

Учреждения образования «БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационных систем и технологий
Специальность 1-40 05 01 Информационные системы и технологии
Специализация 1-40 05 01-03 Информационные системы и технологии (изда-
тельско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
ДИПЛОМНОГО ПРОЕКТА НА ТЕМУ:**

Обучающийся _____ 4 курс, 1 группа _____ А.Н. Халалеев
курс, группа _____ дата _____ подпись _____ И. О. Ф.

Руководитель
дипломного проекта _____ ст. преп. _____ Г.Л. Тимонович
должность, ученая степень, ученое звание _____ дата _____ подпись _____ И. О. Ф.

И.о. заведующего
кафедры _____
_____ Е.А. Блинова
должность, ученая степень, ученое звание дата подпись И. О. Ф.

Консультант _____ ст. преп. _____
 должность, ученая степень, ученое звание _____ дата _____ подпись А.С. Соболевский
 И. О. Ф.

Нормоконтролер _____ асс. _____ В.А. Алешаускас
должность, ученая степень, ученое звание _____ дата _____ И. О. Ф.

Дипломный проект защищен с оценкой _____

Председатель ГЭК _____ В.К. Дюбков
 должность, ученая степень, ученое звание _____ дата _____ подпись _____ И. О. Ф.

Минск 2025

Оглавление

Реферат	5
Abstract	6
Введение.....	7
1 Постановка задачи и аналитический обзор аналогов.....	8
1.1 Описание предметной области	8
1.2 Обзор аналогов	8
1.2.1 Веб-приложение «mooon.by»	9
1.2.2 Веб-приложение «bycard»	10
1.2.3 Веб-приложение «afisha.relax.by».....	11
1.2.4 Анализ по обзору аналогов	13
1.3 Постановка задачи и функциональные требования	13
1.4 Выводы по разделу	14
2 Проектирование веб-приложения.....	15
2.1 Обзор средств разработки	16
2.1.1 Технология Nest.js	17
2.1.2 Технология TypeORM.....	18
2.1.3 Библиотека React	19
2.1.4 База данных PostgreSQL	20
2.1.5 Язык SQL.....	20
2.2 Архитектура приложения.....	21
2.3 Проектирование серверной части	22
2.4 Проектирование клиентской части	23
2.5 Пользовательские роли.....	24
2.6 Проектирование базы данных	26
2.7 Описание алгоритма покупки билета на фильм	29
2.8 Описание алгоритма проверки билетов.....	30
2.9 Выводы по разделу	31
3 Разработка веб-приложения	32
3.1 Разработка серверной части.....	32
3.1.1 Подключение и реализация базы данных	33
3.1.2 Аутентификация и авторизация.....	35
3.1.3 Обработка запросов и валидация.....	35
3.1.4 Обработка ошибок.....	37
3.1.5 Покупка и проверка билетов	37
3.2 Разработка клиентской части.....	39
3.2.1 Взаимодействие с глобальным хранилищем состояний	42
3.2.2 Обращение к REST API	43

					ДП 00.00 ПЗ			
		ФИО	Подпись	Дата	Оглавление			
Разраб.		Халалеенко А.Н.						
Пров		Тимонович Г.Л.						
Н. контр.		Алешаускас В.А.						
Утв.		Блинова Е.А.			БГТУ 1-40 05 01, 2025			

3.2.3 Разработка модуля контроля билетов	44
3.2.4 Использование встроенных компонентов Mantine	45
3.3 Выводы по разделу	46
4 Тестирование веб-приложения	47
4.1 Тестирование страниц авторизации и регистрации	47
4.2 Тестирование добавления фильма	49
4.3 Тестирование добавления сеанса на фильм	50
4.4 Тестирование проверки билетов	51
4.5 Выводы по разделу	51
5 Руководство пользователя.....	52
5.1 Пошаговое руководство	52
5.2 Выводы по разделу	61
6 Техничко-экономическое обоснование работы	62
6.1 Общая характеристика разрабатываемого программного средства	62
6.2 Исходные данные для проведения расчетов	62
6.3 Методика обоснования цены	63
6.3.1 Определение объема программного средства	64
6.3.2 Расчет основной заработной платы	65
6.3.3 Расчет дополнительной заработной платы	65
6.3.4 Расчет отчислений в Фонд социальной защиты населения и по обязательному страхованию.....	66
6.3.5 Расчет суммы прочих прямых затрат	66
6.3.6 Расчет общепроизводственных и общехозяйственных расходов	67
6.3.7 Сумма расходов на разработку программного средства.....	67
6.3.8 Расходы на сопровождение и адаптацию	68
6.3.9 Полная себестоимость	68
6.3.10 Определение цены, оценка эффективности.....	68
6.4 Вывод по разделу	71
Заключение	73
Список использованных источников	74
Приложение А. Диаграмма развертывания	76
Приложение Б. Диаграмма вариантов использования	77
Приложение В. Логическая схема базы данных	78
Приложение Г. Блок-схема алгоритма покупки билета.....	79
Приложение Д. Блок-схема алгоритма проверки билета.....	80
Приложение Е. Реализация функций сервисов	81
Приложение Ж. Реализация классов-валидаторов	85
Приложение И. Реализация проверки билетов.....	101
Приложение К. Главная страница приложения	103
Приложение Л. Таблица расчетов экономических показателей	104

Реферат

Пояснительная записка содержит 74 страницы, 39 рисунков, 15 таблиц, 27 литературных источников, 23 листинга, 9 приложений.

REACT, MANTINE, NESTJS, MULTER, POSTGRESQL, REPOSITORY, AXIOS, TYPEORM, JAVASCRIPT, EMBLA

Цель дипломного проектирования: разработка веб-приложения кинотеатра «Нм». Веб-приложение ориентировано как на большие сети кинотеатров с множеством залов, так и на не большие частные кинотеатры.

Пояснительная записка дипломного проекта состоит из реферата на русском и английском языках, содержания, введения, шести глав, заключения, списка использованных источников и графической части.

Первая глава описывает анализ аналогов по теме приложения и формирование основных задач дипломного проекта.

Во второй главе рассматривается разработка веб-приложения и подробно описываются выбранные архитектурные концепции и решения, принятые в процессе проектирования. Рассмотрены шаги и этапы, которые были предприняты для создания эффективной архитектуры приложения.

Третья глава излагает подробности и особенности процесса разработки веб приложения. В этой главе представлено описание таблиц базы данных, где при водится информация о структуре данных, связях между таблицами и используемых полях. Особое внимание уделяется проектированию базы данных, обеспечению целостности данных.

Четвертая глава посвящена тестированию приложения. В ней рассматривается как положительное, так и негативное тестирование.

В пятой главе представлено руководство пользователя, где объясняются особенности работы приложения. Также в этой главе описываются функциональные возможности приложения.

В шестом разделе приводится расчет экономических показателей для разработанного программного средства.

В заключении подведены результаты разработки приложения.

Объем графической части составляет 1,5 листа А1.

					БГТУ 00.00 ПЗ		
		ФИО	Подпись	Дата	Реферат		
Разраб.		Халалеев А.Н.					
Пров.		Тимонович Г.Л.					
Н. контр.		Алешаускас В.А.					
Утв.		Блинова Е.А.			БГТУ 1-40 05 01, 2025		
					Лит.	Лист	Листов
					У	1	1

Abstract

The explanatory note contains 74 pages, 39 figures, 15 tables, 27 literary sources, 23 listings, and 9 appendices.

REACT, MANTINE, NESTJS, MULTER, POSTGRESQL, REPOSITORY, AXIOS, TYPE ORM, JAVASCRIPT, EMBLA

Main goal: to develop a web application of the cinema "Hm". The web application is aimed at both large cinema chains with multiple halls and small private cinemas. The explanatory note of the graduation project consists of an abstract in Russian and English, a table of contents, an introduction, six chapters, a conclusion, a list of sources used and a graphic part.

The first section describes the analysis of analogues on the topic of the application and the formation of the main objectives of the graduation project.

The second section examines the development of a web application and describes in detail the selected architectural concepts and decisions made during the design process. The steps and stages that have been taken to create an effective application architecture are considered.

The third section outlines the details and specifics of the web application development process. This section describes database tables, which provide information about the data structure, relationships between tables, and fields used. Special attention is paid to database design and ensuring data integrity.

The fourth chapter presents a test case with conducted testing and demonstrates the system's behavior in various situations.

The fifth chapter provides a user's guide that explains how the application works. This chapter also describes the functionality of the application.

The sixth chapter includes a cost analysis of software development and an assessment of its economic efficiency.

In conclusion, the results of the application development are summarized.

The volume of the graphic part is 1,5 A1 sheets.

					БГТУ 00.00 ПЗ		
		ФИО	Подпись	Дата	Abstract		
Разраб.		Халалеенко А.Н.					
Пров.		Тимонович Г.Л.					
Н. контр.		Алешаускас В.А.					
Утв.		Блинова Е.А.			БГТУ 1-40 05 01, 2025		
					Лит.	Лист	Листов
					У	1	1

Введение

В современном мире индустрия кино остается важной частью культурной и развлекательной жизни общества. Кинотеатры предлагают зрителям возможность насладиться новыми фильмами на большом экране, погрузиться в атмосферу визуального искусства и провести время с семьей или друзьями. Развитие цифровых технологий открывает новые пути для повышения качества сервиса и взаимодействия с клиентами в данной сфере.

Среди основных трудностей, с которыми сталкиваются посетители кинотеатров, можно выделить неудобство в поиске информации о сеансах, сложности с покупкой билетов и нехватку интерактивных функций на существующих онлайн-платформах. Пользователи часто вынуждены переходить между несколькими сайтами или использовать устаревшие интерфейсы, что снижает удобство и эффективность взаимодействия с сервисом.

Современные технологии позволяют создавать интуитивно понятные и функциональные веб-приложения, которые обеспечивают быстрый доступ к расписанию показов, онлайн-бронированию мест на сеансах и дополнительной информации о фильмах. Это особенно актуально в условиях растущей цифровизации и привычки пользователей к удобным онлайн-сервисам, доступным с любого устройства.

Актуальность темы дипломного проектирования заключается в необходимости внедрения эффективных веб-инструментов для автоматизации процессов и повышения качества обслуживания в кинотеатрах. С ростом числа пользователей, предпочитающих онлайн-взаимодействие, повышается спрос на современные решения, способные удовлетворить эти потребности.

Целью дипломного проектирования является разработка веб-приложения для управления кинотеатром, а также удобной покупки билетов, «Нм».

Для реализации приложения поставлены следующие задачи:

- провести обзор аналогов и прототипов;
- анализ архитектуры веб-приложения;
- проектирование архитектуры веб-приложения;
- проектирование структуры базы данных;
- разработка веб-приложения;
- тестирование веб-приложения;
- создание руководства пользователю;
- экономические расчеты.

					ДП 00.00 ПЗ		
		ФИО	Подпись	Дата	Введение		
Разраб.	Халалеенко А.Н.						
Пров.	Тимонович Г.Л.						
Н. контр.	Алешаускас В.А.						
Утв.	Блинова Е.А.						
					Лит.	Лист	Листов
					У	1	1
					БГТУ 1-40 05 01, 2025		

1 Постановка задачи и аналитический обзор аналогов

1.1 Описание предметной области

Услуги кинотеатра включают не только показ фильмов, но и создание комфорта для посетителей, начиная с бронирования билетов и заканчивая просмотром фильма. В современном мире, где цифровые технологии играют ключевую роль, кинотеатры также адаптируются к новым стандартам обслуживания клиентов.

Бронирование билетов через интернет или мобильные приложения становится все более популярным. Это позволяет пользователям выбирать удобное время и место просмотра, избегая очередей в кассах. Технологии анализа данных помогают кинотеатрам лучше понимать предпочтения клиентов, что способствует улучшению качества обслуживания и предложений.

Кинотеатры, использующие современные IT-системы, могут предлагать клиентам широкий спектр услуг, включая онлайн-бронирование мест, выбор мест в зале, оплату билетов с помощью банковских карт или электронных кошельков, получение персонализированных рекомендаций по фильмам. Эти возможности делают посещение кинотеатра более комфортным для пользователей.

Администраторы систем управления кинотеатрами обеспечивают бесперебойную работу всех сервисов, от обработки платежей до актуализации информации о прокате. Использование специализированных программных решений позволяет анализировать данные о посещаемости, предпочтениях аудитории и эффективности маркетинговых кампаний, что помогает принимать обоснованные решения по развитию бизнеса.

Комплексные системы управления кинотеатрами включают не только функциональность бронирования билетов, но и инструменты для управления контентом, планирования расписания, учета доходов и расходов, а также анализа эффективности работы кинотеатра. Такие системы позволяют кинотеатрам оставаться конкурентоспособными, предлагая клиентам высококачественный сервис и интересный контент. Все это делает сферу администрирования кинотеатра и покупки билетов онлайн перспективной и востребованной.

1.2 Обзор аналогов

Для создания нового программного продукта для просмотра и заказа билетов на фильмы, а также администрирования кинотеатра, важно провести тщательный анализ существующих решений в этой сфере.

Анализ должен охватывать как достоинства, так и недостатки текущих продуктов и их аналогов, чтобы на основе полученной информации сформулировать четкие требования к проектируемому программному средству.

					ДП 01.00 ПЗ		
		ФИО	Подпись	Дата	1 Постановка задачи и аналитический обзор аналогов		
Разраб.	Халалеев А.Н.						
Пров.	Тимонович Г.Л.						
Н. контр.	Алешаускас В.А.						
Утв.	Блинова Е.А.						
					Лит.	Лист	Листов
					У	1	7
					БГТУ 1-40 05 01, 2025		

Основная цель такого подхода – учесть опыт успешных и менее удачных разработок, внести улучшения или изменения, которые сделают новый продукт более привлекательным и удобным для пользователей.

При выборе программных продуктов для анализа следует обращать внимание на те, которые наиболее тесно связаны с областью применения будущего программного продукта. Это может включать различные платформы для онлайн-просмотра фильмов, сервисы для бронирования билетов в кинотеатрах, а также мобильные приложения, предлагающие подобные услуги.

Источниками информации для анализа могут служить электронные базы данных, доступные в интернете, а также отзывы пользователей и профессиональные обзоры в специализированных изданиях.

Процесс анализа начинается с изучения функциональности и интерфейса каждого из исследуемых продуктов, чтобы определить, какие элементы могут быть полезны для нового проекта, а какие требуют доработки или замены. Особое внимание уделяется вопросам безопасности и удобства использования, поскольку эти аспекты напрямую влияют на восприятие продукта пользователями.

На основе собранной информации формируется список требований к проекту, включающий функциональные, технические и пользовательские требования. Этот список затем используется в процессе разработки приложения, обеспечивая его соответствие ожиданиям целевой аудитории и конкурентоспособность на рынке.

Таким образом, проведение детального анализа существующих программных средств позволяет не только избежать повторения ошибок прошлых разработок, но и создать продукт, который будет отвечать потребностям и ожиданиям пользователей, предлагая им новые возможности и улучшенные условия для просмотра и заказа билетов на фильмы.

1.2.1 Веб-приложение «mooon.by»

Интерфейс веб-приложения «mooon.by» представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс mooon.by

Веб-приложение для покупки билетов tooon.by [1]. Сервис не особо популярен из-за слабой рекламы и продвижения, вы даже можете не найти его на первых страницах поиска (данный сервис служит только для покупок билетов в кинотеатры silverscreen и tooon), но этот сервис обладает максимальной функциональностью и красотой дизайна. На данном сайте достаточно минималистичный дизайн, в то же время, привлекающий внимание пользователей.

Также предоставляется возможность оставить отзывы о фильмах и кинотеатрах, поделиться своим мнением с другими пользователями. Выбор места в зале сопровождается выводом стоимости этого места, что довольно удобно. Помимо этого, сервис предлагает пользователям не только детальные инструкции, но и широкий спектр категорий мест для удобства и персонализированного подхода к каждому пользователю. Этот подход показывает глубину понимания потребностей пользователей, помогая им находить идеальное решение для каждого, по его средствам и запросам. Удачный сервис в плане реализации, функционала и дизайна. Из минусов только малая известность.

1.2.2 Веб-приложение «bycard»

Веб-приложение для покупки билетов bycard.by [2] – один из самых популярных и востребованных сайтов по продаже билетов на любые мероприятия, проходящие на территории Беларуси, на нем можно найти интересующие вас спектакли, фильмы, другие события и купить/забронировать билеты. Сервис отличается интуитивно понятным интерфейсом с удобной системой фильтрации мероприятий по датам, жанрам и локациям. Особенно стоит отметить функцию мгновенного бронирования, которая позволяет закрепить места на 15 минут без immediate оплаты, что дает пользователям время на окончательное решение.

Интерфейс данного сервиса представлен на рисунке 1.2.

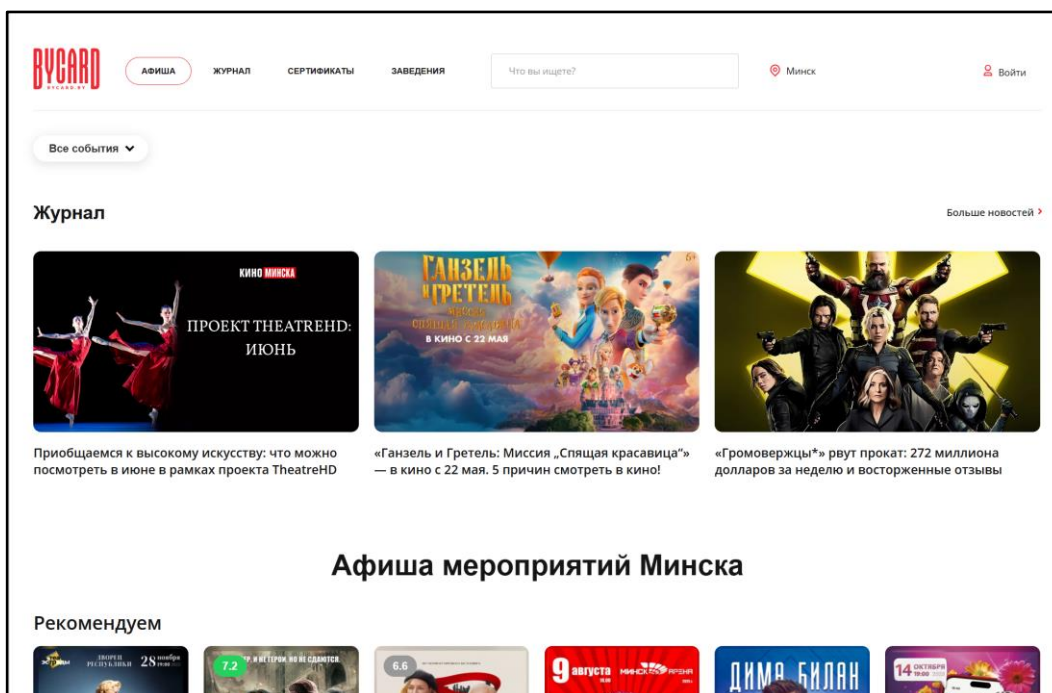


Рисунок 1.2 – Интерфейс Vycard.by

Функционал сервиса включает в себя:

- выбор досуга;
- выбор мест проведения мероприятий;
- выбор организаторов;
- фильтры;
- поисковую систему;
- выбор региона;
- разбивка по мероприятиям.

Отличный сервис, функционал реализован на 100%, есть все нужные сервисы, недостатки крайне малы и нишевы. Например, сам по себе сервис берет процент с каждого проданного билета, за счет чего не все кинотеатры хотят сотрудничать с ним, а также стоимость покупки билета онлайн повышается. Еще небольшим минусом можно считать, что при возврате билета деньги будут возвращены только через несколько дней, но это скорее минус системы оплаты, а не самого сайта.

1.2.3 Веб-приложение «afisha.relax.by»

Главная страница веб-приложения представлена на рисунке 1.3.

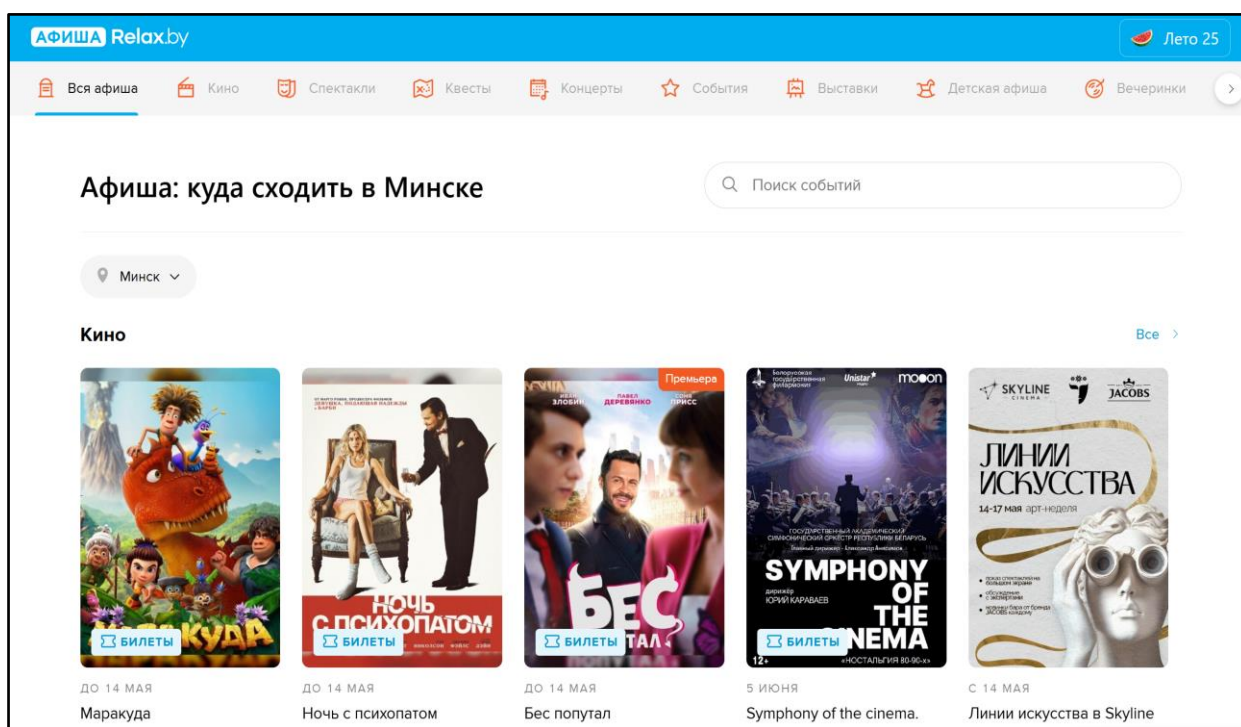


Рисунок 1.3 – Главная страница веб-приложения «afisha.relax.by»

Это веб-приложение представляет собой удобный инструмент для тех, кто ищет новые возможности для отдыха и развлечений [3]. Он предлагает широкий спектр мероприятий, от культурных событий до спортивных соревнований, делая выбор наиболее подходящего варианта максимально простым и быстрым. Возможности для пользователей разнообразны, начиная от просмотра каталога мероприятий и заканчивая прямой регистрацией на интересные вас события.

Одним из ярких преимуществ этого сервиса является особое внимание, уделяемое группам студентов и пенсионеров. Предоставление скидок этим категориям пользователей подчеркивает социальную ответственность сервиса и делает его более доступным для широкого круга людей.

Скидки для студентов и пенсионеров являются важной составляющей стратегии привлечения новых пользователей и укрепления лояльности существующих. Для студентов это возможность экономить на развлечениях и мероприятиях, что особенно актуально во время учебы, когда бюджет часто ограничен. Для пенсионеров же такие скидки позволяют участвовать в культурных и развлекательных мероприятиях без значительных финансовых затрат, что способствует их активному отдыху и социализации. На рисунке 1.4 представлен пример скидок.

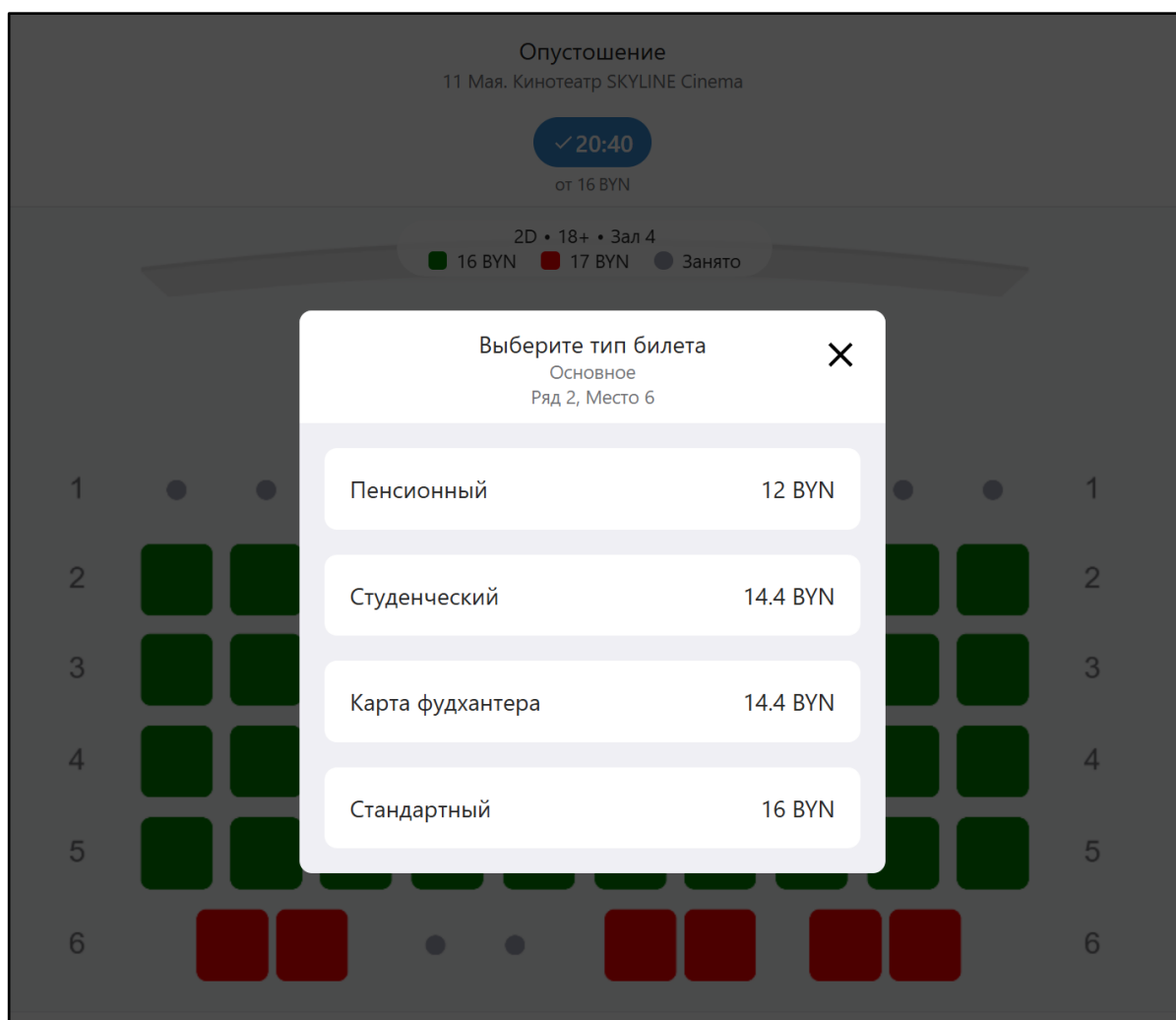


Рисунок 1.4 – Пример скидок в веб-приложении afisha.relax.by

В целом, сервис похож на предыдущий, с похожим функционалом, но более социализированный и направленный на все группы населения. Сервис является эволюционным продолжением предыдущего, но с более широким спектром функциональных возможностей и ориентацией на все слои населения. Кроме того, сервис имеет более простой и интуитивно понятный интерфейс, что делает его доступным для пользователей с любым уровнем технической грамотности.

1.2.4 Анализ по обзору аналогов

Проанализировав данные сервисы, стоит отметить, что общие характеристики это: возможность оформления заказа, поиск фильмов по названию, разбиение фильмов по жанрам, но также ни у одного из них не реализована возможность добавить фильмы во вкладку «Любимые», что я считаю существенным минусом с точки зрения персонализации пользовательского опыта. На основе проведенного анализа можно выделить ключевые характеристики, которые следует учесть при разработке нового приложения для покупки билетов на фильмы. Важными элементами приложения должны быть не только удачная реализация, функциональность и привлекательный дизайн, но и эффективные меры по рекламе и продвижению для повышения его известности среди целевой аудитории. Особое внимание следует уделить широкому функционалу, включая возможность выбора досуга, сохранения понравившихся фильмов, а также разработку удобной поисковой системы с поддержкой умных фильтров.

Кроме того, важными аспектами являются не просто возможность оформления заказа, но и оптимизированный поиск фильмов по различным критериям, и продуманная функция разбиения фильмов по жанрам с учетом современных тенденций кинопроката.

Исключение недостатков, таких как комиссии с продажи билетов и повышение стоимости при онлайн-покупке, а также предоставление удобных механизмов возврата билетов, будет способствовать улучшению общего восприятия приложения пользователями. Исходя из этого был сформирован функционал приложения:

- возврат билетов;
- изменение личных данных;
- выбор одного или нескольких мест;
- покупка билетов;
- просмотр информации о фильмах и поиск по названию;
- добавление фильма в избранные;
- просмотр жанров;
- просмотр статистики по фильмам;
- добавление фильмов/сеансов/залов;
- удаление фильмов/сеансов;
- блокировка/разблокировка пользователей.

Такой комплексный подход к разработке функционала обеспечит удобство использования системы как для пользователей, так и для администраторов.

1.3 Постановка задачи и функциональные требования

Оценка существующих аналогов и их функциональных возможностей дает четкое основание для формулировки требований к разрабатываемой системе. Функциональные требования детально описывают необходимые возможности программного обеспечения, которые должны быть реализованы

разработчиками для обеспечения эффективного выполнения пользователями своих задач в рамках установленных бизнес-целей.

Поскольку дипломный проект разрабатывается без конкретного заказчика, в качестве бизнес-требований были приняты обобщенные требования к подобным решениям, выведенные на основе анализа рыночных потребностей. Функциональные требования систематизированы в виде четких спецификаций необходимых возможностей, включая методы их реализации и способы взаимодействия с системой.

Основной целью приложения является разработка оптимальной системы с интуитивно понятным интерфейсом для взаимодействия администрации кинотеатра с клиентами, со всем необходимым функционалом, достаточным и полноценным для управления кинотеатром, а также понятным и доступным для клиента. На основе поставленной цели можно сформировать основной функционал системы:

- приложение должно поддерживать роли: администратор, пользователь;
- к общим функциям авторизованных пользователей относятся: изменение данных профиля и пароля, просмотр доступных сеансов на фильмы, покупка билетов, добавление фильмов в «Любимые», просмотр фильмов по категориям, просмотр своих билетов, поиск фильмов;
- к функциям администратора относятся: просмотр, создание, удаление фильмов/сеансов, так же добавление залов, блокировка/разблокировка пользователей, просмотр статистики билетов по фильмам, проверка билетов перед сеансом.

Таким образом, решение о разработке веб-приложения для организации продажи билетов в кинотеатр является обоснованным и рациональным. Реализация данного проекта создаст основу для будущего масштабирования системы с учетом развития технологий и изменений в индустрии развлечений.

1.4 Выводы по разделу

В разделе представлены теоретические основы, описывающие предметную область, а также результаты поиска и анализа аналогичных веб-приложений. Данный анализ позволяет выявить преимущества и недостатки существующих решений, чтобы при разработке собственного приложения избежать подобных ошибок.

В частности, было проведено исследование следующих аспектов:

- основы предметной области, включая понятия, модели и принципы, лежащие в основе разработки веб-приложений;
- анализ существующих решений, включая их функциональные возможности, преимущества и недостатки;
- определение функциональных требований к разрабатываемому приложению, включая требования к пользовательскому интерфейсу, функциональности и производительности.

Результаты анализа и поиска помогли сформулировать функциональные требования к разрабатываемому приложению, которые будут учитываться при его разработке. Это позволит создать приложение, которое будет отвечать потребностям пользователей и иметь конкурентные преимущества на рынке.

2 Проектирование веб-приложения

Процесс проектирования архитектуры программного обеспечения состоит в проектировании структуры всех его компонент, функционально связанных с решаемой задачей, включая сопряжения между ними и требования к ним.

Архитектура программного обеспечения в традиционном смысле включает определение всех модулей программ [4], их иерархии и сопряжения между ними и данными. Веб-приложение – это решение, в основе которого лежит взаимодействие браузера и веб-сервера. Такие приложения являются кроссплатформенными сервисами, доступными с любого современного устройства, и не привязаны к архитектуре сети.

Этап проектирования веб-приложения можно логически разбить на три основополагающие части:

- проектирование базы данных;
- проектирование серверной части;
- проектирование клиентской части.

Использование приложения начинается с пользовательского интерфейса, который реализован при помощи React [5]. Пользователь выполняет действие, посылает запрос по адресу развернутого на сервере NestJS [6] веб-API приложения [7]. В свою очередь управление переходит к сервисам NestJS и модели. Если запрос требует работы с данными, используется TypeORM [8]. PostgreSQL [9] выдает необходимые данные. TypeORM передает эти данные веб-API, который формирует ответ пользователю.

Важно отметить, что подобная архитектура является классическим примером «N-layer» архитектуры, где каждый слой зависит только от предыдущего компонента [10]. Следует отметить, что многоуровневой архитектурой часто обозначают два не совсем связанных понятия: n-layer и n-tier. И layer, и tier, как правило, обозначаются словом «Уровень», иногда по отношению к «Layer» еще употребляется слово «Слой». Однако в обоих случаях уровни будут разного порядка. Также взаимодействие происходит последовательно. Так, к примеру, последний компонент не способен взаимодействовать с первым.

Всего выделяют три абстрактных уровня: уровень представления, необходимых для работы пользователя (располагаются интерфейсы, которые пользователь видит и с которыми взаимодействует), уровень бизнес-логики, в котором осуществляется основная обработка данных, и уровень доступа к данным, необходимый для их хранения и использования [11]. При этом следует отметить, что крайние уровни не могут взаимодействовать между собой, то есть уровень представления не может напрямую обращаться к базе данных и даже к уровню доступа к данным, а только через уровень бизнес-логики.

					ДП 02.00 ПЗ		
		ФИО	Подпись	Дата	2 Проектирование веб-приложения		
Разраб.	Халалеев А.Н.						
Пров.	Тимонович Г.Л.						
Н. контр.	Алешаускас В.А.						
Утв.	Блинова Е.А.						
					Лит.	Лист	Листов
					У	1	17
					БГТУ 1-40 05 01, 2025		

2.1 Обзор средств разработки

JavaScript (JS) – это высокоуровневый, динамический и интерпретируемый язык программирования, который в основном используется для создания веб-приложений и добавления интерактивных элементов на веб-сайты [12]. Он был создан в 1995 году Бренданом Эйхом, когда он работал в Netscape Communications Corporation.

Одна из ключевых особенностей JavaScript – его способность выполняться на стороне клиента в браузере, что позволяет создавать динамические и интерактивные веб-страницы. Это означает, что JavaScript может реагировать на ввод пользователя в реальном времени, что делает его идеальным языком для создания веб-приложений, требующих взаимодействия с пользователем.

JavaScript также является универсальным языком, который может использоваться для широкого спектра применений за пределами разработки веб-приложений. Например, он может использоваться для серверной разработки с помощью Node.js [13], разработки игр с помощью фреймворков, таких как Phaser [14], и даже для разработки мобильных и десктопных приложений с помощью фреймворков, таких как Electron [15].

Другая важная особенность JavaScript – его способность работать с асинхронным кодом. Это означает, что JavaScript может обрабатывать несколько задач одновременно без блокировки выполнения других операций. Это особенно полезно для создания высокопроизводительных веб-приложений, требующих быстрой реакции.

JavaScript также является прототипным языком, что означает, что он использует прототипы вместо классов для создания объектов. Это позволяет использовать более гибкую и динамическую объектно-ориентированную парадигму программирования, поскольку объекты могут получать свойства и методы от других объектов во время выполнения.

JavaScript является популярным языком, имеющим большую и активную сообщество разработчиков. Это сообщество создало широкий спектр библиотек, фреймворков и инструментов, которые облегчают разработку сложных веб-приложений. Некоторые из самых популярных фреймворков и библиотек JavaScript включают React, Angular [16], Vue.js [17] и jQuery [18].

Вкратце, JavaScript – мощный и универсальный язык программирования, который в основном используется для создания веб-приложений и добавления интерактивных элементов на веб-сайты. Его способность выполняться на стороне клиента в браузере, работать с асинхронным кодом и использовать прототипы для объектно-ориентированного программирования делают его идеальным языком для создания высокопроизводительных веб-приложений, требующих быстрой реакции.

Кроме того, его большое и активное сообщество разработчиков создало широкий спектр библиотек, фреймворков и инструментов, которые облегчают разработку сложных веб-приложений. Благодаря этим ресурсам разработчики могут использовать готовые решения для множества задач, таких как

аутентификация пользователей, обработка данных, интеграция с базами данных, работа с API, управление состоянием приложений и многое другое.

2.1.1 Технология Nest.js

Nest.js является одним из самых популярных фреймворков для создания серверных приложений на основе Node.js. Он построен на экосистеме Express.js и предоставляет ряд дополнительных возможностей, таких как модульная структура приложения, внедрение зависимостей, декораторы и другие функции, которые делают разработку приложений более простой и эффективной.

Nest.js использует декораторы для определения контроллеров и провайдеров. Контроллеры отвечают за обработку входящих запросов, а провайдеры предоставляют логику приложения. Nest.js также поддерживает использование различных баз данных, таких как MongoDB [19], PostgreSQL, MySQL [20].

Одним из ключевых преимуществ Nest.js является его совместимость с Angular, фреймворком для клиентской разработки от Google. Angular и Nest.js используют общий язык разработки TypeScript [21], что позволяет легко интегрировать клиентскую и серверную части приложения.

Nest.js также предоставляет поддержку микросервисной архитектуры, что позволяет создавать сложные приложения, состоящие из множества взаимосвязанных сервисов. Кроме того, Nest.js имеет активное и растущее сообщество разработчиков, что обеспечивает быстроту использования и доступность множества сторонних модулей и библиотек.

В целом, Nest.js является мощным и гибким фреймворком для разработки серверных приложений на основе Node.js. Его функциональные возможности, совместимость с Angular и активная сообщество делают его одним из лучших выборов для разработки сложных веб-приложений. Основное взаимодействие происходит по протоколу HTTP [22].

При вызове операций с контроллера важным моментом играет routing. Routing позволяет подсистеме WebApi связать Uri адрес и конкретную операцию из контроллера используя внутренние механизмы.

С помощью NestJS можно легко создавать REST-сервисы. До него создание сервисов было более сложным, и эта технология предоставляет широкие возможности хостинга и позволяет выполнять следующие задачи:

- создавать веб-приложения и службы;
- использовать избранные средства разработки в Windows, macOS и Linux;
- выполнять развертывания в облаке или локальной среде;
- запускать в Node.js.

NestJS предоставляет следующие преимущества:

- единое решение для создания пользовательского веб-API;
- разработанное приложение удобно тестируется;
- модульная архитектура приложения;
- поддержка внедрения зависимостей;
- возможность использования декораторов для контроллеров и провайдеров;
- поддержка баз данных, таких как MongoDB, PostgreSQL, MySQL;

- совместимость с фреймворком Angular;
- активное и растущее сообщество разработчиков.

NestJS проект предоставляет следующие возможности размещения:

- Node.js;
- Docker [23].

Кроме того, NestJS также может быть развернут на различных cloud-платформах, таких как:

- Heroku;
- DigitalOcean;
- AWS Elastic Beanstalk;
- Google App Engine;
- Railway;
- Digital Ocean App Platform.

Каждая из этих платформ имеет свои преимущества и недостатки, и выбор за висит от конкретных требований проекта.

Например, AWS Elastic Beanstalk – это управляемый сервис, который позволяет быстро развернуть и управлять приложениями в облаке AWS без необходимости беспокоиться об инфраструктуре, которая запускает эти приложения. Digital Ocean App Platform – это сервис PaaS, который позволяет быстро создавать, развертывать и масштабировать приложения и статические сайты. Railway – это платформа, которая позволяет легко развернуть приложение в облаке за несколько минут, поддерживая множество языков программирования, включая Java, PHP, Deno, Python, Swift, Go и другие.

2.1.2 Технология TypeORM

TypeORM является объектно-реляционной картой (ORM) для TypeScript и JavaScript, которая облегчает взаимодействие с базами данных с помощью объектно-ориентированного синтаксиса. Это может упростить доступ к базе данных и улучшить организацию кода в дипломном проекте.

Одним из преимуществ использования TypeORM является его поддержка многих баз данных, таких как PostgreSQL, MySQL, MariaDB, SQLite, Microsoft SQL Server и других. Это означает, что вы можете выбрать базу данных, которая наилучшим образом подходит для вашего проекта, и TypeORM обеспечит согласованный интерфейс для работы с ней. Это может сэкономить время и усилия, поскольку вы не будете обучаться новому синтаксису или API для каждой базы данных, с которой вы работаете.

TypeORM также предоставляет мощный набор функций для работы с базами данных, включая поддержку сущностей, репозиторий и отношений. Сущности – это классы, которые сопоставляются с таблицами базы данных, а репозитории предоставляют способ взаимодействия с этими сущностями. Отношения позволяют определять связи между сущностями, облегчая работу со сложными структурами данных.

Другое преимущество использования TypeORM – его интеграция с TypeScript. TypeScript – это статически типизированное надмножество

JavaScript, которое добавляет необязательные типы, классы и модули к языку. Используя TypeScript с TypeORM, вы можете воспользоваться системой типов TypeScript для раннего выявления ошибок и улучшения удобочитаемости кода.

Кратко, TypeORM – это мощная и гибкая ORM, которая может упростить доступ к базе данных и улучшить организацию кода в проекте. Его поддержка многих баз данных, сущностей, репозиторий и отношений делает его отличным выбором для работы со сложными структурами данных, а его интеграция с TypeScript может улучшить качество кода и удобочитаемость.

2.1.3 Библиотека React

React – это мощная библиотека JavaScript для создания пользовательских интерфейсов, которая уже более десяти лет остается одним из самых популярных инструментов фронтенд-разработки. Она позволяет создавать повторно используемые компоненты пользовательского интерфейса и эффективно обновлять, и рендерить эти компоненты благодаря использованию виртуального DOM. React позволяет постепенно вводить его в свой проект, используя только необходимую функциональность в любой момент времени, что делает его идеальным выбором как для небольших виджетов, так и для сложных SPA-приложений.

React имеет богатую экосистему с многими передовыми функциями и концепциями для исследования, такими как контекст и ссылки. API-документация и глоссарий предоставляют подробную информацию об этих темах, а также ответы на часто задаваемые вопросы. React позволяет создавать переиспользуемые компоненты, которые могут быть использованы в различных частях приложения. Это упрощает разработку и поддержку приложений, поскольку позволяет избежать дублирования кода и повышает модульность.

Компоненты в React представляют собой независимые единицы, которые могут принимать входные данные в виде пропсов и отображать соответствующий пользовательский интерфейс. Эти компоненты могут быть использованы в других компонентах или страницах, что позволяет легко изменять и обновлять пользовательский интерфейс без необходимости изменять основной код.

В целом, использование компонентов в React позволяет создавать легкие, модульные и поддерживаемые приложения, уменьшая дублирование кода и упрощая разработку.

Во время работы над приложением библиотека React помогла создать динамичный, удобный для поддержки и масштабируемый пользовательский интерфейс. Кроме того, React позволяет легко интегрировать сторонние библиотеки и API, что расширяет функциональность приложения без лишних затрат времени. Его компонентная архитектура и эффективный рендеринг делают его отличным выбором для современной веб-разработки.

Используя возможности React, есть возможность создавать высококачественный и профессиональный проект, демонстрирующий навыки и компетенции разработчика, а также ускорять разработку благодаря переиспользованию существующих компонентов.

2.1.4 База данных PostgreSQL

PostgreSQL является мощной и надежной реляционной базой данных, которая используется в данном дипломном проекте. Она отличается масштабируемостью, безопасностью и функциональностью, что делает ее идеальным выбором для управления данными приложений.

Одна из ключевых особенностей PostgreSQL – это возможность создавать собственные расширения и модули, которые могут быть интегрированы непосредственно в базу данных. Это позволяет нам настраивать и расширять функциональность PostgreSQL в соответствии с нашими потребностями, что особенно полезно в больших и сложных проектах.

Кроме того, PostgreSQL поддерживает множество различных типов данных, включая стандартные типы, такие как целые числа, вещественные числа и строки, а также более сложные типы, такие как JSON, XML и географические данные. Это позволяет нам работать с разнообразными данными в единой системе, что упрощает разработку и поддержку приложений.

В плане производительности PostgreSQL также демонстрирует отличные результаты. База данных может обрабатывать тысячи запросов в секунду, а также поддерживать сотни и даже тысячи одновременных соединений. Кроме того, PostgreSQL поддерживает различные механизмы репликации и отказоустойчивости, которые позволяют нам создавать отказоустойчивые и масштабируемые системы.

Однако, несмотря на все свои преимущества, PostgreSQL также имеет некоторые ограничения и небольшие недостатки. Например, база данных может требовать больше ресурсов, чем другие решения, такие как NoSQL-базы данных. Кроме того, PostgreSQL может быть сложнее в настройке и управлении, особенно для начинающих пользователей.

В целом, PostgreSQL – мощная и функциональная реляционная база данных, которая отлично подходит для многих приложений. Ее возможность настраивать и расширять функциональность, а также поддержка разнообразных типов данных делают ее идеальным выбором для сложных и масштабных проектов. Однако, необходимо учитывать ее ограничения и требования к ресурсам при выборе решения для своего дипломного проекта.

2.1.5 Язык SQL

Общепринятым стандартом языка работы с реляционными базами данных в настоящее время является язык структурированных запросов – SQL [24]. Это универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. Вопреки существующим заблуждениям, SQL является информационно-логическим языком, а не языком программирования.

SQL основывается на реляционной алгебре. Язык SQL делится на три части:

- операторы определения данных;
- операторы манипуляции данными (Insert, Select, Update, Delete);

- операторы определения доступа к данным.

При всех своих изменениях SQL остается единственным механизмом связи между прикладным программным обеспечением и базой данных.

Преимущества использования SQL:

- независимость от конкретной СУБД (несмотря на наличие диалектов и различий в синтаксисе, в большинстве своем тексты SQL-запросов, содержащие DDL и DML, могут быть достаточно легко перенесены из одной системы управления базами данных в другую);

- наличие стандартов (наличие стандартов и набора тестов для выявления совместимости и соответствия конкретной реализации SQL общепринятому стандарту только способствует стабилизации языка);

- декларативность (с помощью SQL программист описывает только то, какие данные нужно извлечь или модифицировать. То, каким образом это сделать, решает СУБД непосредственно при обработке SQL-запроса).

Недостатки:

- сложность. Хотя SQL и задумывался как средство работы конечного пользователя, в конце концов он стал настолько сложным, что превратился в инструмент программиста;

- отступления от стандартов. Многие разработчики СУБД вносят изменения в язык SQL, применяемый в разрабатываемой СУБД. Таким образом появляются специфичные для каждой конкретной СУБД диалекты языка SQL.

2.2 Архитектура приложения

Архитектура программного обеспечения – совокупность важнейших решений об организации программной системы. Основные задачи разработки архитектуры программного средства:

- выделение программных подсистем и отображение на них внешних функций (заданных во внешнем описании) программного средства;
- определение способов взаимодействия между различными выделенными программными подсистемами.

Всего в приложении используется 3 основных сервиса:

- сервис, реализующий клиентскую часть приложения;
- сервис, реализующий серверную часть приложения;
- TypeORM – сервис, реализующий доступ к базе данных.

Диаграмма развертывания – это тип диаграммы в UML, которая демонстрирует физическую структуру системы, включая аппаратное обеспечение и программное обеспечение, на котором она работает. Она показывает, как программное обеспечение развертывается на физических устройствах, называемых узлами, и как эти узлы взаимодействуют друг с другом для выполнения функций системы. Диаграмма развертывания помогает визуализировать архитектуру системы, созданную для физической реализации, и моделирует распределение программного обеспечения по физическим узлам в распределенных системах.

Для разрабатываемого проекта была спроектирована диаграмма развертывания, которая представлена на рисунке 2.1 и в приложении А.

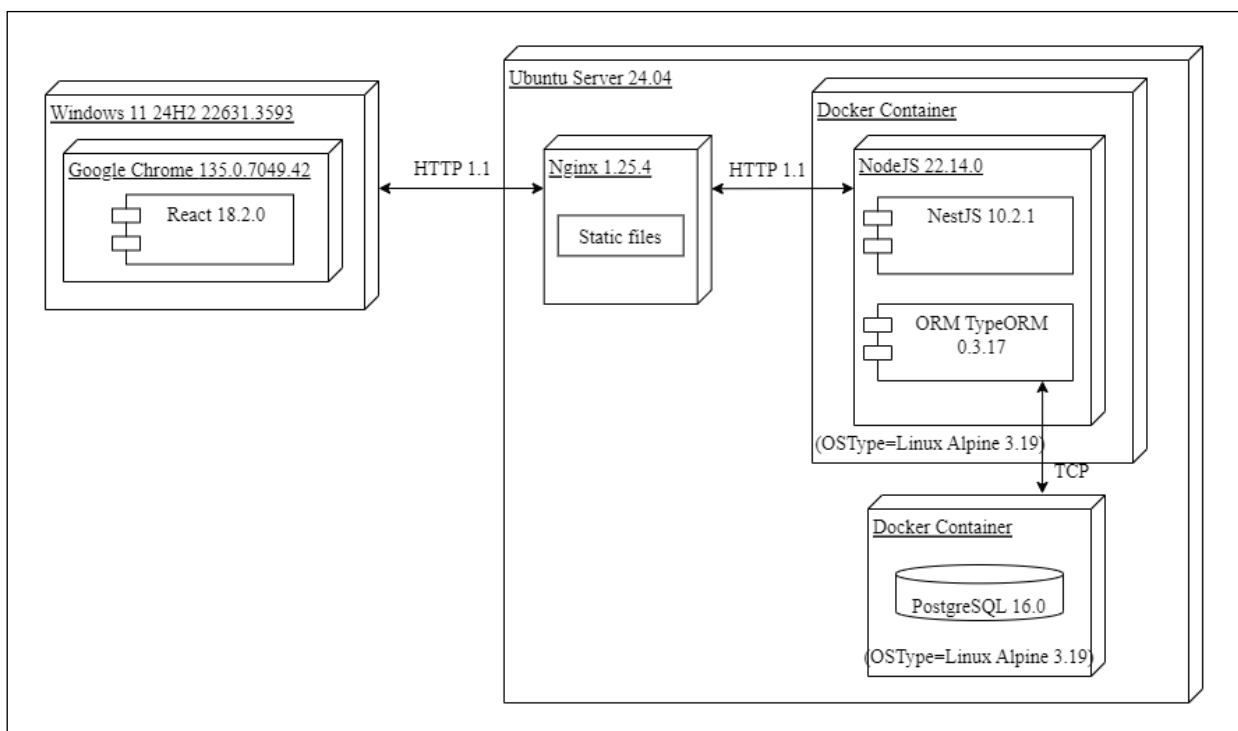


Рисунок 2.1 – Диаграмма развертывания

Каждый компонент программного средства функционирует независимо от других, что обеспечивает модульность системы и дает значительные преимущества при расширении функциональности и техническом обслуживании. Взаимодействие между слоями организовано по четкой схеме: входящие клиентские запросы первоначально обрабатываются серверной частью на базе NestJS, которая выполняет валидацию и бизнес-логику. Затем управление передается клиентскому приложению на React, где формируется динамическое графическое представление данных. Это представление проходит через систему маршрутизации для определения конечной точки вывода и только после этого окончательно отображается в веб-браузере пользователя.

2.3 Проектирование серверной части

Для реализации серверной части клиент-серверного приложения был выбран фреймворк NestJS. NestJS – это гибкий и производительный фреймворк для создания серверных приложений на платформе Node.js, который поддерживает TypeScript и предлагает готовую архитектуру для построения масштабируемых решений. Он предоставляет удобный набор функций и инструментов, включая модульную систему и встроенную поддержку dependency injection, позволяющих разработчикам быстро и эффективно создавать веб-серверы и API. NestJS предоставляет множество расширений и плагинов, которые облегчают разработку различных функциональностей, таких как маршрутизация, обработка запросов, обработка сессий, аутентификация и авторизация, и многое другое.

Архитектура серверной части была разработана на основе «слоистой архитектуры» с некоторыми изменениями для повышения удобства

использования и производительности. Слоистая архитектура представляет собой проверенный подход к проектированию программного обеспечения, при котором приложение разбивается на логические слои или уровни, каждый из которых выполняет определенные функции и взаимодействует только с соседними уровнями через четко определенные интерфейсы.

Слоистая архитектура разделяет систему на несколько уровней, каждый из которых имеет свои конкретные задачи и строгие границы взаимодействия. Внешний уровень обрабатывает пользовательский интерфейс и входящие HTTP-запросы, уровень представления преобразует данные в подходящий формат для клиента, уровень бизнес-логики содержит основную логику приложения и бизнес-правила, а уровень данных работает с базами данных или другими источниками данных, обеспечивая их целостность и согласованность. Такой продуманный подход способствует модульности, упрощает разработку и тестирование отдельных компонентов, а также обеспечивает надежную изоляцию различных частей системы.

Для связи с базой данных была использована ORM TypeORM. TypeORM – это современный ORM (Object-Relational Mapping), предназначенный для связи приложений с базами данных. ORM является программным инструментом, который позволяет разработчикам работать с данными в базе данных, используя объектно-ориентированный подход, вместо прямой работы с SQL-запросами. TypeORM предоставляет удобный и выразительный API для выполнения операций чтения, записи, обновления и удаления данных в базе данных. Он позволяет разработчикам взаимодействовать с базой данных, используя язык программирования, с которым они уже знакомы (например, JavaScript или TypeScript), а не прямо работая с SQL-запросами.

2.4 Проектирование клиентской части

Для проектирования клиентской части приложения была выбрана такая библиотека, как React. React – это JavaScript-библиотека с открытым исходным кодом для создания пользовательских интерфейсов. Она позволяет разрабатывать динамичные и высокопроизводительные веб-приложения благодаря своей компонентной архитектуре и виртуальному DOM, что значительно упрощает управление состоянием интерфейса. Кроме того, React имеет активное сообщество разработчиков, что обеспечивает постоянное обновление библиотеки и обширную базу готовых решений.

Для визуального оформления и организации компонентов на странице сайта была использована библиотека Mantine UI [25]. Этот фреймворк не только предоставляет готовые UI-элементы, но и включает в себя мощные инструменты для кастомизации. Mantine UI – это популярный инструмент для разработки веб-интерфейсов, который предлагает набор предварительно созданных компонентов, стилей и скриптов. Его модульная структура позволяет легко интегрировать только необходимые компоненты, уменьшая итоговый размер бандла. Он облегчает и ускоряет процесс создания и поддержки современных веб-страниц, которые легко адаптируются для различных устройств.

2.5 Пользовательские роли

При проектировании приложения, его серверной и клиентской части, а также реализации его функционала была учтена диаграмма вариантов использования, указанная в приложении Б.

В разрабатываемом веб-приложении присутствует 2 роли. В зависимости от роли определяются функциональные возможности в рамках разрабатываемой системы. В рамках системы существует администратор и пользователь.

Фрагмент диаграммы вариантов использования администратором приложения представлен на рисунке 2.2.

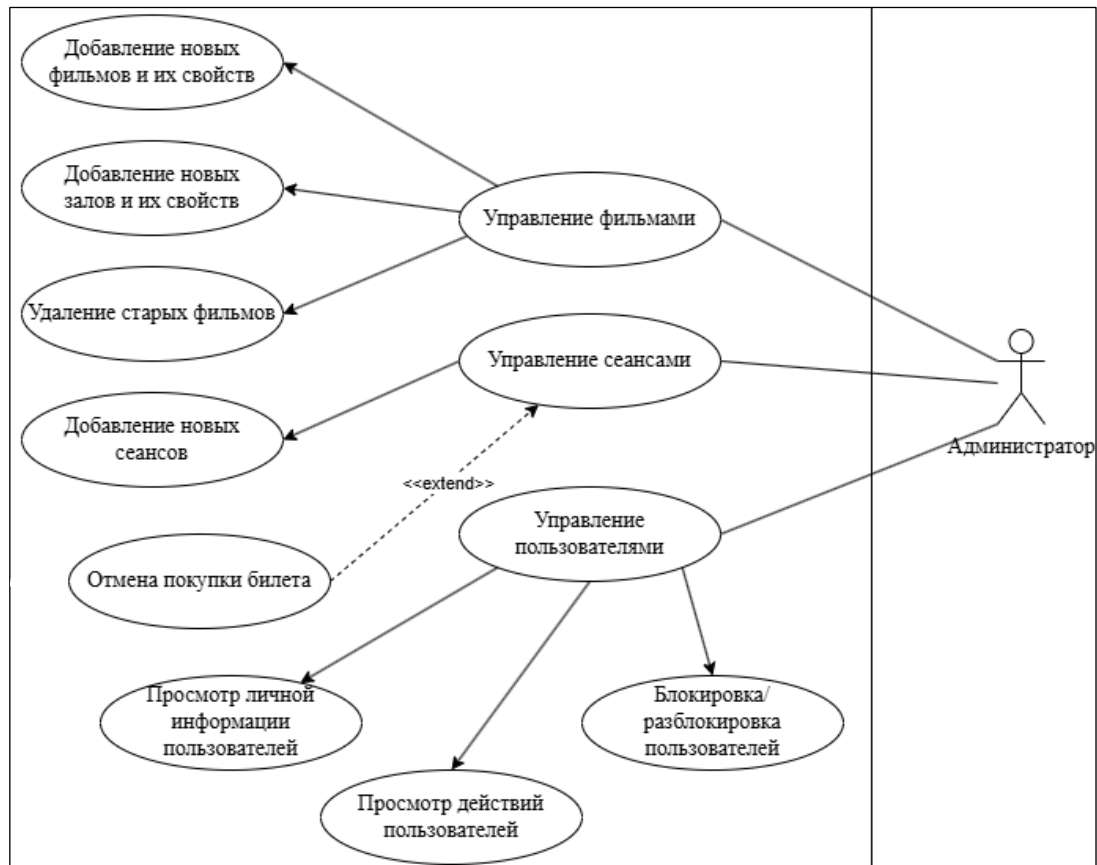


Рисунок 2.2 – Фрагмент диаграммы вариантов использования администратора

Рассмотрим функциональность, которая предоставляется администратору: данный пользователь имеет определенное количество функций и выполняет работу по поддержанию работоспособности веб-приложения. Ему доступны следующие функции:

- поиск фильмов;
- возврат билетов;
- изменение личных данных;
- покупка билетов;
- просмотр фильмов;
- добавление фильма в избранные;
- просмотр истории действий;
- добавление и удаление сеансов/фильмов;

- просмотр статистики по фильмам;
- блокировка\разблокировка пользователей;
- проверка билетов на сеанс по QR-code.

У авторизованного пользователя в соответствии и диаграммой должны быть доступен следующий функционал:

- поиск фильмов;
- возврат билетов;
- изменение личных данных;
- покупка билетов;
- просмотр фильмов;
- добавление фильма в избранные.

Описание вышеперечисленных функциональных возможностей пользователя представлено на рисунке 2.3.

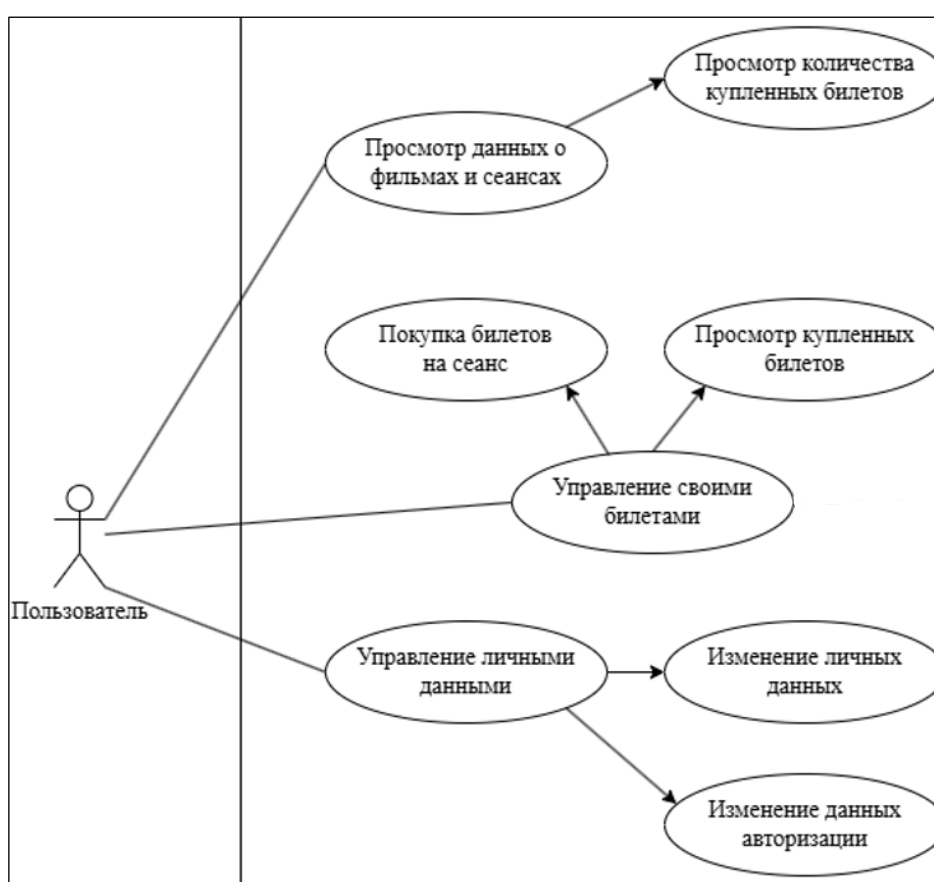


Рисунок 2.3 – Фрагмент диаграммы вариантов использования пользователя

Основные функции администратора – это проверка билетов, а основная функция пользователя – это покупка билетов. Покупка билетов происходит по следующему алгоритму: сначала пользователь выбирает билет или билеты, далее нажимает на кнопку оплатить, после этого его перенаправляет на платежную систему, пользователь вводит реквизиты своей карты и после успешной оплаты его перенаправляет обратно на страницу фильма. Проверка билетов осуществляется с помощью сканера, который использует камеру для получения данных с qr-code, после этого данные билета

сравниваются с данными из базы данных и при их совпадении пользователя пропускают на сеанс. Более подробно расписаны алгоритмы проверки билетов и их покупки в подразделах 2.7 и 2.8.

2.6 Проектирование базы данных

Для хранения данных в приложении было выбрано реляционное базовое хранилище PostgreSQL. PostgreSQL представляет собой систему управления реляционными базами данных, обладающую широким спектром функциональности для работы с данными в структурированном формате.

Выбор PostgreSQL в качестве реляционной базы данных был обусловлен его высокой производительностью и гибкостью настройки. Данная СУБД предоставляет возможности для оптимизации и настройки конфигурации, что позволяет достичь высокой скорости работы в соответствии с требованиями нашего дипломного проекта.

Схема базы данных представлена на рисунке 2.4, а также в приложении В.

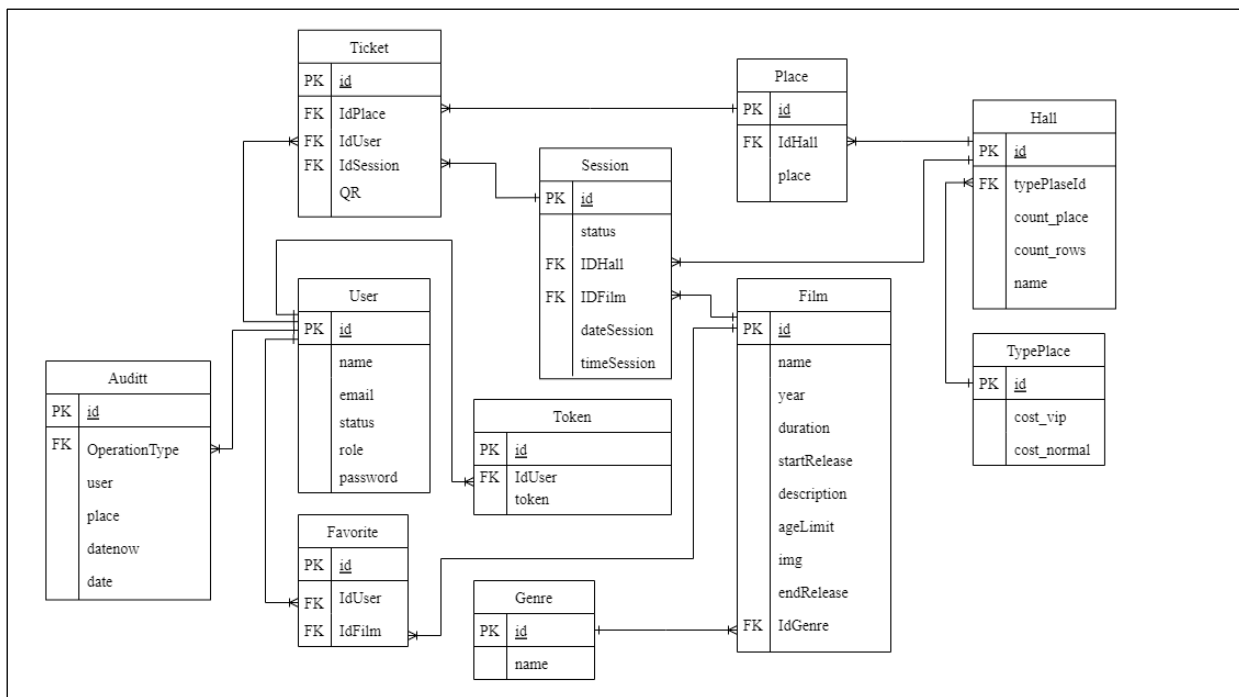


Рисунок 2.4 – Логическая схема базы данных

При проектировании данной базы данных были сознательно применены принципы нормализации, последовательно реализуются первая, вторая и третья нормальные формы (1NF, 2NF, 3NF), чтобы в итоге получить оптимальную структуру, полностью соответствующую требованиям третьей нормальной формы. Такой тщательно продуманный подход обеспечивает несколько ключевых преимуществ: во-первых, он эффективно минимизирует избыточность данных за счет правильного распределения информации между таблицами; во-вторых, предотвращает аномалии и неожиданное поведение при операциях обновления, вставки и удаления записей. Эти

характеристики особенно критичны для высоконагруженных систем, таких как платформы онлайн-продажи билетов, где целостность данных и производительность являются определяющими факторами успешной работы.

Для хранения данных приложение использует базу данных. База данных состоит из 11 таблиц: Ticket, Session, Film, Genre, Place, Hall, TypePlace, Users, Favorite, Auditt, Token. Ниже приведены их описания, структура и связи. Данные в таблицы были внесены во время тестирования.

Таблица Ticket хранит список проданных билетов на данный момент на доступные сеансы, состоит из столбцов (таблица 2.1).

Таблица 2.1 – Столбцы таблицы Ticket

Наименование	Описание	Тип
ID	Идентификатор билета, первичный ключ	число
IDSession	Идентификатор сеанса, внешний ключ для связи с таблицей «Session»	число
IDPlace	Идентификатор места, внешний ключ для связи с таблицей «Place»	число
IDUsers	Идентификатор данных пользователя, внешний ключ для связи с таблицей «Users»	число
QR	QR-code для прохода на сеанс	строка

Таблица Users данные авторизации пользователей (таблица 2.2).

Таблица 2.2 – Столбцы таблицы Users

Наименование	Описание	Тип
ID	Идентификатор пользователя, первичный ключ	число
Password	Пароль пользователя	строка
Name	Логин пользователя	строка
Email	Электронная почта пользователя	строка

Таблица Genre содержит список всех жанров (таблица 2.3).

Таблица 2.3 – Столбцы таблицы Genre

Наименование	Описание	Тип
ID	Идентификатор жанра, первичный ключ	число
Name	Название жанра	строка

Таблица Film содержит список всех фильмов, внесенных в базу (таблица 2.4).

Таблица 2.4 – Столбцы таблицы Film

Наименование	Описание	Тип
ID	Идентификатор фильма, первичный ключ	число
Name	Название фильма	строка
StartRelease	Дата начала проката фильма	дата
EndRelease	Дата окончания проката фильма	дата
Description	Описание фильма	строка
AgeLimit	Возрастное ограничение	число
IDGenre	Идентификатор жанра, внешний ключ для связи с таблицей «Genre»	число
IMG	Афиша фильма	число

Продолжение таблицы 2.4

Наименование	Описание	Тип
Year	Год выпуска фильма	число
Duration	Продолжительность фильма	время

Таблица Hall содержит информацию о залах кинотеатра (таблица 2.5).

Таблица 2.5 – Столбцы таблицы Hall

Наименование	Описание	Тип
ID	Идентификатор зала, первичный ключ	число
Name	Название зала	строка
Capacity	Общее число мест в зале	число

Таблица Session содержит сеансы фильмов, которые сейчас находятся в прокате в кинотеатре (таблица 2.6).

Таблица 2.6 – Столбцы таблицы Session

Наименование	Описание	Тип
ID	Идентификатор пользователя, первичный ключ	число
DateSession	Дата сеанса	дата
TimeSession	Время сеанса	время
IDHall	Идентификатор зала, внешний ключ для связи с таблицей «Hall»	число
IDFilm	Идентификатор фильма, внешний ключ для связи с таблицей «Film»	число

Таблица Place содержит информацию о местах в добавленных залах. Описание полей этой сущности отражено в таблице 2.7.

Таблица 2.7 – Столбцы таблицы Place

Наименование	Описание	Тип
ID	Идентификатор места, первичный ключ	число
Place	Номер места	число
IDHall	Идентификатор зала, внешний ключ для связи с «Hall»	число

Таблица TypePlace включает в себя список всех типов мест, соответствующих таблице Place (таблица 2.8).

Таблица 2.8 – Столбцы таблицы TypePlace

Наименование	Описание	Тип
ID	Идентификатор типа места	число
Cost_vip	Стоимость вип места	число
Cost_normal	Стоимость обычного места	число

Таблица Favorite включает в себя список избранных фильмов (таблица 2.9).

Таблица 2.9 – Столбцы таблицы Favorite

Наименование	Описание	Тип
ID	Идентификатор любимого фильма	число

Продолжение таблицы 2.4

Наименование	Описание	Тип
IDUsers	Идентификатор пользователя	число
IDFilm	Идентификатор фильма	число

Таблица AUDITT включает в себя данные о действиях всех пользователей приложения (таблица 2.10).

Таблица 2.10 – Столбцы таблицы AUDITT

Наименование	Описание	Тип
OperationType	Тип действия	число
FilmName	Название фильма	строка
Date	Дата изменения	дата
Datenow	Текущая дата	дата
Place	Место	строка
Users	Пользователь, выполнивший действие	строка

Таблица Token включает в себя список токенов (таблица 2.11).

Таблица 2.11 – Столбцы таблицы Token

Наименование	Описание	Тип
ID	Идентификатор токена	число
IDUsers	Идентификатор пользователя	число
Token	Токен	строка

В данном пункте были рассмотрены структуры всех таблиц базы данных. Данная структура обеспечивает корректное и эффективное хранение, а также взаимосвязь всех сущностей системы.

2.7 Описание алгоритма покупки билета на фильм

Первым шагом работы алгоритма пользователь авторизуется в приложении и ему доступны функции роли пользователя. Затем пользователь переходит на главную страницу, откуда есть возможность попасть в меню.

На начальном этапе осуществляется получение всех доступных сеансов для выбранного фильма. Далее производится перебор всех сеансов, среди которых отбираются только актуальные, то есть те, на которые еще можно приобрести билеты. После отбора подходящих сеансов начинается работа с каждым из них.

Пользователь выбирает интересующий сеанс. Для данного сеанса выполняется перебор мест в зале. Каждому месту присваивается определенная цена в зависимости от его категории. Одновременно обозначаются места, которые уже были выкуплены ранее. Это позволяет пользователю видеть доступные для бронирования места.

Следующим шагом становится выбор одного или нескольких мест для покупки. После выбора производится оформление заказа и переход к оплате. На этом этапе система проверяет, была ли оплата успешной. В случае удачного проведения платежа билеты становятся доступны в личном

кабинете пользователя. В противном случае процесс покупки можно повторить. Блок-схема алгоритма покупки билета на фильм изображена в приложении Г и на рисунке 2.5.

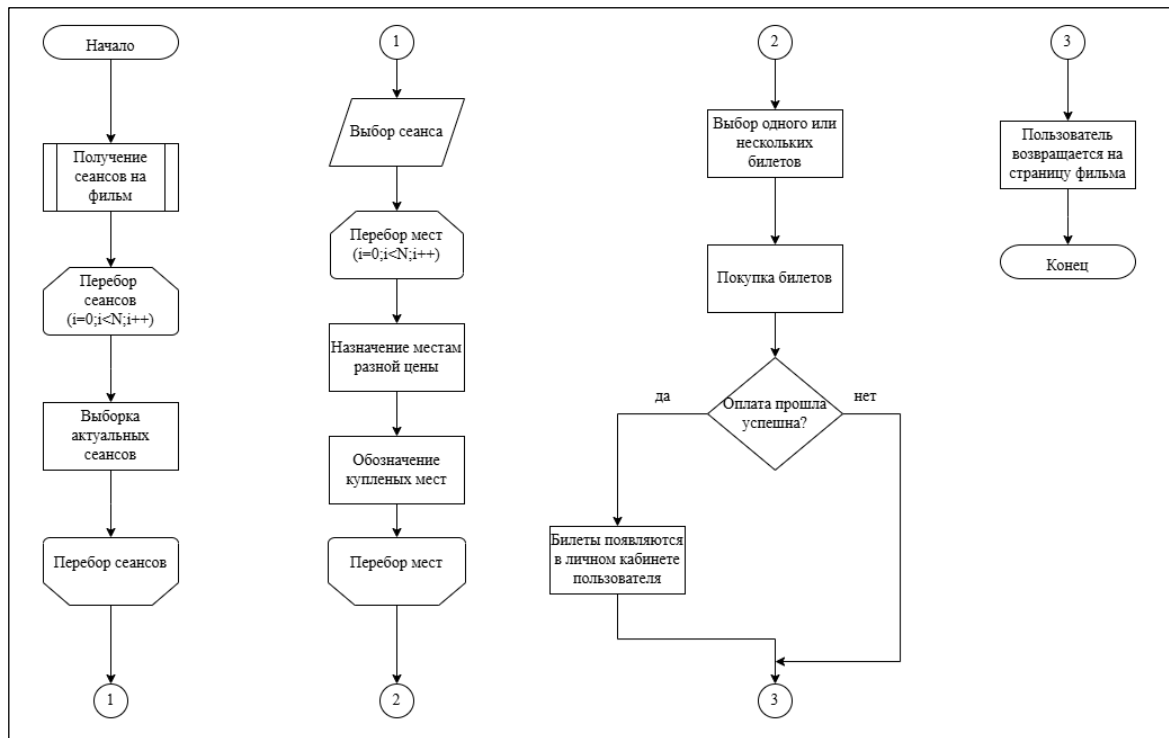


Рисунок 2.5 – Блок-схема алгоритма покупки билета на фильм

Завершением сценария является возврат пользователя на страницу фильма, где он может продолжить выбор других сеансов или фильмов.

2.8 Описание алгоритма проверки билетов

Первым шагом работы алгоритма пользователь авторизуется в приложении и ему доступны функции роли пользователя. Затем пользователь переходит на главную страницу, откуда есть возможность попасть в меню.

Процесс начинается с того, что пользователь предъявляет билет для входа на сеанс. Система инициирует сканирование QR-кода, содержащего уникальный идентификатор билета. Этот идентификатор используется для проверки его подлинности и актуальности.

После сканирования происходит загрузка полного списка билетов, оформленных на конкретный сеанс. Все билеты из этого списка перебираются с целью найти тот, который соответствует считанному QR-коду. Если такой билет найден, осуществляется его верификация.

На следующем этапе система проверяет статус найденного билета. В случае, если статус обозначен как активный, это означает, что билет еще не использовался, и пользователь получает соответствующее разрешение на вход в кинозал. После этого, чтобы предотвратить повторное использование, билет переводится в статус неактивного.

Если же при проверке выясняется, что билет уже имеет неактивный статус, система уведомляет ответственного сотрудника о недопустимости допуска пользователя на мероприятие. Это может означать, что билет уже был использован ранее или аннулирован.

Если ни один из билетов в списке не соответствует QR-коду, также формируется сообщение о невозможности допуска на сеанс. Такой сценарий может возникнуть, если пользователь предъявил поддельный или просроченный билет.

На этом процесс проверки завершается, и система фиксирует результат проверки – успешный вход или отказ в допуске. Блок-схема алгоритма проверки билетов изображена в приложении Д и на рисунке 2.6.

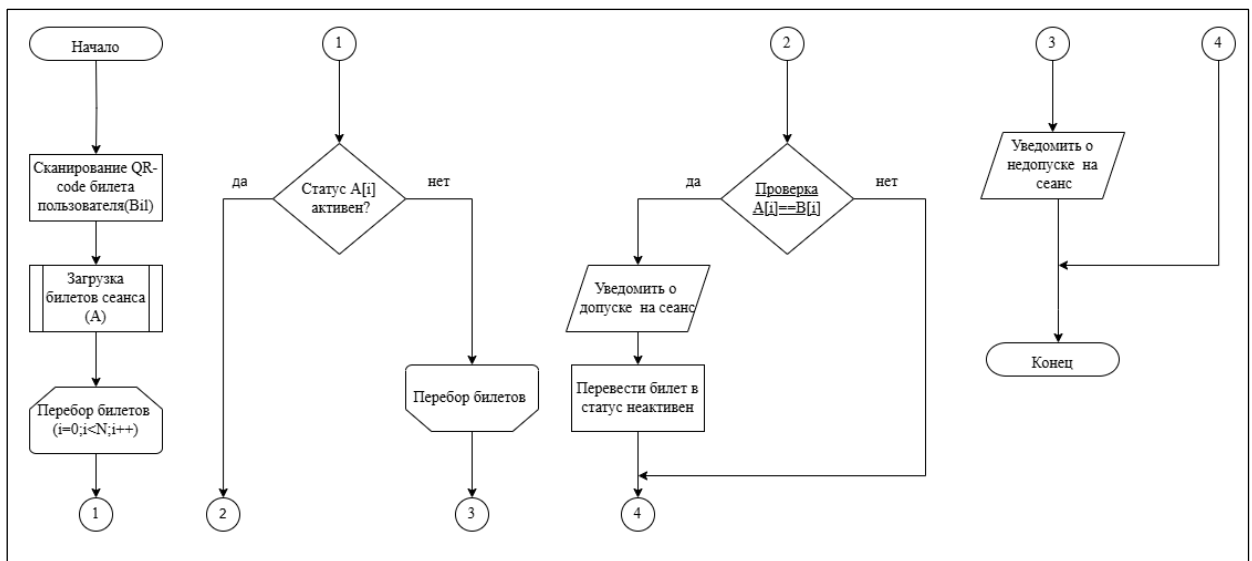


Рисунок 2.6 – Блок-схема алгоритма проверки билетов

Реализации данного алгоритма и предыдущего будут представлены в следующем разделе, где будет рассмотрена разработка веб-приложения.

2.9 Выводы по разделу

В ходе данного раздела была выполнена комплексная проработка архитектуры веб-приложения, включая проектирование клиентской и серверной частей, а также детальное построение структуры базы данных. Особое внимание уделено проектированию ключевых компонентов базы данных.

Кроме того, проведен анализ технологий, используемых в проекте. Для серверной части выбран NestJS – строго типизированный фреймворк на TypeScript, который обеспечивает высокую надежность, безопасность и масштабируемость благодаря модульной архитектуре. Клиентская часть реализуется на React, что позволяет создавать интерактивный и производительный интерфейс с эффективным рендерингом и переиспользованием компонентов.

Таким образом, проведенное проектирование заложило прочную основу для дальнейшей разработки функционального, безопасного и удобного веб-приложения онлайн-кинотеатра, способного эффективно обрабатывать данные и обеспечивать комфортное взаимодействие с пользователями.

3 Разработка веб-приложения

3.1 Разработка серверной части

В данном разделе рассматриваются следующие этапы разработки:

- подключение и реализация базы данных;
- аутентификация и авторизация;
- обработка запросов и валидация;
- обработка ошибок;
- передача картинок на сервер.

Для реализации серверной части клиент-серверного приложения использовался фреймворк NestJS описанный в главе 2.2. Для конфигурирования серверной части приложения использовался env-файл. Файл .env (Environment) является текстовым файлом, который используется для хранения конфигурационных переменных среды в проекте. Он содержит пары «ключ=значение», где каждая переменная среды имеет свое уникальное имя (ключ) и соответствующее значение. В дальнейшем эти значения используются в коде для легкой перенастройки используемых значений проекта.

В проекте используются следующие переменные среды для приложения:

- порт для запуска сервера;
- ключ для генерации access-токена;
- ключ для генерации refresh-токена;
- адрес страницы авторизации клиентского сервера;
- строка подключения к базе данных.

Файл переменных среды представлен в листинге 3.1.

```
HOST='localhost'
DATABASE_PORT='5432'
USER='Kir'
PASSWORD="Kirill-17"
DATABASE_NAME="dream_cinema_db"
EXPIRATION_TIME=1h
JWT_EXPIRES_IN=60s
JWT_REFRESH_PASS_TO_CALLBACK=true
```

Листинг 3.1 – Файл переменных среды

Для доступа к хранилищу используется синтаксис, как в листинге 3.2.

```
http://localhost:${process.env.PORT}/
```

Листинг 3.2 – Получение переменной из хранилища

					ДП 03.00 ПЗ		
		ФИО	Подпись	Дата	3 Разработка веб-приложения		
Разраб.		Халалеев А.Н.					
Пров.		Тимонович Г.Л.					
Н. контр.		Алешаускас В.А.					
Утв.		Блинова Е.А.					
					Лит.	Лист	Листов
					У	1	15
					БГТУ 1-40 05 01, 2025		

Был использован env в данном приложении по причине того, что он предоставляет возможность безопасно хранить различную авторизационную информацию. Данные, которые хранятся в env можно получить только способом выше и никак иначе. Именно с этой целью там и хранятся данные подключения к базе данных и другие данные.

3.1.1 Подключение и реализация базы данных

Для подключения базы данных используется ORM «TypeORM», которая описана выше. TypeORM была добавлена при помощи пакетного менеджера npm.

Для разработки базы данных был использован подход Code First.

Code First – это подход к разработке программного обеспечения, при котором модели данных определяются в первую очередь, а затем база данных и код доступа к данным генерируются автоматически на основе этих моделей. Данный подход удобен своей динамичностью и легкой интеграцией, что понадобилось мне в процессе разработки.

Процесс создания и использования схемы TypeORM будет продемонстрирован на основе модели пользователей (Users).

Для начала был создан класс, который будет описывать все необходимые поля, их типы данных, значения по умолчанию, связи и определения в Entity.

Пример создания класса представлен в листинге 3.3.

```
@Entity()
export class User {
  @PrimaryGeneratedColumn(UUID_NAME)
  id: string;

  @Column({ length: 50 })
  email: string;

  @Column({ length: 255 })
  password: string;

  @Column({ length: 50 })
  name: string;
  @Column({ default: 'Active', length: 10 })
  status: string;
  @Column({ default: 'User', length: 50 })
  role: string;
  @OneToMany(() => Auditt, ticket => ticket.user, { onDelete:
'CASCADE' })
  auditt: Auditt[];
}
```

Листинг 3.3 – Класс пользователей

Таким образом были созданы все классы и настроены между ними связи. Далее была выполнена миграция из TypeORM в базу данных. Создание

происходит автоматически, что позволяет нам подключить наш сервер к любой базе данных и там тут же будут созданы данные таблицы.

Далее для обращения к базе данных будут использоваться данные классы, что позволит защитить созданную базу данных.

Пример функции получения всех билетов, купленных на сеанс, использующей запрос к базе данных с использованием TypeORM приведен в листинге 3.4.

```
async GetUserSession(idsession) {
  const userinfo = await this.databaseService.ticket.find({
    where: { session: { id: idsession },
    },
    relations: ['user', 'place', 'session'],
  });
  return userinfo;
},
}
```

Листинг 3.4 – Функция получения всех билетов из базы данных

В данной функции был сделан запрос к базе данных с использованием метода `find` для получения всех записей из таблицы и получения данных из связанных моделей. Так же с помощью метода `relations` выбираются какие именно данные будут получены, а также `where` для условий.

Далее необходимо создать контроллер для каждого сервиса для получения данных из запросов и передачи необходимых данных в функции сервисов. Пример функции контроллера на получение всех билетов в листинге 3.5.

```
async GetTicketUser(idUser: string) {
  await this.databaseService.session.query('SELECT CheckSession()');
  const tick = await this.databaseService.ticket.find({
    where: { user: { id: idUser } },
    relations: ['place', 'user', 'session', 'session.film', 'session.hall'],
  });
  const filteredData = tick.filter(item => item.session.status === 'Active');
  return filteredData;
} }
```

Листинг 3.5 – Пример функции получения всех билетов в контроллере

В данной функции происходит асинхронное обращение к сервису базы данных с помощью функции `find`. Передаются условия отбора, чтобы идентификатор пользователя был таким же, как в параметрах, а также настраиваются связи с другими таблицами, для получения полной информации. После фильтрации по активности возвращается ответ. Данные действия по созданию сервисов и контроллеров с передачей параметров и обработкой значений были проделаны для всех моделей базы данных.

3.1.2 Аутентификация и авторизация

В данной главе описан механизм аутентификации и авторизации пользователей, а также представлена его реализация.

Для аутентификации используется механизм jwt-токенов [26]. Для работы с токенами был использован пакет «jsonwebtoken».

Для реализации аутентификации и авторизации был разработан соответствующий функционал, предоставляющий необходимые функции для аутентификации и также авторизации. Список функций представлен ниже:

- проверка на существование, статус блокировки пользователя в базе данных;
- генерация jwt access и refresh токенов;
- обновление access токена;
- получение данных из jwt-токена.
- использование стратегий.

Реализация функций сервисов пользователя представлена в приложении Е

При разработке приложения была заложена следующая логика. При аутентификации пользователя проверяются все необходимые данные допускающее последующую авторизацию пользователя.

Для начала осуществляются проверки на существование пользователя. После успешного прохождения всех проверок происходит генерация токенов в функции generateTokens. В данной функции идет так же обращение к функции сервиса auth generateTokens для генерации непосредственно токенов на основании данных пользователя (уникальный идентификатор пользователя и роль) и использованием соответствующих ключей для токенов.

При необходимости обновления access токена происходит запрос, который вызывает функцию refresh, это происходит через некоторый промежуток времени. В данной функции сначала происходит проверка что refresh токен был передан, после чего происходит обращение к функции сервиса сессий getRefreshToken и передача токена в нее. Функция getRefreshToken ищет в базе данных переданный токен и возвращает уникальный идентификатор пользователя, после чего с помощью функции getUserById по полученному идентификатору происходит получение данных пользователя. Далее на основании этих данных происходит генерация нового токена с его сохранением и передачей в соответствующий контроллер.

3.1.3 Обработка запросов и валидация

В данной главе происходит разработка обработчиков запросов клиента. В NestJS маршруты (routes) определяются с использованием декораторов и модулей. Маршруты представляют собой пути, по которым приложение обрабатывает входящие HTTP-запросы. В листинге 3.6 приведен пример маршрутов.

```
@Controller('film')
export class FilmController {
```

Листинг 3.6 – Маршрут контроллера фильмов

Следует обратить внимание, что в некоторых маршрутах используется проверка роли пользователя для доступа к определенным запросам и проверка авторизации пользователя.

Также данные проходят валидацию перед тем, как попасть в обработку. Я использовал Class-validator. Class-validator – это мощный и гибкий инструмент для валидации классов в TypeScript и JavaScript приложениях. Он позволяет легко определить правила валидации непосредственно в деклараторах свойств класса, что делает код более читаемым и поддерживаемым. Использование этого инструмента позволило быстро провалидировать все данные, а также защищает базу данных от sql-инъекций так как в нем есть встроенная защита. В листинге 3.7 представлен пример такой валидации.

```
import { Transform } from 'class-transformer';
import { IsDefined, IsNotEmpty, IsString, IsUUID, Length, }
from 'class-validator';

import { UUID_LENGTH } from 'common';

export class CreateFavoriteDto {
  @IsDefined({ message: 'Идентификатор должен быть указан' })
  @IsNotEmpty({ message: 'Идентификатор не должен быть пустым' })
  @IsString({ message: 'Идентификатор должен быть строкой' })
  @Transform(({ value }) => value.trim()) // Если нужно убрать
про
белы в начале и конце
  @IsUUID(4, {
message: 'Некорректный формат UUID'
})
  @Length(UUID_LENGTH, UUID_LENGTH, {
    message: `Идентификатор должен содержать ${UUID_LENGTH}
симво
лов`,
  })
  IdFilm: string;
  @IsDefined({ message: 'Идентификатор должен быть указан' })
  @IsNotEmpty({ message: 'Идентификатор не должен быть пустым' })
  @IsString({ message: 'Идентификатор должен быть строкой' })
  @Transform(({ value }) => value.trim()) // Если нужно убрать
про
белы в начале и конце
  @IsUUID(4, { message: 'Некорректный формат UUID' })
  @Length(UUID_LENGTH, UUID_LENGTH, {
    message: `Идентификатор должен содержать ${UUID_LENGTH}
символов`,
  })
  IdUser: string;
}
export class FilmController
```

Листинг 3.7 – Валидация при добавлении фильмов в любимые

Этот листинг представляет `CreateFavoriteDto`, предназначенный для валидации данных при добавлении фильма в избранное пользователем. Он проверяет, что идентификаторы фильма (`IdFilm`) и пользователя (`IdUser`) заданы, не пусты, соответствуют строковому формату UUID версии 4 и имеют строго заданную длину. Основные реализованные функции представлены в приложении Ж.

3.1.4 Обработка ошибок

В данной главе представлен функционал обработки и вывода ошибок с использованием соответствующих кодов ошибок. Принцип обработки ошибок, приведенный в данной главе, используется во всей структуре приложения.

Для упрощенной обработки ошибок и их вывода используется специальная функция, код которой приведен на листинге 3.8.

```
@Post('AddFilm')
@UseInterceptors(FileInterceptor('image'))
async creates(
  @Body() createFilmDto: CreateFilmDto,
  @UploadedFile() file: any, ) {
  if (!file) {
    throw new BadRequestException('Добавьте постер', 'img');
  }
  createFilmDto.img = file.buffer;
  return await this.filmService.create(createFilmDto); }
```

Листинг 3.8 – Обработка ошибок при добавлении постера

Данный метод позволяет нам отлавливать пользовательские ошибки на сервере и в дальнейшем отправлять их на клиент.

3.1.5 Покупка и проверка билетов

В данной главе представлен функционал обработки покупки билета, а также предварительной проверки его перед сеансом.

Для покупки билетов я использую сервис оплаты `stripe`, с помощью него пользователи могут оплачивать билеты онлайн. Чтобы произвести оплату пользователь должен выбрать билеты и нажать оплатить, после этого его переадресует на страницу оплаты, код перенаправления представлен на листинге 3.9.

```
const session = await stripe.checkout.sessions.create({
  payment_method_types: ['card'],
  line_items: lineItems,
  mode: 'payment',
  success_url: `http://localhost:3000/Film?ID=${IdFilm}`,
  cancel_url: 'http://localhost:3000/',
});
```

Листинг 3.9 – Перенаправление на сервис оплаты

Также, после успешной оплаты генерируется QR-code билета, в котором хранится информация о сканируемом билете, а также его идентификатор, но идентификатор не в открытом виде, а в хешированном, чтобы злоумышленники не могли получить доступ к данным. Код, реализующий создание QR-code представлен на листинге 3.10.

```
const session = await stripe.checkout.sessions.create({
  const secretKey = 'mySecretKey';
  const originalText = tick.id;
  const cipher = crypto.createCipher('aes-256-cbc', secret-
Key);
let encryptedText = cipher.update(originalText, 'utf8', 'hex');

encryptedText += cipher.final('hex');

const updatedQRCodeData = await qrcode.toDataURL(
`Зал:${pl.hall.name}; Место:${pl.place}; ID:${encryptedText}`,
  );
}
```

Листинг 3.10 – Генерация QR-code билета

Далее, данные сохраняются в базе данных, код этой функциональности представлен на листинге 3.11.

```
const tick = await this.databaseService.ticket.save({
  user: { id: id },
  place: { id: pl.id },
  session: { id: sessions },
  QR: qrCodeData,
  chatr_id: chatr_id,
});
```

Листинг 3.11 – Сохранение данных о билете в базу данных

Одна из основных функций администратора – это проверка билетов перед сеансом, для этой цели я использую библиотеку, которая подключается к камере и считывает данные с qr-code билета для проверки. Реализация данного функционала представлена на листинге 3.12.

```
const config = { fps: 10, qrbox:{width: 600, height: 600}};
const html5QrCode = new Html5Qrcode("qrCodeContainer");
const qrScannerStop = () => {
  if (html5QrCode && html5QrCode.isScanning) {
    html5QrCode.stop().then(() => console.log("ok"))
      .catch(() => console.log("error"));
  }
};
```

Листинг 3.12 – Считывания qr-code билета

Далее происходит чтение и сверка данных, полученных из qr-code, с данными, которые лежат в базе данных, код представлен на листинге 3.13.

```
const userinfo = await this.databaseService.ticket.find({
  where: {
    id: decryptedText,
  }
});
```

Листинг 3.13 – Поиск данных билета в базе данных

И при всех верных данных зрителя пропускают, если же нет, то администратору приходит уведомление, о том, что данный билет фальшивый и этого зрителя нельзя пускать в зал. Полный код реализации представлен в приложении И

3.2 Разработка клиентской части

В данной главе представлена реализация взаимодействия между клиентом и сервером, а также поясняется процесс разработки компонентов. Клиентская часть приложения выполнена на библиотеке React, используя интерпретируемый язык программирования JavaScript.

Прежде чем перейти к детальному описанию взаимодействия между клиентом и сервером, следует рассмотреть реализацию компонентов в приложении. Для стилизации пользовательского интерфейса использовалась библиотека Mantine, а также применялись собственные стили, основанные на возможностях библиотеки. Подробную информацию о возможностях Mantine можно найти в ее официальной документации.

Ключевым элементом разработки клиентской части стало использование библиотеки React, которая позволяет строить интерфейс из независимых и многократно используемых компонентов. Такой подход упрощает разработку, облегчает сопровождение проекта и способствует его масштабируемости.

Для организации структуры и управления визуальными элементами была выбрана библиотека Mantine, предоставляющая готовые стили и настраиваемые компоненты, что значительно ускоряет процесс создания интерфейса и делает его более целостным и современным.

В дальнейшем будет подробно рассмотрено, как реализовано взаимодействие между клиентской частью и сервером, а также будет разобрана реализация ключевых компонентов и особенности стилизации с использованием Mantine UI и собственных решений. На листинге 3.14 представлен корневой компонент приложения.

```
import ReactDOM from 'react-dom/client';
import App from './App';
const container = document.createElement("div");
document.body.appendChild(container);
ReactDOM.createRoot(container).render(<App />);
```

Листинг 3.14 – Корневой компонент сервера

Как можно заметить, здесь подключаются корневой компонент App. В данном компоненте определен Routes для создания маршрутизации приложения. Фрагмент компонента App представлен на листинге 3.15.

```
function App() {
  return (
    <MantineProvider
      defaultColorScheme="light"
      theme={{
        primaryColor: "teal",
        fontFamily: 'Inter, sans-serif',
      }}>
      <Notifications/>
      <BrowserRouter>
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Registration
/>} />
          <Route path="/" element={<MainLayout />}>
            <Route index element={<Navigate to="/AdminHome"
replace />} />
            <Route path="AdminHome" element={<AdminHome />}
/>
            <Route path="AdminAddFilm" element={<AdminAdd-
Film />} />
            <Route path="AdminAddHall" element={<AdminAd-
dHall />} />
            <Route path="AdminAddSession" element={<Ad-
minAddSession />} />
            <Route path="AdminAllFilm" element={<AdminAll-
Film />} />
            <Route path="AdminAllHall" element={<AdminAll-
Hall />}></Route>
            <Route path="Controller" element={<TheControl-
ler />} />
          </Route>
        </Routes>
      </BrowserRouter>
    </MantineProvider>
  );
}
```

Листинг 3.15 – Компонент App

Маршрутизация в приложении организована таким образом, что при успешной аутентификации пользователь перенаправляется на соответствующий его роли интерфейс. Первоначально запрос на авторизацию обрабатывается компонентом Login, который взаимодействует с серверной частью для проверки учетных данных. В случае успешной проверки система определяет уровень доступа пользователя и выполняет переход на соответствующий раздел.

После авторизации в пользовательском интерфейсе отображаются дополнительные элементы навигации, включая верхнюю панель и боковое меню. Навигационная система автоматически адаптируется под права пользователя, скрывая или отображая функциональные элементы. Для администраторских учетных записей доступен расширенный набор возможностей управления системой.

Механизм маршрутизации включает встроенную проверку прав доступа, что предотвращает переход пользователей на страницы, не соответствующие их уровню авторизации. В случае попытки доступа к закрытому разделу система автоматически перенаправляет пользователя на доступный ему интерфейс. Все переходы между разделами сопровождаются проверкой актуальности авторизационных данных.

Для создания общей разметки для всех страниц приложения был создан компонент `MainLayout`, код которого отображен в листинге 3.16.

```
import {AppShell} from '@mantine/core';
import Header from '../components/header/header';
import {Outlet} from "react-router-dom";
import logo from "../images/logo.png";

export function MainLayout() {
  return (
    <AppShell
      header={{ height: 60 }}
      navbar={{
        width: 260,
        breakpoint: 'sm',
      }}
      aside={{
        width: 260,
        breakpoint: 'sm',
      }}
      padding="md"
    >
      <Header logo={logo}/>
      <AppShell.Navbar></AppShell.Navbar>
      <AppShell.Aside></AppShell.Aside>
      <AppShell.Main h={"100%"}>
        <Outlet />
      </AppShell.Main>
    </AppShell>
  );
}
```

Листинг 3.16 – Компонент `MainLayout`

Можно отметить, что основная разметка задается при помощи встроенного в Mantine компонента `AppShell`, что позволяет создать общую совместимость с остальными стилями компонентов библиотеки. При помощи параметров есть возможность устанавливать ширину навигационного меню и правого дополнительного меню при различных поведениях экрана.

3.2.1 Взаимодействие с глобальным хранилищем состояний

Поскольку React основан на повторном использовании компонентов, возникает необходимость в передаче состояний из одного компонента к другому. Но проблема с отслеживанием состояния может возникнуть, если у компонентов будет глубокая вложенность. В таком случае самым рациональным решением будет создание глобального хранилища, откуда есть возможность получить состояние без передачи параметра через многие компоненты.

В данном веб-приложении используется библиотека MobX, которая реализует функцию глобального хранилища. Глобальное хранилище в React – это механизм для централизованного управления состоянием приложения, позволяющий сохранять и обновлять данные, доступные ко всему приложению, и обеспечивая их синхронизацию между компонентами. Для реализации этой логики был реализован класс, отвечающий за сохранение состояния.

Также этот класс реализует функциональность промежуточной обработки данных для передачи в сервисы. Данный класс используется для более практичного применения при необходимости использования функций нескольких сервисов одновременно.

При авторизации пользователя, а также при выходе из приложения, для получения его данных из любого места в приложении и корректной работы с условной отрисовкой элементов, стоит записывать статус авторизации (авторизован или нет) и данные пользователя в глобальное хранилище. Листинг 3.17 отображает пример работы с хранилищем состояний.

```
const store = new Store();
useEffect(() => {
  const fetchData = async () => {
    try {
      const users = await store.GetUserById();
      setData(users.data);
      await store.checkAuth();
    } catch (error) {
      if (error.response.data.statusCode === 401) {
        store.logout();
        navigate("/login");
      } else {
        console.log(error.response.data);
      }
    }
  }
});
```

Листинг 3.17 – Установка состояния пользователя в хранилище

С помощью функции `GetUserById` происходит получение данных о пользователе, а затем с помощью функции `checkAuth` устанавливается в `localStorage` токен и данные пользователя, а также устанавливается статус авторизации. С помощью этих данных затем есть возможность использовать их для запроса. Реализация функции из хранилища представлена в листинге 3.18.

```

async checkAuth() {
  const response = await AuthService.checkAuth();
  localStorage.setItem("token", response.data.accessToken);
  this.setAuth(true);
  this.setUser(response.data.user);
  return response;
}

```

Листинг 3.18 – Функция checkAuth

Также в дальнейшем созданные функции из хранилища состояний используются для работы с фильмами, сессиями, залами, и другими функциональными объектами приложения.

3.2.2 Обращение к REST API

Обращение к серверу происходит с помощью axios. Axios – это библиотека для выполнения HTTP-запросов в среде JavaScript. Она предоставляет удобные методы для отправки запросов на сервер и обработки ответов. Axios может использоваться в различных окружениях, таких как браузер и Node.js, и является популярным инструментом во фронтенд и бэкенд разработке. Выбор axios для выполнения HTTP-запросов в проекте подтверждает предпочтение разработчиков к этому инструменту за счет его мощных возможностей, легкости интеграции и универсальности, что позволяет эффективно обрабатывать как простые GET и POST запросы, так и более сложные операции с данными, включая работу с JSON форматом и формами, а также поддержку промисов и асинхронных операций.

Внутри директории файл index.ts отвечает за инициализацию объекта axios. Поскольку весь функционал требует авторизации пользователя, разделения на авторизованные запросы и неавторизованные не производилось. С помощью интерцептора authInterceptor, к запросу прибавляется заголовок authorization с bearer токеном пользователя. Код общей функции осуществления запросов приведен на листинге 3.19.

```

import axios from "axios";
export const API_URL = `http://localhost:5000/`;
const $api = axios.create({
  withCredentials: true,
  baseURL: API_URL,
});

$api.interceptors.request.use((config) => {
  config.headers.Authorization = `Bearer ${localStorage.getItem("token")}`;
  return config;
});
export default $api;

```

Листинг 3.19 – Функция отправки запросов на сервер

В зависимости от содержания токена авторизации в запросах пользователя, система может динамически изменять интерфейс, предоставляя доступ к различным возможностям в соответствии с уровнем доступа пользователя. К примеру, кнопка для добавления фильма или для управления сеансами является невидимой для пользователей, которые не имеют роли администратора.

Далее в листинге 3.20 представлена реализация серверной функции, созданной для покупки билетов.

```
static async BuyTicket(tickets, IdFilm, fullPrice) {
    return $api.post("ticket/BuyTicket", {
        tickets: tickets,
        IdFilm: IdFilm,
        fullPrice: fullPrice,
    });
}
```

Листинг 3.20 – Функция покупки билетов

Если запрос на сервер был отправлен успешно с клиентской стороны, то возвращаются либо нужные данные и успешный код ответа, либо код ошибки вместе с пояснением к данному коду.

3.2.3 Разработка модуля контроля билетов

В рамках клиентской части приложения был реализован ключевой функциональный модуль для сканирования и верификации билетов с использованием QR-кодов. Данный компонент демонстрирует интеграцию с аппаратными возможностями устройства (камерой), реализацию сложной клиент-серверной логики проверки билетов, обработку различных сценариев взаимодействия пользователя с системой.

Реализация функции проверки билетов отображена в листинге 3.21.

```
const qrCodeSuccess = async (decodedText) => {
    const res = await store.CheckTicketCon(decodedText);
    const isValid = res.data.length > 0;

    setLastScanResult({
        isValid,
        message: isValid ? `Билет действителен` : "Билет недействителен",
        place: isValid ? res.data[0].place.place : null
    });
};
```

Листинг 3.21 – Функция проверки билетов

Данный фрагмент кода иллюстрирует:

- асинхронное взаимодействие с сервером для проверки билета;
- обработку результатов сканирования;

– обновление состояния интерфейса на основе полученных данных.

Для работы с камерой приложения была использована библиотека `html5-qrcode`. Затем объект этой библиотеки принимает как параметр идентификатор элемента `Card`, который отображает рамку с камерой, внутрь которой помещается QR-код. Затем этот расшифрованный код отправляется на сервер приложения в функцию `CheckTicketCon`.

Важной частью реализации является система уведомлений, которая реализована с помощью интегрированного под капотом дополнения библиотеки `Mantine`, отображенная на листинге 3.22.

```
notifications.show({
  title: isValid ? 'Билет действителен' : 'Билет недействителен',
  message: isValid ? `Место: ${data.place.place}` : 'QR-код не найден',
  color: isValid ? 'green' : 'red'
});
```

Листинг 3.22 – Показ уведомления о билете

Данный компонент доступен лишь администратору приложения, который проверяет билеты на входе в сеанс. Если билет действителен, то приходит уведомление об успехе, в противном случае, если билет недействителен, приходит отказ.

3.2.4 Использование встроенных компонентов `Mantine`

При разработке клиентского сервиса было реализовано динамическое обновление данных при необходимости и логически необходимые уведомления пользователя о его действиях. Использованы различные механизмы обработки как данных для отправки запросов, так и данных для получения соответствующих ответов и их вывод. Были использованы такие механизмы `React` как: `useEffect`, `useRef`, `useState`, `props`, `hooks`.

В ходе разработки были использованы отдельные функциональные возможности из библиотеки компонентов `Mantine`, к примеру `Carousel`, фрагмент использования этого компонента представлен в листинге 3.23.

```
<Carousel
  height={280}
  slideSize="100%"
  slideGap="sm"
>
  {getCarouselSlides(genreFilms[genre.id])}
</Carousel>
```

Листинг 3.23 – Использование компонента `Carousel`

Данный компонент позволяет отображать постеры и картинки фильмов компактным и удобным способом для пользователя.

3.3 Выводы по разделу

Разработанное веб-приложение успешно реализует все поставленные функциональные требования. Архитектура системы построена с соблюдением современных принципов разработки, включая четкое разделение клиентской и серверной частей. Серверная часть, реализованная на NestJS, обеспечивает надежную работу с данными через TypeORM, безопасную аутентификацию на основе JWT-токенов и валидацию входящих запросов. Особое внимание уделено безопасности - все конфиденциальные данные хранятся в переменных окружения, а запросы проходят строгую проверку.

Клиентское приложение на React с использованием библиотеки Mantine предоставляет интуитивно понятный интерфейс для различных категорий пользователей. Реализована сложная бизнес-логика, включая систему онлайн-покупки билетов с интеграцией платежного сервиса Stripe, механизм проверки билетов через QR-коды, а также разграничение прав доступа. Глобальное состояние приложения эффективно управляется через MobX, что обеспечивает согласованность данных между компонентами.

Дополнительно следует отметить, что в процессе разработки были успешно решены несколько технических задач. Во-первых, реализована комплексная система обработки ошибок на всех уровнях приложения. Во-вторых, обеспечена высокая производительность интерфейса благодаря оптимизированным запросам к API и эффективному управлению состоянием. В-третьих, достигнута хорошая адаптивность интерфейса для различных устройств. Все эти аспекты в совокупности позволили создать полнофункциональное решение, отвечающее современным требованиям к веб-приложениям.

Таким образом, разработанное веб-приложение демонстрирует высокий уровень зрелости как с технической, так и с пользовательской точки зрения. Применение современных технологий и подходов обеспечило не только стабильную и безопасную работу системы, но и её масштабируемость, что позволяет легко внедрять дополнительные функции в будущем. Кроме того, модульная структура кода способствует удобству поддержки и дальнейшего развития проекта. Полученный результат подтверждает практическую значимость проведённой разработки и её готовность к реальному использованию в условиях действующего кинотеатра.

4 Тестирование веб-приложения

В данной главе приводятся тесты приложения как под ролью администратора, так и под ролью пользователя. Тестирование будет ручным. Ручное тестирование – это процесс проверки качества программного продукта, который включает в себя выполнение тестовых сценариев человеком. Этот метод особенно полезен для проверки интерфейсов, взаимодействия с пользователем и других аспектов, которые сложно автоматизировать. При ручном тестировании под разными ролями, например, администратором и пользователем, важно следовать определенным шагам для обеспечения эффективности процесса.

4.1 Тестирование страниц авторизации и регистрации

На рисунке 4.1 представлена страница входа в приложение и введенные учетные данные пользователя «andrew1@gmail.com».

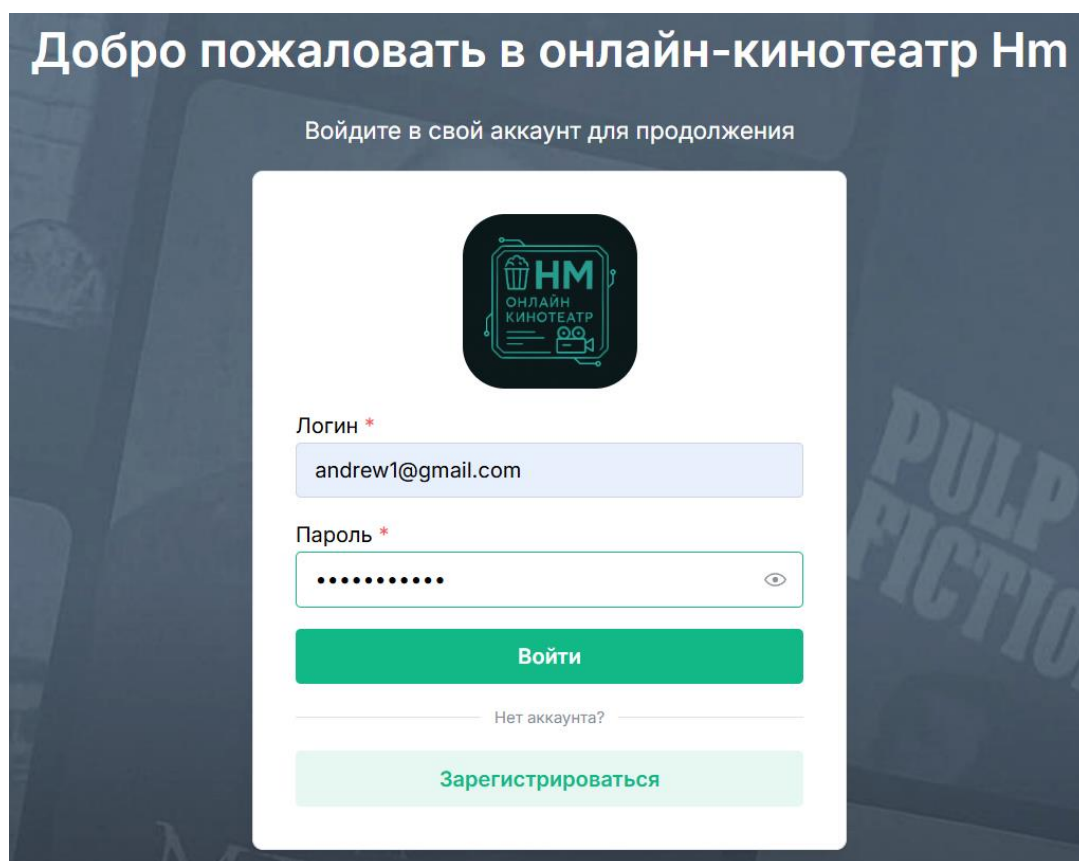


Рисунок 4.1 – Страница входа в приложение

В случае неуспешного входа будет отображаться одна ошибка рисунок 4.2.

					ДП 04.00 ПЗ		
		ФИО	Подпись	Дата	4 Тестирование веб-приложения		
Разраб.	Халалеенко А.Н.						
Пров.	Тимонович Г.Л.						
Н. контр.	Алешаускас В.А.						
Утв.	Блинова Е.А.						
					Лит.	Лист	Листов
					У	1	5
					БГТУ 1-40 05 01, 2025		

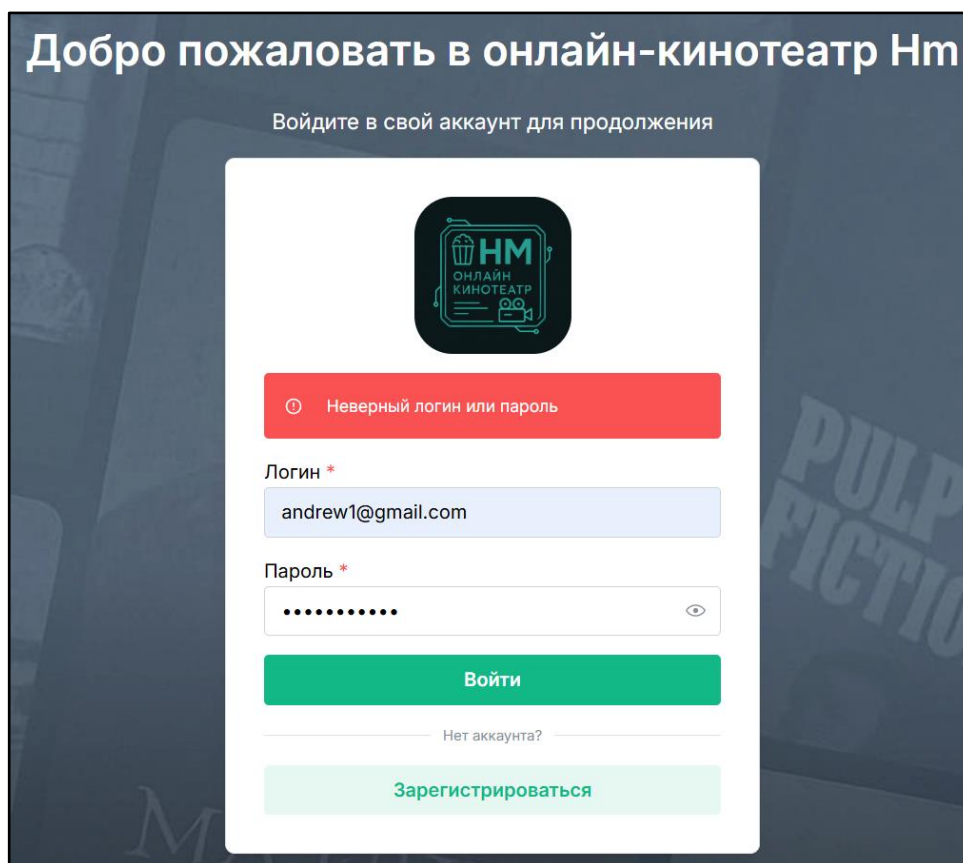


Рисунок 4.2 – Отображение ошибки при входе

В этом случае выводится общий текст ошибки с той целью, чтобы методом перебора неверных вариантов было сложнее узнать логин и пароль. При попытке входа пользователя, которого заблокировал администратор, будет отображена ошибка, которая представлена на рисунке 4.3.

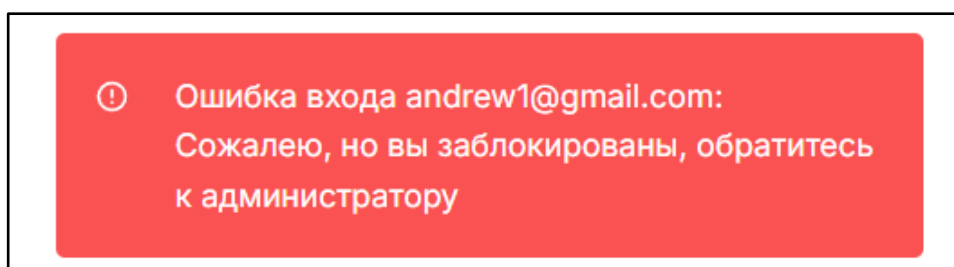


Рисунок 4.3 – Сообщение при блокировке

Если пользователь попытается зарегистрироваться под электронной почтой, которая уже зарегистрирована в системе, будет отображена ошибка с рисунка 4.4.



Рисунок 4.4 – Ошибка регистрации двух аккаунтов

В данном подразделе было рассмотрено тестирование страниц входа и регистрации в приложение, а также были рассмотрены возможные ошибки и предусмотрены возможные случаи их возникновения.

4.2 Тестирование добавления фильма

Для администраторов приложения предоставлена возможность добавления нового фильма в каталог доступных фильмов. Форма для заполнения данных о фильме представлена на рисунке 4.5.

Название фильма *

Год выпуска *

Продолжительность *

Возрастное ограничение *

Дата начала проката *

Дата окончания проката *

Жанр *

Постер фильма

Описание *

Отмена

Добавить фильм

Рисунок 4.5 – Страница добавления фильма

Если администратор попытается добавить фильма с датой начала проката позже, чем дата окончания проката, ему выведется соответствующая ошибка, представленная на рисунке 4.6.

Дата начала проката *

Дата окончания проката *

23.06.1994

21.06.1994

Дата окончания проката не может быть раньше даты начала

Рисунок 4.6 – Ошибка окончания даты проката

Подобным образом обрабатывается каждое из полей формы, такие как ограничение по возрасту, год выпуска, название фильма, а также обработана ситуация попытки отправки формы с пустыми полями.

4.3 Тестирование добавления сеанса на фильм

В данном подразделе отображены основные сценарии возможных ошибок при добавлении сеанса и их корректная обработка. Один из таких случаев (добавление сеанса раньше либо позже даты проката) отображен на рисунке 4.7.

Добавить новый сеанс

Ошибка
Прокат фильма уже закончен

Фильм *
Побег из Шоушенка

Зал *
Oktyabr

Дата *
12.09.2036

Время *
15:00

Добавить сеанс

Рисунок 4.7 – Ошибка даты проката фильма

Также предусмотрен и обработан вариант того, что администратор попытается добавить сеанс в то время пока уже идет фильм в данном зале в указанное время. Обработка этой ситуации отображена на рисунке 4.8.

Добавить новый сеанс

Ошибка
В данное время в данном зале идет фильм

Фильм *
Форрест Гамп

Зал *
Oktyabr

Дата *
14.05.2025

Время *
15:00

Добавить сеанс

Рисунок 4.8 – Ошибка времени сеанса

Таким образом были обработаны все возможные ошибки при добавлении сеанса. Также предусмотрена обязанность заполнения всех полей формы.

4.4 Тестирование проверки билетов

Администраторам приложения предоставлена функция сканирования и проверки билетов, при нажатии на кнопку проверки запрашивается доступ к камере. Это разрешение запрашивается единоразово. Если при сканировании билета оказывается, что этот билет уже был возвращен, выводится соответствующее уведомление. Обработка этой ситуации отображена на рисунке 4.9.

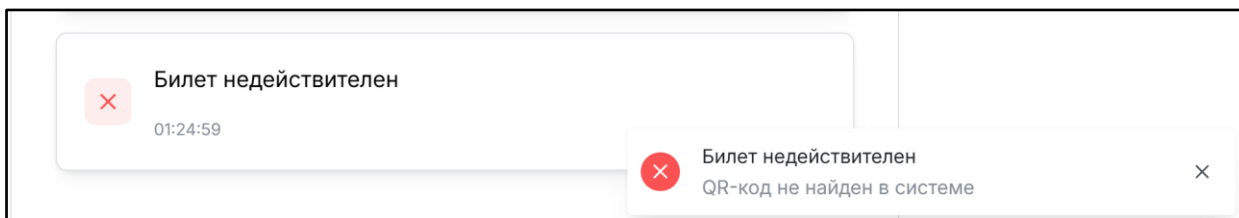


Рисунок 4.9 – Ошибка при сканировании недействительного билета

При этом если пользователь предъявляет актуальный QR-код, происходит отображение соответствующего уведомления об успехе активации, которое представлено на рисунке 4.10.

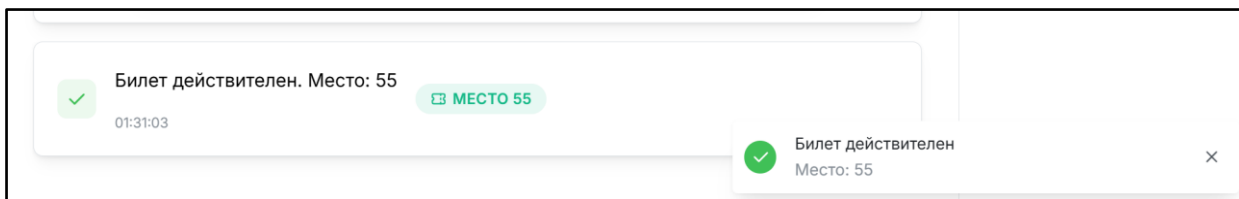


Рисунок 4.10 – Уведомление об успешном сканировании

Таким образом было рассмотрено тестирование проверки билетов.

4.5 Выводы по разделу

В данном разделе было проведено ручное тестирование разработанного веб-приложения. Таким образом рассмотрено тестирование основных страниц и компонентов приложения. Было продемонстрировано отображение ошибок и ошибочных сценариев различного характера. Также реализованы уведомления информационного характера и уведомления об успешном выполнении какого-либо действия.

5 Руководство пользователя

5.1 Пошаговое руководство

Руководство пользователя необходимо для предоставления пользователям инструкций и информации о том, как эффективно и качественно использовать приложение. Хорошо написанное руководство пользователя может помочь пользователям узнать о расширенных или малоизвестных функциях приложения, что улучшит их общий опыт и повысит производительность. Написание руководства пользователя может улучшить опыт пользователя, снизить требования к поддержке и повысить способность пользователя эффективно использовать программное обеспечение.

В данном разделе пояснительной записки рассмотрены основные возможности обычного пользователя и администратора.

При входе в приложение пользователю отображается страница авторизации, где отображены поля для ввода логина и пароля. Страница входа представлена на рисунке 5.1.

Рисунок 5.1 – Страница авторизации

					ДП 05.00 ПЗ		
		ФИО	Подпись	Дата			
Разраб.		Халалеев А.Н.			5 Руководство пользователя		
Пров.		Тимонович Г.Л.					
Н. контр.		Алешаускас В.А.					
Утв.		Блинова Е.А.					
					Лит.	Лист	Листов
					У	1	10
					БГТУ 1-40 05 01, 2025		

Поскольку приложение доступно лишь авторизованным пользователям, на странице входа снизу есть кнопка позволяющая перейти на страницу регистрации пользователя. Данная страница отображена на рисунке 5.2.

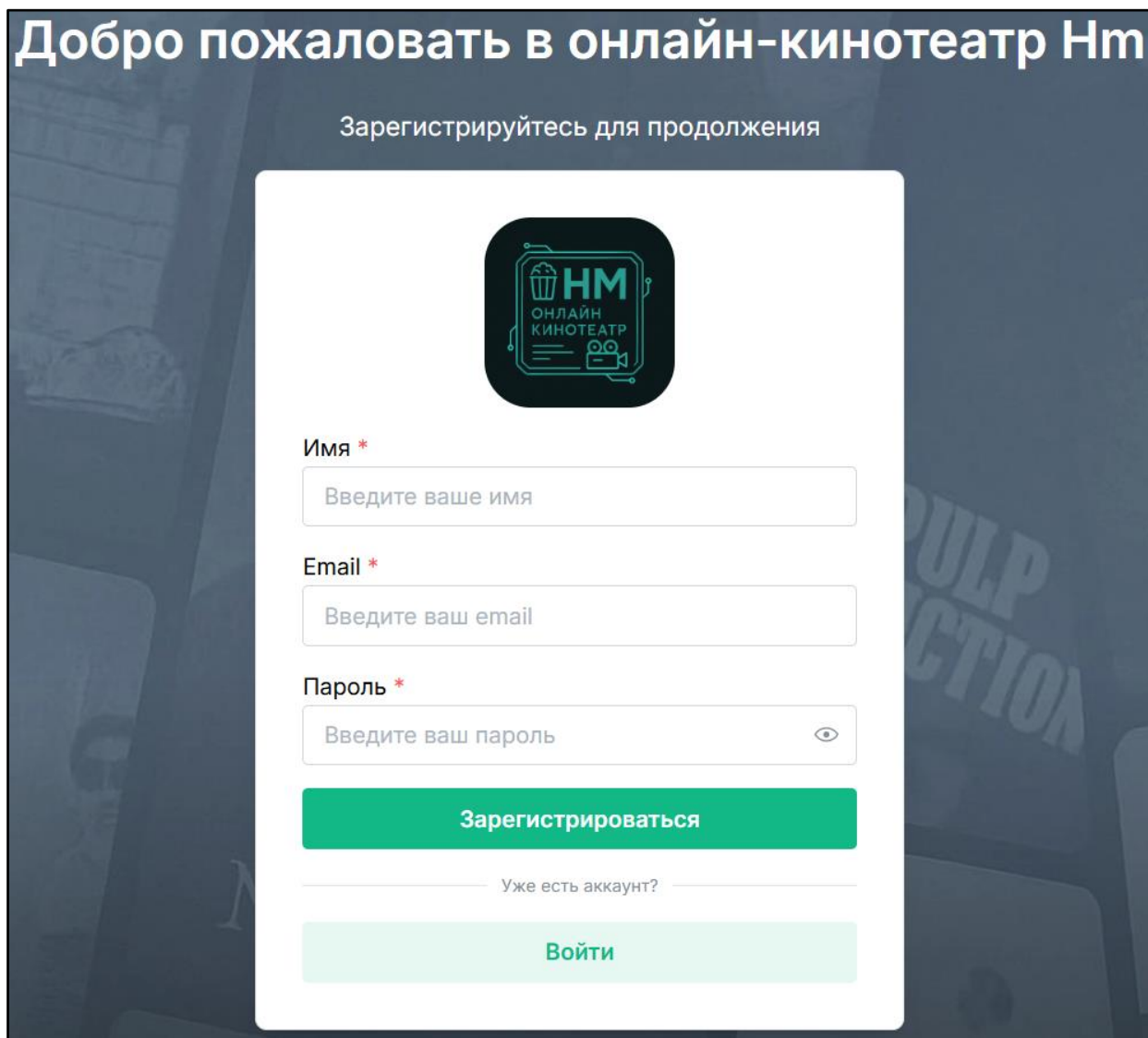


Рисунок 5.2 – Страница регистрации

На странице регистрации необходимо обязательно ввести имя, электронную почту и задать пароль. После успешной регистрации пользователь перенаправляется на страницу авторизации в приложение.

После успешной авторизации пользователь попадает на главную страницу, где отображена основная картотека фильмов. Отсюда он может попасть в любую часть приложения: в карточку любого фильма, чтобы получить информацию о нем, а также купить билеты. Имеется возможность перейти во вкладку со своими любимыми фильмами, во вкладку жанры, где ему предоставляется возможность выбора фильмов по жанрам, а также в свой личный кабинет с возможностью редактирования личных данных и изменением пароля или же выйти из приложения. Данная страница и меню пользователя без администраторских возможностей отражены на рисунке 5.3.

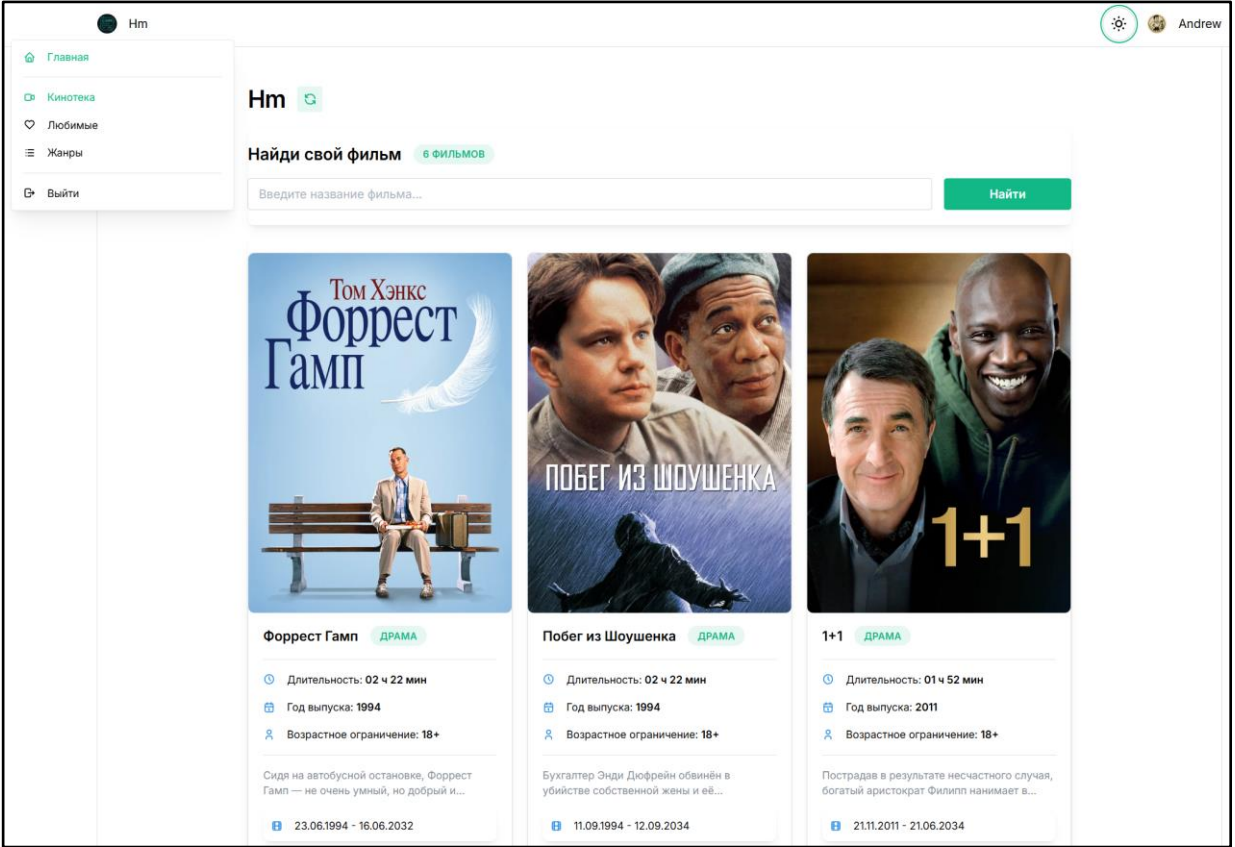


Рисунок 5.3 – Главная страница

Далее при нажатии на карточку с изображением и краткими данными интересующего фильма отображается страница с подробными данными о выбранном фильме. Данная страница изображена на рисунке 5.4.

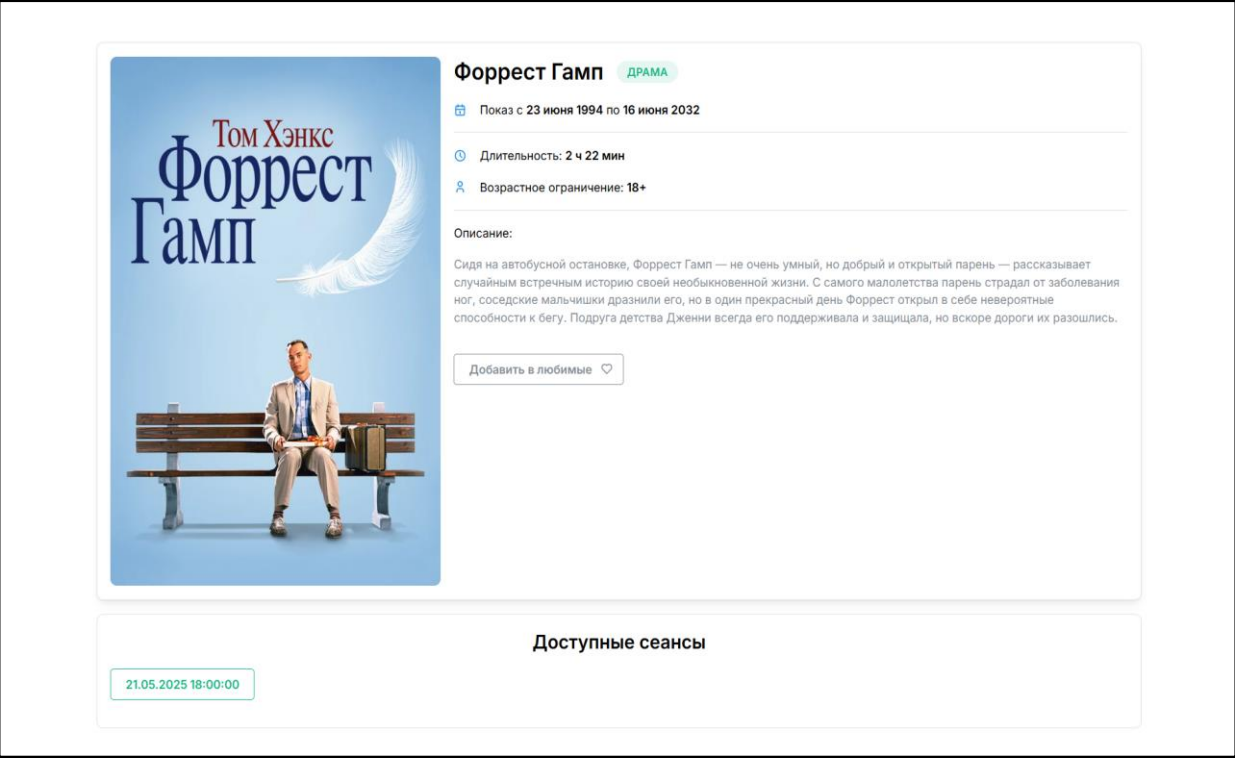


Рисунок 5.4 – Страница с данными о фильме

Как можно отметить, ниже есть поле с доступными сеансами на просмотр данного фильма. При нажатии на интересующую дату сеанса открывается схема мест в кинотеатре. Ниже отображаются подсказки с обозначением мест, где синий – доступное обычное место, серый – недоступное место и желтый – доступное место VIP. Также при выборе мест, снизу отображаются карточки с выбранными местами и их ценами. Весь описанный выше функционал отражен на рисунке 5.5.

Ряд 1	1	2	3	4	5	6	7	8	9	10	11	12
Ряд 2	13	14	15	16	17	18	19	20	21	22	23	24
Ряд 3	25	26	27	28	29	30	31	32	33	34	35	36
Ряд 4	37	38	39	40	41	42	43	44	45	46	47	48
Ряд 5	49	50	51	52	53	54	55	56	57	58	59	60
Ряд 6	61	62	63	64	65	66	67	68	69	70	71	72
Ряд 7	73	74	75	76	77	78	79	80	81	82	83	84
Ряд 8	85	86	87	88	89	90	91	92	93	94	95	96

Выбранные места:

Ряд: 4
Место: 43
Тип: VIP
Цена: 25 ₽

Ряд: 3
Место: 31
Тип: Standart
Цена: 10 ₽

Ряд: 3
Место: 32
Тип: Standart
Цена: 10 ₽

Обычные места

VIP места

Занятые места

Оформить заказ (45 ₽)

Рисунок 5.5 – Функциональность выбора мест

Выбор мест происходит по нажатию на интересующее место и при выборе оно подсвечивается заполненным синим цветом. При повторном нажатии на выбранное место выбор снимается и пропадает карточка из графы выделенных мест.

При нажатии на кнопку оформления заказа происходит перенаправление на сервис оплаты, в котором требуется ввести данные платежного средства.

Наглядная цветовая индикация статусов мест, динамическое отображение выбранных билетов и их стоимости, а также мгновенная реакция интерфейса на действия пользователя обеспечивают комфортный процесс бронирования.

Страница для оплаты выбранных билетов отображена на рисунке 5.6.

← Назад

TEST MODE

К оплате

45,00 BYN

Оплатить через link

Или

Эл. почта

andrew1@gmail.com

Данные карты

4242 4242 4242 4242

05 / 29 111

Имя владельца карты

NAME JOHN

Страна или регион

Нидерланды

☐ Надежно сохранить мои данные для оформления платежа одним щелчком

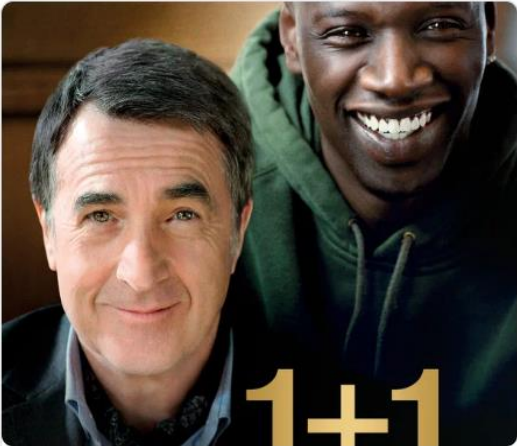
Выполняйте оплату быстрее в Vitebsk и везде, где принимают Link.

Оплатить

Рисунок 5.6 – Страница оплаты билетов


После прохождения оплаты пользователь возвращается на страницу фильма, а билет появляется в его личном кабинете. Также на странице фильма есть возможность добавить его в любимые, и фильм будет отображаться на соответствующей странице. Страница любимых фильмов отображена на рисунке 5.7.

Любимые фильмы



1+1 2011

Пострадав в результате несчастного случая, богатый аристократ Филипп нанимает в...



Форрест Гамп 1994

Сидя на автобусной остановке, Форрест Гамп — не очень умный, но добрый и открытый...

Рисунок 5.6 – Страница любимых фильмов

У пользователя также есть возможность просмотра личного кабинета, где будет отображена информация о пользователе такая как имя, почта, возможность изменения этих данных и пароля. Помимо этого, на странице есть небольшая таблица, которая содержит информацию о купленных пользователем билетах и информацию о них: название фильма, дата и время сеанса, зал в котором будет проходить фильм, место и QR-код, который пользователь должен предъявить контролеру при входе на сеанс. Вся эта функциональность отображена на рисунке 5.7.

Профиль Andrew ACTIVE

Ваши заказы

Фильм	Дата	Время	Зал	Место	QR-код	Вернуть
Jeremy	10.05.2025	18:00	BelCinema	42	QR-код	↩
Jeremy	10.05.2025	18:00	BelCinema	43	QR-код	↩
Форрест Гамп	21.05.2025	18:00	AbsoluteCinema	29	QR-код	↩
Форрест Гамп	21.05.2025	18:00	AbsoluteCinema	41	QR-код	↩
Форрест Гамп	21.05.2025	18:00	AbsoluteCinema	30	QR-код	↩

Изменение данных

Имя *

Email *

Пароль

Сохранить изменения

Рисунок 5.7 – Личный кабинет пользователя

На данной странице пользователь может как просмотреть свои билеты, так же и предъявить QR-код для проверки, а также вернуть купленный билет и деньги за него, но если же он его уже отсканировал при входе в зал, то он не сможет его вернуть. Данный интерфейс представлен на рисунке 5.8.

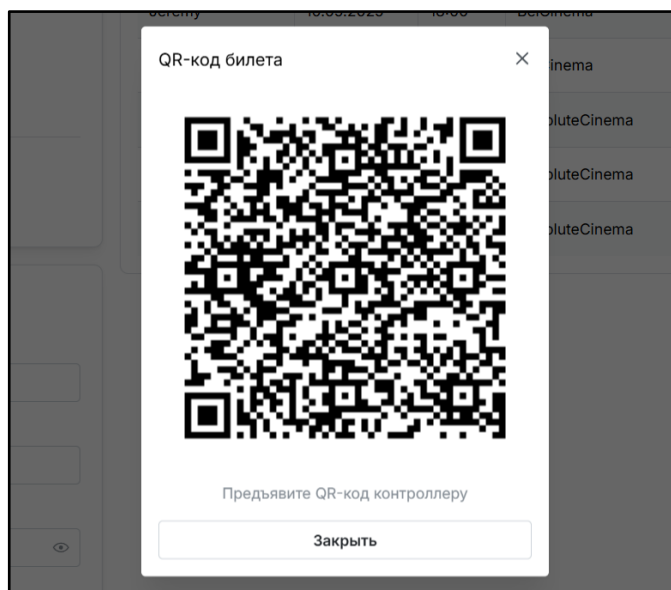


Рисунок 5.8 – QR-код билета

Если же зайти в приложение под администратором, то будут все те же возможности, что и у пользователя, но с дополнениями, нужными для администратора, такими как: добавление фильмов, залов, сеансов, просмотр купленных билетов на фильм, просмотр пользователей и сведений о них, а так же блокировка их в приложения, просмотр статистики купленных билетов, проверка билетов перед сеансом, изменение личной информации. Главная страница и меню администратора представлены на рисунке 5.9 и в приложении К.

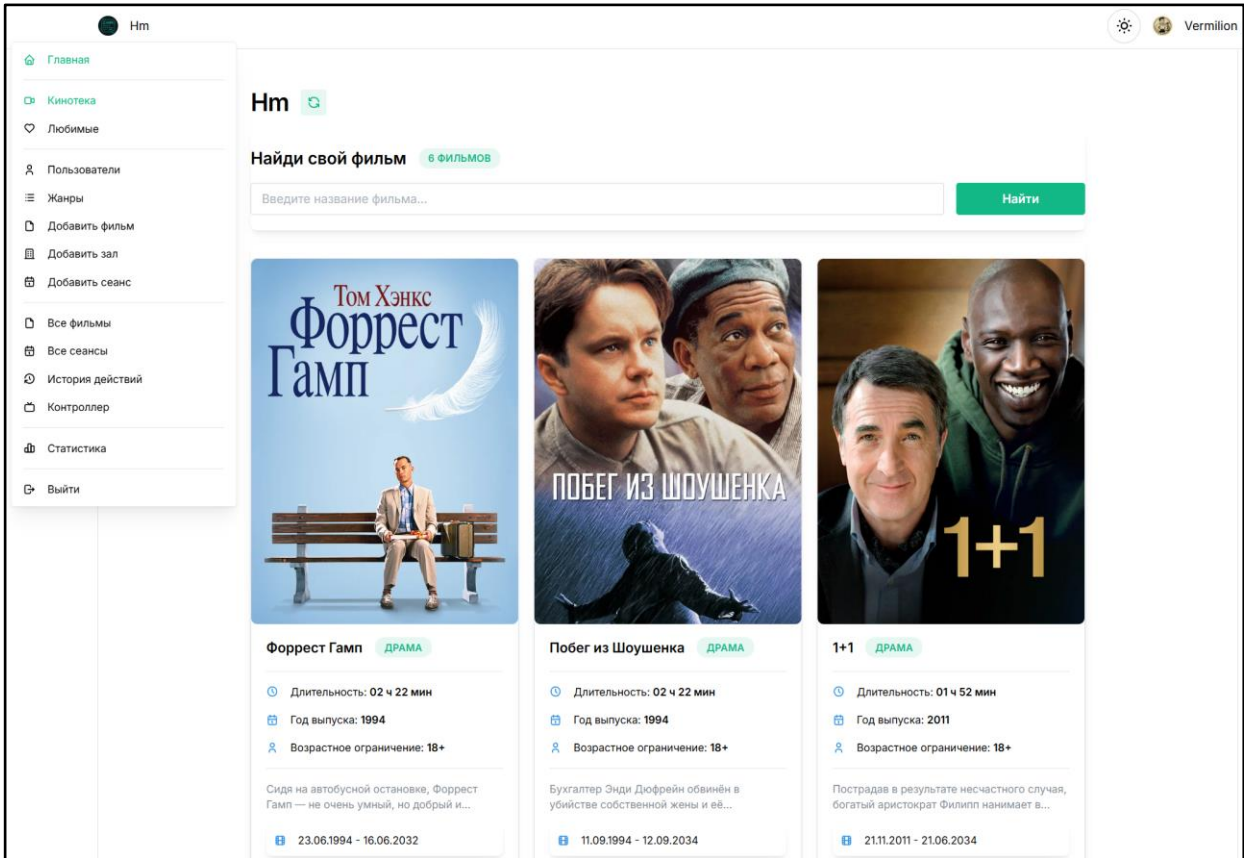


Рисунок 5.9 – Главная страница администратора

Первое из отличий, администратор видит всех пользователей и так же видит статистику по ним: количество купленных и количество возвращенных билетов. И на основе данной информации может их заблокировать, так же, как и разблокировать, что представлено на рисунке 5.10.

Список пользователей						
Поиск по имени или email						
Имя	Email	Билетов куплено	Билетов возвращено	Роль	Статус	Действия
DjKhaled	khaled@mail.ru	1	0	User	ЗАБЛОКИРОВАН	🔒
Lamar	lamar@gmail.com	3	0	User	АКТИВЕН	🔒
Andrew	andrew1@gmail.com	5	0	User	АКТИВЕН	🔒
Илья	sergo@mail.ru	0	0	User	АКТИВЕН	🔒
Kir	demo@devias.io	0	0	Admin	АКТИВЕН	🔒

Рисунок 5.10 – Общий список пользователей приложения

Так же есть вкладки по добавлению: фильмов, сеансов, залов, которые были продемонстрированы на этапе тестирования. Далее администратор имеет следующие возможности:

- просмотр всех сеансов на фильмы;
- просмотр фильмов, которые сейчас в прокате и которые планируются;
- просмотр информации о фильмах и сеансах;
- удаление фильмов;
- отмена планируемых сеансов.

Пример управления активными сеансами в кинотеатре и просмотр информации о них представлен на рисунке 5.11.

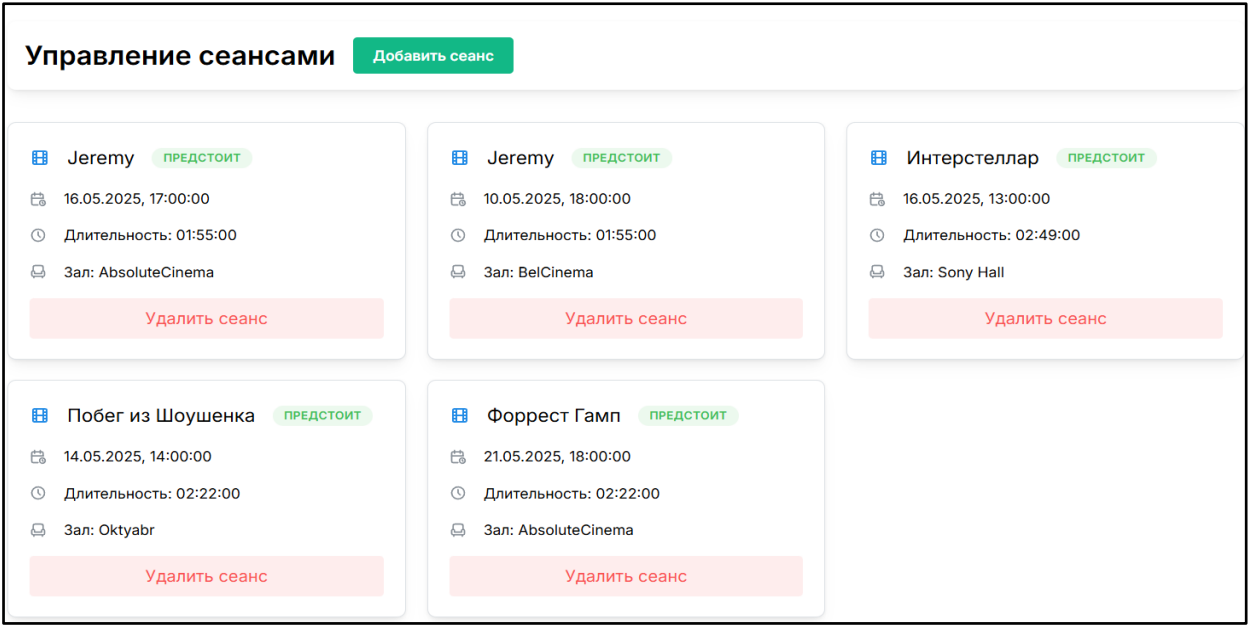


Рисунок 5.11 – Все активные сеансы в кинотеатре

На вкладке фильмов администратор может видеть такие данные как: название, описание, жанр и длительность фильма. Также есть возможность удаления фильма из общего каталога доступных фильмов (рисунок 5.12).

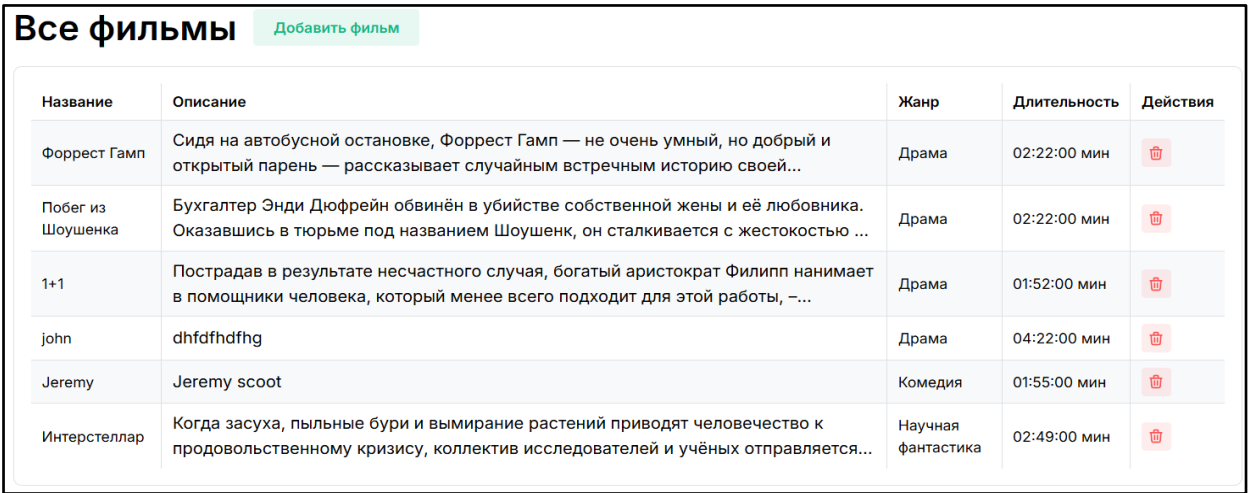


Рисунок 5.12 – Все доступные фильмы в кинотеатре

Также у администратора есть возможность проверять билеты, что позволяет исключить возможность прохода людей без билета, а также людей, которые хотят воспользоваться билетом повторно. Данное действие отражено на рисунке 5.13.

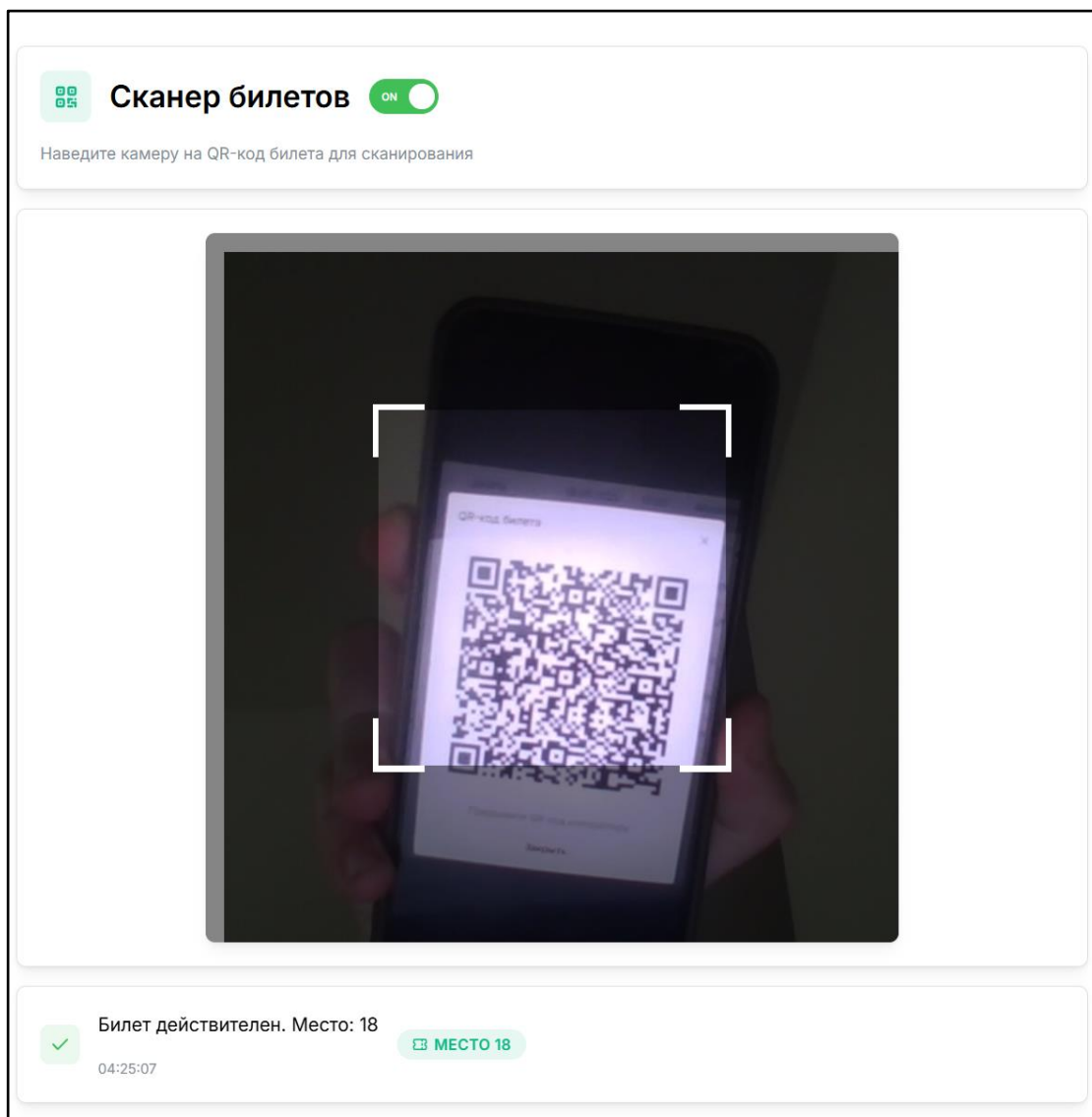


Рисунок 5.13 – Проверка билетов

При успешном сканировании происходит уведомление о действительности билета и клиент проходит в зал. Если же код оказался недействительным, то администратор это сразу видит и пользователь не проходит на сеанс. Уведомление о недействительности сканируемого кода отображено на рисунке 5.14.

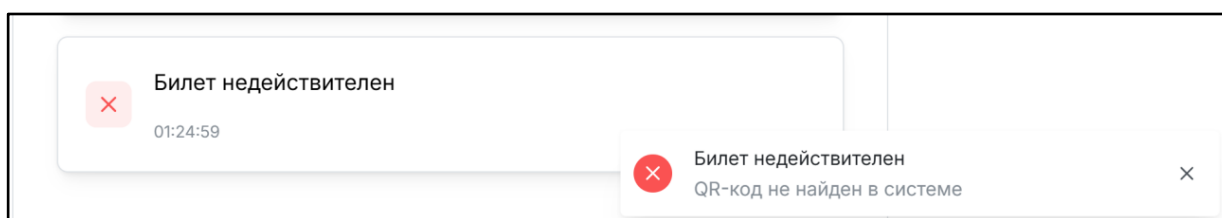


Рисунок 5.14 – Недействительность кода

К тому же у администратора есть возможность просматривать графическое представление статистики по фильмам и принимать решения по продлению проката и по увеличению или уменьшению количества сеансов.

Статистика по фильмам показана на рисунке 5.15.

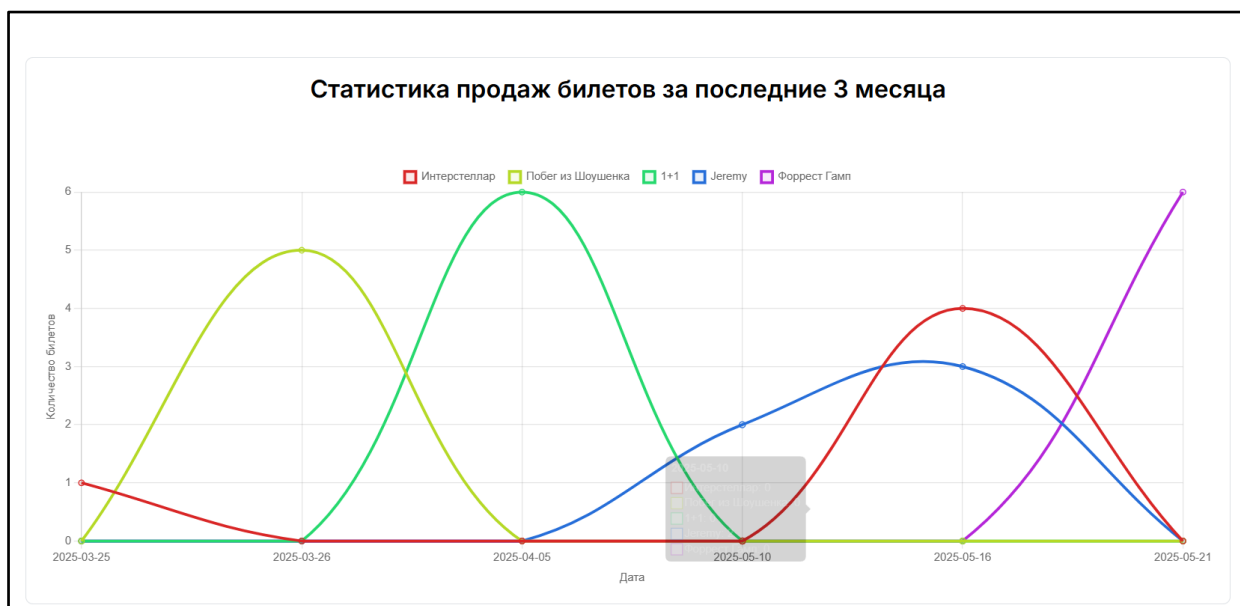


Рисунок 5.15 – Статистика по фильмам

Был показан весь функционал, реализованный в данном приложении. Данный функционал дает возможность пользователю быстро и удобно выбрать и купить билет на интересующие его фильм, а администратору отслеживать статистику по кинотеатру.

5.2 Выводы по разделу

В данном разделе были рассмотрены ключевые аспекты использования веб-приложения для администратора и пользователя. Для более ясного представления взаимодействия пользователей с приложением, предоставлены скриншоты с подробными пояснениями к алгоритмам взаимодействия. Были детально описаны содержание и функциональность различных экранов, позволяя пользователям лучше понимать, что они увидят на каждом этапе использования веб-приложения.

Интерфейс приложения достаточно прост и интуитивно понятен, поэтому у пользователя не должно возникнуть трудностей с его эксплуатацией.

6 Технико-экономическое обоснование работы

6.1 Общая характеристика разрабатываемого программного средства

Разрабатываемое в рамках дипломного проекта программное обеспечение представляет собой веб-приложение для кинотеатра. Оно предназначено для возможности бронирования и приобретения билетов на сеансы просмотра фильмов. Основной целью разработки является создание удобного, интуитивно понятного сервиса, который позволит пользователям комфортно просматривать каталог фильмов кинотеатра, отмечать их в качестве любимых, приобретать билеты на сеансы, а также позволит создать удобную инфраструктуру для администраторов для удобной проверки актуальности билетов и просмотра статистики.

С помощью данного программного средства пользователь может:

- регистрироваться и авторизоваться в системе;
- выбирать понравившиеся фильмы
- просматривать основную информацию о фильмах;
- выбирать и приобретать билеты на фильмы;
- просматривать и редактировать собственный профиль;
- фильтровать фильмы по жанрам и производить поиск по названию;
- подавать на проверку QR-код билета.

Разрабатываемое веб-приложение выгодно отличается от существующих решений глубокой интеграцией функций, удобством взаимодействия и адаптацией под современные ожидания пользователей. Представленная система сосредоточена на полноте пользовательского опыта: от интуитивного интерфейса и стабильного взаимодействия с серверной частью до индивидуализированных функций вроде избранных фильмов и просмотра статистики по просмотрам. За счёт применения современных технологий, таких как NestJS, React и PostgreSQL, удалось достичь высокой отзывчивости, безопасности и масштабируемости системы, при этом сохранив простоту для пользователей.

Способом монетизации данного веб-приложения является передача прав на программное обеспечение заказчику.

В рамках данного раздела необходимо определить затраты, произведенные на всех стадиях разработки описанного выше веб-приложения онлайн кинотеатра.

6.2 Исходные данные для проведения расчетов

Разрабатываемое веб-приложение не является принципиально новым решением, поскольку на рынке уже существуют аналогичные веб-приложения. Тем не менее, представленный проект включает уникальные функции.

					ДП 06.00 ПЗ			
		ФИО	Подпись	Дата				
Разраб.	Халалеевко А.Н.				6 Технико-экономическое обоснование работы	Лит.	Лист	Листов
Провер.	Тимонович Г.Л.					У	1	11
Консульт.	Соболевский А.С.					БГТУ 1-40 05 01, 2025		
Н. контр.	Алешаускас В.А.							
Утв.	Блинова Е.А.							

Веб-приложение имеет расширенное взаимодействие с пользователями, что выделяет его среди существующих решений. Программное средство относится ко второй группе сложности, так как реализует клиент-серверную архитектуру, базу данных и широкий спектр пользовательских функций.

Основные исходные данные, использованные для расчета экономических показателей, приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

Наименование показателя	Единица измерения	Условные обозначения	Норматив
Численность разработчиков	чел.	$Ч_p$	5
Стоимость одного машино-часа	руб.	$C_{м.ч.}$	0,05
Норматив дополнительной заработной платы	%	$H_{доп.з.п.}$	15
Ставка отчислений в Фонд социальной защиты населения	%	$H_{фсзн.}$	34
Ставка отчислений по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний	%	$H_{бгс.}$	0,6
Норматив прочих затрат	%	$H_{пз.}$	25
Норматив общепроизводственных и общехозяйственных расходов	%	$H_{обп.обх.р.}$	30
Ставка НДС	%	$H_{ндс}$	20,0
Налог на прибыль	%	$H_{п}$	20,0

Выбранные нормативные величины соответствуют стандартам и будут использоваться для последующих расчетов себестоимости разработки созданного веб-приложения [27].

6.3 Методика обоснования цены

В условиях рыночной экономики веб-приложения представляют собой законченные программные продукты, которые могут быть реализованы пользователям или организациям на коммерческой основе.

Постоянное развитие ИТ-сферы требует регулярного обновления и создания новых и интересных решений. Для разработчика экономическая эффективность заключается в возможности получения прибыли от реализации веб-приложения. Для пользователя экономическая эффективность веб-приложения заключается в экономии времени и ресурсов, которая достигается за счёт данных факторов:

- снижения автоматизации управления информацией о фильмах;
- упрощения бронирования билетов;
- легкого взаимодействия с другими пользователями и обмена опытом.

Оценка стоимости программного продукта со стороны разработчика включает определение всех понесённых затрат, среди которых:

- основная и дополнительная заработная плата исполнителей;

- отчисления в фонды социального страхования;
- расходы на эксплуатацию техники (машинное время);
- прочие производственные и общехозяйственные расходы;
- затраты на сопровождение программного продукта после сдачи.

На основе этих затрат рассчитываются себестоимость и отпускная цена веб-приложения. Такой подход позволяет объективно оценить эффективность проекта и определить его коммерческую привлекательность.

6.3.1 Определение объема программного средства

В таблице 6.2 указаны в укрупненном виде все работы и этапы разработки, в действительности выполненные для создания указанного в дипломном проекте программного средства и количество рабочих дней, которые были реально затрачены для выполнения всех этих работ.

Были учтены затраты на разработку программного средства, поиск нужных компонентов платформы, с сервисами которой будет интегрировано программное средство, а также тестирование и написание руководства пользователя.

Таблица 6.2 – Затраты рабочего времени на разработку ПС

Содержание работ	Исполнитель	Затраты рабочего времени, час.
Исследование и анализ требований	Бизнес-аналитик	12
Анализ аналогичных веб-приложений	Бизнес-аналитик	10
Подготовка технического задания	Бизнес-аналитик	16
Разработка UI/UX дизайна	Дизайнер	42
Проектирование базы данных и архитектуры веб-приложения	Бизнес-аналитик	38
Реализация модели базы данных	Backend-разработчик	22
Разработка серверной части веб-приложения	Backend-разработчик	185
Тестирование серверной части веб-приложения	Тестировщик	24
Разработка клиентской части веб-приложения	Frontend-разработчик	260
Тестирование клиентской части веб-приложения	Тестировщик	30
Интеграционное тестирование	Тестировщик	15
Отладка и исправление ошибок	Backend-разработчик	20
Подготовка к развертыванию на сервере	Backend-разработчик	6
Запуск готового веб-приложения	Backend-разработчик	2
Всего		682

Таким образом, занятость на проекте бизнес-аналитика составляет 76 час., дизайнера – 42 час., frontend-разработчика – 260 час., backend-разработчика – 235 час., тестировщика – 69 час.

6.3.2 Расчет основной заработной платы

После определения часовых ставок и трудозатрат исполнителей определяются основные заработные платы всех исполнителей. Основная заработная плата отдельного специалиста будет рассчитываться по формуле 6.1. Результаты подсчетов всех исполнителей проекта представлены в таблице 6.4.

$$C_{\text{оз}} = T_{\text{раз}} \cdot C_{\text{зп}}, \quad (6.1)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$T_{\text{раз}}$ – трудоемкость (чел./час.);

$C_{\text{зп}}$ – средняя часовая ставка руб./час.

Результаты подсчетов представлены в таблице 6.4.

Таблица 6.4 – Расчет основной заработной платы специалистов

Исполнитель	Затраты рабочего времени, час.	Средняя часовая ставка, руб./час	Основная заработная плата, руб.
Бизнес-аналитик	66	13,85	1 052,6
Дизайнер	40	10,55	443,1
Frontend-разработчик	280	14,17	3 684,2
Backend-разработчик	184	15,23	3 579,05
Тестировщик	59	10,68	736,92
Всего	629		9 495,87

При разработке программного средства основная заработная плата бизнес-аналитика составит 1 052,6 руб., дизайнера – 443,1 руб., frontend-разработчика – 3 684,2 руб., backend-разработчика – 3 579,05 руб., тестировщика – 736,92 руб. Суммарная основная заработная плата всех специалистов проекта по разработке веб-приложения по администрированию кинотеатра составит 9 495,87 руб.

6.3.3 Расчет дополнительной заработной платы

Законодательство о труде предусматривает наличие выплат дополнительной заработной платы. Дополнительная заработная плата – это дополнительные средства, выплачиваемые работнику сверх основной заработной платы за выполнение определенной работы или за достижение определенных целей. Такие выплаты могут включать оплату за сверхурочные часы, работу в выходные и праздничные дни, а также за выполнение обязанностей временно отсутствующего работника.

Размер дополнительной заработной платы устанавливается в процентах от основной и зависит от норм, принятых на конкретном предприятии или внутри отрасли. Определяется по нормативу в процентах к основной заработной плате по формуле 6.2. При этом конкретные условия и порядок начисления таких выплат должны быть четко прописаны в трудовом договоре или локальных нормативных актах организации.

$$C_{\text{доп.з.п.}} = \frac{C_{\text{осн.з.п.}} \cdot N_{\text{доп.з.п.}}}{100}, \quad (6.2)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$N_{\text{доп.з.п.}}$ – норматив дополнительной заработной платы, %.

$$C_{\text{доп.з.п.}} = 9\,495,87 \cdot 15 / 100 = 1\,424,38 \text{ руб.}$$

В итоге вычисления была получена сумма дополнительной заработной платы за время разработки данного проекта.

6.3.4 Расчет отчислений в Фонд социальной защиты населения и по обязательному страхованию

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются законодательными актами.

Вычисляются отчисления по формуле 6.3.

$$C_{\text{фсзн}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot N_{\text{фсзн}}}{100}, \quad (6.3)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$C_{\text{дз}}$ – дополнительная заработная плата на конкретное ПС, руб.;

$N_{\text{фсзн}}$ – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.4.

$$C_{\text{бгс}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot N_{\text{бгс}}}{100}, \quad (6.4)$$

$$C_{\text{фсзн}} = \frac{(9\,495,87 + 1\,424,38) \cdot 34}{100} = 3\,712,89 \text{ руб.}$$

$$C_{\text{бгс}} = \frac{(9\,495,87 + 1\,424,38) \cdot 0,6}{100} = 65,52 \text{ руб.}$$

Таким образом общие отчисления в БРУСП «Белгосстрах» составили 65,52 руб., а в фонд социальной защиты населения – 3 712,89 руб.

6.3.5 Расчет суммы прочих прямых затрат

Сумма прочих затрат $C_{\text{пз}}$ определяется как произведение основной заработной платы исполнителей на конкретное программное средство $C_{\text{оз}}$ на норматив прочих затрат в целом по организации $N_{\text{пз}}$, и находится по формуле 6.5. $N_{\text{пз}}$ устанавливается в соответствии с учетной политикой предприятия и может

включать в себя расходы на обслуживание оборудования, амортизацию, административные и другие накладные издержки.

$$C_{пз} = \frac{C_{оз} \cdot H_{пз}}{100}. \quad (6.5)$$

$$C_{пз} = 9\,495,87 \cdot 25 / 100 = 2\,373,97 \text{ руб.}$$

В итоге получена сумма прочих прямых затрат равная 2 373,97 руб.

6.3.6 Расчет общепроизводственных и общехозяйственных расходов

Сумма накладных расходов $C_{н.р.}$ – произведение основной заработной платы исполнителей на конкретное программное средство $C_{оз}$ на норматив накладных расходов в целом по организации $H_{н.р.}$.

Сумму накладных расходов можно рассчитать по формуле 6.6.

$$C_{н.р.} = \frac{C_{оз} \cdot H_{н.р.}}{100}. \quad (6.6)$$

Сумма накладных расходов составит:

$$C_{н.р.} = \frac{9\,495,87 \cdot 30}{100} = 2\,848,76 \text{ руб.}$$

Таким образом, была получена сумма норматива накладных расходов по организации на сумму 2 848,76 руб. Эти затраты включают в себя административные, хозяйственные и другие сопутствующие расходы, необходимые для поддержания рабочего процесса. Учет накладных расходов позволяет более точно оценить себестоимость разработки программного средства и оптимизировать бюджет проекта.

6.3.7 Сумма расходов на разработку программного средства

Сумма расходов на разработку программного средства C_p определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, суммы прочих затрат и суммы накладных расходов.

Общая сумма расходов на разработку программного средства вычисляется по формуле 6.7.

$$C_p = C_{оз} + C_{дз} + C_{фсзн} + C_{бгс} + C_{пз} + C_{н.р.} \quad (6.7)$$

Все данные, необходимые для вычисления, есть, поэтому можно определить сумму расходов на разработку программного средства.

$$C_p = 9\,495,87 + 1\,424,38 + 3\,712,89 + 65,52 + 2\,373,97 + 2\,848,76 = 19\,921,39 \text{ руб.}$$

Сумма расходов на разработку программного средства вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 19 921,39 рублей.

6.3.8 Расходы на сопровождение и адаптацию

Сумма расходов на сопровождение и адаптацию программного средства C_{pca} определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию H_{pca} , и находится по формуле 6.8.

$$C_{pca} = \frac{C_p \cdot H_{pca}}{100}, \quad (6.8)$$

$$C_{pca} = 19\,921,39 \cdot 10 / 100 = 1\,992,14 \text{ руб.}$$

Сумма расходов на сопровождение и адаптацию была вычислена на основе данных, рассчитанных ранее в данном разделе.

Все проведенные выше расчеты необходимы для вычисления полной себестоимости разрабатываемого проекта.

6.3.9 Полная себестоимость

Полная себестоимость программного средства C_{Π} определяется как сумма двух элементов: суммы расходов на разработку C_p и суммы расходов на сопровождение и адаптацию программного средства C_{pca} .

Полная себестоимость C_{Π} вычисляется по формуле 6.9.

$$C_{\Pi} = C_p + C_{pca}, \quad (6.9)$$

$$C_{\Pi} = 19\,921,39 + 1\,992,14 = 21\,913,53 \text{ руб.}$$

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

6.3.10 Определение цены, оценка эффективности

Для оценки отпускной цены разработки необходимо рассмотреть цены разработки приложений, обладающих аналогичным функционалом. Это позволит оценить потенциальную прибыль от продажи продукта, изучить конкурентов на рынке и принять обоснованные решения относительно стратегии маркетинговых действий по отношению к веб-приложению.

При формировании стоимости на подобных онлайн-калькуляторах учитываются такие параметры, как количество и сложность функциональных модулей, тип платформы, наличие административной панели, дизайн, требования к безопасности, а также техническая поддержка. Эти ресурсы основываются на

статистике выполненных проектов и предоставляют усреднённую оценку стоимости, что делает их удобным инструментом для предварительной калькуляции. Однако необходимо учитывать, что итоговая сумма может варьироваться в зависимости от специфики реализации и региона разработки.

Первым ресурсом является сайт estimatemyapp.com. Выбрав все необходимые параметры, выходит, что на разработку схожего ресурса в среднем должен уйти 84 дня разработки, а также стоимость в 28 800 \$, что составит 86 423 рублей. Наглядная демонстрация результатов работы ресурса представлена на рисунке 6.1.

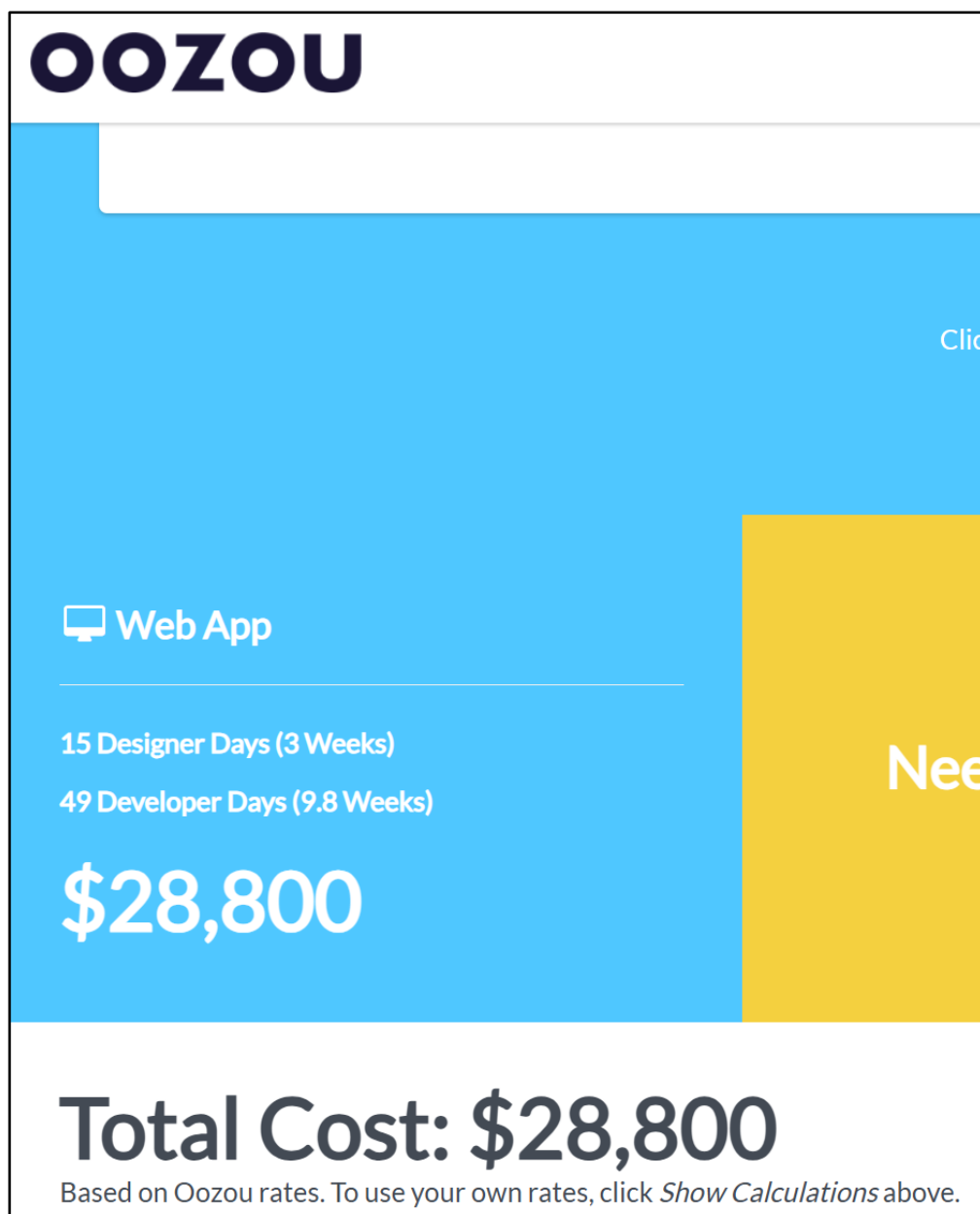


Рисунок 6.1 – Результат работы сайта estimatemyapp.com

Вторым ресурсом является сайт ios-lab.ru. Этот ресурс работает по аналогии с предыдущим, за исключением того, что итоговая цена будет представлена в российских рублях. На рисунке 6.2 представлен результат работы сайта-агрегатора услуг по разработке различных программных средств.

Рисунок 6.2 – Результат работы сайта ios-lab.ru

Как видно, примерная сумма похожего ресурса обойдется в 1 011 701 рублей, что в пересчете на белорусские рубли составляет 38 240,27 рублей.

Последним ресурсом является сайт thebestapp.ru. Этот ресурс похож на два ранее рассмотренных аналога, но, как и в предыдущем, итоговая цена представлена в российских рублях. Ниже на рисунке 6.3 приведен результат подсчетов.

Рисунок 6.3 – Результат работы сайта thebestapp.ru

Выбрав все необходимые компоненты системы, приблизительная стоимость проекта составляет 1 610 000 российских рублей, что в пересчете в белорусскую валюту составляет 60 854,78 белорусских рубля.

Как видно из результатов, отпускная цена на сайтах-агрегаторах услуг по разработке программного обеспечения не сильно отличается друг от друга.

Из проведенной выше информации видно, что цена разработки подобного продукта варьируется в диапазоне от 32919,86 до 113475 белорусских рублей. Средняя цена разработки приложения с аналогичным функционалом рассчитывается как среднее арифметическое указанных цен:

$$Ц_{\text{Аср}} = (86\,423 + 38\,240,27 + 60\,854,78) / 3 = 61\,839,35 \text{ руб.}$$

Минимальная цена разработки веб-приложения с аналогичным функционалом составляет 38 240,27 белорусских рубля.

Т. к. полная себестоимость разработки и поддержки ПС составляет 21 913,53 рубля, есть возможность установить цену соответствующую самой низкой представленной цене. Установим $Ц_{\text{с НДС}} = 38\,240,27$ рублям.

Цена ПС без НДС рассчитывается по формуле 6.10.

$$Ц_{\text{без НДС}} = Ц_{\text{с НДС}} \cdot \frac{100}{(Н_{\text{НДС}} + 100)}, \quad (6.10)$$

$$Ц_{\text{без НДС}} = 38\,240,27 \cdot (100 / 120) = 31\,866,89 \text{ руб.}$$

Прибыль от реализации ПС рассчитывается по формуле 6.11.

$$П_{\text{р.}} = Ц_{\text{без НДС}} - С_{\text{п.}}, \quad (6.11)$$

$$П_{\text{р.}} = 31\,866,89 - 21\,913,53 = 9\,953,36 \text{ руб.}$$

Зная общую сумму на разработку, с затратами на сопровождение и адаптацию, веб-приложения и прибыль от реализации ПС можно рассчитать рентабельность реализации разрабатываемого приложения.

Рентабельность ПС рассчитывается по формуле 6.12.

$$P = \frac{П_{\text{р.}}}{С_{\text{п.}}} \cdot 100\%, \quad (6.12)$$

$$P = (9\,953,36 / 21\,913,53) \cdot 100 = 45,42 \text{ \%}.$$

В результате расчетов было определено, что цена без НДС составляет 31 866,89 руб., рентабельность – 45,42 %.

6.4 Вывод по разделу

В ходе тщательного анализа были определены ключевые параметры, влияющие на стоимость разработки веб-приложения. Отпускная цена, которая отражает общую стоимость проекта, включая затраты на разработку, тестирование и последующую поддержку, была установлена на основе изучения всех факторов. Этот анализ позволил учесть все аспекты, начиная от

первоначальных затрат на разработку и заканчивая долгосрочными расходами на обслуживание и развитие проекта.

В таблице 6.5 представлены результаты расчетов основных экономических показателей, включая стоимость разработки и ожидаемую прибыль. Эти данные дают четкое представление о финансовой целесообразности проекта, позволяя оценить, насколько быстро проект окупит свои начальные затраты и начнет приносить прибыль. Такой подход к анализу позволяет не только оценить текущую стоимость проекта, но и прогнозировать его будущую доходность, что является важным фактором при принятии решений о разработке новых продуктов или услуг.

Таблица 6.5 – Результаты расчетов основных экономических показателей разработки веб-приложения кинотеатра

Наименование показателя	Значение
Время разработки, часов	682
Количество разработчиков, чел.	5
Основная заработная плата, руб.	9 495,87
Дополнительная заработная плата, руб.	1 424,38
Отчисления в Фонд социальной защиты населения, руб.	3 712,89
Отчисления в БРУСП «Белгосстрах», руб.	65,52
Прочие прямые затраты, руб.	2 373,97
Накладные расходы, руб.	2 848,76
Себестоимость разработки программного средства, руб.	19 921,39
Расходы на сопровождение и адаптацию, руб.	1 992,14
Полная себестоимость, руб.	21 913,53
Цена реализации, сформированная на основе проведенного маркетингового анализа рынка, руб.	38 240,27
Прибыль от реализации, руб.	9 953,36
Рентабельность разработки, %	45,42

Данная таблица с результатами расчетов основных экономических показателей также представлена в приложении Л.

На основании представленных в таблице 6.5 данных можно сделать обоснованные выводы о затратах, потраченных на разработку веб-приложения, а также о его экономической эффективности.

Рентабельность разработки составила 33,27 %, что является высоким показателем для индивидуального проекта или продукта, реализуемого небольшой командой. Это свидетельствует о том, что затраты на создание программного средства оправданы, а при установленных рыночных условиях проект способен приносить стабильную прибыль. Такой уровень рентабельности позволяет рассматривать продукт как коммерчески целесообразный и потенциально успешный при дальнейшей реализации. Высокая экономическая эффективность проекта может служить основанием для масштабирования, инвестирования в расширение функциональности или выхода на новые сегменты рынка.

Заключение

Перед началом разработки дипломного проекта были проанализированы аналоги веб-приложения по администрированию кинотеатров, выявлены их положительные и отрицательные стороны.

Также была выбрана подходящая технологическая стековая модель, включая СУБД для организации базы данных, язык программирования для серверной части и технологии для создания клиентского интерфейса.

На этапе планирования проекта были разработаны следующие документы:

- логическая схема базы данных;
- диаграмма вариантов использования;
- диаграмма развертывания;
- блок-схема алгоритма приобретения лицензии.

В процессе выполнения дипломного проекта было разработано клиент-серверное приложение с использованием фреймворка NestJS для написания серверной части и библиотеки React в совокупности с библиотекой компонентов Mantine для написания клиентского приложения. Разработан механизм аутентификации и авторизации с использованием jwt-токена. Разработана база данных, использующая третью нормальную форму. Подключение к разработанной базе данных осуществляется с использованием такой ORM, как TypeORM. Так же был разработан функционал, позволяющий администратору проводить сканирование и проверку билетов с помощью QR-code и камеры, сканирующей их. Был подключен сервис онлайн оплаты, позволяющий пользователям осуществлять покупку билетов онлайн. В пояснительной записке подробно описано, как использовать веб-приложение, включая основные сценарии работы, что облегчает понимание и эффективное использование приложения со стороны пользователей.

После тщательного тестирования всех элементов приложения и получения удовлетворительных результатов можно сделать вывод, что все ключевые требования были выполнены, и приложение функционирует без ошибок.

В руководстве пользователя были описаны основные действия, доступные различным пользователям в системе в зависимости от их роли.

В ходе дипломного проектирования был произведен расчет стоимости разработки представленного в дипломном проекте программного продукта, что подтверждает его экономическую целесообразность.

Результатом является готовое веб-приложение по администрированию кинотеатра, которое полностью соответствует заявленным требованиям и обладает гибкостью для будущих расширений функционала.

					ДП 00.00 ПЗ		
		ФИО	Подпись	Дата	Заключение		
Разраб.		Халалеенко А.Н.					
Провер.		Тимонович Г.Л.					
Н. контр.		Алешаускас В.А.					
Утв.		Блинова Е.А.			БГТУ 1-40 05 01, 2025		
					Лит.	Лист	Листов
					У	1	1

Список использованных источников

1. mooon.by : [сайт]. – Минск, 2020 – 2025 – URL: <https://mooon.by/> (дата обращения: 13.02.2025).
2. ByCard : [сайт]. – Минск, 2012 – 2025 – URL: <https://bycard.by/> (дата обращения: 17.02.2025).
3. Афиша Relax.by : [сайт]. – Минск, 2025 – URL: <https://afisha.relax.by/kino/minsk/> (дата обращения: 18.02.2025).
4. Habr – Проектирование ПО: [сайт]. – Сан-Франциско, 2006 – 2025. – URL: <https://habr.com/ru/articles/74330/> (дата обращения: 01.03.2025).
5. React documentation : [сайт]. – Сан-Франциско, 2013 – 2025 – URL: <https://ru.react.js.org/> (дата обращения: 20.04.2025)
6. NestJS documentation : [сайт]. – Варшава, 2017 – 2025 – URL: <https://nestjs.com/> (дата обращения: 01.04.2025).
7. Web API's Basics : [сайт]. – Нью-Йорк, 2025 – URL: <https://tayfun-guven25.medium.com/web-apis-basics-698e09f090ba> (дата обращения: 01.04.2025).
8. TypeORM documentation : [сайт]. – Берлин, 2025 – URL: <https://docs-nestjs.netlify.app/techniques/database> (дата обращения: 02.04.2025).
9. PostgreSQL : [сайт]. – Брик, Нью-Джерси, 1997 – 2025 – URL: <https://www.postgresql.org/> (дата обращения: 05.03.2025).
10. Layered (N-Layer) Architecture : [сайт]. – Сан-Франциско, 2025 – URL: <https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-e15ffdb7fa42> (дата обращения: 04.04.2025).
11. Архитектура веб-приложения: компоненты, слои и типы : [сайт]. – Ростов-на-Дону, 2025 – URL: <https://itanddigital.ru/webapplications> (дата обращения: 04.03.2025).
12. Documentation JS : [сайт]. – Лондон, 2021 – 2025 – URL: <https://jsdocumentation.netlify.app/> (дата обращения: 26.02.2025).
13. Documentation NodeJS : [сайт]. – Сан-Франциско, 2009 – 2025 – URL: <https://nodejs.org/en/> (дата обращения: 26.02.2025).
14. Phaser : [сайт]. – Лондон, 2025 – URL: <https://phaser.io/> (дата обращения: 08.03.2025).
15. Electron : [сайт]. – Сан-Франциско, 2025 – URL: <https://www.electronjs.org/> (дата обращения: 08.03.2025).
16. Angular documentation : [сайт]. – Маунтин-Вью, 2010 – 2025 – URL: <https://v17.angular.io/docs/> (дата обращения: 11.03.2025).
17. Vue documentation : [сайт]. – Сан-Франциско, 2014 – 2025 – URL: <https://vuejs.org/api/> (дата обращения: 13.03.2025).

					ДП 00.00 ПЗ		
		ФИО	Подпись	Дата	Список использованных источников		
Разраб.	Халалеенко А.Н.						
Провер.	Тимонович Г.Л.						
Н. контр.	Алешаускас В.А.						
Утв.	Блинова Е.А.				БГТУ 1-40 05 01, 2025		
					Лит.	Лист	Листов
					У	1	2

18. jQuery documentation : [сайт]. – Сан-Хосе, 2006 – 2025 – URL: <https://api.jquery.com/> (дата обращения: 16.03.2025).
19. MongoDB documentation : [сайт]. – Нью-Йорк, 2009 – 2025 – URL: <https://www.mongodb.com/docs/> (дата обращения: 15.03.2025).
20. MySQL documentation : [сайт]. – Сан-Матео, 1995 – 2025 – URL: <https://dev.mysql.com/doc/> (дата обращения: 15.03.2025).
21. TypeScript documentation : [сайт]. – Редмонд, 2012 – 2025 – URL: <https://www.typescriptlang.org/docs/> (дата обращения: 15.03.2025).
22. HTTP documentation : [сайт]. – Женева, 1991 – 2025 – URL: <https://httpwg.org/specs/> (дата обращения: 15.03.2025).
23. Docker documentation : [сайт]. – Сан-Франциско, 2011 – 2025 – URL: <https://docs.docker.com/> (дата обращения: 15.03.2025).
24. SQL (ISO standard) : [сайт]. – Женева, 1992 – 2025 – URL: <https://www.iso.org/standard/45498.html> (дата обращения: 14.04.2025).
25. Mantine UI documentation : [сайт]. – Сан-Франциско, 2021 – 2025 – URL: <https://mantine.dev/> (дата обращения: 14.03.2025).
26. JWT (JSON Web Tokens) : [сайт]. – Сан-Франциско, 2015 – 2025 – URL: <https://jwt.io/introduction> (дата обращения: 30.03.2025).
27. Экономическое обоснование дипломных проектов : методические указания для студентов специальностей 1-47 01 01 «Издательское дело», 1-47 02 01 «Технология полиграфического производства», 1-36 06 01 «Полиграфическое оборудование и средства обработки информации», 1-46 01 02 «Информационные системы и технологии»; сост. Т. В. Каштелян. – Минск : БГТУ, 2013. – 86 с.

Приложение А. Диаграмма развертывания

Приложение Б. Диаграмма вариантов использования

Приложение В. Логическая схема базы данных

Приложение Г. Блок-схема алгоритма покупки билета

Приложение Д. Блок-схема алгоритма проверки билета

Приложение Е. Реализация функций сервисов

Сервис User

```

@Injectable()
export class UserService {
  constructor(private readonly dataService: DatabaseService) {}

  async create(createUserDto: RegisterUserDto) {
    const newUser = await this.dataService.user.save(createUserDto);

    return newUser;
  }
  async findUserEmail(email: string) {
    const newUser = await this.dataService.user.find({
      where: { email: email },
    });

    return newUser;
  }
  async GetUsers(id: string) {
    const foundUsers = await this.dataService.user.find({
      where: {
        id: Not(id),
      },
      relations: ['auditt'],
    });
    const newArray = foundUsers.map(obj => {
      return { ...obj, good: 0, bad: 0 };
    });
    for (let i = 0; i < foundUsers.length; i++) {
      let good = 0;
      let bad = 0;
      const obj = foundUsers[i];
      // Перебор вложенного массива в каждом объекте
      for (let j = 0; j < obj.auditt.length; j++) {
        if (obj.auditt[j].OperationType === 'Покупка билета') {
          good++;
        } else {
          bad++;
        }
      }
      newArray[i].good = good;
      newArray[i].bad = bad;
    }
    return newArray;
  }

  public async findOneById(id): Promise<User> {
    const foundUser = await this.dataService.user.findOne({
      where: { id } });
    return foundUser;
  }

```

```

    }

    public async findOneByEmail(email: string): Promise<User> {
        const foundUser = await this.dataService.user.findOne({
where: { email } });
        return foundUser;
    }

    async DeleteUser(idUser: string, id: string) {
        await this.dataService.user.update(idUser, {status: 'DisAc-
tive'} );
        const foundUsers = await this.dataService.user.find({
            where: {
                id: Not(id),
            },
            relations: ['auditt'],
        });
        const newArray = foundUsers.map(obj => {
            return { ...obj, good: 0, bad: 0 };
        });
        for (let i = 0; i < foundUsers.length; i++) {
            let good = 0;
            let bad = 0;
            const obj = foundUsers[i];
            // Перебор вложенного массива в каждом объекте
            for (let j = 0; j < obj.auditt.length; j++) {
                if (obj.auditt[j].OperationType === 'Покупка билета') {
                    good++;
                } else {
                    bad++;
                }
            }
            newArray[i].good = good;
            newArray[i].bad = bad;
        }
        return newArray;
    }

    async UnDeleteUser(idUser: string, id: string) {
        await this.dataService.user.update(idUser, { status: 'Ac-
tive' });

        const foundUsers = await this.dataService.user.find({
            where: {
                id: Not(id),
            },
            relations: ['auditt'],
        });
        const newArray = foundUsers.map(obj => {
            return { ...obj, good: 0, bad: 0 };
        });
        for (let i = 0; i < foundUsers.length; i++) {
            let good = 0;
            let bad = 0;

```

```

    const obj = foundUsers[i];
    // Перебор вложенного массива в каждом объекте
    for (let j = 0; j < obj.auditt.length; j++) {
        if (obj.auditt[j].OperationType === 'Покупка билета') {
            good++;
        } else {
            bad++;
        }
    }
    newArray[i].good = good;
    newArray[i].bad = bad;
}
return newArray;
}

public async logout({ res, cookies }: RequestWithUser) {
    res.clearCookie(ACCESS_TOKEN);
    res.clearCookie(REFRESH_TOKEN);
    const foundUser = await this.dataService.token.delete({
        token: cookies.refreshToken,
    });

    return foundUser;
}

async ChangeUserInfo(
    idUser: string,
    name: string,
    email: string,
    password: string,
) {
    const foundUserNow = await this.dataService.user.findOne({
        where: { id: idUser },
    });
    if (password === '') {
        password = foundUserNow.password;
    } else {
        const passwordRegex =
            /^[a-zA-Z0-9!@#$$%^&*()_+{}[\]:;<>,.?~\\\/`' "|-]{8,30}$/;

        if (passwordRegex.test(password)) {
            const saltOrRounds = SALT_OR_ROUNDS;
            const hashedPassword = await bcrypt.hash(password, saltOrRounds);
            password = hashedPassword;
        } else {
            throw new BadRequestException(
                `Пароль должен содержать только латинские буквы,
                цифры и символы. Длина пароля от 8 до 30 символов`,
                'password',
            );
        }
    }
}
}

```

```

    if (name === '') {
      name = foundUserNow.name;
    } else {
      const nameRegex = /^[a-zA-Za-яA-Я]{3,30}$/;
      if (!nameRegex.test(name)) {
        throw new BadRequestException(
          `Имя должно содержать только буквы русского или ан-
            глийского алфавита`,
          'name',
        );
      }
    }

    if (email === '') {
      email = foundUserNow.email;
    } else {
      const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-
        ]+\.[a-zA-Z]{2,}$/;

      if (!emailRegex.test(email)) {
        throw new BadRequestException(
          'Некорректный адрес электронной почты',
          'email',
        );
      }
    }

    const foundUser = await this.dataService.user.up-
date(idUser, {
      name: name,
      email: email,
      password: password,
    });
    const foundUsers = await this.dataService.user.findOne({
      where: { id: idUser },
    });

    return foundUsers;
  }
}

```

Приложение Ж. Реализация классов-валидаторов

Сервис Film

```

@Injectable()
export class FilmService {
  constructor(private readonly databaseService: DatabaseService) {}

  async create(createFilmDto: CreateFilmDto) {
    if (createFilmDto.startRelease >= createFilmDto.endRelease)
    {
      throw new BadRequestException(
        'Дата начала проката не может быть позже или равна даты
оканчания',
        'дата',
      );
    }
    const filmName = await this.databaseService.film.findOne({
      where: {
        name: createFilmDto.name,
        year: createFilmDto.year,
      },
    });
    if (filmName) {
      if (
        filmName.name == createFilmDto.name &&
        filmName.year === filmName.year
      ) {
        throw new BadRequestException(
          'Не может быть фильма с одинаковым названием в один
год',
          'name',
        );
      }
    }
    const genre = await this.databaseService.genre.findOne({
      where: {
        name: createFilmDto.genre,
      },
    });
    if (!genre) {
      const genre = await this.databaseService.genre.save({
        name: createFilmDto.genre,
      });

      createFilmDto.genre = genre.id;
    } else {
      createFilmDto.genre = genre.id;
    }
  }
}

```

```

    const filmData: DeepPartial<Film> = classToPlain(createFilm-
mDto);

    const createFilm = await this.databaseServ
ice.film.save(filmData);
    return createFilm;
    async findAll() {
        const films = await this.databaseService.film.find({
            relations: ['genre'],
        });
        return films;
    }
    async GetFilmHall() {
        return { films, halls };
    }
    async GetDataFilm(IdFilm: string) {
        await this.databaseService.session.query('SELECT CheckSes
sion()');
        let places = [];
        const films = await this.databaseService.film.findOne({
            where: {
                id: IdFilm,
            },
            relations: ['genre'],
        });

        const sessions = await this.databaseService.session.find({
            where: {
                film: { id: IdFilm },
            }

            return { films, sessions };
        }

        GetFilmGenre(idGenre: string) {
            return this.databaseService.film.find({
                where: { genre: { id: idGenre } },
                relations: ['genre'],
            });
        }

        async DeleteFilm(idFilm: string) {
            await this.databaseService.film.delete({
                id: idFilm,
            });
            const films = await this.databaseService.film.find({});
            return films;
        }

        async GetFilmByName(idFilm: string) {
            const films = await this.databaseService.film.find({
                where: {
                    name: Like(`%${idFilm}%`),
                },
            });

```

```

    });

    return films;  async loginCheck(login) {
    async BuyTicket(sessions, tickets, id) {
    let massOccupied = [];

    for (let i = 0; i <= tickets.length - 1; i++) {
        const pl = await this.databaseService.place
            .createQueryBuilder('place')
            .leftJoinAndSelect('place.hall', 'hall')
            .leftJoinAndSelect('hall.sessions', 'session')
            .where('place.place = :ticket', { ticket: tickets[i] })
            .andWhere('session.id = :sessionId', { sessionId: ses-
sessions })
            .getOne();

        const checkTicket = await this.databaseServ-
ice.ticket.findOne({
            where: {
                place: { id: pl.id },
                session: { id: sessions },
            },
            relations: ['place'],
        });

        if (checkTicket == null) {
            const tick = await this.databaseService.ticket.save({
                user: { id: id },
                place: { id: pl.id },
                session: { id: sessions },
            });
        } else {
            massOccupied.push(checkTicket);
        }
    }
    return massOccupied;
}

async GetUserЧислооSession(idsession) {
    const userinfo = await this.databaseService.ticket.find({
        where: {
            session: { id: idsession },
        },
        relations: ['user', 'place', 'session'],
    });
    return userinfo;
}

async GetTicketUser(idUser: string) {
    await this.databaseService.session.query('SELECT
CheckSes
sion()');
    const tick = await this.databaseService.ticket.find({
        where: { user: { id: idUser } },
    });
}

```

```

        relations: ['place', 'user', 'session', 'session.film',
'ses
sion.hall']],
        const filteredData = tick.filter(item => item.session.status
===
'Active');
        return filteredData;

```

Сервис Hall

```

@Injectable()
export class HallService {
    constructor(private readonly databaseService: DatabaseServ-
ice) {}
    private id: string;
    async create(createHallDto: CreateHallDto) {
        const checkHallName = await this.databaseServ-
ice.hall.findOne({
            where: {
                name: createHallDto.name,
            },
        });
        if (checkHallName) {
            throw new BadRequestException(
                'Зал с таким название уже существует',
                'hall',
            );
        }
        const checkTypePlace = await this.databaseServ
ice.typeplace.findOne({
            where: {
                cost_normal: createHallDto.coast_norm,
                cost_vip: createHallDto.coast_vip,
            },
        });
        if (!checkTypePlace) {
            const checkTypePlaces = await this.databaseServ
ice.typeplace.save({
                cost_normal: createHallDto.coast_norm,
                cost_vip: createHallDto.coast_vip,
            });
            this.id = checkTypePlaces.id;
        } else {
            this.id = checkTypePlace.id;
        }

        const createHall = await this.databaseService.hall.save({
            name: createHallDto.name,
            count_rows: createHallDto.count_rows,
            count_place: createHallDto.count_place,
            type_Place: this.id,
        });
    }
}

```



```

    const rowCount = createHallDto.count_place * create
    HallDto.count_rows;
    for (let i = 1; i <= rowCount; i++) {
        await this.databaseService.place.save({
            place: i,
            hall: {id:createHall.id},
        });
    }
    return createHall;
}

```

Контроллер Ticket

```

const stripe = require('stripe')(
    'sk_test_51P8Jvj2MUFusCVLsvhlbRhqtZ-
LXPFPkgjwCGsR24tbDpiQN4PO7L3NTRw3w1LLFQ9x4MK7h7TW0aFOxR6y7jJjzP0
0zJkBI9UQ',
);
@Controller('ticket')
export class TicketController {
    constructor(private readonly ticketService: TicketService)
    {}

    @UseGuards(JwtAuthGuard)
    @Post('BuyTicket')
    async BuyTicket(
        @Req() req: RequestWithUser,
        @Res() res: Response,
        @Body() data: any,
    ) {
        const lineItems = [
            {
                price_data: {
                    currency: 'byn',
                    product_data: {
                        name: 'К оплате',
                    },
                },
                unit_amount: Math.round(data.fullPrice * 100),
            },
            {
                quantity: 1,
            },
        ];

        const IdFilm = data.IdFilm;
        const session = await stripe.checkout.sessions.create({
            payment_method_types: ['card'],
            line_items: lineItems,
            mode: 'payment',
            success_url: `http://localhost:3000/Film?ID=${Id-
Film}` ,
            cancel_url: 'http://localhost:3000/',
        });

        res.send(session.url);
    }
}

```

```

        const chatr_id = await this.waitForPaymentCompletion(session.id);
        return this.ticketService.BuyTicket(
            data.sessions,
            data.tickets,
            req.user.id,
            chatr_id
        );
    }

    async waitForPaymentCompletion(sessionId) {
        let tries = 0;
        const maxTries = 25;
        const delay = 1000;

        while (tries < maxTries) {
            const session = await stripe.checkout.sessions.retrieve(sessionId);
            const status = session.status;

            if (status !== 'open') {
                const paymentIntentId = session.payment_intent;

                const paymentIntent =
                    await stripe.paymentIntents.retrieve(paymentIntentId);

                return paymentIntent.latest_charge;
            }

            tries++;
            await new Promise(resolve => setTimeout(resolve, delay));
        }
    }

    @Post('GetUserIntoSession')
    findOne(@Body() data: any) {
        return this.ticketService.GetUserIntoSession(data.idsession);
    }

    @Post('CheckTicketCon')
    CheckTicketCon(@Body() data: any) {
        return this.ticketService.CheckTicketCon(data.data.decodedText);
    }

    @UseGuards(JwtAuthGuard)
    @Get('GetTicketUser')
    async GetTicketUser(@Req() req: RequestWithUser) {

```

```

        const result = await this.ticketService.GetTicketUser(req.user.id);
        return result;
    }

    @Delete('Dellorder')
    async Dellorder(@Body() data: any) {
        const refund = await stripe.refunds.create({
            charge: data.chatr_id,
            amount: Math.round(data.coast_set * 100),
        });
        return this.ticketService.Dellorder(
            data.id,
            data.idsession,
        );
    }
    @UseGuards(JwtAuthGuard)
    @Delete('DeleteTicket')
    async DeleteTicket(@Body() data: any, @Req() req: Request-
    WithUser) {

        const refund = await stripe.refunds.create({
            charge: data.chatr_id,
            amount: Math.round(data.coast_set * 100),
        });
        return this.ticketService.DeleteTicket(data.idTicket,
        req.user.id);
    }
}

```

Сервис Ticket

```

@Injectable()
export class TicketService {
    constructor(private readonly databaseService: DatabaseService) {}
    async BuyTicket(sessions, tickets, id, chatr_id, coast_set) {
        let massOccupied = [];
        for (let i = 0; i <= tickets.length - 1; i++) {
            const pl = await this.databaseService.place
                .createQueryBuilder('place')
                .leftJoinAndSelect('place.hall', 'hall')
                .leftJoinAndSelect('hall.sessions', 'session')
                .where('place.place = :ticket', { ticket: tickets[i] })
                .andWhere('session.id = :sessionId', { sessionId: sessions })
                .getOne();
            const checkTicket = await this.databaseService.ticket.findOne({
                where: {
                    place: { id: pl.id },
                    session: { id: sessions },
                },
                relations: ['place'],
            });

```

```

    });

    if (checkTicket == null) {
        let qrCodeData = await qrcode.toDataURL(
            `Зал:${pl.hall.name}; Место:${pl.place};`,
        );
        const tick = await this.databaseService.ticket.save({
            user: { id: id },
            place: { id: pl.id },
            session: { id: sessions },
            QR: qrCodeData,
            chatr_id: chatr_id,
            coast_set: coast_set[i],
        });
        const secretKey = 'mySecretKey';

        const originalText = tick.id;
        const cipher = crypto.createCipher('aes-256-cbc',
secretKey);
        let encryptedText = cipher.update(originalText, 'utf8',
'hex');
        encryptedText += cipher.final('hex');
        const updatedQRCodeData = await qrcode.toDataURL(
            `Зал:${pl.hall.name}; Место:${pl.place};
ID:${encryptedText}`,
        );
        // Затем обновите запись в базе данных с новыми данными
        QR
        кода
        const foundUser = await this.databaseService.ticket.up
date(tick.id, {
            QR: updatedQRCodeData,
        });
    } else {
        massOccupied.push(checkTicket);
    }
}
return massOccupied;
}

async GetUserIntoSession(idsession) {
    const userinfo = await this.databaseService.ticket.find({
        where: {
            session: { id: idsession },
        },
        relations: ['user', 'place', 'session'],
    });
    return userinfo;
}

async CheckTicketCon(decodedText) {

    const secretKey = 'mySecretKey';

```

```

    const decipher = crypto.createDecipher('aes-256-cbc',
secretKey);

    const startIndex = decodedText.indexOf('ID:')+3;

    const idText = decodedText.substring(startIndex);
    let decryptedText = decipher.update(idText, 'hex', 'utf8');
    decryptedText += decipher.final('utf8');

    const userinfo = await this.databaseService.ticket.find({
      where: {
        id: decryptedText,
      },
      relations: ['user', 'place'],
    });
    if(userinfo){
      const foundUser = await this.databaseService.ticket.up-
date(
        decryptedText,
        {
          status: 'DisActive',
        },
      );
    }
    return userinfo;
  }
  async GetTicketUser(idUser: string) {
    await this.databaseService.session.query('SELECT CheckSes
sion()');

    const tick = await this.databaseService.ticket.find({
      where: { user: { id: idUser } },
      relations: ['place', 'user', 'session', 'session.film',
'ses
sion.hall'],
    });

    const filteredData = tick
      .filter(item => item.session.status === 'Active')
      .map(item => {
        delete item.session.film.img;
        return item;
      });
    return filteredData;
  }

  async Dellorder(id: string, idsession: string) {
    const filmDel = await this.databaseService.ticket.delete({
      id: id,
    });

    const userinfo = await this.databaseService.ticket.find({

```

```

        where: {
            session: { id: idsession },
        },
        relations: ['user', 'place', 'session'],
    });
    return userinfo;
}

async DeleteTicket(idTicket: string, id: string) {

const filmdel = await this.databaseService.ticket.delete({
id: idTicket,
});
const tick = await this.databaseService.ticket.find({
where: { user: { id: id } },
relations: ['place', 'user', 'session', 'session.film', 'session.hall'],
});
const filteredDatas = tick
.filter(item => item.session.status === 'Active')
.map(item => {
delete item.session.film.img;
return item;
});
return filteredDatas;
}
}

```

Контроллер Session

```

@Controller('session')
export class SessionController {
    constructor(
        private readonly sessionService: SessionService,
        private readonly filmService: FilmService,
    ) {}

    @Post('AddSession')
    async create(@Body() createSessionDto: CreateSessionDto) {
        const FreeHall = await this.sessionService.getDataFreeHall(
            createSessionDto.dateSession,
            createSessionDto.hall,
        );
        const cinema = await this.filmService.GetDataFilm(createSessionDto.film);

        if (
            new Date(cinema.films.endRelease) < new Date(createSessionDto.dateSession)
        ) {
            throw new BadRequestException(`Прокат фильма уже закончен`, 'date');
        }
    }
}

```

```

    if (
        new Date(cinema.films.startRelease) >
        new Date(createSessionDto.dateSession)
    ) {
        throw new BadRequestException(`Прокат фильма еще не
начался `, 'date');
    }

    const timeString = `${createSessionDto.timeSession}`;
    const cinDuration = `${cinema.films.duration}`;
    const [hourss, minutess] = cinDuration.split(':');
    const cinDurations = new Date();
    cinDurations.setUTCHours(Number(hourss), Number(minutess),
0, 0);
    const [hours, minutes] = timeString.split(':');
    const times = new Date();
    times.setUTCHours(Number(hours), Number(minutes), 0, 0);
    const StartTime = times.getUTCHours() * 60 +
times.getMinutes();
    const timeFilm =
        cinDurations.getUTCHours() * 60 + cinDura-
tions.getMinutes();
    FreeHall.map(item => {
        const cinDurationi = `${item.timeSession}`;
        const [hoursi, minutesi] = cinDurationi.split(':');
        const cinDurationsi = new Date();
        cinDurationsi.setUTCHours(Number(hoursi), Num-
ber(minutesi), 0, 0);
        const cinDurationis = `${item.film.duration}`;
        const [hoursis, minutesis] = cinDurationis.split(':');
        const cinDurationsis = new Date();
        cinDurationsis.setUTCHours(Number(hoursis), Num-
ber(minutesis), 0, 0);
        const timeOtherSession =
            cinDurationsi.getMinutes() + cinDura-
tionsi.getUTCHours() * 60;
        const timeStartOtherSession =
            cinDurationsis.getMinutes() + cinDuration-
sis.getUTCHours() * 60;

        if (
            timeFilm + StartTime >= timeStartOtherSession &&
            StartTime <= timeOtherSession + timeStartOtherSession
        ) {
            throw new BadRequestException(
                `В данное время в данном зале идет фильм`,
                'hall',
            );
        }
    });
    const filmData = await this.sessionService.create(createSessionDto);

```

```

    return filmData;
}

@Get('GetSession')
getSession() {
    return this.sessionService.getSession();
}

@Get('/:id')
findOne(@Param('id') id: string) {
    return this.sessionService.findOne(+id);
}

@Patch('/:id')
update(@Param('id') id: string, @Body() updateSessionDto: UpdateSessionDto) {
    return this.sessionService.update(+id, updateSessionDto);
}

@Delete('DeleteSession')
deleteSession(@Body() data: any) {
    return this.sessionService.deleteSession(data.idSession);
}
}

```

Сервис Auth

```

@Injectable()
export class AuthService {
    constructor(
        private readonly jwtService: JwtService,
        private readonly configService: ConfigService,
        private readonly databaseService: DatabaseService,
    ) {}

    public getRefreshToken() {
        return this.configService.get<string>('REFRESH_TOKEN_SECRET');
    }

    public getJWTPathToCallback() {
        return this.configService.get<boolean>('JWT_REFRESH_PASS_TO_CALLBACK');
    }

    async validateUser(email: string, password: string): Promise<User> {

        const user = await this.databaseService.user.findOne({
            where: { email } });
        if (!user) {
            throw new NotAcceptableException('Неверный логин или пароль');
        }
    }
}

```



```

    }

    const passwordValid = await bcrypt.compare(password,
user.password);

    const isAuthenticated = user && passwordValid;
    if (!isAuthenticated) {
        throw new UnauthorizedException(`Неверный логин или
пароль`);
    }

    return user;
}

async login({ user, res }: RequestWithUser): Promise<LoginRe-
sponse> {
    try {
        const { email, id } = user;
        const foundedUser = await this.databaseServ-
ice.user.findOne({
            where: { id },
        });
        if (!foundedUser)
            throw new NotFoundException(`User with such id ${id}
does not exist`);
        if (!user)
            throw new NotFoundException(`User with such id ${id}
does not exist`);
        if (foundedUser.status==="DisActive"){
            throw new NotAcceptableException('Сожалею, но вы
заблокированы, обратитесь к администратору');
        }
        const { role } = user;

        const payload = { id, email, role };

        const refreshToken = this.jwtService.sign(payload, {
            secret: this.configService.get<string>(
                ENV_VARIABLES_NAMES.REFRESH_TOKEN_SECRET,
            ),
            expiresIn: this.configService.get<string>(
                ENV_VARIABLES_NAMES.REFRESH_TOKEN_EXPIRATION,
            ),
        });

        const accessToken = this.jwtService.sign(payload, {
            secret: this.configService.get<string>(
                ENV_VARIABLES_NAMES.ACCESS_TOKEN_SECRET,
            ),
            expiresIn: this.configService.get<string>(
                ENV_VARIABLES_NAMES.ACCESS_TOKEN_EXPIRATION,
            ),
        });
    }
}

```

```

    const session = await this.databaseService.token.findOne({
      where: { user: { id: payload.id } },
    });

    if (!session) {
      this.databaseService.token.save({
        user: { id: payload.id },
        token: refreshToken,
      });
    }

    this.databaseService.token.update(
      { user: { id: payload.id } },
      { token: refreshToken },
    );

    res.cookie(AccessToken, accessToken);
    res.cookie(RefreshToken, refreshToken);

    const accessTokenAuthorized = this.jwtService.sign(payload, {
      secret: this.configService.get<string>(
        EnvVariablesNames.ACCESS_TOKEN_SECRET,
      ),
      expiresIn: this.configService.get<string>(
        EnvVariablesNames.ACCESS_TOKEN_EXPIRATION,
      ),
    });

    return {
      accessToken: accessTokenAuthorized,
      refreshToken,
      user,
    };
  } catch (error) {
    throw new BadRequestException(
      `Ошибка входа ${user.email}: ${error.message}`,
    );
  }
}

public async getUserIfRefreshTokenMatches(
  refreshToken: string,
  userId: string,
): Promise<User> {
  const refreshTokenFromDb = await this.databaseService.token.findOne({
    where: { user: { id: userId } },
  });

```

```

    if (!refreshTokenFromDb)
        throw new NotFoundException(
            `Refresh token for user with id ${userId} does not exist`,
        );

    const { token } = refreshTokenFromDb;

    if (refreshToken !== token) {
        throw new ForbiddenException(`Access denied, userId ${userId}`);
    }

    const user = await this.databaseService.user.findOne({
        where: { id: userId },
    });
    return user;
}

public async getCurrentUser(req: RequestWithUser): Promise<User> {
    const user = await this.databaseService.user.findOne({
        where: { id: req.user.id },
    });

    return user;
}

public async refresh(req: RequestWithUser): Promise<AccessTokenResponse> {
    const payload = {
        email: req.user.email,
        id: req.user.id,
        role: req.user.role,
    };

    const accessToken = this.jwtService.sign(payload, {
        secret: this.configService.get<string>(
            ENV_VARIABLES_NAMES.ACCESS_TOKEN_SECRET,
        ),
        expiresIn: this.configService.get<string>(
            ENV_VARIABLES_NAMES.ACCESS_TOKEN_EXPIRATION,
        ),
    });

    const refreshToken = this.jwtService.sign(payload, {
        secret: this.configService.get<string>(
            ENV_VARIABLES_NAMES.REFRESH_TOKEN_SECRET,
        ),
        expiresIn: this.configService.get<string>(
            ENV_VARIABLES_NAMES.REFRESH_TOKEN_EXPIRATION,
        ),
    });
}

```

```

const stats = await this.databaseService.token.update(
  { user: { id: payload.id } },
  { token: refreshToken },
);

if (!stats.affected)
  throw new NotFoundException(
    `Not found session with user id ${payload.id} to update
it`,
  );

const refreshTokenCookie = cookie.serialize('refreshToken',
refreshToken, {
  httpOnly: this.configService.get<boolean>(
    ENV_VARIABLES_NAMES.COOKIE_HTTP_ONLY,
  ),
  path: this.configService.get<string>(ENV_VARIABLES_NAMES.COOKIE_PATH),
  maxAge: this.configService.get<number>(
    ENV_VARIABLES_NAMES.COOKIE_MAX_AGE,
  ),
});

req.res.setHeader('Set-Cookie', refreshTokenCookie);

return { accessToken };
}

```

Приложение II. Реализация проверки билетов

```

useEffect(() => {
  const config = {
    fps: 10,
    qrbox: { width: 300, height: 300 },
    aspectRatio: 1.0
  };

  const html5QrCode = new Html5Qrcode("qrCodeContainer");

  const qrScannerStop = () => {
    if (html5QrCode && html5QrCode.isScanning) {
      html5QrCode
        .stop()
        .then(() => console.log("Scanner stopped"))
        .catch((err) => console.error("Error stopping scanner:", err));
    }
  };

  const showNotification = (isValid, data) => {
    // Очищаем предыдущее уведомление, если оно есть
    if (lastNotificationId) {
      notifications.hide(lastNotificationId);
    }

    // Показываем новое уведомление
    const id = notifications.show({
      id: `scan-${Date.now()}`, // Уникальный ID для уведомления
      title: isValid ? 'Билет действителен' : 'Билет недействителен',
      message: isValid
        ? `Место: ${data.place.place}`
        : 'QR-код не найден в системе',
      color: isValid ? 'green' : 'red',
      icon: isValid ? <IconCheck size="1.1rem" /> : <IconX size="1.1rem" />,
      autoClose: 5000, // Автоматически закрывать через 3 секунды
    });

    setLastNotificationId(id);
  };

  const qrCodeSuccess = async (decodedText) => {
    setQRMessage(decodedText);
    try {
      const res = await store.CheckTicketCon(decodedText);
      const isValid = res.data.length > 0;

      setLastScanResult({

```

```

        isValid,
        message: isValid
          ? `Билет действителен. Место:
${res.data[0].place.place}`
          : "Билет недействителен",
        timestamp: new Date(),
        place: isValid ? res.data[0].place.place : null
      });

      showNotification(isValid, res.data[0]);

    } catch (error) {
      console.error('Error checking ticket:', error);
      // Очищаем предыдущее уведомление перед показом ошибки
      if (lastNotificationId) {
        notifications.hide(lastNotificationId);
      }
      const id = notifications.show({
        title: 'Ошибка',
        message: 'Не удалось проверить билет',
        color: 'red',
        autoClose: 3000,
      });
      setLastNotificationId(id);
    }
  };

  if (isEnabled) {
    html5QrCode.start(
      { facingMode: "environment" },
      config,
      qrCodeSuccess,
      (errorMessage) => {
        console.log(errorMessage);
      }
    );
    setQRMessage("");
  } else {
    qrScannerStop();
  }

  return () => {
    qrScannerStop();
    if (lastNotificationId) {
      notifications.hide(lastNotificationId);
    }
  };
}, [isEnabled, lastNotificationId]);

```

Приложение К. Главная страница приложения

Приложение Л. Таблица расчетов экономических показателей