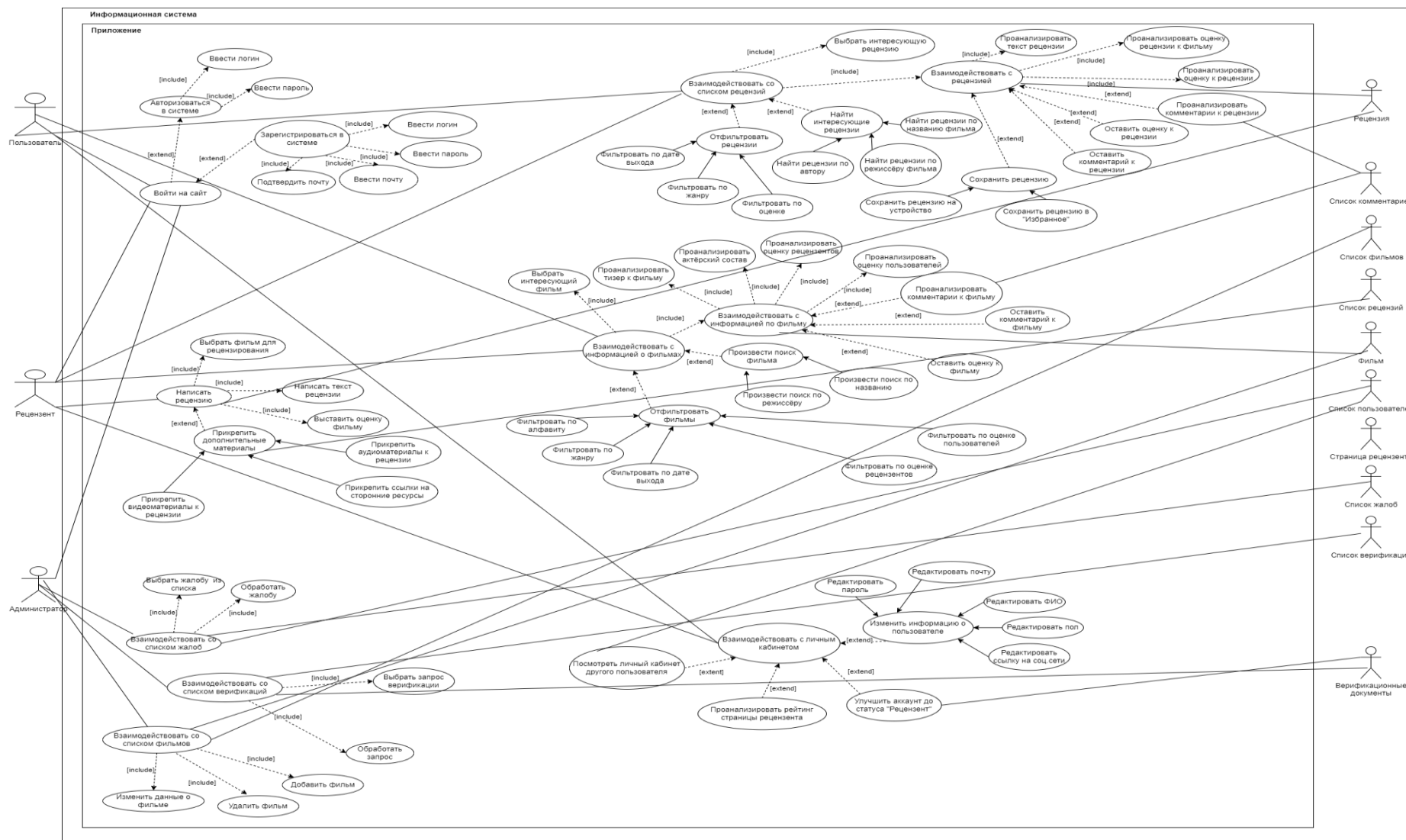


Рисунок 4 – Диаграмма вариантов использования для информационного сервиса обсуждения и оценки кинофильмов

На рисунке 5 представлен сценарий варианта использования, описывающий последовательность действий пользователя при входе на сайт.

Войти на сайт
Id:1
Описание: Пользователь входит на сайт
Действующие лица: Пользователь
Предусловие: Пользователь открывает сайт
Основной поток: 1.Пользователь хочет авторизоваться в системе: – вводит логин; – вводит пароль. Альтернативные потоки: 1а. Пользователь хочет зарегистрироваться в системе: – вводит логин; – вводит пароль; – вводит почту; – подтверждает почту. 1в.Пользователь заходит без авторизации.
Постусловие: Пользователь находится на главной странице сайта

Рисунок 5 – Сценарий варианта использования «Войти на сайт»

На рисунке 6 представлен сценарий варианта использования, описывающий последовательность действий администратора при обработке жалобы.

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 6 – Сценарий варианта использования «Взаимодействовать со списком жалоб»

На рисунке 7 представлен сценарий варианта использования, описывающий последовательность действий рецензента при написании рецензии.

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

Рисунок 7 – Сценарий варианта использования «Написать рецензию»

На рисунке 8 представлен сценарий варианта использования, описывающий последовательность действий рецензента при написании рецензии.

Взаимодействовать со списком фильмов
Id:4
Описание: Администратор добавляет/удаляет/изменяет 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 8 – Сценарий варианта использования «Взаимодействовать со списком фильмов»

На рисунке 9 представлен сценарий варианта использования, описывающий последовательность действий администратора при работе со списком верификаций.

Взаимодействовать со списком верификаций
Id:5
Описание: Администратор хочет обработать запрос с верификацией
Действующие лица: Администратор
Предусловие: Открыта страница с запросами верификаций
Основной поток: <ul style="list-style-type: none"> – выбирает запрос с верификацией; – обрабатывает запрос с верификацией;
Постусловие: <p>Администратор обработал запрос с верификацией</p>

Рисунок 9 – Сценарий варианта использования «Взаимодействовать со списком верификаций»

На рисунке 10 представлен сценарий варианта использования «Взаимодействовать с личным кабинетом».

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: <ol style="list-style-type: none"> 1. Пользователь/рецензент изменяет информацию о себе в своем личном кабинете; Альтернативные потоки: <ol style="list-style-type: none"> 1а. Улучшает аккаунт до статуса «Рецензент»; 1б. Анализирует рейтинг страницы рецензента; 1в. Смотрит личный кабинет другого;
Постусловие: Данные в личном кабинете просмотрены или отредактированы

Рисунок 10 – Сценарий варианта использования «Взаимодействовать с личным кабинетом»

На рисунке 11 представлен сценарий варианта использования «Взаимодействовать с информацией о фильмах».

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: 1. Выбирает интересующий фильм 2. Взаимодействует с информацией по фильму; Альтернативные потоки: 1а. Фильтрует фильмы; 1б. Производит поиск;
Постусловие: Переход на страницу с описанием фильма

Рисунок 11 – Сценарий варианта использования «Взаимодействовать с информацией о фильмах»

На рисунке 12 представлен сценарий варианта использования «Взаимодействовать со списком рецензий».

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: 1. Выбирает интересующую рецензию 2. Взаимодействует с рецензией; Альтернативные потоки: 1а. Фильтрует рецензии; 1б. Производит поиск;
Постусловие: Переход на страницу с описанием рецензии

Рисунок 12 – Сценарий варианта использования «Взаимодействовать со списком рецензий»

Все сценарии вариантов использования детально определяют, как пользователи будут взаимодействовать с системой. Они охватывают различные сценарии, начиная от входа на сайт до взаимодействия с различными аспектами системы, такими как добавление/удаление/изменение фильмов, формирование рецензий и обработка жалоб. Эти сценарии являются ценным инструментом для понимания функциональности системы и предоставляют уверенность в том, что основные потоки и альтернативные потоки были учтены.

2.2 Разработка концептуальной модели данных

Концептуальная модель – это определенная схема. Она в целях формирования смысловой структуры некоего объекта использует различные понятия и связи между ними. Однако стоит учитывать тот момент, что концептуальная модель системы абстрактна. Но это не единственное значение термина. Кроме того, существует понятие «концептуальная модель предметной области». Смысл данного термина состоит в том, что для описания какой-либо сферы используется перечень связанных между собой понятий. В этих целях используются классификация определений, их характеристики и свойства, а также законы происходящих в них процессов [4].

Диаграмма «сущность-связь» концептуальной модели (схемы) базы данных представлена на рисунке 13. Данная диаграмма наглядно отображает основные сущности и связи между ними в рамках рассматриваемой предметной области. Она служит основой для дальнейшего проектирования логической и физической структуры базы данных.

Концептуальная модель предметной области является важным этапом в процессе разработки информационных систем. Она позволяет четко определить и структурировать ключевые понятия, их атрибуты и взаимосвязи, что способствует лучшему пониманию предметной области. Это, в свою очередь, облегчает дальнейшее проектирование и реализацию системы.

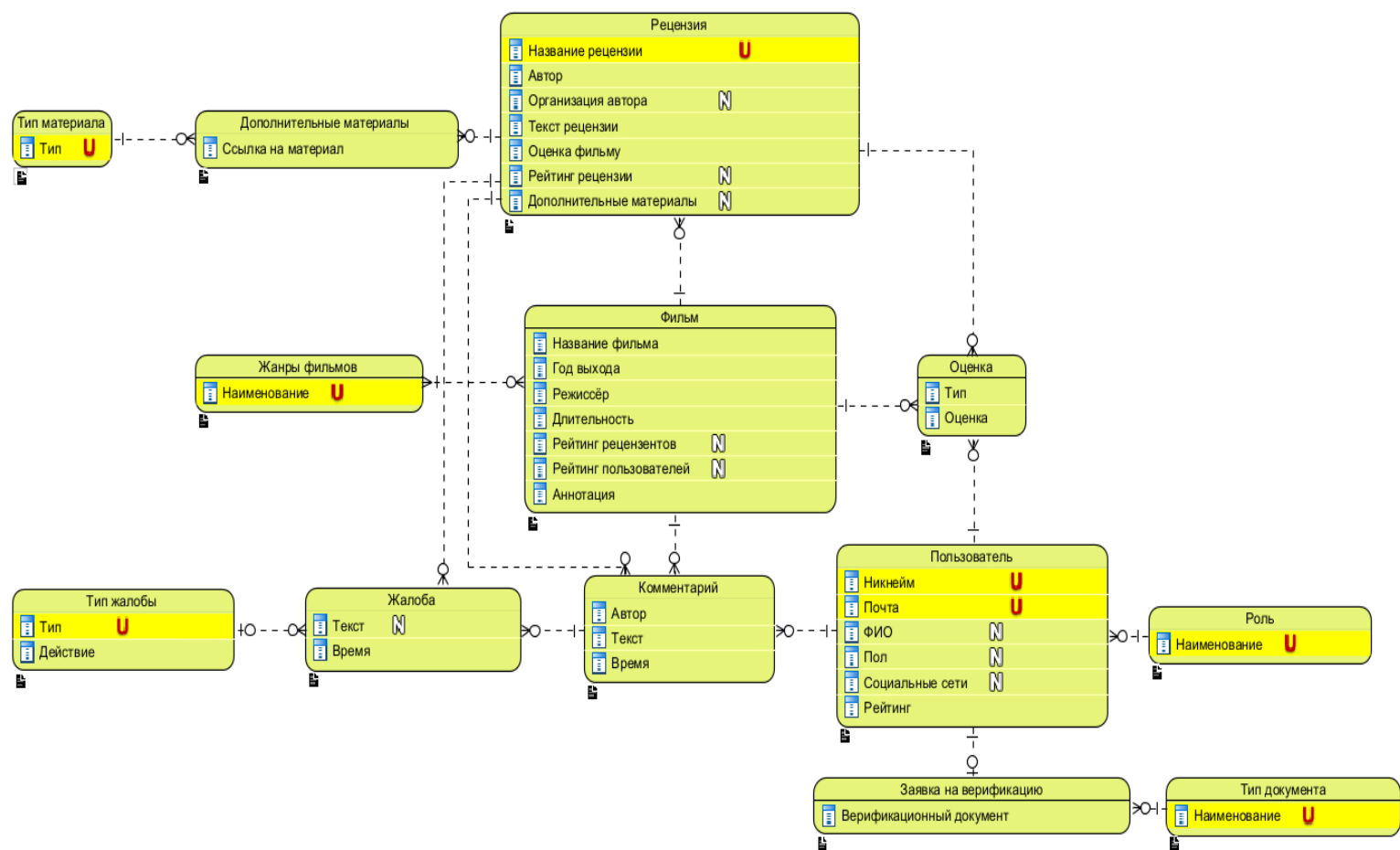


Рисунок 13 – Диаграмма «сущность-связь» концептуальной модели базы данных

Табличное описание сущностей представлено в таблице 6

Таблица 6 – Описание сущностей

№ п/п	Наименование	Описание	Категории	Количество экземпляров	Режим обновления
1	Рецензия	Список созданных рецензий	Стержневая	1000000	30% за год
2	Дополнительные материалы	Список дополнительных материалов, прикрепляемых к рецензиям	Стержневая	1000000	20% за год
3	Тип материала	Типы прикрепляемых материалов	Справочная	4	Нет
4	Фильм	Список фильмов, представленных на сайте	Стержневая	600000	15% за год
5	Жанры фильмов	Список жанров фильмов	Справочная	19	Нет
6	Оценка	Список оценок, выставленных к фильмам и рецензиям	Стержневая	1000000	30% за год
7	Комментарий	Список комментариев, оставленных к фильмам и рецензиям	Стержневая	1000000	40% за год
8	Жалоба	Список жалоб на комментарии или рецензии	Стержневая	1000000	100% за год
9	Тип жалобы	Типы жалоб и реакции на них	Справочная	4	Нет
10	Пользователь	Список пользователей, зарегистрированных на сайте	Стержневая	1000000	25% за год
11	Роль	Виды ролей пользователей	Справочная	3	Нет
12	Заявка на верификацию	Список заявок пользователей на верификацию	Стержневая	10000	3% за год
13	Тип документа	Виды подаваемых на верификацию документов	Справочная	5	Нет

Спецификации связей между сущностями представлены в таблице 7

Таблица 7 – Спецификации связей между сущностями

№ п/п	Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
1	Тип материала	Дополнительные материалы	Материал имеет определённый тип. Есть несколько материалов одного типа.	1:M	Обязательная для дополнительного материала – материал не может существовать без определённого типа Необязательная для типа материала – может не быть ни 1 материала типа	Один материал может быть лишь 1 типа
2	Дополнительные материалы	Рецензия	К рецензии может быть прикреплено несколько дополнительных материалов. Один дополнительный материал соответствует одной рецензии.	M:1	Обязательная для дополнительного материала – материал не может существовать без прикрепления к рецензии Необязательная для рецензии – рецензия может существовать без дополнительных материалов	К 1 рецензии можно прикрепить не более 10 дополнительных материалов
3	Рецензия	Фильм	К одному фильму может быть написано несколько рецензий. Одна рецензия соответствует 1 фильму	M:1	Обязательная для рецензии – рецензия не может быть написана без выбора фильма Необязательная для фильма – фильм может существовать без единой рецензии	Одной рецензии соответствует 1 фильм.
4	Рецензия	Оценка	К одной рецензии может быть выставлено несколько оценок. 1 оценка соответствует только 1 рецензии	1:M	Необязательная для рецензии – рецензия может не иметь ни 1 оценки. Необязательная для оценки – оценка может быть выставлена не рецензии	Одной оценке может соответствовать не более 1 рецензии

Продолжение таблицы 7

№ п/п	Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
5	Рецензия	Комментарий	К одной рецензии может быть написано несколько комментариев. Одному комментарию соответствует не более 1 рецензии	1:М	Необязательная для рецензии – к рецензии может быть написано ни 1 комментария Необязательная для комментария – комментарий может быть написан не только к рецензии	Одному комментарию может соответствовать не более 1 рецензии
6	Рецензия	Жалоба	К одной рецензии может быть написано несколько жалоб. Одной жалобе может соответствовать не более 1 рецензии	1:М	Необязательная для рецензии – на рецензию может быть написано ни 1 жалобы Необязательная для жалобы – жалоба может быть написана не на рецензию	Одной жалобе может соответствовать не более 1 рецензии
7	Фильм	Жанры фильмов	Один фильм может иметь несколько жанров. Одному жанру соответствует несколько фильмов	М:М	Обязательная – фильм не может существовать без жанра Необязательная – может не быть ни 1 фильма к определённому жанру	Одному фильму должен соответствовать минимум 1 жанр
8	Фильм	Оценка	Одному фильму может быть выставлено несколько оценок. Одна оценка соответствует не более 1 фильму	1:М	Необязательная – фильм может быть никем не оценён Необязательная – оценка может быть выставлена не фильму	Оценке может соответствовать не более 1 фильма

Продолжение таблицы 7

№ п/п	Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
9	Фильм	Комментарий	К одному фильму может быть написано несколько комментариев. Одному комментарию соответствует не более 1 фильма	1:M	Необязательная для фильма – к фильму может быть написано ни одного комментария Необязательная для комментария – комментарий может быть написан не к фильму	Один комментарий может быть написан не более чем к 1 фильму
10	Оценка	Пользователь	Одна оценка может быть выставлена лишь 1 пользователем. Пользователь может выставить несколько оценок	M:1	Обязательная для оценки – оценка обязательно должна иметь оценивающего Необязательная для пользователя – пользователь может не выставить ни одной оценки	У оценки должен обязательно быть ровно 1 автор
11	Жалоба	Комментарий	Жалоба может соответствовать не более одного комментария. К одному комментарию может быть написано несколько жалоб	M:1	Необязательная для жалобы – жалоба может быть написана не на комментарий Необязательная для комментария – на комментарий может быть ни одной жалобы	Жалоба должна соответствовать не более чем к одному комментарию
12	Жалоба	Тип жалобы	Жалоба может быть лишь одного типа. Одному типу может соответствовать несколько жалоб	M:1	Обязательная для жалобы – жалоба не может не иметь типа Необязательная для типа жалобы – типу жалобы может не соответствовать ни одна из жалоб	Жалоба может быть лишь одного типа

Продолжение таблицы 7

№ п/п	Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
13	Комментарий	Пользователь	Комментарий может быть оставлен лишь 1 пользователем. Одним пользователем может быть написано несколько комментариев	M:1	Обязательная для комментария – комментарий обязан иметь комментирующего Необязательная для пользователя – пользователь может не оставить ни 1 комментария	Одному комментарию должен соответствовать ровно один пользователь
14	Пользователь	Роль	Один пользователь может иметь лишь одну роль. Одной роли может соответствовать несколько пользователей	M:1	Обязательная для пользователя – пользователь не может существовать без роли Необязательная для роли – роли может соответствовать ни одного пользователя	Пользователь обязан иметь одну роль
15	Пользователь	Заявка на верификацию	Пользователь может иметь несколько заявок. Одной заявке соответствует ровно один пользователь	1:M	Необязательная для пользователя – пользователь может не оставить ни одной заявки на верификацию Обязательная для заявки на верификацию – заявка обязана иметь заявителя	Заявка обязана иметь ровно одного заявителя. Пользователем может быть отправлено не более одной заявки в день

Окончание таблицы 7

№ п/п	Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
16	Заявка на верификацию	Тип документа	Одной заявке соответствует один тип документа. Одному типу документа может соответствовать несколько заявок на верификацию	М:1	Обязательная для заявки на верификацию – заявка обязана иметь определённый тип документа Необязательная для типа документа – одному типу может соответствовать ни одной заявки	Одной заявке соответствует один тип документа.

В ходе раздела, посвященного разработке концептуальной модели базы данных с использованием средств Visual Paradigm, мы смогли охватить широкий спектр действий – начиная от определения сущностей и их атрибутов до установления связей между ними. Visual Paradigm предоставил удобный и интуитивно понятный интерфейс для проектирования моделей баз данных, позволяя нам визуализировать и улучшить понимание структуры данных, а также отношений между различными сущностями.

2.3 Проектирование пользовательского интерфейса

Проектирование и дизайн пользовательского интерфейса являются неотъемлемой частью создания любого современного web-приложения. Качественно разработанный интерфейс не только обеспечивает удобство использования продукта, но и повышает его конверсию и уровень удовлетворенности пользователей [5].

Работа платформы-агрегатора начинается с главной страницы, где представлены все отзывы. В шапке платформы расположены навигация по платформе, поле ввода для поиска, фильтрация отзывов и личный кабинет пользователя. Главная страница платформы представлена на рисунке 14.

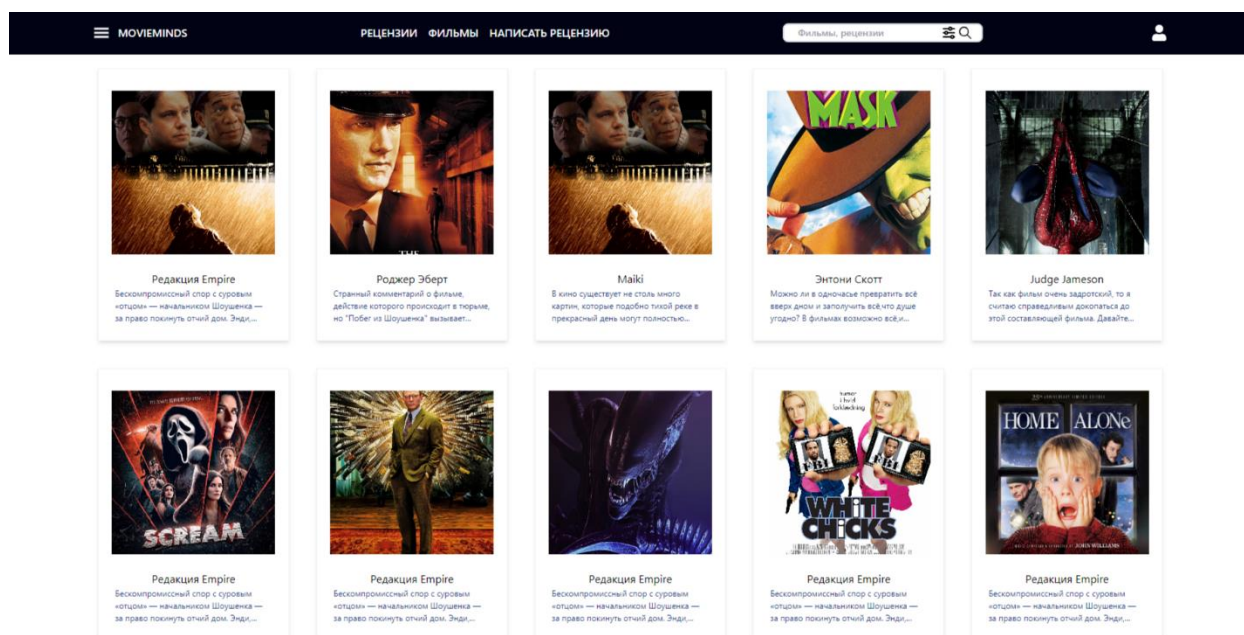


Рисунок 14 – Главная страница платформы

При нажатии на элемент с рецензией про исходит переход на страницу с подробным описанием этой рецензии. Данная страница представлена на рисунке 15.

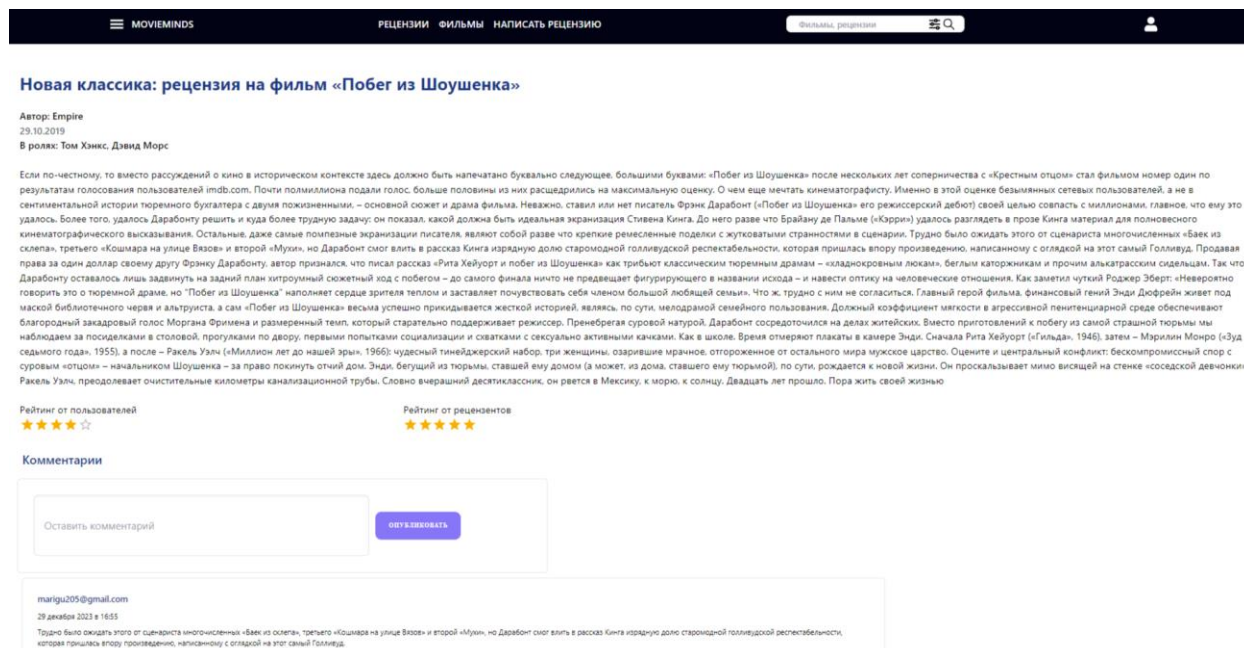


Рисунок 15 – Страница описания рецензии

Навигационный блок состоит из трех элементов: страницы списка рецензий, страница со списком фильмов и страницы для написания рецензий – «Написать рецензию». Последний элемент навигационного блока доступен

только для пользователя, роль которого является «рецензентом». Страницы «Фильмы» и «Написать рецензию» представлены на рисунках 16 и 19.

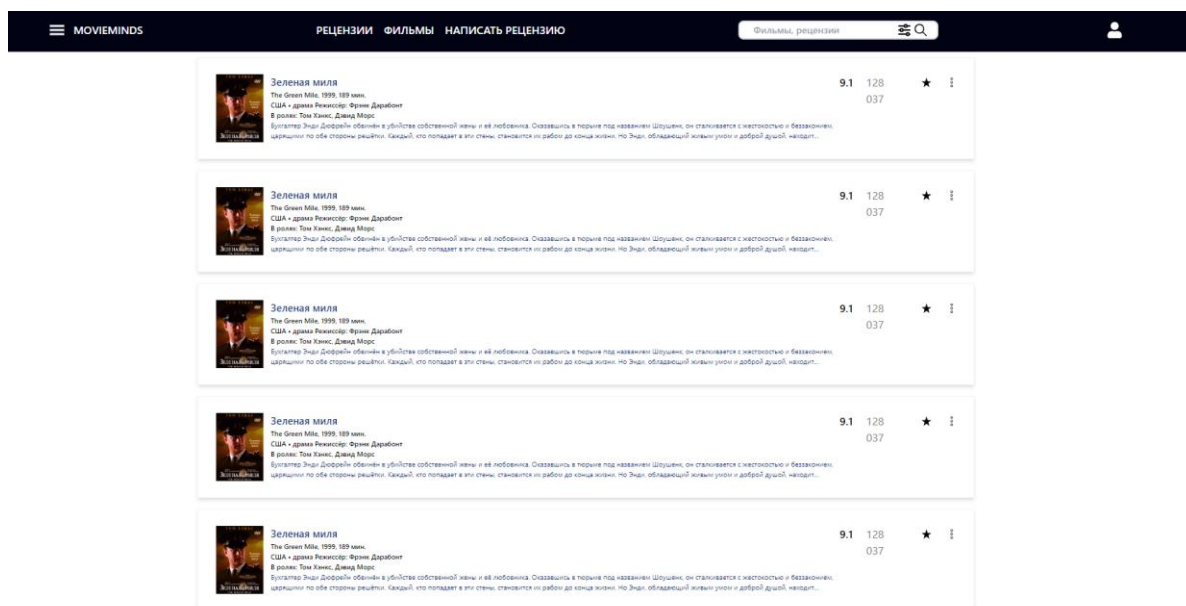


Рисунок 16 – Страница «Фильмы»

Блок с фильмами состоит из названия, краткого описания, страны-производителя, актеров, длительности фильма, просмотров и оценки от пользователей. При нажатии на фильм происходит переход на страницу описания фильма. Данная страница представлена на рисунках 17 и 18.

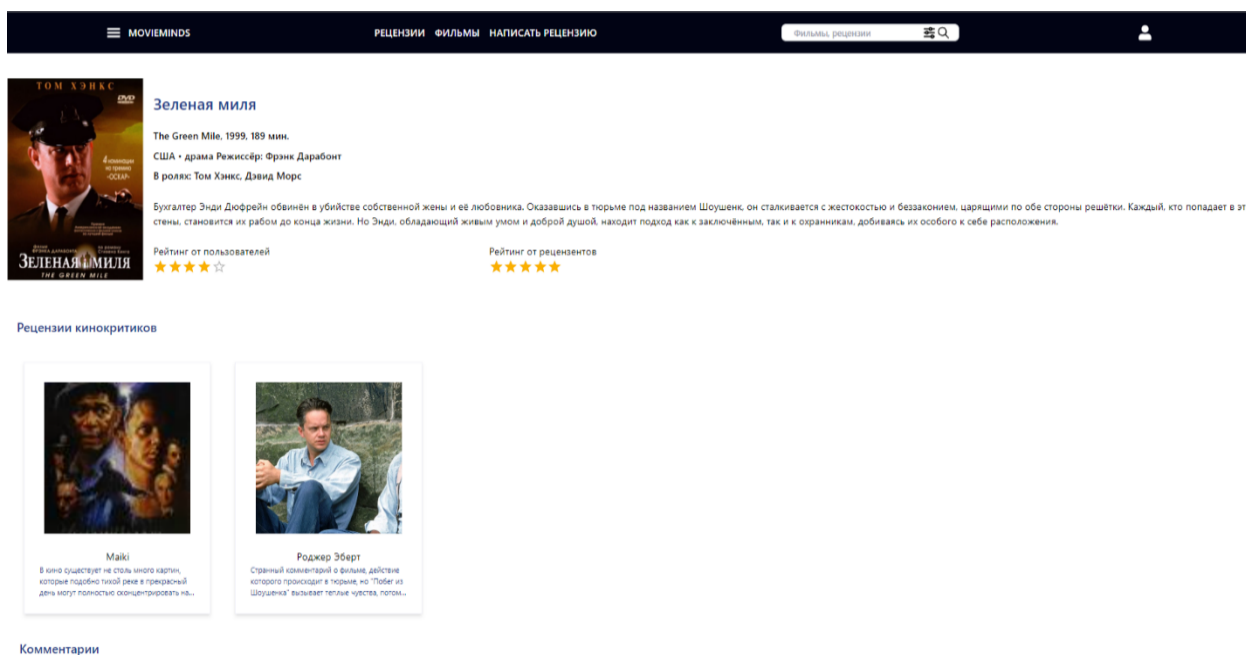


Рисунок 17 – Страница с описанием фильма

Страница с описанием фильма содержит всю необходимую информацию для пользователя: подробное описание фильма, блок с рецензиями по данному фильму от профессиональных рецензентов.

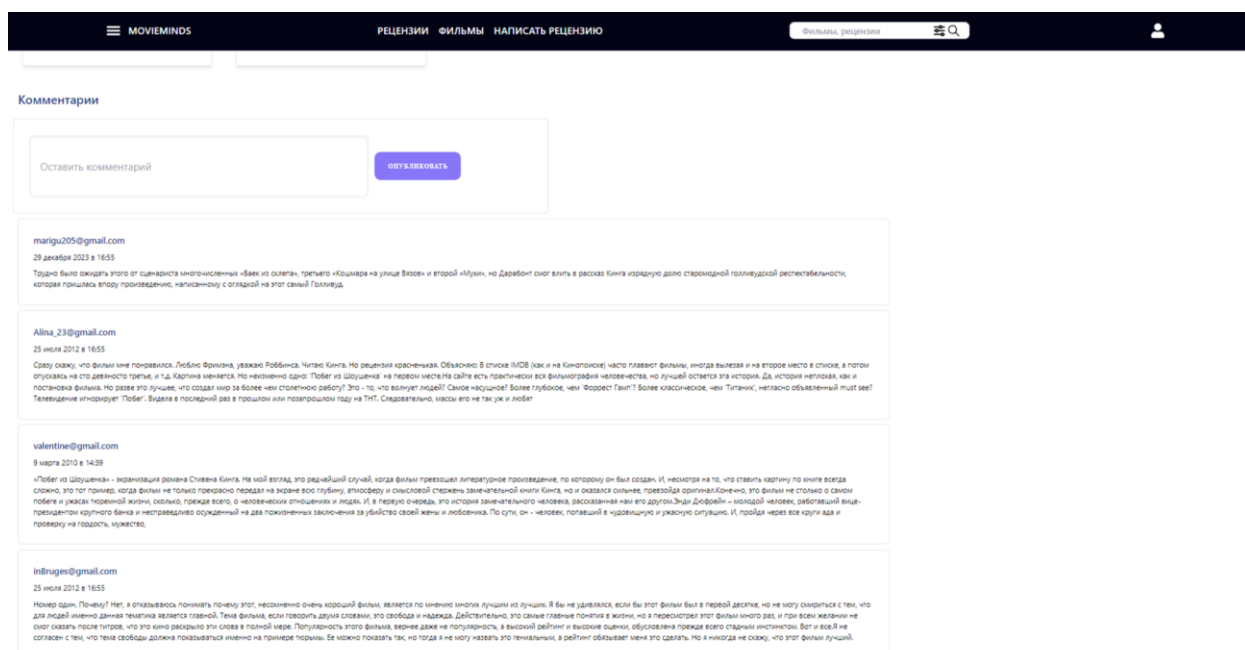


Рисунок 18 – Продолжение страницы с описанием фильма

На странице с описанием фильма также предусмотрен раздел для комментариев, где пользователи могут выразить свое мнение и ознакомиться с отзывами других пользователей.

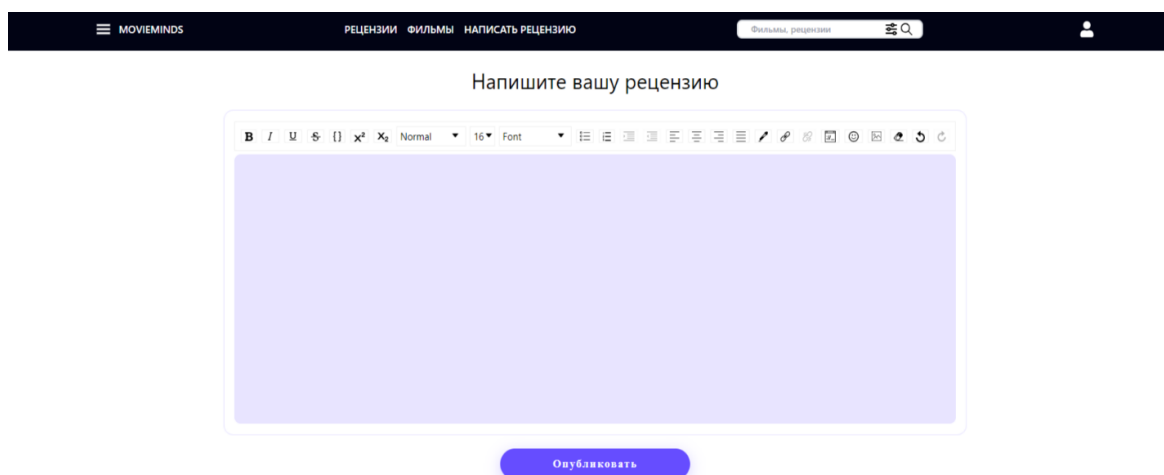


Рисунок 19 – Страница «Написать рецензию»

Данная страница состоит из двух пользовательских компонентов: редактора рецензии и кнопки «Опубликовать», при нажатии которой происходит создание рецензии.

Для того, чтобы пользователю стать рецензентом, платформа предлагает удобный личный кабинет, где он может отправить запрос на роль рецензента. Если пользователь не авторизован, то система перенаправит на страницу регистрации/авторизации. Страницы регистрации/авторизации представлены на рисунках 20 и 21, а страница личного кабинета представлена на рисунке 22.

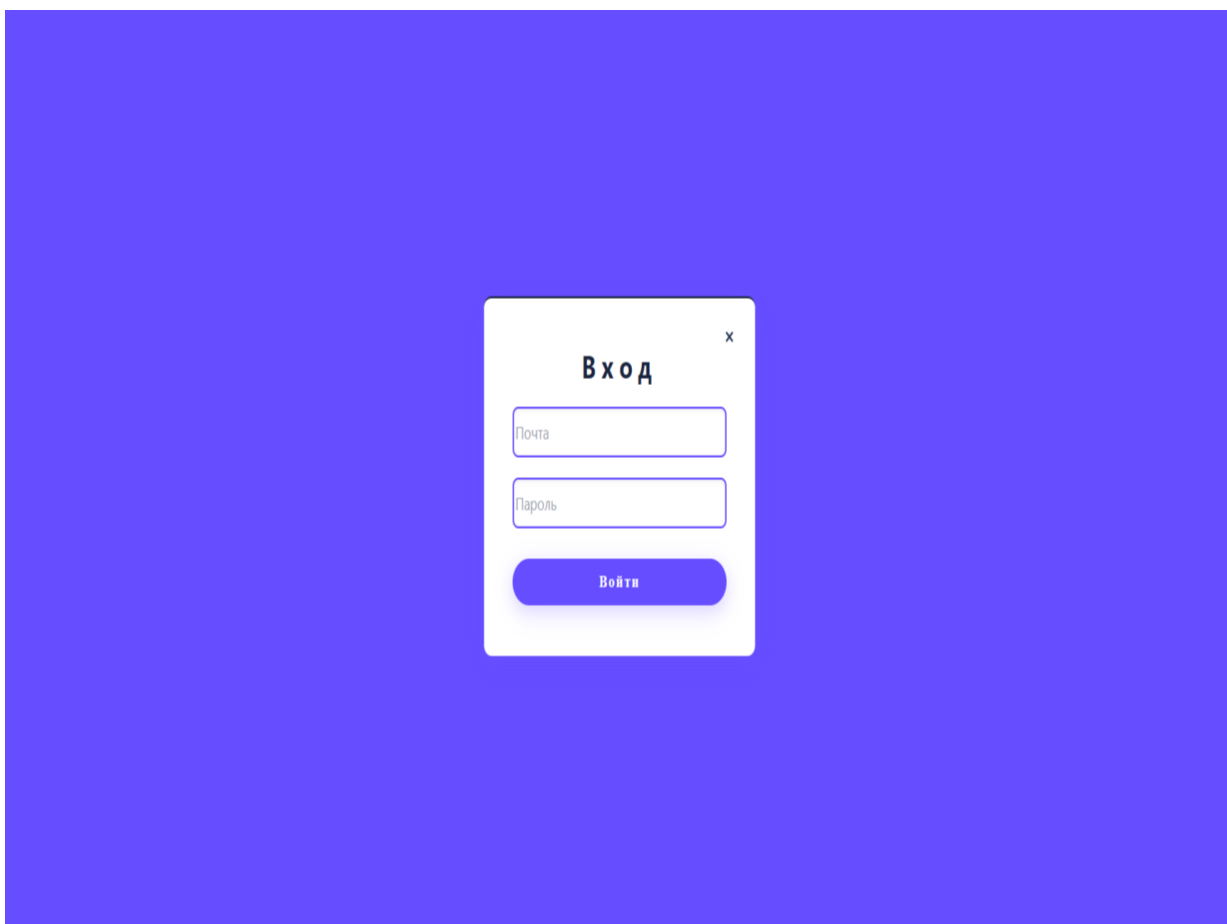


Рисунок 20 – Страница входа на платформу

Страница входа в систему является точкой входа на платформу, которая позволяет пользователям идентифицировать себя и получить доступ к своим учетным записям. Пользователи могут входить в систему, используя адрес электронной почты и пароль.

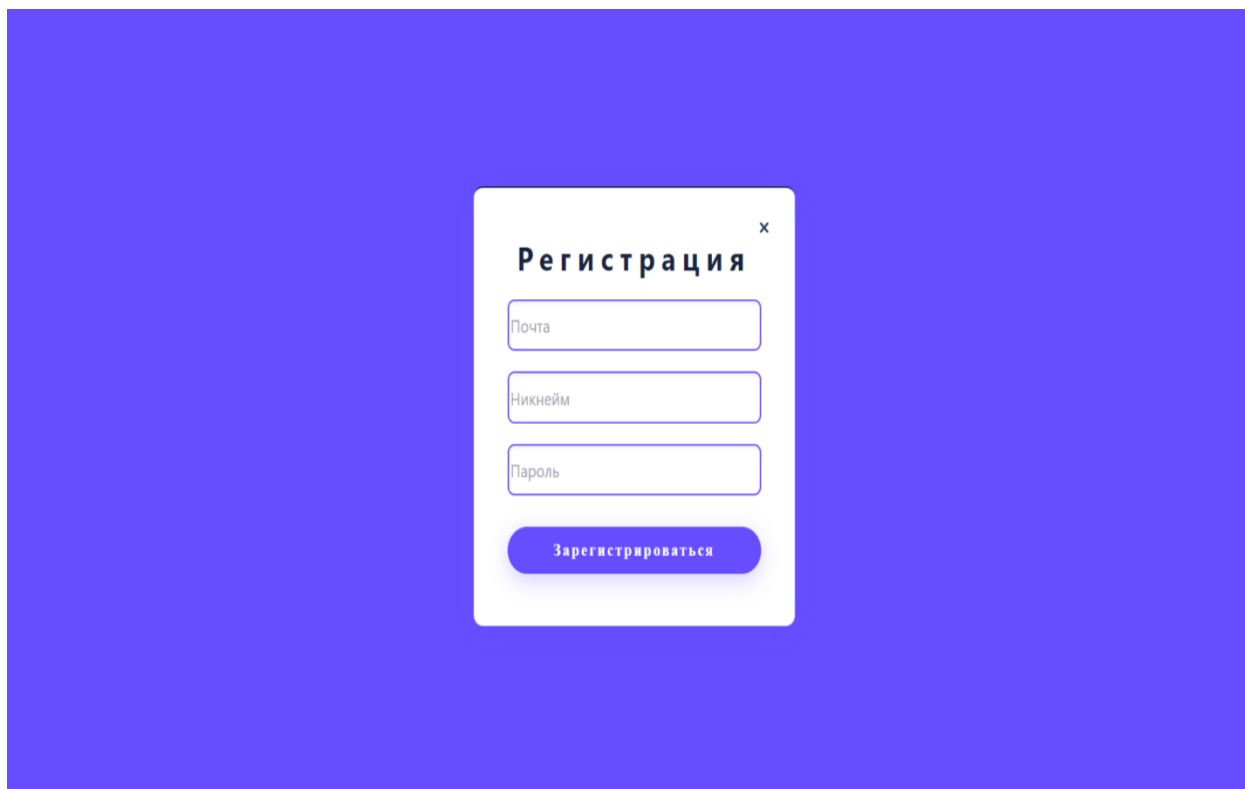


Рисунок 21 – Страница регистрации

Страница регистрации позволяет новым пользователям создавать учетные записи на платформе и получать доступ к ее функциям и сервисам. Данная страница разработана таким образом, чтобы быть понятной и удобной для пользователей всех уровней технических знаний.

После успешной регистрации/авторизации система перенаправляет пользователя на главную страницу пользователя.

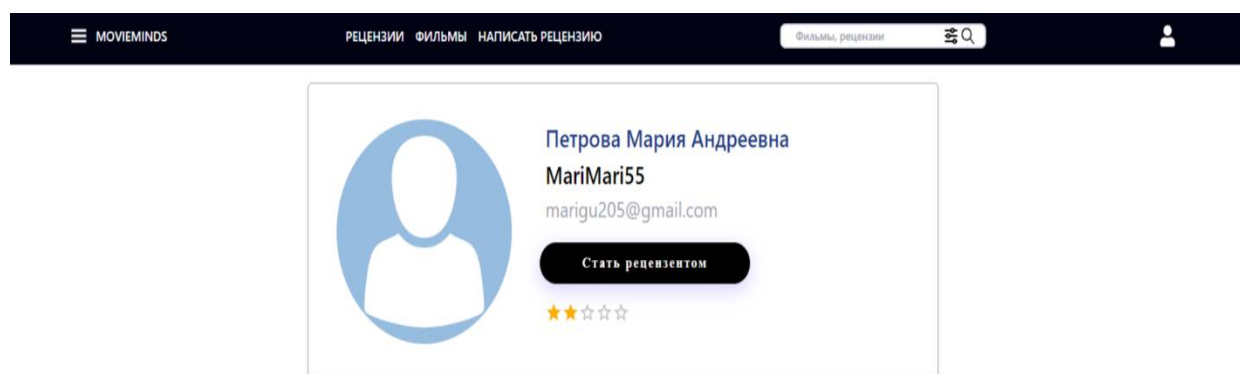


Рисунок 22 – Личный кабинет пользователя

Личный кабинет пользователя на платформе предоставляет пользователям персонализированный и удобный способ управления своими учетными записями. При нажатии на кнопку «Стать рецензентом» отправляется запрос на роль рецензента. После отправки запроса пользователи обычно должны пройти процесс проверки, который может включать оценку их опыта, квалификации и соответствия критериям платформы для рецензентов.

Также для платформы спроектирована админ-панель. Админ-панель на платформе. Админ-панель на платформе предоставляет администраторам централизованный интерфейс для управления различными аспектами платформы. Вот обзор возможностей админ-панели:

- управление пользователями: администраторы могут блокировать пользователей, назначать роли и разрешения, а также отслеживать активность пользователей;
- модерация контента: админ-панель позволяет администраторам модерировать пользовательский контент, такой как публикации, комментарии и загрузки, администраторы могут удалять, скрывать или помечать контент, который нарушает правила платформы или не соответствует ее стандартам;
- аналитика и отчетность: админ-панель предоставляет администраторам доступ к аналитическим данным об использовании платформы и активности пользователей, они могут создавать отчеты для отслеживания показателей, выявления тенденций и принятия обоснованных решений;
- управление ролями и разрешениями: администраторы могут создавать и управлять пользовательскими ролями с различными уровнями доступа и разрешениями, это позволяет им делегировать задачи и обязанности другим членам команды.

Админ-панель обычно доступна только авторизованным администраторам с соответствующими разрешениями. Доступ к админ-панели может быть защищен паролем или другими мерами безопасности для предотвращения несанкционированного доступа.

Страница админ-панели для просмотра рейтинга плохих представлена на рисунке 23.

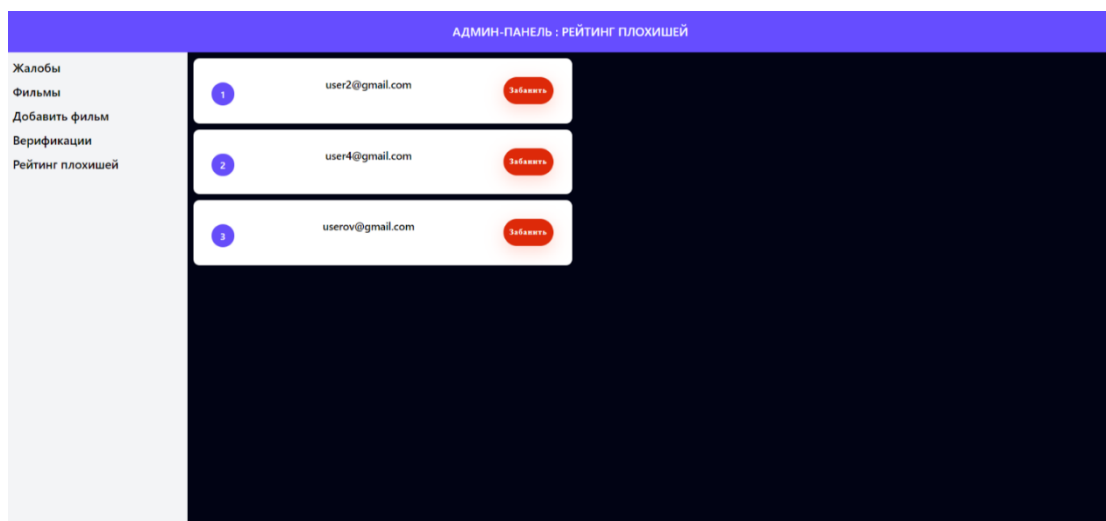


Рисунок 23 – Страница админ-панели: рейтинг плохих

Страница «рейтинг плохих» в админ-панели платформы представляет собой список пользователей, которые получили достаточное количество жалоб от других пользователей или нарушили правила платформы. Эта функция предоставляет администраторам возможность отслеживать проблемных пользователей и принимать соответствующие меры, такие как бан.

Страница для работы со списком верификаций представлена на рисунке 24.

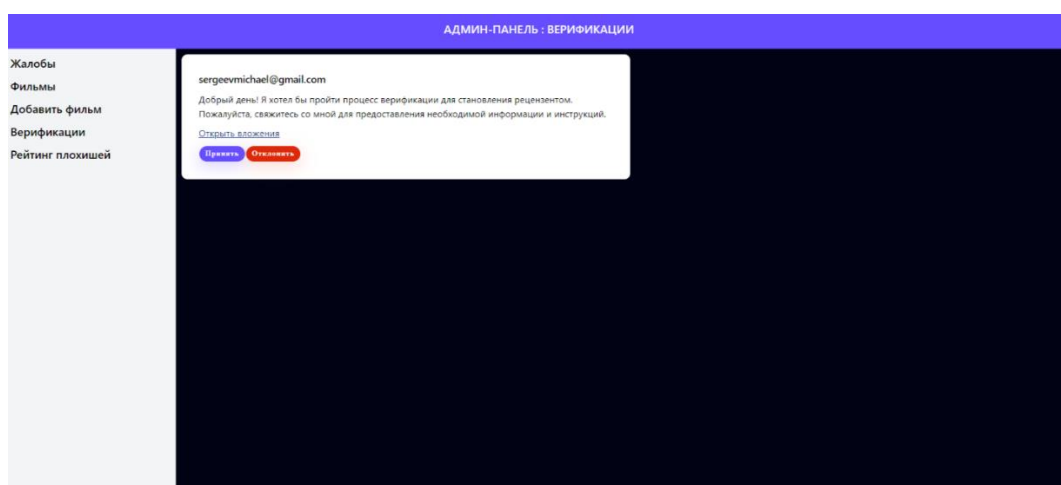


Рисунок 24 – Страница админ-панели: верификация

Страница верификаций в админ-панели платформы позволяет администраторам просматривать и управлять запросами пользователей на получение статуса рецензента. Пользователи могут подавать запросы на получение статуса рецензента, предоставив ссылку на документы, подтверждающие их опыт, квалификацию и соответствие критериям платформы.

Страница для работы с фильмами представлена на рисунке 25.

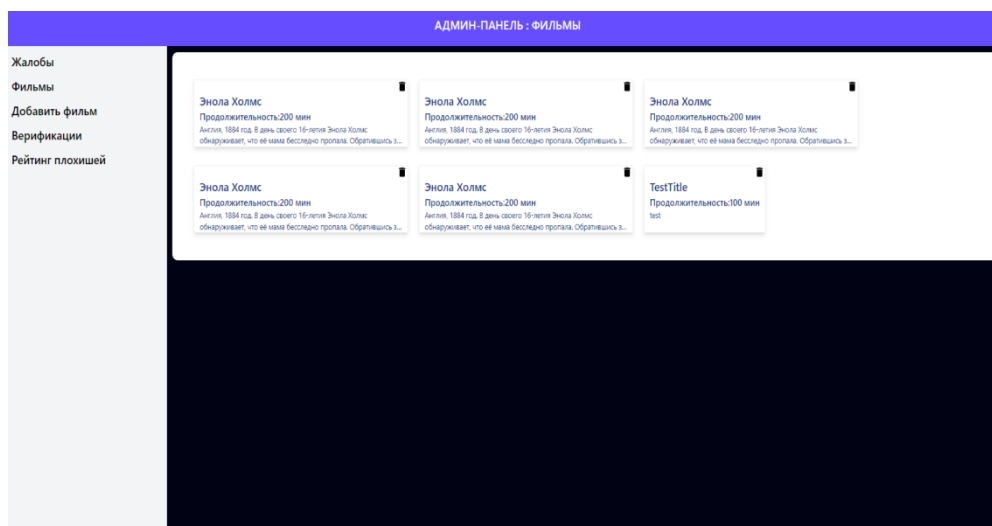


Рисунок 25 – Страница админ-панели: фильмы

Страница фильмов в админ-панели предоставляет администраторам полный список фильмов, которые в настоящее время доступны на платформе.

Страница для добавления фильма представлена на рисунке 26.

Рисунок 26 – Страница админ-панели: добавить фильм

Страница добавления фильмов в админ-панели позволяет администраторам добавлять новые фильмы в каталог платформы. Эта страница предоставляет все необходимые поля и инструменты для создания подробного описания фильма с точной и актуальной информацией.

Страница для работы с жалобами представлена на рисунке 27.

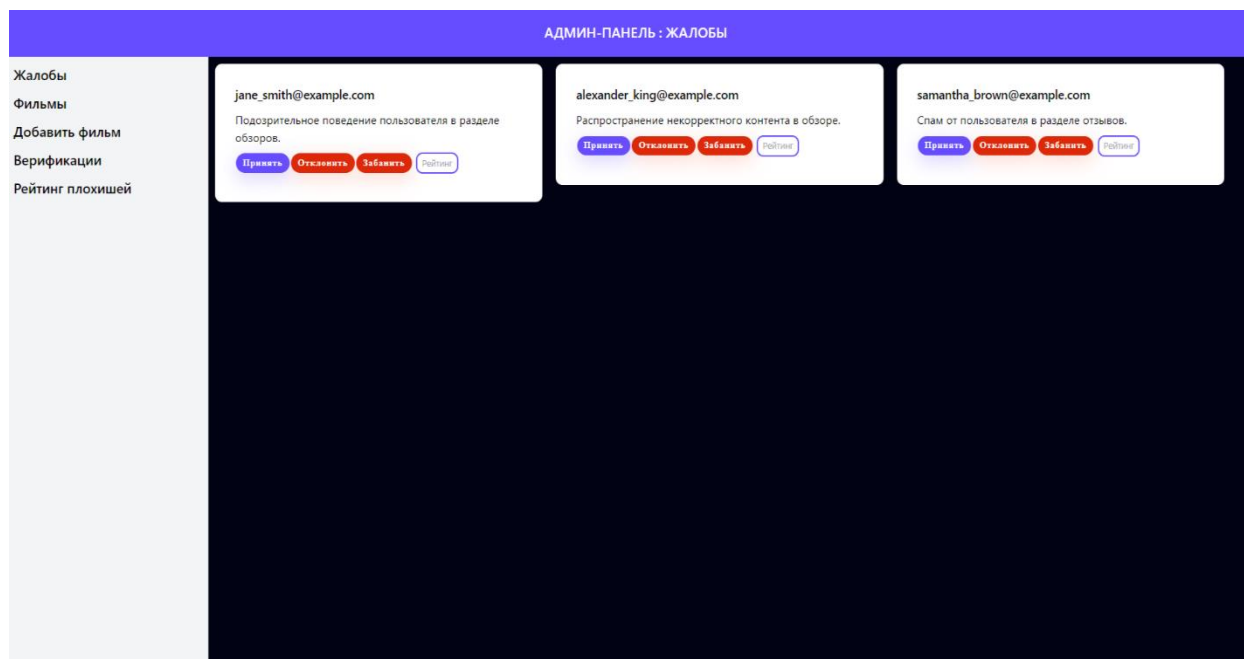


Рисунок 27 – Страница админ-панели: жалобы

Страница жалоб в админ-панели предоставляет администраторам централизованный обзор всех жалоб, поданных пользователями на рецензентов или других пользователей. Эта страница позволяет администраторам расследовать жалобы, принимать соответствующие меры и поддерживать безопасную и здоровую среду на платформе.

В разделе проектирования пользовательского интерфейса платформы подробно рассмотрены страницы и функциональные возможности, которые будут реализованы в будущей системе.

Шаблоны страниц, представленные в этом разделе, служат ориентиром для дальнейшей разработки и реализации. Они обеспечивают основу для создания целостного и привлекательного пользовательского интерфейса, который будет соответствовать потребностям наших пользователей и целям платформы.

2.4 Выбор технологий и средств разработки клиентского приложения

В связи с выбором Javascript в качестве основного языка программирования для разработки пользовательского интерфейса, возникла задача выбора наиболее подходящего фреймворка для реализации проекта.

Для того, чтобы выяснить какие фреймворки приоритетны в России, обратимся к сервису Google Trends.

Сервис Google Trends позволяет анализировать запросы пользователя по частоте, относительно любого региона мира. Данные не являются приоритетными, так как запросы могут происходить по разным причинам. Например, разработчик, который ищет ответы на вопросы относительно используемого фреймворка в текущий момент. Однако, такая сравнительная характеристика может показать интересные данные.

На рисунке 28 представлены данные поисковых запросов в России:

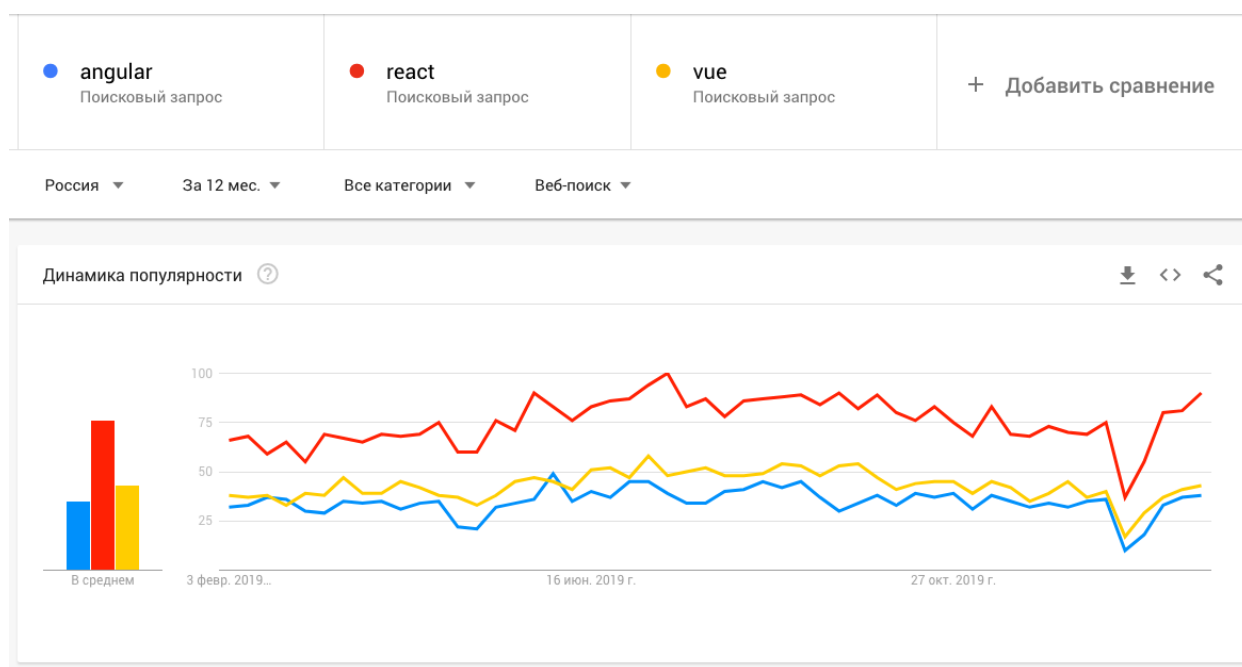


Рисунок 28 – Данные поисковых запросов в России

Среди трех вышеупомянутых фреймворков, React имеет большое количество преимуществ:

- высокая производительность;

- компонентная архитектура, способствующее повторному использованию кода и созданию модульных приложений;
- огромное и активное сообщество разработчиков, предоставляющее множество библиотек и компонентов;
- поддержка от Facebook, обеспечивающая стабильность и надежность фреймворка;
- гибкость в интеграции с различными технологиями и платформами, включая React Native.

Для разработки клиентского приложения был выбран редактор кода VSCode, разработка осуществляется под управлением операционной системы Windows 10, запуск и просмотр разрабатываемого приложения – в браузере Chrome.

3 Разработка клиентского приложения

3.1 Разработка архитектуры клиентского приложения на основе шаблона MVC

MVC – это шаблон проектирования, который разделяет логику приложения на три части:

- модель: серверная часть, содержащая всю логику данных;
- представление: внешний интерфейс или графический пользовательский интерфейс (GUI);
- контроллер: «Мозг» приложения, который управляет отображением данных.

Изоляция выполняется для распределения конкретных обязанностей для каждой части [6]. На рисунке 29 представлена схема работы шаблона проектирования MVC.

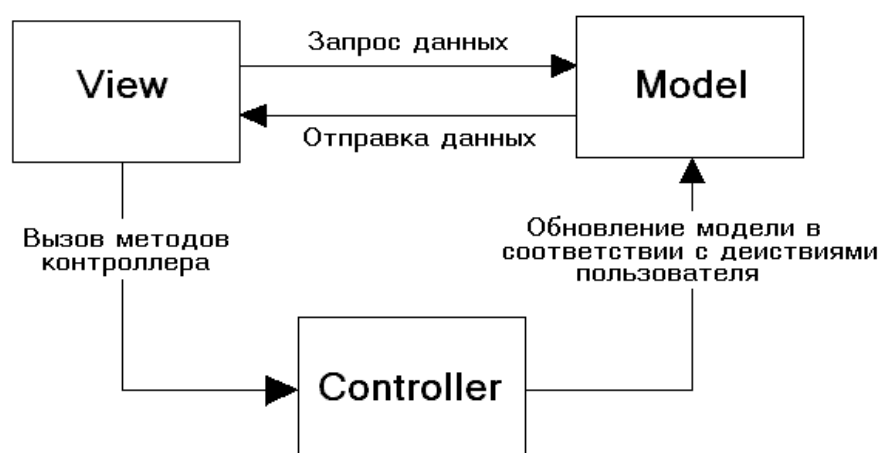


Рисунок 29 – Схема работы шаблона проектирования MVC

В архитектуре React, Redux и Redux-toolkit выполняют роли, соответствующие шаблону проектирования MVC.

Представление (View) – React используется в качестве компонента представления в архитектуре MVC. Он отвечает за создание и отображение пользовательского интерфейса (GUI) приложения на основе состояния модели (Redux). React позволяет разработчикам создавать динамические и

интерактивные компоненты пользовательского интерфейса, используя декларативный синтаксис.

Модель (Model) – Redux выполняет роль модели в архитектуре MVC. Он отвечает за управление состоянием приложения. Redux хранит централизованное состояние, к которому может обращаться любая часть приложения. Изменения состояния с помощью функций-редукторов запускают перерисовку соответствующих компонентов React.

Контроллер – (Controller): Redux Toolkit, расширение для Redux, предоставляет набор вспомогательных функций и инструментов для управления состоянием приложения. Он упрощает разработку функций-редукторов, действий и селекторов, которые являются основными компонентами архитектуры Redux. Redux Toolkit также включает в себя полезные паттерны, такие как состояния срезов (slices), которые позволяют разбивать состояние приложения на более управляемые модули.

Это разделение логики приложения на три части позволяет изолировать различные аспекты приложения и способствует разработке более управляемого и поддерживаемого кода.

Используемая структура файловой системы проекта MVC представлена на рисунке 30.

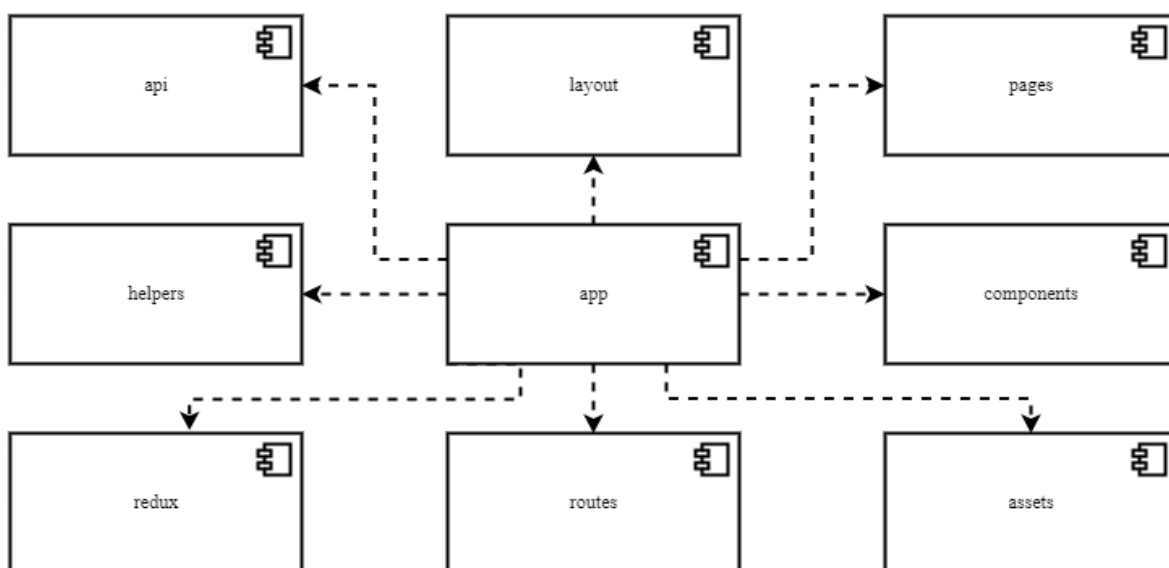


Рисунок 30 – Структура файловой системы проекта MVC

Подробно опишем структуру файловой системы проекта:

- `redux` содержит файлы контроллера, которые обрабатывают запросы, манипулируют данными и выбирают соответствующие представления для отображения; каждый файл контроллера должен соответствовать одному или нескольким действиям или состояниям приложения;
- `components` содержит коллекцию компонентов пользовательского интерфейса, таких как `button`, `custom input field`, `modal`, которые будут общими и использоваться во всех файлах проекта;
- `pages` содержит компоненты `React`, представляющие отдельные страницы или маршруты приложения; каждый компонент страницы отвечает за отображение определенного экрана или части пользовательского интерфейса;
- `layout` содержит различные макеты страниц приложения;
- `api` содержит методы, которые отвечают за выполнение запросов `API` и обмен данными с сервером;
- `helpers` содержит общие функции, классы и утилиты, которые могут использоваться несколькими частями приложения, помогает поддерживать `DRY`-код (не повторяться) и централизовать общую функциональность;
- `routes` содержит файлы маршрутизации, которые определяют соответствия между `URL`-адресами и действиями контроллера; помогает централизовать и организовать маршруты приложения;
- `assets` содержит статические файлы, такие как изображения, шрифты, которые доступны публично; помогает отделить статические файлы от файлов приложения.

Эта структура файловой системы предоставляет логичную и упорядоченную организацию для проекта `MVC`, что упрощает навигацию и обслуживание.

3.2 Разработка пользовательского интерфейса клиентского приложения

Для разработки пользовательского интерфейса была выбрана библиотека TailwindCSS. Она позволяет использовать набор готовых классов утилит, которые могут быть скомбинированы для создания гибких и адаптивных макетов. TailwindCSS значительно упрощает разработку интерфейса, предоставляя следующие преимущества:

- модульность: классы утилит можно применять к различным элементам, что позволяет легко поддерживать и повторно использовать стили;
- атомарность: каждый класс утилиты предназначен для одной конкретной цели, что обеспечивает большую гибкость и контроль над стилем;
- реактивность: классы утилит автоматически реагируют на изменения в состоянии приложения, что позволяет создавать динамические интерфейсы;
- экосистема: TailwindCSS имеет активную экосистему плагинов и расширений, что расширяет ее возможности и позволяет использовать инструменты других сообществ.

Использование TailwindCSS позволяет создать четкую иерархию классов, что делает код более читабельным и понятным. Кроме того, TailwindCSS использует систему сетки на основе flexbox, которая обеспечивает гибкую и адаптивную раскладку интерфейсов [7].

Пример стилизации компонента с использованием Tailwind CSS приведен в листинге 1.

Листинг 1 – Стилизация компонента с использованием TailwindCSS

```
/* eslint-disable react/prop-types */
import { Button } from "../button";
import { Input } from "../input";

export const ReportCard = ({text,user='user@gmail.com'}) => {
  return (
    <div className='h-[15%] w-[31%] bg-white rounded-lg p-6 m-2'>
      <div className='text-[14px] font-semibold'>{user}</div>
      <div className='text-[13px] mt-2'>{text}</div>
      <div className="flex mt-2">
```

```

        <Button name="Принять" className="btn-admin btn-admin-
primary" />
        <Button name="Отклонить" className="btn-admin btn-admin-
red"/>
        <Button name="Забанить" className="btn-admin btn-admin-
red"/>
        <Input placeholder="Рейтинг" style="w50"/>
      </div>
      <div className="flex mt-2">
      </div>
    </div>
  );
};

```

Компонент `ReportCard` – это React-компонент, используемый для отображения карточек с жалобами от пользователей. Он принимает два свойства: `text` (текст отчета) и `user` (имя пользователя, отправившего жалобу).

3.3 Разработка средств взаимодействия с серверным приложением

Для отправки запросов на сервер было решено использовать `axios`.

`Axios` – это библиотека JavaScript с открытым исходным кодом для совершения запросов к HTTP-конечным точкам REST API. Она предоставляет простой и универсальный способ взаимодействия с серверным приложением [8].

Для того, чтобы начать отправлять запросы, необходимо настроить `axios` с помощью следующих параметров:

- `baseUrl`: базовый URL-адрес вашего серверного приложения;
- `timeout`: время ожидания в миллисекундах;
- `headers`: заголовки по умолчанию для всех запросов;
- `auth`: конфигурация аутентификации, если требуется.

Пример настройки `axios` в проекте представлен в листинге 2.

Листинг 2 – Настройка `axios`

```

const instance = axios.create({
  baseUrl: process.env.API,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
  auth: {
    username: process.env.user,
    password: process.env.password,
  }, });

```

После создания экземпляра `Axios` с параметрами аутентификации его можно использовать для отправки запросов, как показано в листинге 3 – получения пользователей.

Листинг 3 – Создание экземпляра `axios`

```
instance.get('/users')
  .then((response) => {
    return response.data;
  })
  .catch((error) => {
    throw new Error(error);});
```

В проекте методы, предназначенные для отправки запросов, хранятся в директории `api`. Структура директории `api` представлена на рисунке 31:

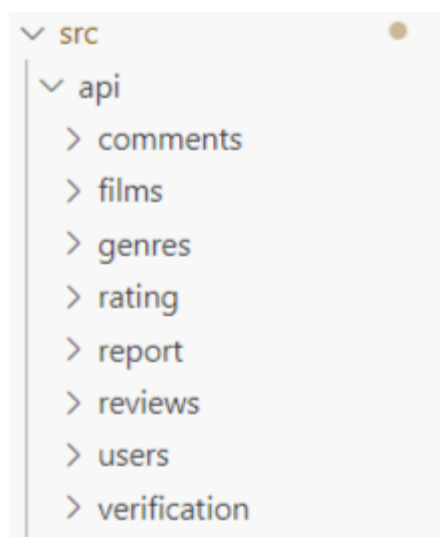


Рисунок 31 – Структура директории `api`

Перечисленные выше директории включают методы для отправки HTTP-запросов к серверной части приложения. Однако некоторые из них имеют ограниченный доступ. Например, метод `verification` предназначен для административных действий. Обычные авторизованные пользователи не могут управлять запросами на получение статуса рецензента.

3.4 Разработка бизнес-логики клиентского приложения

3.4.1 Управление состоянием

Управление состоянием веб-приложения с помощью `Redux-Toolkit` предоставляет ряд преимуществ таких, как: неизменяемость состояния, что

делает код более надежным и понятным; использование типизации, которая помогает избежать ошибок; встроенная поддержка асинхронных операций.

Файлы, содержащие данные о состоянии приложения, размещены в каталоге «redux». Использование отдельного файла для каждого фрагмента состояния привело к созданию архитектуры с высокой степенью абстракции и модульности. Структура каталога проекта «redux» представлена на рисунке 32.

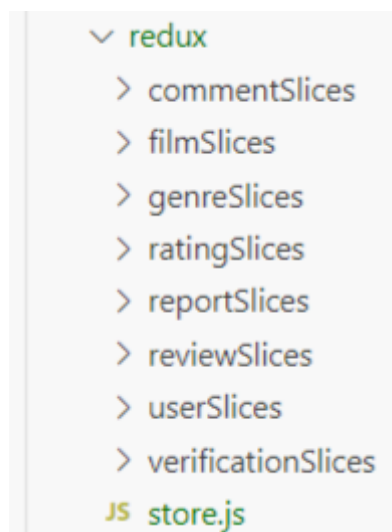


Рисунок 32 – Структура каталога «redux»

Файл store.js играет роль хранилища Redux, которое будет содержать глобальное состояние приложения. Содержание файла store.js представлен в листинге 4.

Листинг 4 – Содержание файла store.js

```
import { configureStore } from '@reduxjs/toolkit';  
const store = configureStore({  
  reducer: rootReducer,  
});
```

В данном листинге, rootReducer – это корневой редьюсер, который объединяет все редьюсеры приложения.

Состояния приложения определяются с помощью срезов (slice). Срезы – это отдельные модули, каждый из которых управляет определенной частью состояния. Например, можно создать срез для управления списком фильмов и срез для управления текущим фильмом.

Действия используются для изменения состояния. В Redux Toolkit действия создаются с помощью функций-генераторов действий. Код с созданием действия для добавления фильма в список представлен в листинге 5.

Листинг 5 – Код с созданием действия для добавления фильма в список

```
import { createSlice } from '@reduxjs/toolkit';
const moviesSlice = createSlice({
  name: 'movies',
  initialState: [],
  reducers: {
    addMovie: (state, action) => {
      state.push(action.payload);
    },
  },
});
export const { addMovie } = moviesSlice.actions;
```

Редьюсеры – это функции, которые принимают текущее состояние и действие и возвращают новое состояние. Корневой редьюсер объединяет все редьюсеры приложения в один редьюсер, который управляет всем состоянием.

Селекторы – это функции, которые извлекают данные из состояния. Они используются для доступа к данным состояния в компонентах React [9]. Например, можно создать селектор для получения списка фильмов:

Чтобы подключить Redux к приложению React, можно использовать хук `useSelector` для доступа к состояниям и хук `useDispatch` для отправки действий [8]. В листинге 6 представлен пример использования `useSelector` приведен в листинге.

Листинг 6 – Пример использования `useSelector`

```
import { useSelector, useDispatch } from 'react-redux';
import { addFilm, removeFilm, updateFilm } from
'./reduxtoolkit/filmSlice';
const FilmList = () => {
  const films = useSelector((state) => state.films.list);
  const dispatch = useDispatch();
  const onFilmChange = (films) => {
    dispatch(updateFilm(films));
  };
  return (
    <div>
      {films.map((film) => (
        <div key={film.id}>
          <h3>{film.title}</h3>
          <p>{film.year}</p>
        )}
      )}
    </div>
  );
};
```

```

        <button onClick={() =>
dispatch(removeFilm(film.id))}>Удалить</button>
      </div>
    )}
    <button onClick={() => dispatch(addFilm({ title: "Новый фильм",
year: 2023 })))>Добавить</button>
  </div>
);
};
export default FilmList;

```

Ниже на рисунке 33 приведена диаграмма классов, представляющая схему работы фрагмента состояния Film.

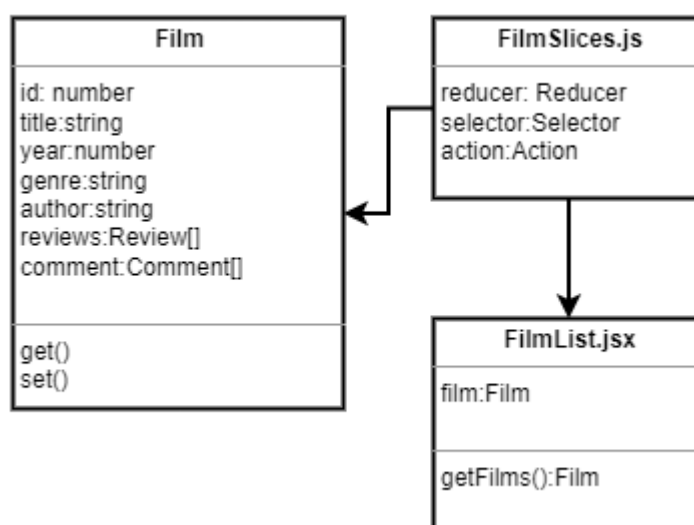


Рисунок 33 – Диаграмма классов «Схема работы фрагмента Film»

Таким образом, применение Redux Toolkit в веб-приложении обеспечивает надежное и масштабируемое управление состоянием, упрощая разработку и повышая качество кода.

3.4.2 Навигация

Для навигации в веб-приложении используется библиотека `react-router-dom`.

React Router DOM – мощная библиотека для управления навигацией и маршрутизацией в веб-приложениях React. Она предоставляет целый спектр функций, позволяющих создавать сложные и динамичные приложения с интуитивной системой навигации [10].

В веб-приложении реализованы три типа маршрутизации для пользователей:

- маршруты для неавторизованного пользователя;
- маршруты для авторизованного пользователя;
- маршруты для администратора.

Пример маршрутизации администратора приведен в листинге 7.

Листинг 7 – Пример маршрутизации администратора

```
import { Navigate } from "react-router-dom";
export const AdminRoutes = ({
  admin,
  redirectPath = '/login',
  children,
}) => {
  if (!admin) {
    return <Navigate to={redirectPath} replace />;
  }

  return children;
};
```

Основным компонентом для отображения страниц является файл App.jsx. Реализация навигации в React Router DOM осуществляется с помощью следующих основных компонентов:

- **BrowserRouter**: обертка вокруг приложения, которая предоставляет контекст маршрута;
- **Route**: компонент, который отображает дочерний компонент только тогда, когда URL приложения соответствует определенному пути;
- **Link**: компонент ссылки, который позволяет перемещаться по маршрутам без перезагрузки страницы.

BrowserRouter является обязательной оберткой для всего приложения. Он создает контекст маршрута, к которому могут обращаться дочерние компоненты для получения информации о текущем URL и маршруте. Пример представлен в листинге 8.

Листинг 8 – Пример работы с BrowserRouter

```
import { BrowserRouter } from "react-router-dom";
function App() {
  return (
    <BrowserRouter>
      {/* Дочерние маршруты здесь */}
    </BrowserRouter>
  );
};
```

```
}
```

Компонент `Route` используется для определения маршрутов в приложении. Он отображает предоставленный дочерний компонент только когда соответствующий путь совпадает с текущим URL. Пример представлен в листинге 9.

Листинг 9 – Пример использования `Route`

```
<Route
  path="/films/:id"
  element={
    <ProtectedRoutes user={user}>
      <FilmDescription />
    </ProtectedRoutes>
  }
/>
```

Компонент `Link` позволяет создавать ссылки для перехода по маршрутам без перезагрузки страницы.

`React Router DOM` является незаменимым инструментом для создания динамичных и удобных в использовании веб-приложений `React`. Он обеспечивает полный контроль над навигацией, что позволяет создавать сложные маршрутные системы с интуитивно понятным поведением. Следуя приведенным рекомендациям, вы сможете реализовать эффективную и гибкую навигацию в своих `React`-приложениях.

3.4.3 Валидация полей данных

Валидация на стороне клиента играет важную роль в обеспечении качества данных, отправляемых на сервер. Она гарантирует, что формы заполнены корректным образом, без пропущенных или некорректно заполненных полей. Этот процесс помогает предотвратить неполные или неточные данные, прежде чем они достигнут сервера. Для валидации полей данных было решено использовать библиотеку `Yup`.

`Yup` – это библиотека проверки данных для `JavaScript`, которая предоставляет широкий спектр возможностей для валидации данных на стороне клиента. Она позволяет разработчикам легко и эффективно проверять входные

данные и гарантировать их соответствие определенным критериям. Yup широко используется в таких фреймворках, как React, Angular и Vue.js, и является ценным инструментом для обеспечения целостности данных и повышения удобства использования приложений [11]. На рисунке 34 продемонстрирована процесс валидации библиотеки Yup.

The image shows a registration form with a blue background. The form is white and contains the following elements:

- Title: **Регистрация** (Registration)
- Input field: Почта (Email)
- Input field: Никнейм (Nickname)
- Input field: Пароль (Password)
- Error message: **Заполните поле "Никнейм"** (Fill in the "Nickname" field) in red text below the Nickname field.
- Button: **Зарегистрироваться** (Register)

Рисунок 34 – Процесс валидации библиотеки Yup

Реализации валидации функционала регистрации представлен в листинге 10.

Листинг 10 – Валидация функционала регистрации

```
export const registerUserSchema = yup.object().shape({
  email: yup.string().email('Некорректный формат почты').required("Заполните поле 'Почта'"),
  password: yup.string()
    .required('Заполните поле "Пароль"')
    .min(6, 'Пароль должен содержать не менее 6 символов'),
  nickname: yup.string()
    .required('Заполните поле "Никнейм"')
    .matches(/^[a-zA-Z0-9]+$/, 'Никнейм должен содержать только английские буквы и цифры')
});

const getUser = () => {
  registerUserSchema.validate(user).then(() => {
    setDataError([]);
    console.log(user)
  });
}
```

```

    })
    .catch((error) => {
      setDataError(error.errors)})})

```

В качестве примера использования Yup приведен код валидации функционала регистрации (листинг 1). Схема валидации определяет правила проверки для полей «Почта», «Пароль» и «Никнейм». Результаты валидации обрабатываются функцией `getUser()`, которая либо регистрирует пользователя, если валидация пройдена, либо отображает сообщения об ошибках, если валидация не выполнена.

4 Тестирование клиентского приложения

4.1 Функциональное тестирование

Функциональное тестирование является распространенным видом проверки программного обеспечения, которое оценивает соответствие функциональных возможностей приложения его изначальным требованиям. В рамках функционального тестирования применяются различные методики. Рассмотрим некоторые из них более подробно:

- положительное тестирование;
- негативное, или отрицательное тестирование;
- техники тест-дизайна;
- классы эквивалентности;
- метод граничных значений;
- техника состояния переходов;
- техника попарного тестирования, или pairwise testing.

Положительное тестирование – проверка положительных сценариев: работает ли система так, как было задумано. Например, получается ли указать адрес при заказе доставки.

Негативное, или отрицательное тестирование – проверяет, как система ведёт себя в нестандартных ситуациях, при введении невалидных значений с точки зрения формата или бизнес-логики. Например, в интернет-магазине пользователь пытается повторно вернуть товар. Ожидается, что система вычислит эту ошибку и не будет второй раз возвращать деньги.

Техники тест-дизайна – это готовые стратегии и приёмы, которые позволяют оптимизировать трудозатраты на создание тестов, соответствующих определённым целям и задачам.

Классы эквивалентности – входящие данные, например дата или сумма, делятся на классы эквивалентности. Подразумевается, что в одном классе вся система функционирует одинаково.

Метод граничных значений – обычно используется вместе с классами эквивалентности. Эта техника строится на подтверждённой теории: ошибки при разработке скапливаются на границах классов.

Техника состояния переходов – используется для сложных систем, например административных панелей или CRM-систем. Есть некая сущность, которая от создания до окончания существования меняет состояния. Тестировщик проводит с ней позитивные и негативные проверки и отслеживает, как сущность меняет состояние в системе.

Техника попарного тестирования – проверка всех возможных комбинаций системы по заданным параметрам. Например, в сервисе записи к врачу пациенты выбирают день, время, специализацию врача и клинику, всего четыре поля. В сервисе зарегистрированы врачи двадцати специализаций от окулиста до дерматолога и три клиники – уникальных комбинаций можно составить сотни. Техника попарного тестирования позволяет скомбинировать сочетание полей таким образом, чтобы протестировать все пары, но при меньшем количестве вариантов. Это можно сделать вручную, также для этого есть специальные инструменты: достаточно внести все значения, чтобы получить на выходе автоматически рассчитанный результат. Например, это PICT у Microsoft и IBM FoCuS у IBM [12].

4.1.1 Определение классов эквивалентности

На основе требований к программе и ее спецификаций были определены классы эквивалентности:

- никнейм должен содержать от 3 до 20 символов включительно;
- допустимые символы: буквы латинского алфавита (A-Za-z), цифры, символы нижнего подчеркивания и дефис;
- никнейм может начинаться с буквы, цифры и символа нижнего подчеркивания, не должен содержать специальные символы кроме дефиса и символа нижнего подчеркивания.

Таблица 8 – Классы эквивалентности: Табличное представление

	Правильные КЭ	Неправильные КЭ
Длина никнейма	3 – 20 (1)	0 – 1 (2) 21 – ∞ (3)
Структура	Начинается с буквы (4), цифры (5) или символа нижнего подчеркивания (6)	Начинается со спец символа (7)
Содержание	Латинские буквы (8), цифры (9), символ нижнего подчеркивания (10) и дефиса (11)	Остальное (12)
Длина никнейма	3 – 20 (1)	0 – 1 (2) 21 – ∞ (3)

Ниже представлены тесты и их описание, в дальнейшем они позволят оценить надежность и корректность работы приложения, обнаружить потенциальные проблемы и обеспечить высокое качество программного обеспечения в соответствии с требованиями и ожиданиями пользователей.

Таблица 9 – Тест -кейсы

№	КЭ	Входные данные	Выходные данные
1	Все правильные (1,4,5,6,8,9,10,11)	houstonia, _my-nickname733_	Успешно
2	4,5,2	js, j7	Ошибка, меньше трех символов
3	4,5,3	nickname13456789087665678212 212121211	Ошибка, больше 20 символов
4	1,4,5,6,7	-houstonia	Ошибка, начинается со спец символа
5	1,8,9,10,11	hello_world@@!!!!	Ошибка, содержит спец символы

При использовании методики классов эквивалентности для тестирования функциональности, связанной с никнеймами пользователей, были определены различные категории входных данных. Результаты тестирования показали, что программа корректно обрабатывает различные типы никнеймов в соответствии с их характеристиками.

4.1.2 Метод попарного тестирования

Метод попарного тестирования AllPairs – это техника проверки, которая тестирует все возможные комбинации условий для данной функции. Он основан

на принципе, что большинство ошибок возникают из-за взаимодействия нескольких входных условий.

Цель метода AllPairs – обеспечить всестороннее покрытие тестирования, выявляя скрытые взаимосвязи и потенциальные проблемы в программном обеспечении.

Метод AllPairs подходит для тестирования функций с небольшим количеством входных условий, для которых необходимо всестороннее покрытие. Он особенно полезен, когда существуют подозрения на наличие скрытых взаимосвязей и потенциальных проблем во взаимодействиях условий.

Определим основные случаи, которые необходимо проверить:

- никнейм, состоящий из корректных символов, корректной длины;
- пароль, состоящий из корректных символов, корректной длины;
- короткий никнейм, состоящий из корректных символов;
- короткий пароль, состоящий из корректных символов;
- длинный никнейм, состоящий из корректных символов;
- длинный пароль, состоящий из корректных символов;
- никнейм, имеющий в составе некорректные символы;
- пароль, имеющий в составе некорректные символы.

Исходные данные для формирования тестов представлены на рисунке 35.

Nickname	Password
cor_length	cor_length
correct	correct
free	wr_length
wr_length	wrong
wrong	

Рисунок 35 – Исходные данные для формирования тестов

Тесты, сгенерированные при помощи инструмента «AllPairs» представлены на рисунках 36 и 37.

24						
25	PAIRING DETAILS					
26	var1	var2	value1	value2	appearanc	cases
27	Nickname	Password	cor_length	cor_length	1	1
28	Nickname	Password	cor_length	correct	1	2
29	Nickname	Password	cor_length	wr_length	1	3
30	Nickname	Password	cor_length	wrong	1	4
31	Nickname	Password	correct	cor_length	1	5
32	Nickname	Password	correct	correct	1	6
33	Nickname	Password	correct	wr_length	1	7
34	Nickname	Password	correct	wrong	1	8
35	Nickname	Password	free	cor_length	1	9
36	Nickname	Password	free	correct	1	10
37	Nickname	Password	free	wr_length	1	11
38	Nickname	Password	free	wrong	1	12
39	Nickname	Password	wr_length	cor_length	1	13
40	Nickname	Password	wr_length	correct	1	14
41	Nickname	Password	wr_length	wr_length	1	15
42	Nickname	Password	wr_length	wrong	1	16
43	Nickname	Password	wrong	cor_length	1	17
44	Nickname	Password	wrong	correct	1	18
45	Nickname	Password	wrong	wr_length	1	19
46	Nickname	Password	wrong	wrong	1	20

Рисунок 36 – Тесты, сгенерированные при помощи инструмента «AllPairs»

1					
2	TEST CASES				
3	case	Nickname	Password	pairings	
4	1	cor_length	cor_length	1	
5	2	cor_length	correct	1	
6	3	cor_length	wr_length	1	
7	4	cor_length	wrong	1	
8	5	correct	cor_length	1	
9	6	correct	correct	1	
10	7	correct	wr_length	1	
11	8	correct	wrong	1	
12	9	free	cor_length	1	
13	10	free	correct	1	
14	11	free	wr_length	1	
15	12	free	wrong	1	
16	13	wr_length	cor_length	1	
17	14	wr_length	correct	1	
18	15	wr_length	wr_length	1	
19	16	wr_length	wrong	1	
20	17	wrong	cor_length	1	
21	18	wrong	correct	1	
22	19	wrong	wr_length	1	
23	20	wrong	wrong	1	
24					

Рисунок 37 – Тесты, сгенерированные при помощи инструмента «AllPairs»

Применение метода попарного тестирования с помощью инструмента AllPairs позволило определить оптимальный набор тестовых случаев для

проверки корректности программы при заданных параметрах. Результаты тестирования подтвердили, что программа функционирует корректно при указанных комбинациях значений параметров

4.2 Структурное тестирование

Тестирование представляет собой процедуру выполнения программы для выявления неисправностей. Эта процедура осуществляется на основе тестов, которые устанавливают следующее:

- комплект исходных данных и условий для запуска программы;
- ожидаемый результат и рабочие характеристики программы.

Для иллюстрации структурного тестирования приводится блок-схема алгоритма работы с жалобами. На рисунке 38 отражен полный цикл работы алгоритма работы с жалобами.

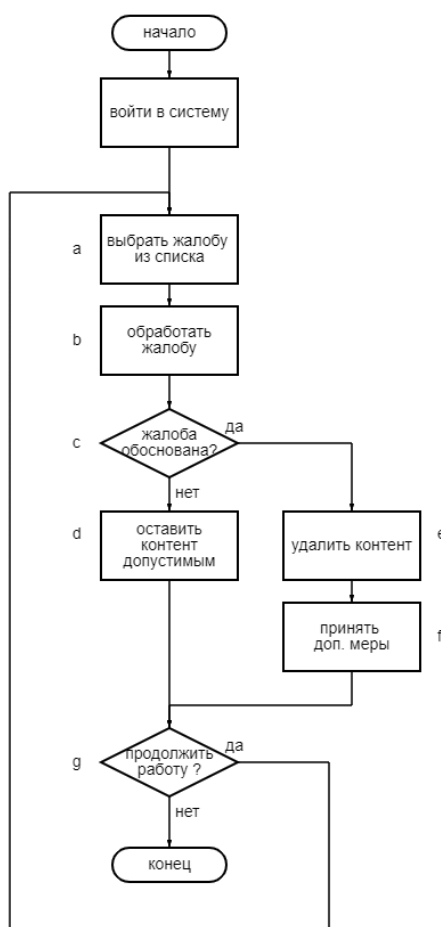


Рисунок 38 – Блок-схема алгоритма работы с жалобами

Управляющий граф представляет собой преобразованную блок – схему, где узлы соответствуют операциям, а ребра – условиям переходов.

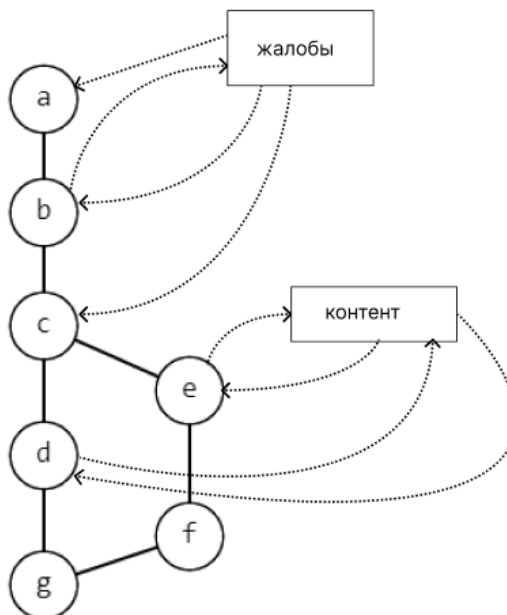


Рисунок 39 – Управляющий граф

Программа работает с объектами – жалобы и контент. В пунктах а, b, с происходит работа с объектом «жалобы», а в d и e – с объектом «контент».

Исходя из блок-схемы и управляющего графа, можно перейти к формированию тестов.

4.2.1 Покрытие операторов, решений и условий

Критерий покрытия операторов предполагает исполнение каждого оператора программы как минимум один раз. Чтобы выполнить каждый оператор не менее одного раза, нужно составить единственный тест со следующими значениями входных данных. Результаты тестирования, основанные на критерии покрытия операторов, представлены в таблице 10.

Таблица 10 – Результаты тестирования, основанные на критерии покрытия операторов

№	Операторы	Входные данные	Выходные данные
1	a,b,e,f	Жалоба не обоснована	Жалоба обоснована

Как видно из таблицы, каждый оператор был выполнен хотя бы один раз при различных наборах входных данных, что свидетельствует о достижении критерия покрытия операторов

В соответствии с покрытием решений необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное значение, так и ложное значение (таблица 11).

Таблица 11 – Покрытие решений

№	Решения	Входные данные	Выходные данные
1	с – да	Жалоба обоснована	Контент удален
2	с – нет	Жалоба не обоснована	Контент допустим

Комбинаторное покрытие условий требует, чтобы все возможные комбинации результатов условий в каждом решении, а также каждый оператор выполнились по крайней мере один раз.

В нашем примере должны быть покрыты 2 комбинаций:

- жалоба обоснована.
- жалоба не обоснована.

Для покрытия этих двух комбинаций достаточно 2 тестов, так как всего возможно 2 результата: контент допустим, контент удален (таблица 12).

Таблица 12 – Покрытие условий

№	Условие	Входные данные	Выходные данные
1	с, d	Жалоба обоснована	Контент удален
2	с, e, f	Жалоба не обоснована	Контент допустим

Тестирование на основе критериев покрытия показало удовлетворительные результаты, охватив все операторы, решения и комбинации условий в программе. Это дает уверенность в том, что программа была протестирована всесторонне, и снижает риск остаться незамеченными критическими ошибками.

4.3 Расчет метрик

Метрики кода позволяют анализировать качество программного кода и оценивать его сложность, объем и удобство сопровождения. Рассмотрим анализ качества кода на примере листинга 11, который реализует функционал фильтрации фильмов по жанру.

Листинг 11 – Код функционала фильтрации фильмов по жанру

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const MovieList = () => {
  const [movies, setMovies] = useState([]);
  const [genres, setGenres] = useState([]);
  const [selectedGenre, setSelectedGenre] = useState('all');

  useEffect(() => {
    const fetchMoviesAndGenres = async () => {
      try {
        const moviesResponse = await axios.get('/api/movies');
        const genresResponse = await axios.get('/api/genres');
        setMovies(moviesResponse.data);
        setGenres(['all', ...genresResponse.data]);
      } catch (error) {
        console.error('Error fetching movies and genres:', error);
      }
    };
    fetchMoviesAndGenres();
  }, []);

  const handleGenreChange = (event) => {
    setSelectedGenre(event.target.value);
  };

  const filteredMovies = selectedGenre === 'all'
    ? movies
    : movies.filter((movie) => movie.genres.includes(selectedGenre));

  return (
    <div>
      <h1>Expert Film Reviews</h1>
      <div>
        <label htmlFor="genre-select">Filter by genre:</label>
        <select id="genre-select" value={selectedGenre}
onChange={handleGenreChange}>
          {genres.map((genre) => (
            <option key={genre} value={genre}>
              {genre}
            </option>))}
        </select>
      </div>
      <ul>
        {filteredMovies.map((movie) => (
          <li key={movie.id}>
            <h2>{movie.title}</h2>
          </li>
        ))}
      </ul>
    </div>
  );
};
```

```

        <p>Expert Rating: {movie.expertRating}/10</p>
        <p>{movie.review}</p>
    </li>
    )})
</ul>
</div>);};
export default MovieList;

```

Для анализа качества кода будем использовать следующие метрики:

- метрика Холстеда;
- метрика Чепина;

Метрики кода помогают выявлять потенциальные риски, оценивать текущее состояние проекта и отслеживать прогресс во время разработки программного обеспечения [13].

4.3.1 Метрика Холстеда

Метрика Холстеда – это набор мер, используемых для оценки сложности программного кода. Метрика получила свое название в честь ее разработчика, Эдварда Холстеда [14].

Метрика Холстеда основана на двух основных понятиях:

- операторы: ключевые слова, управляющие операторы и структурные символы программы;
- операнды: идентификаторы, константы и литералы.

Расчет количественных метрик кода листинга 11 приведены в таблице 13.

Таблица 13 – Количественные метрики кода листинга 11

Операторы		Операнды	
Имя	Количество	Имя	Количество
import	2	React	1
export	1	useState	4
from	2	useEffect	2
const	9	«react»	1
=	9	«axios»	1
async	1	axios	3
await	2	MovieList	1
get	2	movies	3
try	1	setMovies	2
catch	1	genres	3
console.error	1	setGenres	2

Продолжение таблицы 13

Имя	Количество	Имя	Количество
===	1	selectedGenre	4
?:	1	setSelectedGenre	2
filter	1	«all»	3
includes	1	fetchMoviesAndGenres	2
map	1	«/api/movies»	1
onChange	1	«api/genres»	1
		moviesResponse	2
		genresResponse	2
		data	2
		error	2
		«Error fetching movies and genres:»	1
		handleChangeError	1
		event	2
		filteredMovies	2
		movie	7
		genre	4
$n_1 = 17$	$N_1 = 37$	$n_2 = 28$	$N_2 = 61$

Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2,$$

где n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Подставив соответствующие значения в формулы, получаем: $n = 45$.

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2,$$

где N_1 – общее число простых операторов и операций в программе;

N_2 – общее число простых операндов в программе.

В данном случае: $N = 98$.

Объем программы (Halstead Volume) рассчитывается по формуле:

$$HV = N \times \log_2 n = 98 \times 5,491 = 538,118,$$

где N – длина программы;

n – словарь программы.

Сложность программы (Halstead Difficulty) рассчитывается по формуле:

$$HDiff = \frac{n_1}{2} \times \frac{N_2}{n_2},$$

где n_1 – общее число простых операторов и операций в программе;

n_2 – общее число простых операндов в программе;

N_2 – общее число простых операндов в программе.

В данном случае : $HDiff = 18,5$.

Полученные результаты программы находятся в приемлемом диапазоне и соответствуют установленным стандартам для метрик программного кода.

4.3.2 Метрика Чепина

Метрика Чепина – это метрика оценки сложности программного модуля, которая оценивает информационную прочность модуля, анализируя использование переменных во входных и выходных списках.

Список переменных входных и выходных данных разбивается на четыре функциональные группы:

- P (вводимые переменные): использование для вычислений и обеспечения вывода;
- M (модифицируемые переменные): создаются или изменяются в программе;
- C (управляющие переменные): участвуют в управлении потоком выполнения;
- T (паразитные переменные): не используются в программе.

Метрика Чепина рассчитывается по формуле:

$$Q = a1 \times P + a2 \times M + a3 * C + a4 * T,$$

где $a1, a2, a3, a4$ — весовые коэффициенты.

Весовые коэффициенты отражают разное влияние каждой функциональной группы на сложность программы. Согласно Чепину, управляющие переменные имеют наибольшее влияние, поэтому им присваивается вес 3. Остальные весовые коэффициенты распределены следующим образом:

С учетом весовых коэффициентов формула метрики становится:

$$Q = P + 2M + 3C + 0,5T.$$

Более высокое значение Q указывает на более сложный программный модуль. Высокие значения M и C могут указывать на модуль, который трудно модифицировать или поддерживать. Высокие значения T могут указывать на наличие неиспользуемого или лишнего кода [15].

В представленном листинге 11:

- P (вводимые переменные для расчетов и вывода): `movies` – список фильмов, полученный из API, `genres` – список жанров, полученный из API, `selectedGenre` – выбранный жанр для фильтрации;
- M (модифицируемые переменные или создаваемые внутри программы): `filteredMovies` – отфильтрованный список фильмов на основе выбранного жанра;
- C (управляющие переменные): `selectedGenre` – используется для управления отображением фильмов в зависимости от выбранного жанра;
- T (неиспользуемые переменные или «паразитные»): в данном коде нет неиспользуемых переменных.

$$Q = 3 + 2 * 1 + 3 * 1 + 0.5 * 0 = 8$$

Значение метрики Чепина равно 8, что указывает на относительно низкую информационную прочность модуля. Это означает, что модуль имеет простую структуру и выполняет ограниченное количество функций.

Низкая информационная прочность обычно является признаком хорошо структурированного и простого в обслуживании кода.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано клиентское приложение для web-портала агрегации зрительской и экспертной кинокритики. Разработка проекта включала анализ предметной области, в результате которого были изучены конкурирующие программные продукты.

На основе проектирования было проведено функциональное моделирование, в котором были проанализированы бизнес-процессы и потоки данных разрабатываемой платформы и разработана концептуальная модель данных. Кроме того, был разработан программный интерфейс приложения.

Разработка велась с помощью библиотеки React на языке программирования JavaScript.

Корректность работы программы была проверена с помощью функционального и структурного тестирования. Также был проведен анализ качества кода с использованием метрик, который показал, что разработанное приложение обладает хорошей сопровождаемостью кода.

Кроме того, было спланировано создание web-приложения и оценено ее бюджетирование с помощью программного обеспечения ProjectLibre. В ходе этого процесса был разработан график выполнения каждой задачи проекта, а также рассчитаны затраты на ресурсы.

Список использованных источников

1. Кино. Основные понятия. [Электронный ресурс]. URL: <https://www.calc.ru/463.html#:~:text=КИНОДРАМАТУРГИЯ%20%20литературн%20кинематографический%20вид%20творчества%3В,или%20воссозданных%20средствами%20мультипликации%20событий> (дата обращения 11.04.2024)
2. О профессии кинокритика [Электронный ресурс]. URL: <https://postupi.online/professiya/kinokritik/> (дата обращения 20.04.2024)
3. Прошкина Е.Н. Язык моделирования информационных процессов UML 2.0 / Прошкина Е.Н., Балашова И. Ю. – Пенза: Изд-во ПГУ, 2018. – 112с.
4. Концептуальная модель [Электронный ресурс]. <https://fb.ru/article/144365/что-такое-kontseptualnaya-model> (дата обращения 29.04.2024)
5. Разработка дизайна интерфейса [Электронный ресурс]. URL: <https://terabit.ai/research/polzovatelskij-interfejs-dizajn-razrabotka> (дата обращения 04.05.2024)
6. Макеев, Г.А. Объектно – ориентированное программирование с нуля к SOLID и MVC / Макеев Г.А. – Изд – во – BHV, 2024. – 272с
7. Руководство по TailwindCSS – фреймворку для быстрой стилизации [Электронный ресурс]. URL: <https://webformyself.com/v-dobryj-put-tailwind-css/>
8. Использование Axios с React [Электронный ресурс]. URL: <https://timeweb.cloud/tutorials/react/ispolzovanie-axios-s-react>
9. React и Redux [Электронный ресурс]. URL: <https://react-redux.js.org/>
10. Навигация через react-router-dom [Электронный ресурс]. URL: <https://ru.hexlet.io/blog/posts/react-router-v6>
11. Валидация данных через Yup [Электронный ресурс]. URL: <https://www.npmjs.com/package/yup>
12. Функциональное тестирование ПО [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/funkcionalnoe-testirovanie-po/> (дата обращения 22.05.2024)

13. Выпускная квалификационная работа: стандарты и программные технологии. Учебно – методическое пособие для бакалавров по направлениям 09.03.04 «Программная инженерия», 09.03.02 «Информационные системы и технологии» / под ред. П.П. Макарычева. – Пенза: Изд – во ПГУ, 2020. – 182с.

14. Метрика Холстеда [Электронный ресурс]. URL: <https://studfile.net/preview/10101914/page:5/>

15. Метрика Чепина [Электронный ресурс]. URL: <https://studfile.net/preview/6196490/page:8/>

Приложение А
(рекомендуемое)
ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

Листинг A.1 – Файл button.jsx

```
/* eslint-disable react/prop-types */
import './button.css'

export const Button = ({name='', onClick, className=''})=>{
  return <button className={` ${className.length>0? `${className}` : 'btn'}` `} onClick={onClick}>{name}</button>
}
```

Листинг A.2 – Файл button.css

```
.btn{
  border-radius: 33.778px;
  border: none;
  background: #664DFF;
  box-shadow: 0px 6.333px 21.111px 0px rgba(102, 77, 255, 0.25);
  min-width: 300px;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 13px 24px;
  margin: 20px 0;
  color: #FFF;
  font-family: "Helvetica Neue";
  letter-spacing: 2px;
  font-size: 16.889px;
  font-style: normal;
  font-weight: 600;
  line-height: normal;
}
.btn-admin{
  border-radius: 33.778px;
  border: none;
  box-shadow: 0px 6.333px 21.111px 0px rgba(102, 77, 255, 0.25);

  margin: 1px;
  padding: 5px 8px;
  color: #FFF;
  font-family: "Helvetica Neue";
  letter-spacing: 1px;
  font-size: 10px;
  font-style: normal;
  font-weight: 600;
  line-height: normal;
}
.btn-admin-primary{
  background: #664DFF;
}
.btn-admin-red{
  background: #dd2909;

  box-shadow: 0px 6.333px 21.111px 0px rgba(255, 107, 77, 0.25);
}
.btn-comment{
  width: 140px;
```

```

height: 45px;
text-transform: uppercase;
background: #8776f7;
letter-spacing: 1px;
display: flex;
justify-content: center;
align-items: center;
padding: 13px 24px;
color: #FFF;
font-family: "Helvetica Neue";
font-size: 10px;
font-style: normal;
font-weight: 600;
line-height: normal;
border-radius: 10px;
border: none;
}

.btn-lk{
border-radius: 33.778px;
border: none;
background: #000000;
box-shadow: 0px 6.333px 21.111px 0px rgba(102, 77, 255, 0.25);
min-width: 300px;

display: flex;
justify-content: center;
align-items: center;
padding: 13px 24px;

margin: 20px 0;
color: #FFF;
font-family: "Helvetica Neue";
letter-spacing: 2px;
font-size: 16.889px;
font-style: normal;
font-weight: 600;
line-height: normal;
}

.btn-query{
border-radius: 33.778px;
border: none;
background: #818181;
box-shadow: 0px 6.333px 21.111px 0px rgba(102, 77, 255, 0.25);
min-width: 300px;
display: flex;
justify-content: center;
align-items: center;
padding: 13px 24px;
margin: 20px 0;
color: #FFF;
font-family: "Helvetica Neue";
letter-spacing: 2px;
font-size: 16.889px;
font-style: normal;
font-weight: 600;

```

```

    line-height: normal;
}

```

Листинг А.3 – Файл base-film-card.jsx

```

import { CiMenuKebab } from 'react-icons/ci';
import { MdStarRate } from 'react-icons/md';

export const BaseFilmCard = ({onClick, values}) => {
  return (
    <div className="border border-solid border-[#F3F3FA] rounded-5
shadow-md h-[15%] w-[61%] bg-white p-6 m-2 flex justify-between"
onClick={onClick}>
      <div className="flex">
        <img
          className="max-h-[100px]"
          src={values.src}
          alt=""
        />
        <div>
          <p className="text-[#1C3988] text-[15px] font-semibold text-start mx-2">
            {values.name}
          </p>
          <p className="text-gray text-[10px] font-semibold text-start mx-2">
            {values.duration}
          </p>
          <p className="text-gray text-[10px] font-semibold text-start mx-2">
            {values.author}
          </p>
          <p className="text-gray text-[10px] font-semibold text-start mx-2">
            {values.roles}
          </p>
          <p className="text-[#1C3988] text-[10px] mx-2 line-clamp-2">
            {values.desc}
          </p>
        </div>
      </div>
      <div className="flex">
        <p className="font-semibold mr-5">9.1</p>
        <p className="text-[gray] mr-10">128 037</p>
        <MdStarRate size={values.ratingUser} className="mr-5" />
        <CiMenuKebab size={values.ratingReviewer} />
      </div>
    </div>
  );
};

```

Листинг А.4 – Файл film-card.jsx

```

/* eslint-disable react/prop-types */

```

```

import { Inset, Card } from "@radix-ui/themes"
import { MdDelete } from "react-icons/md";
export const FilmCard = ({annot="",title='Энола
Холмс',filmTiming=200,img='https://narrative.onlinemodule.ru/wp-
content/uploads/2021/10/9Vs7ybgN2Ij2TzTjabWEFdKbm6G-1024x576.jpg'}) => {
  return <Card size="2" className="border border-solid border-
[#F3F3FA] max-w-[320px] mx-2 my-3 rounded-5 shadow-md">
    <Inset clip="padding-box" side="top" pb="current">
      <div className='flex justify-end' >
        <MdDelete />
      </div>
      <img
        src={img}
        alt="Bold typography"
        className='block object-cover max-h-[249px]'
      />
    </Inset>
    <div className="text-[#1C3988] text-[15px] font-semibold text-
start mx-2">
      {title}
    </div>
    <div className="text-[#1C3988] text-[12px] font-semibold text-
start mx-2">Продолжительность:{filmTiming} мин </div>
    <div className=" text-[#1C3988] text-[10px] mx-2 line-clamp-
2">{annot.length>0?annot:annotation}</div>
  </Card>
}

```

Листинг А.5– Файл report-card.jsx

```

/* eslint-disable react/prop-types */
import { Button } from "../button";
import { Input } from "../input";

```

```

export const ReportCard = ({text,user='user@gmail.com'}) => {
  return (
    <div className='h-[15%] w-[31%] bg-white rounded-lg p-6 m-2'>
      <div className='text-[14px] font-semibold'>{user}</div>
      <div className='text-[13px] mt-2'>{text}</div>
      <div className="flex mt-2">
        <Button name="Принять" className="btn-admin btn-
admin-primary" />
        <Button name="Отклонить" className="btn-admin btn-
admin-red"/>
        <Button name="Забанить" className="btn-admin btn-
admin-red"/>
        <Input placeholder="Рейтинг" style="w50"/>
      </div>
      <div className="flex mt-2">
        </div>
    </div>
  );
};

```

Листинг А.6– Файл report-card.jsx

```

export const ReviewCard = ({onClick, img="",author=" ", text=" => {

```

```

    return <div onClick={onClick} className="border border-solid
border-[#F3F3FA] rounded-5 shadow-md h-[15%] w-[15%] bg-white p-6 m-5
flex flex-col items-center">
      <div className="mb-4 h-64 w-64 p-2">
        <img className="h-full w-full object-cover" src={img}
alt="" />
      </div>
      <div>{author}</div>
      <div className=" text-[#1C3988] text-[12px] line-clamp-
3">{text}</div>
    </div>
  }

```

Листинг А.7– Файл user-card.jsx

```

/* eslint-disable react/prop-types */

import { Button } from "../button";

export const UserCard=({text,user="user@gmail.com"})=>{
  return (
    <div className='h-[15%] w-[40%] bg-white rounded-lg p-6 m-2
flex justify-between'>
      <div className='w-[30px] h-[30px] text-[13px] text-white
font-semibold flex justify-center items-center mt-2 rounded-[100px] bg-
[#664DF9]'>{text}</div>
      <div className='text-[14px] font-semibold'>{user}</div>
      <Button name="Забанить" className="btn-admin btn-admin-
red"/>
    </div>
  );
}

```

Листинг А.8– Файл verification-card.jsx

```

/* eslint-disable react/prop-types */
import { Button } from "../button";

export const VerificationCard=({text,user='user@gmail.com'})=>{
  return (
    <div className='h-[15%] w-[50%] bg-white rounded-lg p-6 m-2'>
      <div className='text-[14px] font-semibold'>{user}</div>
      <div className='text-[13px] mt-2'>{text}</div>
      <div className='text-[13px] mt-2 text-[#1C3988]
underline'>Открыть вложения</div>
      <div className="flex mt-2">
        <Button name="Принять" className="btn-admin btn-
admin-primary" />
        <Button name="Отклонить" className="btn-admin btn-
admin-red"/>
      </div>
    </div>
  );
}

```

Листинг А.9– Файл comment-list.jsx

```

import { Comment } from "../comment"
import { Button } from "../button"

```

```

import { Input } from "../input"
export const CommentList=()=>{
  return <div className="m-5">
    <p className="text-[#1C3988] text-[20px] font-semibold text-
start mx-2 pb-4">Комментарии</p>
    <div className="flex items-center border rounded-md p-4 w-
[43%]">
      <Input placeholder="Оставить комментарий"
style="comment"/>
      <Button name="Опубликовать" className="btn-comment"/>
    </div>
  </div>
  {comments.map(i=>{
    <Comment key={i} text={i.text} user={i.user}
date={i.date}/>
  })}
</div>
</div>
}

```

Листинг А.10– Файл comment.jsx

```

export const Comment=({text,user,date})=>{
  return <div className="flex flex-col w-[70%] border rounded-md p-4
m-2">
    <p className="text-[#1C3988] text-[14px] font-semibold text-
start m-2">{user}</p>
    <p className="text-[12px] text-start ml-2">{date}</p>
    <p className="text-[12px] text-start m-2">{text}</p>
  </div>
}

```

Листинг А.11 – Файл editor.jsx

```

// import { Editor, EditorState } from 'draft-js';
import { Button } from "../button/button"
import { useState } from 'react';
import { EditorState } from 'draft-js';
import { Editor } from 'react-draft-wysiwyg';
import "../editor.css"
import 'react-draft-wysiwyg/dist/react-draft-wysiwyg.css';

export const MyEditor=()=> {
  const [editorState, setEditorState] = useState(
    () => EditorState.createEmpty(),
  );

  return (
    <div className="flex flex-col items-center">
      <h2 className="text-[32px] my-[20px]">
        Напишите вашу рецензию
      </h2>
      <Editor
        toolbarClassName="toolbarClassName"
        wrapperClassName="wrapperClassName"
        editorClassName="editorClassName"
        editorState={editorState}

```

```

        onEditorStateChange={setEditorState}
      />
      <Button name="Опубликовать" className="btn"/>
    </div>
  )
}

```

Листинг А.12– Файл editor.css

```

.toolbarClassName {
  border-radius: 10px;
}
.wrapperClassName {
  border: 1px solid #654dff26;
  padding: 1rem;
  border-radius: 15px;
}
.editorClassName {
  background-color: #654dff26;
  border-radius: 10px;
  padding: 1rem;
  min-height: 400px;
}

```

Листинг А.13– Файл input.jsx

```

/* eslint-disable react/prop-types */

export const Input = ({ placeholder = '', value, setValue, style =
'primary' }) => {
  return <input value={value} onChange={setValue}
    className={styles[style]}
    type="text" placeholder={placeholder} />
}

const styles = {
  primary: 'w-[300px] h-[50px] m-[10px] px-[2px] shadow-inner border-
2 border-[#664dff] outline-none rounded-[8px] bg-[#ffffff] text-[#0A1223]
font-inter text-[16.889px]',
  w50: 'mx-1 w-[50px] rounded-[8px] text-[10px] border-2 border-
[#664dff] outline-none text-center',
  comment: "min-w-[550px] h-[100px] m-[10px] px-4 shadow-inner border
border-1 border-gray outline-none rounded-[8px] text-[#0A1223] font-
inter text-[16.889px]"
}

```

Листинг А.14– Файл search-input.jsx

```

import { IoIosSearch } from "react-icons/io";
import { TbAdjustmentsHorizontal } from "react-icons/tb";

export const SearchInput = () => {
  return (
    <div className="flex items-center border-2 border-gray-300 bg-white
rounded-lg h-7 pr-3">
      <input type="text" placeholder="Фильмы, рецензии"
        className="border-none px-5 text-black text-sm w-[230px] outline-none"/>
    </div>
  )
}

```

```

        <TbAdjustmentsHorizontal size={24} color="black"/>
        <IoIosSearch size={24} color="black"/>
    </div>
  );
};

```

Листинг А.15– Файл reviews-list.jsx

```

import { ReviewCard } from "../card"

export const ReviewList = ({reviews}) => {
  return <div className="flex flex-col ml-4">
    <p className="text-[#1C3988] text-[20px] font-semibold text-start mx-2 my-5">Рецензии кинокритиков </p>
    <div className="flex">
      {
        reviews.map(i => (
          <ReviewCard text={i.text} key={i.img}
img={i.img} author={i.author}/>
        ))
      }
    </div>
  </div>
}

```

Листинг А.16– Файл schema.js

```

import * as yup from 'yup';

export const schema = yup.object().shape({
  email: yup.string().email('Некорректный формат почты').required("Заполните поле 'Почта'"),
  password: yup.string()
    .required('Заполните поле "Пароль"')
    .min(6, 'Пароль должен содержать не менее 6 символов'),
});

export const registerUserSchema = yup.object().shape({
  email: yup.string().email('Некорректный формат почты').required("Заполните поле 'Почта'"),
  password: yup.string()
    .required('Заполните поле "Пароль"')
    .min(6, 'Пароль должен содержать не менее 6 символов'),
  nickname: yup.string()
    .required('Заполните поле "Никнейм"')
    .matches(/^[a-zA-Z0-9]+$/, 'Никнейм должен содержать только английские буквы и цифры')
});

export const filmSchema = yup.object().shape({
  filmTitle: yup.string().required("Укажите название фильма"),
  director: yup.string().required("Укажите режиссера фильма"),
  filmTiming: yup.string().required("Укажите продолжительность фильма"),
  annotation: yup.string().required("Добавьте аннотацию к фильму"),
  releaseDate: yup.string().required("Укажите год выпуска фильма"),
  reviewerRating: yup.number().integer().min(0).max(10).required("Укажите оценку критика для фильма"),
});

```

```

    usersRating: yup.number().integer().min(0).max(10).required("Укажите
пользовательскую оценку для фильма"),
});

```

Листинг А.17– Файл admin-layout.jsx

```

import { useState } from "react";
import { NavLink, Outlet } from "react-router-dom";

export const AdminLayout = () => {
  const [currentTitle, setCurrentTitle] = useState('')
  const testMenuItems = [
    {
      href: 'report',
      title: 'Жалобы',
    },
    {
      href: 'film',
      title: 'ФИЛЬМЫ',
    },
    {
      href: 'add-film',
      title: 'Добавить фильм',
    },
    {
      href: 'verification',
      title: 'Верификации',
    },
    {
      href: 'user-rating',
      title: 'Рейтинг плохишей',
    },
  ],

  <li><NavLink to="/film">ФИЛЬМЫ</NavLink></li>
  return (
    <div className='min-h-screen flex flex-col'>
      <header className='bg-[#664DFF] font-bold text-white
sticky top-0 h-14 flex justify-center items-center font-semibold
uppercase'>
        Админ-панель {currentTitle}
      </header>
      <div className='flex flex-col md:flex-row flex-1'>
        <aside className='bg-gray-100 w-full md:w-60'>
          <nav>
            <ul>
              {testMenuItems.map(({ href, title
            }) => (
              <li key={title} className=' m-
2' onClick={()=>setCurrentTitle(`: ${title}`)}>
                <NavLink to={href} >
                  <p
className={'text-black font-semibold'}>{title}</p>
                </NavLink>
              </li>
            ))}

```

```

        </ul>
      </nav>
    </aside>
    <main className={'flex-1 bg-[#010314] w-[100%]' }>
      <Outlet />
    </main>
  </div>
</div>
);
}

```

Листинг А.18– Файл base-layout.jsx

```

import { GiHamburgerMenu } from "react-icons/gi";
import { FaUser } from "react-icons/fa";
import { NavLink, Outlet } from "react-router-dom";
import { SearchInput } from "../components";

export const BaseLayout = () => {
  const testMenuItems = [
    {
      href: 'reviews',
      title: 'Рецензии',
    },
    {
      href: 'films',
      title: 'ФИЛЬМЫ',
    },
    {
      href: 'films',
      title: 'Написать рецензию',
    }
  ];

  <li><NavLink to="/film">ФИЛЬМЫ</NavLink></li>
  return (
    <div className='min-h-screen flex flex-col'>
      <header className='px-6 bg-[#010314] text-white sticky
top-0 h-16 flex items-center justify-around font-semibold uppercase'>
        <div className="flex">
          <GiHamburgerMenu size={24} />
          <p className="ml-3">MovieMinds</p>
        </div>
        <nav>
          <ul className="flex">
            {testMenuItems.map(({ href, title }) =>
              (
                <li key={title} className='m-2'>
                  <NavLink to={href} >
                    <p className={'text-
white font-semibold'}>{title}</p>
                  </NavLink>
                </li>
              )))
          </ul>
        </nav>
        <SearchInput/>
        <NavLink to="/lk' >

```

```

                <FaUser size={24} />
            </NavLink>
        </header>
        <Outlet />
    </div>
);
}

```

Листинг A.19– Файл admin-routes.jsx

```

/* eslint-disable react/prop-types */

import { Navigate } from "react-router-dom";

export const AdminRoutes = ({
  admin,
  redirectPath = '/login',
  children,
}) => {
  if (!admin) {
    return <Navigate to={redirectPath} replace />;
  }

  return children;
};

```

Листинг A.20– Файл protected-routes.jsx

```

/* eslint-disable react/prop-types */

import { Navigate } from "react-router-dom";

export const ProtectedRoutes = ({
  user,
  redirectPath = '/login',
  children,
}) => {
  if (!user) {
    return <Navigate to={redirectPath} replace />;
  }

  return children;
};

```

Листинг A.21– Файл App.jsx

```

import { Route, Routes, BrowserRouter as Router } from "react-router-dom"
import { AdminLayout, BaseLayout } from "../layout"
import { AddFilm, EditorPage, Film, FilmDescription, Home, Lk, LkUser,
LoginPage, RegisterPage, Report, ReviewDescription, Reviews, UserRating,
Verification } from "../page"
import { ProtectedRoutes } from '../routes'

const App = () => {
  let user = true

```

```

return <div>
  <Router>
    <Routes>
      <Route path="register" element={<RegisterPage />} />
      <Route path="login" element={<LoginPage />} />
      <Route element={<BaseLayout />}>
        <Route
          index
          element={
            <ProtectedRoutes user={user}>
              <Home />
            </ProtectedRoutes>
          }
        />
        <Route
          path="films/:id"
          element={
            <ProtectedRoutes user={user}>
              <FilmDescription />
            </ProtectedRoutes>
          }
        />
        <Route
          path="films"
          element={
            <ProtectedRoutes user={user}>
              <Home />
            </ProtectedRoutes>
          }
        />
        <Route
          path="reviews"
          element={
            <ProtectedRoutes user={user}>
              <Reviews />
            </ProtectedRoutes>
          }
        />
        <Route
          path="reviews/:id"
          element={
            <ProtectedRoutes user={user}>
              <ReviewDescription />
            </ProtectedRoutes>
          }
        />
        <Route
          path="editor"
          element={
            <ProtectedRoutes user={user}>
              <EditorPage />
            </ProtectedRoutes>
          }
        />
        <Route
          path="lk"

```

```

        element={
          <ProtectedRoutes user={user}>
            <Lk />
          </ProtectedRoutes>
        }
      />
    <Route
      path="my-lk"
      element={
        <ProtectedRoutes user={user}>
          <LkUser />
        </ProtectedRoutes>
      }
    />
  </Route>
  <Route element={<AdminLayout />}>
    <Route
      path="report"
      element={
        <ProtectedRoutes user={user}>
          <Report />
        </ProtectedRoutes>
      }
    />
    <Route
      path="film"
      element={
        <ProtectedRoutes user={user}>
          <Film />
        </ProtectedRoutes>
      }
    />
    <Route
      path="verification"
      element={
        <ProtectedRoutes user={user}>
          <Verification />
        </ProtectedRoutes>
      }
    />
    <Route
      path="add-film"
      element={
        <ProtectedRoutes user={user}>
          <AddFilm />
        </ProtectedRoutes>
      }
    />
    <Route
      path="user-rating"
      element={
        <ProtectedRoutes user={user}>
          <UserRating />
        </ProtectedRoutes>
      }
    />
  </Route>

```

```

        </Route>
      </Routes>
    </Router>
  </div>
}
export default App;

```

Листинг А.22– Файл `eslinttrc.cjs`

```

module.exports = {
  root: true,
  env: { browser: true, es2020: true },
  extends: [
    'eslint:recommended',
    'plugin:react/recommended',
    'plugin:react/jsx-runtime',
    'plugin:react-hooks/recommended',
  ],
  ignorePatterns: ['dist', '.eslinttrc.cjs'],
  parserOptions: { ecmaVersion: 'latest', sourceType: 'module' },
  settings: { react: { version: '18.2' } },
  plugins: ['react-refresh'],
  rules: {
    'react-refresh/only-export-components': [
      'warn',
      { allowConstantExport: true },
    ],
  },
},

```

Листинг А.23– Файл `prettierrc`

```

{
  "printWidth": 80,
  "tabWidth": 2,
  "bracketSameLine": false,
  "arrowParens": "avoid",
  "bracketSpacing": true,
  "endOfLine": "crlf",
  "singleQuote": true
}

```

Листинг А.24– Файл `tailwind.config.js`

```

/** @type {import('tailwindcss').Config} */
export default {
  content: [
    './index.html',
    './src/**/*.{js,ts,jsx,tsx}',
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}

```

Листинг А.25– Файл `add.jsx`

```

import { useState } from 'react'
import { Button, Input } from '../../components/'
import { filmSchema } from '../../helpers/schema';

export const AddFilm = () => {
  const [dataError, setDataError] = useState([]);
  const [film, setFilm] = useState({
    filmTitle: '',
    releaseDate: '',
    director: '',
    filmTiming: '',
    annotation: '',
    reviewerRating: '',
    usersRating: ''
  })
  const AddFilm = () => {
    filmSchema.validate(film).then(() => {
      setDataError([]);
      console.log(film)
    })
    .catch((error) => {
      setDataError(error.errors);
    });
  }
  return <div>
    <div className='bg-white rounded-lg p-6 m-2'>
      <div className='text-[16px] font-semibold'>Добавить
фильм</div>
      <div className="text-[12px] text-[#dd2909] font-semibold">
        {dataError.map((i) => i)}
      </div>
      <Input setValue={e => { setFilm({ ...film, filmTitle:
e.target.value }}} value={ film.filmTitle}placeholder='Название' />
      <Input setValue={e => { setFilm({ ...film, annotation:
e.target.value }}} value={ film.annotation}placeholder='Аннотация' />
      <Input setValue={e => { setFilm({ ...film, filmTiming:
e.target.value }}} value={
film.filmTiming}placeholder='Продолжительность' />
      <Input setValue={e => { setFilm({ ...film, director:
e.target.value }}} value={ film.director}placeholder='Режиссер' />
      <Input setValue={e => { setFilm({ ...film, usersRating:
e.target.value }}} value={ film.usersRating}placeholder='Рейтинг от
пользователей' />
      <Input setValue={e => { setFilm({ ...film, reviewerRating:
e.target.value }}} value={ film.reviewerRating}placeholder='Рейтинг от
рецензентов' />
      <Button name='Добавить' onClick={AddFilm} />
    </div>
  </div>
}

```

Листинг А.26– Файл Film.jsx

```

import { FilmCard } from "../../components"

export const Film=(ar)=>{

```

```

return <div>
  <div className='flex flex-wrap bg-white rounded-lg p-6 m-2'>
    {
      ar.map( (_, ind)=>(
        <FilmCard key={ind}/>
      ))
    }
    <FilmCard key={100} title="TestTitle" annot='test'
filmTiming={100}
img="https://i.artfile.ru/2560x1638_110517_[www.ArtFile.ru].jpg"/>
  </div>
</div>
}

```

Листинг A.27– Файл report.jsx

```

import { ReportCard } from "../../components"

export const Report = ({arr}) => {
  return <div>
    <div className="flex flex-wrap ">
      {arr.map((report, ind) => (
        <ReportCard key={ind} text={report.reportText}
user={report.user} />
      ))}
    </div>
  </div>
}

```

Листинг A.28– Файл user-rating

```

import { UserCard } from "../../components/card"

export const UserRating = ({arr}) => {
  return <div>
    <div>
      {
        arr.map(i => (
          <UserCard key={i.id} text={i.id-1}
user={i.name} />
        ))
      }
    </div>
  </div>
}

```

Листинг A.29– Файл verification.jsx

```

import { VerificationCard } from "../../components"

export const Verification = (arr) => {
  return <div>
    <div>
      {
        arr.map((i, ind) => (

```

```

                                <VerificationCard text={i.verificationText}
key={ind} user={i.user}/>
                                ))
                                }
                                </div>
                                </div>
                                }

```

Листинг A.30– Файл FilmDescription.jsx

```

import { Rating } from "@mui/material";
import { useState } from "react";
import { ReviewList } from "../components";
import { CommentList } from "../components/comments/comment-list"

export const FilmDescription = () => {
  const [userRating, setUserRating] = useState(4)
  const [reviewerRating, setReviewerRating] = useState(5)
  return <div>
    <div className="flex my-10">
      
    </div>
    <p className="text-[#1C3988] text-[26px] font-
semibold text-start mx-2 my-5">
      Зеленая миля
    </p>
    <p className="text-gray text-[16px] font-semibold
text-start m-2">
      The Green Mile, 1999, 189 мин.
    </p>
    <p className="text-gray text-[16px] font-semibold
text-start m-2">
      США • драма Режиссёр: Фрэнк Дарабонт
    </p>
    <p className="text-gray text-[16px] font-semibold
text-start m-2">
      В ролях: Том Хэнкс, Дэвид Морс
    </p>
    <p className="text-[16px] mx-2 my-6">
      Бухгалтер Энди Дюфрейн обвинён в убийстве
      собственной жены и её
      любовника. Оказавшись в тюрьме под названием
      Шоушенк, он
      сталкивается с жестокостью и беззаконием,
      царящими по обе стороны
      решётки. Каждый, кто попадает в эти стены,
      становится их рабом до
      конца жизни. Но Энди, обладающий живым умом и
      доброй душой, находит
      подход как к заключённым, так и к охранникам,
      добиваясь их особого к
      себе расположения.
    </p>
  </div>
}

```

```

        </p>
        <div className="mx-2 flex w-[40%] justify-between
items-center">
            <div>
                <p>Рейтинг от пользователей</p>
                <Rating
                    name="simple-controlled"
                    value={userRating}
                    onChange={ (event, newValue) => {
                        setUserRating(newValue);
                    }}
                />
            </div>
            <div>
                <p>Рейтинг от рецензентов</p>
                <Rating
                    name="simple-controlled"
                    value={reviewerRating}
                    onChange={ (event, newValue) => {
                        setReviewerRating(newValue);
                    }}
                />
            </div>
        </div>
    </div>
    <ReviewList />
    <CommentList />
</div>
}

```

Листинг А.31– Файл home.jsx

```

import { useNavigate } from 'react-router-dom';
import { BaseFilmCard } from '../components/card/base-film-card';

let films = [1, 2, 3, 4, 5];
export const Home = () => {
    const nav=useNavigate()
    return (
        <div className="flex flex-col items-center">
            {films.map(i => (
                <BaseFilmCard key={i} onClick={()=>nav(`/films/${i}`)} />
            ))}
        </div>
    );
};

```

Листинг А.32– Файл lk-user.jsx

```

import { Rating } from "@mui/material";
import { useState } from "react";
import { Input, ReviewCard } from "../components";
import { Button } from "../components";

export const LkUser = () => {
    const [userRating, setUserRating] = useState(2)
    const [showReviewBlock, setShowReviewBlock] = useState(false)

```

```

const [sended, setSended] = useState(false)
function send(){
  setSended(true)
  setShowReviewBlock(false)
}
function becomeReviewer(){
  if(!sended){
    setShowReviewBlock(!showReviewBlock)
  }else{
    setShowReviewBlock(false)
  }
}
return <div className="flex justify-center">
  <div className="flex w-[900px] border border-gray-400 m-5 p-10
rounded-md shadow-md">
    
    <div className="ml-10">
      <p className="text-[#1C3988] text-[27px] font-
semibold text-start mx-2">Петрова Мария Андреевна</p>
      <p className="text-gray-400 text-[27px] font-
semibold text-start mx-2">MariMari55</p>
      <p className="text-gray-400 text-[24px] text-start
mx-2">marigu205@gmail.com</p>
      <Button name={` ${sended?'Запрос в обработке':'Стать
рецензентом'}}`} className={` ${sended?'btn-query':'btn-lk'}}`}
onClick={becomeReviewer} />
      <Rating
        className="mx-2"
        name="simple-controlled"
        value={userRating}
        onChange={(event, newValue) => {
          setUserRating(newValue);
        }}
      />
    </div>
  </div>
  {
    showReviewBlock &&
    <div className="w-[25%] flex flex-col items-center
border border-gray-400 m-5 p-10 rounded-md shadow-md">
      <p className="text-[#1C3988] text-[35px] font-
semibold text-start mx-2 mb-4">Стать рецензентом</p>
      <div>
        <Input placeholder="ссылка" />
        <Button name="отправить запрос" onClick={send}
      />
      </div>
    </div>
  }
</div>
}

```

Листинг А.33– Файл lk.jsx

```
import { Rating } from "@mui/material";
import { useState } from "react";
import { ReviewCard } from "../..components";

export const Lk = () => {
  const [userRating, setUserRating] = useState(4)
  return <div className="flex flex-col items-center">
    <div className="flex border border-gray-400 m-5 p-10 rounded-
md shadow-md">
      
      <div className="ml-10">
        <p className="text-[#1C3988] text-[27px] font-semibold
text-start mx-2">Иванов Сергей Андреевич</p>
        <p className="text-gray-400 text-[27px] font-
semibold text-start mx-2">sergeiIvanov75</p>
        <p className="text-gray-400 text-[22px] text-start
mx-2">ivanov@gmail.com</p>
        <p className="text-gray-400 text-[22px] text-start
mx-2">Ученая степень: Доктор философии (Ph.D.)</p>
        <p className="text-gray-400 text-[22px] text-start
mx-2 mb-5">Место работы: Университет Пензы, Кафедра журналистики</p>
        <Rating
          className="mx-2"
          name="simple-controlled"
          value={userRating}
          onChange={(event, newValue) => {
            setUserRating(newValue);
          }}
        />
      </div>
    </div>
    <div className="border border-gray-400 m-5 p-10 rounded-md
shadow-md">
      <p className="text-[#1C3988] text-[25px] font-semibold
text-start mx-2">История Рецензирования</p>
      <div className="flex">
        <ReviewCard author="sergeiGorelov75"/>
        <ReviewCard author="sergeiGorelov75"/>
        <ReviewCard author="sergeiGorelov75"/>
      </div>
    </div>
  </div>
}
```

Листинг А.34– Файл register.jsx

```
import { Button, Input } from '../..components'
import Close from '../..assets/xmark-solid.svg'
import { useState } from 'react';
```

```

import { registerUserSchema } from '../helpers/schema';

export const RegisterPage = () => {
  const [user, setUser] = useState({ email: '', password:
'', nickname: '' })
  const [dataError, setDataError] = useState([]);
  const getUser = () => {
    registerUserSchema.validate(user).then(() => {
      setDataError([]);
      console.log(user)
    })
    .catch((error) => {
      setDataError(error.errors);
    });
  }
  return <div className='login'>
    <div className='login-form'>
      <div className='close-icon' >
        <img src={Close} alt="" />
      </div>
      <p className='login-title'>Регистрация</p>
      <Input placeholder='Почта' value={user.email}
setValue={e => { setUser({ ...user, email: e.target.value }) }}/>
      <Input placeholder='Никнейм' value={user.nickname}
setValue={e => { setUser({ ...user, nickname: e.target.value }) }}/>
      <Input placeholder='Пароль' value={user.password}
setValue={e => { setUser({ ...user, password: e.target.value }) }} />
      <div className="text-[12px] text-[#dd2909] font-
semibold">
        {dataError.map((i) => i)}
      </div>
      <Button name='Зарегистрироваться' onClick={getUser} />
    </div>
  </div>
}

```

Листинг А.35– Файл login.jsx

```

import { Button, Input } from '../components'
import Close from '../assets/xmark-solid.svg'
import './login.css'
import { useState } from 'react';
import { schema } from '../helpers/schema';

export const LoginPage = () => {
  const [user, setUser] = useState({ email: '', password: '' })
  const [dataError, setDataError] = useState([]);
  const getUser = () => {
    schema.validate(user).then(() => {
      setDataError([]);
      console.log(user)
    })
    .catch((error) => {
      setDataError(error.errors);
    });
  }
}

```

```

return <div className='login'>
  <div className='login-form'>
    <div className='close-icon' >
      <img src={Close} alt="" />
    </div>
    <p className='login-title'>Вход</p>
    <Input placeholder='Почта' value={user.email}
setValue={e => { setUser({ ...user, email: e.target.value }) }}/>
    <Input placeholder='Пароль' value={user.password}
setValue={e => { setUser({ ...user, password: e.target.value }) }} />
    { /* <Input /> */ }
    <div className="text-[12px] text-[#dd2909] font-
semibold">
      {dataError.map((i) => i)}
    </div>
    <Button name='Войти' onClick={getUser} />
  </div>
</div>
}

```

Листинг A.36– Файл login-auth-slices.jsx

```

import { createAsyncThunk } from '@reduxjs/toolkit';
import { RootState } from '../store';
import { AuthData, User } from '../types';

// Создание асинхронного запроса для регистрации
export const register = createAsyncThunk<
  User, // Тип возвращаемого значения
  AuthData, // Тип аргумента
  { state: RootState } // Тип дополнительной информации
>('auth/register', async (authData, thunkAPI) => {
  // Запрос к серверу для регистрации пользователя
  const response = await fetch('/api/auth/register', {
    method: 'POST',
    body: JSON.stringify(authData),
    headers: {
      'Content-Type': 'application/json',
    },
  });

  // Возврат данных пользователя, если регистрация прошла успешно
  if (response.ok) {
    return await response.json();
  }

  // Ошибка, если регистрация не удалась
  return thunkAPI.rejectWithValue(new Error('Registration failed'));
});

// Создание асинхронного запроса для авторизации
export const login = createAsyncThunk<
  User, // Тип возвращаемого значения
  AuthData, // Тип аргумента
  { state: RootState } // Тип дополнительной информации
>('auth/login', async (authData, thunkAPI) => {

```

```

// Запрос к серверу для авторизации пользователя
const response = await fetch('/api/auth/login', {
  method: 'POST',
  body: JSON.stringify(authData),
  headers: {
    'Content-Type': 'application/json',
  },
});

// Возврат данных пользователя, если авторизация прошла успешно
if (response.ok) {
  return await response.json();
}

// Ошибка, если авторизация не удалась
return thunkAPI.rejectWithValue(new Error('Login failed'));
});

// Создание асинхронного запроса для выхода из системы
export const logout = createAsyncThunk<
  void, // Тип возвращаемого значения
  undefined, // Тип аргумента
  { state: RootState } // Тип дополнительной информации
>('auth/logout', async () => {
  // Запрос к серверу для выхода из системы
  await fetch('/api/auth/logout');
});

```

Листинг А.37– Файл lk-slices.jsx

```

// Создание асинхронного запроса для обновления данных пользователя
export const updateProfile = createAsyncThunk<
  User, // Тип возвращаемого значения
  Partial<User>, // Тип аргумента
  { state: RootState } // Тип дополнительной информации
>('user/updateProfile', async (userData, thunkAPI) => {
  // Запрос к серверу для обновления данных пользователя
  const response = await fetch('/api/user/updateProfile', {
    method: 'PUT',
    body: JSON.stringify(userData),
    headers: {
      'Content-Type': 'application/json',
    },
  });

  // Возврат данных пользователя, если обновление прошло успешно
  if (response.ok) {
    return await response.json();
  }

  // Ошибка, если обновление не удалось
  return thunkAPI.rejectWithValue(new Error('Profile update failed'));
});

// Создание асинхронного запроса для удаления пользователя
export const deleteUser = createAsyncThunk<
  void, // Тип возвращаемого значения

```

```

        undefined, // Тип аргумента
        { state: RootState } // Тип дополнительной информации
>('user/deleteUser', async () => {
    // Запрос к серверу для удаления пользователя
    await fetch('/api/user/deleteUser', {
        method: 'DELETE',
    });
});

// Создание асинхронного запроса для получения отзывов пользователя
export const getReviews = createAsyncThunk<
    Review[], // Тип возвращаемого значения
    undefined, // Тип аргумента
    { state: RootState } // Тип дополнительной информации
>('user/getReviews', async () => {
    // Запрос к серверу для получения отзывов пользователя
    const response = await fetch('/api/user/getReviews');

    // Возврат отзывов, если запрос прошел успешно
    if (response.ok) {
        return await response.json();
    }
    // Запрос не удалось выполнить
    return [];
});

// Создание асинхронного запроса для повышения уровня пользователя
export const verifyUser = createAsyncThunk<
    User, // Тип возвращаемого значения
    undefined, // Тип аргумента
    { state: RootState } // Тип дополнительной информации
>('user/verifyUser', async () => {
    // Запрос к серверу для повышения уровня пользователя
    const response = await fetch('/api/user/verifyUser', {
        method: 'PUT',
    });
});

```

Листинг А.38– Файл lk.jsx

```

import { Rating } from "@mui/material";
import { useState } from "react";
import { ReviewCard } from "../components";

export const Lk = () => {
    const [userRating, setUserRating] = useState(4)
    return <div className="flex flex-col items-center">
        <div className="flex border border-gray-400 m-5 p-10 rounded-
md shadow-md">
            
            <div className="ml-10">
                <p className="text-[#1C3988] text-[27px] font-semibold
text-start mx-2">Иванов Сергей Андреевич</p>

```

```

        <p className="text-gray-400 text-[27px] font-
semibold text-start mx-2">sergeiIvanov75</p>
        <p className="text-gray-400 text-[22px] text-start
mx-2">ivanov@gmail.com</p>
        <p className="text-gray-400 text-[22px] text-start
mx-2">Ученая степень: Доктор философии (Ph.D.)</p>
        <p className="text-gray-400 text-[22px] text-start
mx-2 mb-5">Место работы: Университет Пензы, Кафедра журналистики</p>
        <Rating
            className="mx-2"
            name="simple-controlled"
            value={userRating}
            onChange={(event, newValue) => {
                setUserRating(newValue);
            }}
        />
    </div>
</div>
<div className="border border-gray-400 m-5 p-10 rounded-md
shadow-md">
    <p className="text-[#1C3988] text-[25px] font-semibold
text-start mx-2">История Рецензирования</p>
    <div className="flex">
        <ReviewCard author="sergeiGorelov75"/>
        <ReviewCard author="sergeiGorelov75"/>
        <ReviewCard author="sergeiGorelov75"/>
    </div>
</div>
</div>
}

```