МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационных систем и технологий
Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА

по дисциплине «Программирование мобильных систем»
Тема «Приложения для просмотра погоды»

Исполнитель
студент 3 курса 7 группы _____ С. Ю. Кунцов
подпись, дата

Руководитель
ассистент _____ Н. И. Уласевич
преподавателя
должность, ученая степень, ученое звание подпись, дата

Допущен(а) к защите _____ _____
_____ _____
_____ _____
дата, подпись

Курсовой проект защищен с оценкой _____

Руководитель _____ _____ Н. И. Уласевич
подпись дата инициалы и фамилия

Минск 2025

Содержание

Введение

В современном мире мобильные приложения стали неотъемлемой частью повседневной жизни, предоставляя пользователям удобный и мгновенный доступ к развлекательным сервисам. Одним из наиболее востребованных сервисов являются приложения для заказа билетов в кинотеатре, которые позволяют не только быстро приобретать билеты в любой момент, но и анализировать расписание сеансов и премьеры новых фильмов. Такие приложения стали незаменимыми инструментами для миллионов любителей кино: от случайных зрителей и киноманов до семей, планирующих совместный досуг, и компаний друзей.

Разработка мобильного приложения для заказа билетов в кинотеатре даёт пользователям возможность всегда быть в курсе текущих и предстоящих кинопремьер. Такое приложение удобно использовать в любом месте: дома, в транспорте, на работе или во время встречи с друзьями. Оно помогает планировать досуг, выбирать оптимальные места в зале и избегать очередей в кассах кинотеатров.

Целью данного курсового проекта является разработка мобильного приложения для заказа билетов в кинотеатре, которое обеспечит пользователей удобным и эффективным инструментом для организации киноразвлечений.

В рамках проекта будут детально исследованы ключевые аспекты разработки: от проектирования функциональных возможностей, таких как отображение расписания сеансов и схемы зала, до интеграции с внешними API кинотеатров (например, Киноход или Рамблер/Касса) для получения актуальной информации, обработку ошибок, валидацию данных и создание интуитивно понятного пользовательского интерфейса.

Курсовой проект должен быть завершен выводами по каждому разделу работы, обоснование выбора технологий и заключением включающий вывод по проделанной работе.

По завершении изучения и формулирования задач курсового проекта, окончательная цель проекта заключается в предоставлении пользователям удобного и функционального инструмента для заказа билетов в кинотеатре с помощью мобильного приложения.

1. Постановка задачи и анализ аналогичных решений

1.1 Постановка задачи

Целью курсового проекта является разработка программного средства, представляющего собой мобильное приложение, которое должно предоставлять возможности заказа билетов в кинотеатре. Функционально должны быть выполнены следующие задачи:

– возможность выбора, сортировки, поиска фильмов и сеансов;

– управление фильмами, сеансами, задами, бронированиями (добавление, удаление, изменение);

– анализ статистики (количество купленных билетов по периодам, популярные фильмы, общее количество бронирований);

– обеспечение авторизации через Google;

1.2 Актуальность задачи

Развитие мобильных технологий и изменение потребительских привычек создают объективную потребность в цифровизации процесса покупки билетов в кинотеатры. Традиционный способ приобретения билетов через кассы кинотеатров имеет ряд ограничений, включая необходимость физического присутствия, ограниченные часы работы касс и возможные очереди в популярные дни.

Современные пользователи ценят возможность планирования досуга заранее и предпочитают решать бытовые задачи с помощью мобильных устройств. Мобильное приложение для заказа билетов решает практические проблемы пользователей, предоставляя удобный инструмент для выбора фильма, времени сеанса и места в зале без необходимости посещения кинотеатра до начала просмотра.

Основными преимуществами мобильного решения являются экономия времени пользователей, возможность заблаговременного планирования походов в кино и снижение нагрузки на кассы кинотеатров. Для владельцев кинотеатров такие приложения представляют дополнительный канал продаж и инструмент для сбора данных о предпочтениях аудитории;

1.3 Обзор аналогичных решений

В современном мире приложения для заказа билетов в кинотеатре являются достаточно популярными веб–сервисами. Их основная задача – это приобретение билетов на сеансы в нужном кинотеатре. При разработке концепции для своего приложения я обратил внимание на самые успешные примеры популярнейших приложений для бронирования билетов. Аналоги рассмотрены ниже.

1.3.1 Киноафиша

Киноафиша – это мобильное приложение для поиска фильмов и оформления билетов для разных городов России и Беларуси. Предлагает широкий спектр функций для организации киноразвлечений, включая подробные описания

фильмов, информацию о сеансах в реальном времени, и возможность выбора конкретных мест в зале. Особенности этого приложения охватывают наличие подробных расписаний на несколько дней вперед.

Интерфейс приложения показан на рисунке 1.1.

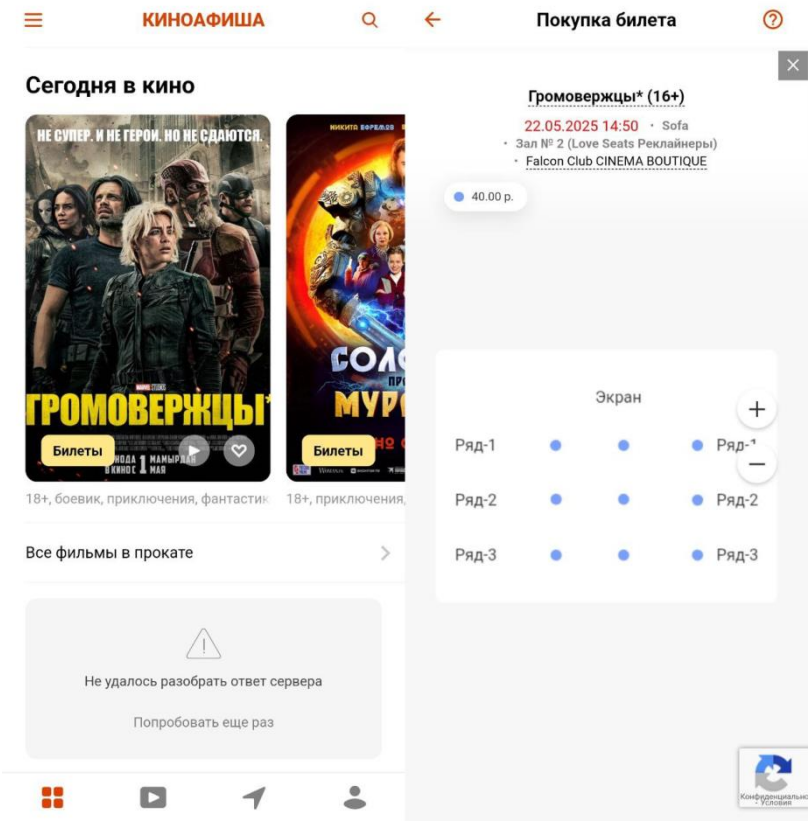


Рисунок 1.1 – Интерфейс приложения «Киноафиша»

Технологически приложение выделяется своей способностью к интеграции с различными кинотеатральными сетями и обновлением информации о доступности мест в реальном времени. Однако критическим недостатком является зависимость от сторонних веб-сервисов при выборе мест в зале. При попытке забронировать билет пользователь перенаправляется на веб-страницу официального сайта кинотеатра, что нарушает целостность пользовательского опыта и создает дополнительные точки отказа системы. Также пользователи могут столкнуться с недостатками, такими как наличие рекламы в бесплатной версии и нестабильная работа при слабом интернет-соединении.

1.3.2 Relax.by

Мобильное приложение «Relax.by» представляет собой платформу для бронирования развлекательных мероприятий в Беларуси, включая заказ билетов в кинотеатры. Приложение предлагает обширную базу данных киносеансов, подробную информацию о фильмах с трейлерами и рецензиями, а также возможность сравнения цен в различных кинотеатрах города. Особенности этого приложения включают интеграцию с системами лояльности кинотеатров,

накопление бонусных баллов за покупки и уникальную функцию групповых заказов, позволяющую организовывать коллективные походы в кино.

Интерфейс приложения показан на рисунке 1.2.

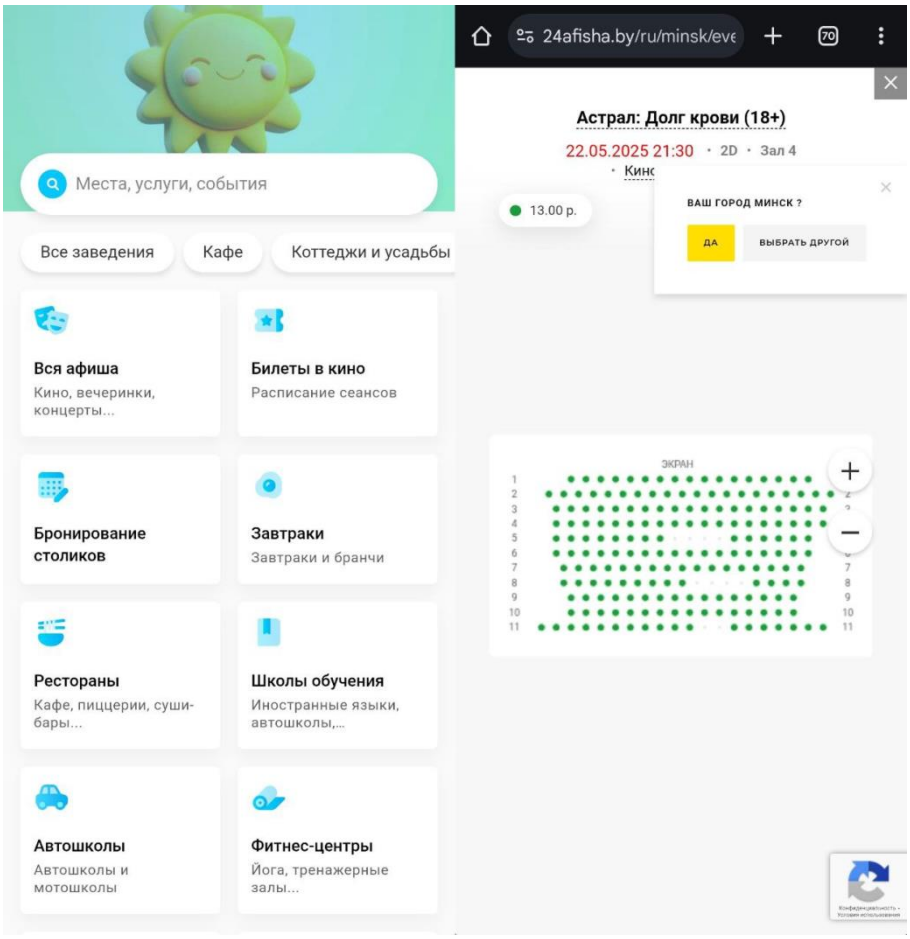


Рисунок 1.2 – Интерфейс приложения «Relax.by»

Архитектурно приложение страдает от аналогичной проблемы зависимости от внешних систем бронирования. При выборе конкретных мест в зале приложение открывает встроенный браузер с веб-страницей партнерского кинотеатра, что создает разрыв в пользовательском интерфейсе и снижает общее качество взаимодействия. Среди других недостатков можно выделить относительно медленную загрузку контента при большом количестве доступных сеансов и периодические проблемы с синхронизацией данных между мобильным приложением и веб-сервисами кинотеатров. Кроме того, некоторые пользователи отмечают сложность навигации при работе с большим количеством фильтров и опций поиска.

Общим существенным недостатком обеих рассмотренных платформ является их критическая зависимость от сторонних веб-сервисов на этапе финального бронирования билетов. Это создает потенциальные проблемы с производительностью, безопасностью данных и общим пользовательским опытом, поскольку стабильность работы приложения напрямую зависит от надежности внешних систем, находящихся вне контроля разработчиков

1.4 Вывод

В ходе анализа прототипов и литературных источников выявлены основные задачи и функциональные требования проекта. Необходимо разработать мобильное приложение для заказа билетов в кинотеатре, предоставляющее пользователям следующие функции:

– простая и быстрая регистрация и авторизация;

– интуитивно понятный и ненагруженный интерфейс;

– просмотр расписания киносеансов и бронирование билетов прямо в приложении.

Разработка данного приложения направлена на создание удобного сервиса для заказа билетов в кинотеатре, который позволит пользователям эффективно планировать свой киноразвлекательный досуг.

2. Проектирование приложения

Проектирование является критическим этапом в процессе разработки программного обеспечения, которым часто пренебрегают неопытные разработчики. Это чревато отсутствием четкого плана действий, что может привести к тому, что проект будет отложен на неопределенное время.

2.1 Диаграмма вариантов использования

Обычно при проектировании разработчики представляют систему в графическом виде, так как этот формат легко воспринимается человеком. Именно поэтому вместо того, чтобы писать объемные тексты о каждой функции будущей программы, разработчики создают различные диаграммы для описания своих систем.

Диаграмма вариантов использования (use-case diagram) – это диаграмма, которая описывает, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

В ходе разработки программного средства будут реализованы три пользователея: пользователь, администратор и гость.

Диаграмма вариантов использования представлена в приложении А.

2.2 Проектирование базы данных

Модель данных – это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь.

2.2.1 Проектирование базы данных сервиса Пользователей

Для реализации сервиса пользователей в рамках данного курсового проекта была выбрана реляционная система управления базами данных PostgreSQL.

PostgreSQL предоставляет развитые механизмы безопасности, что критически важно для хранения персональных данных пользователей, включая электронные адреса, пароли и личную информацию.

Логическая модель базы данных представлена на рисунке 2.1.

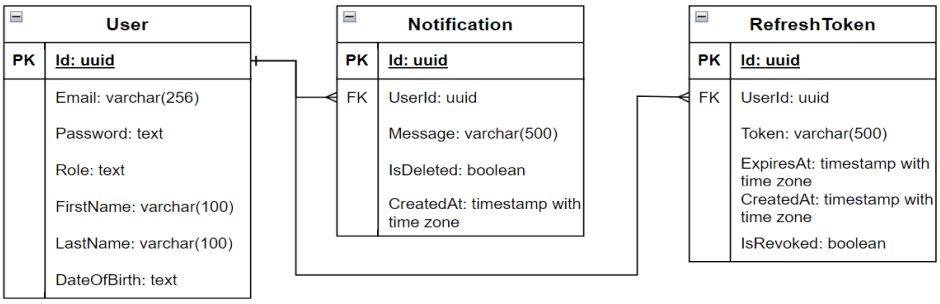


Рисунок 2.1 – Модель БД сервиса Пользователей

Таблица «User» будет содержать информацию о пользователях. Структура представлена в таблице 2.1.

Таблица 2.1 – Описание таблицы User

Название	Тип данных	Описание
Id	uuid	Идентификатор пользователя
Email	varchar(256)	Электронная почта пользователя
Password	text	Захэшированный пароль пользователя
Role	varchar	Роль пользователя
FirstName	varchar(100)	Имя пользователя
LastName	varchar(100)	Фамилия пользователя
DateOfBirth	text	Дата рождения пользователя

Таблица «Notification» отвечает за хранение уведомлений для пользователей. Перечень и описание ее столбцов приведены в таблице 2.2.

Таблица 2.2 – Описание таблицы Notification

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор уведомления
UserId	uuid	Внешний ключ, ссылающийся на пользователя
Message	varchar(500)	Текст уведомления
IsDeleted	boolean	Флаг удаления уведомления
CreatedAt	timestamp with time zone	Дата и время создания уведомления

Таблица «RefreshToken» предназначена для хранения токенов обновления для авторизации пользователей. Структура представлена в таблице 2.3.

Таблица 2.3 – Описание таблицы RefreshToken

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор токена
UserId	uuid	Внешний ключ, ссылающийся на пользователя
Token	varchar(500)	Строка токена обновления
ExpiresAt	timestamp with time zone	Дата и время истечения токена
CreatedAt	timestamp with time zone	Дата и время создания токена
IsRevoked	boolean	Флаг отзыва токена

2.2.2 Проектирование базы данных сервиса Фильмов

Для реализации сервиса фильмов в рамках данного курсового проекта также была выбрана реляционная система управления базами данных PostgreSQL. Выбор PostgreSQL для данного сервиса обусловлен теми же техническими преимуществами, что и для сервиса пользователей, включая высокую производительность при работе с большими объемами данных о фильмах, расписании сеансов и информации о кинотеатрах. Данная СУБД обеспечивает

эффективную обработку сложных запросов для поиска и фильтрации фильмов по различным критериям.

Логическая модель базы данных представлена на рисунке 2.2.

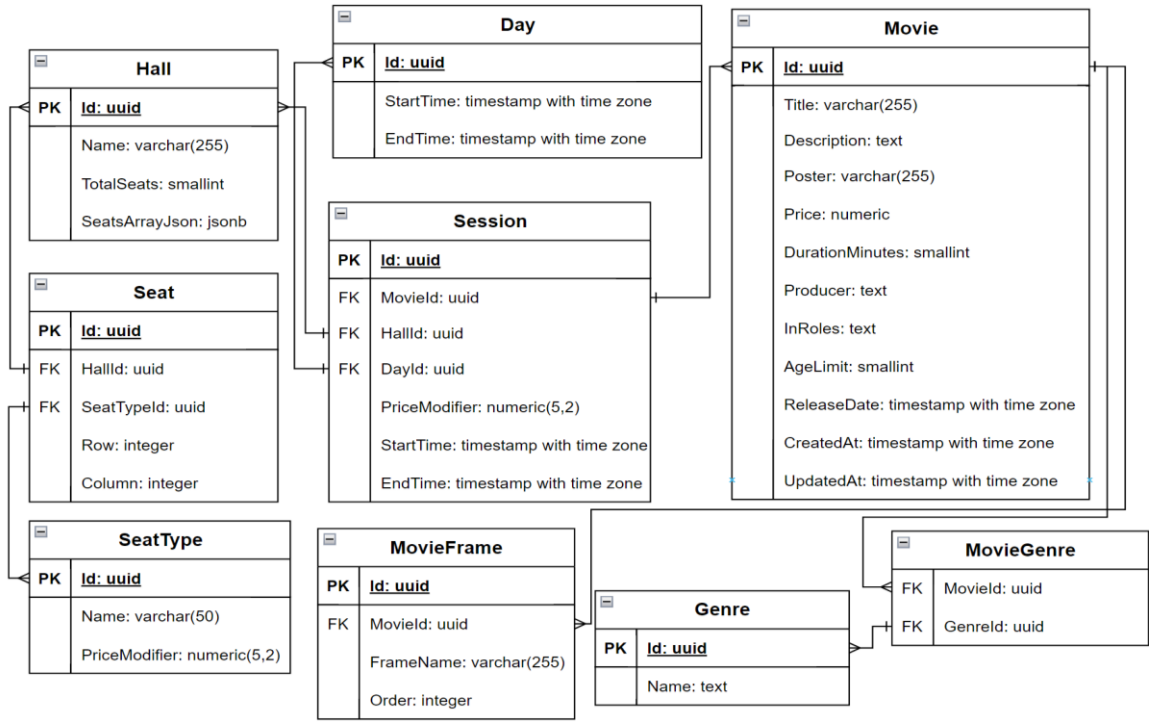


Рисунок 2.2 – Модель БД сервиса Фильмов

Таблица «Movie» будет содержать информацию о фильмах в системе. Структура представлена в таблице 2.4.

Таблица 2.4 – Описание таблицы Movie

Название	Тип данных	Описание
Id	uuid	Идентификатор фильма
Title	varchar(255)	Название фильма
Description	text	Описание сюжета фильма
Poster	varchar(255)	Ссылка на постер фильма
Price	numeric	Базовая цена билета на фильм
DurationMinutes	smallint	Продолжительность фильма в минутах
Producer	text	Информация о продюсере фильма
InRoles	text	Информация об актерском составе
AgeLimit	smallint	Возрастное ограничение для просмотра
ReleaseDate	timestamp with time zone	Дата выхода фильма в прокат
CreatedAt	timestamp with time zone	Дата добавления фильма в систему
UpdatedAt	timestamp with time zone	Дата последнего обновления информации

Таблица «Hall» отвечает за хранение информации о кинозалах. Перечень и описание ее столбцов приведены в таблице 2.5.

Таблица 2.5 – Описание таблицы Hall

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор зала
Name	varchar(255)	Название кинозала
TotalSeats	varchar(500)	Текст уведомления
IsDeleted	smallint	Общее количество мест в зале
SeatsArrayJson	jsonb	JSON-структура с расположением мест

Таблица «Day» предназначена для хранения информации о днях показов. Структура представлена в таблице 2.6.

Таблица 2.6 – Описание таблицы Day

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор дня
StartTime	timestamp with time zone	Время начала рабочего дня кинотеатра
EndTime	timestamp with time zone	Время окончания рабочего дня кинотеатра

Таблица «Session» содержит информацию о киносеансах. Структура представлена в таблице 2.7.

Таблица 2.7 – Описание таблицы Session

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор дня
MovieId	uuid	Внешний ключ, ссылающийся на фильм
HallId	uuid	Внешний ключ, ссылающийся на зал
DayId	uuid	Внешний ключ, ссылающийся на день
PriceModifier	numeric(5,2)	Коэффициент изменения цены для сеанса
StartTime	timestamp with time zone	Время начала рабочего дня кинотеатра
EndTime	timestamp with time zone	Время окончания рабочего дня кинотеатра

Таблица «Seat» отвечает за хранение информации о местах в залах. Перечень и описание ее столбцов приведены в таблице 2.8.

Таблица 2.8 – Описание таблицы Seat

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор места
HallId	uuid	Внешний ключ, ссылающийся на зал
SeatTypeId	uuid	Внешний ключ, ссылающийся на тип места
Row	integer	Номер ряда
Column	integer	Номер места в ряду

Таблица «SeatType» предназначена для хранения типов мест в кинозале. Структура представлена в таблице 2.9.

Таблица 2.9 – Описание таблицы SeatType

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор типа места
Name	varchar(50)	Название типа места
PriceModifier	numeric(5,2)	Коэффициент изменения цены для типа места

Таблица «Genre» содержит информацию о жанрах фильмов. Структура представлена в таблице 2.10.

Таблица 2.10 – Описание таблицы Genre

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор жанра
Name	text	Название жанра

Таблица «MovieGenre» реализует связь многие-ко-многим между фильмами и жанрами. Структура представлена в таблице 2.11.

Таблица 2.11 – Описание таблицы MovieGenre

Название	Тип данных	Описание
MovieId	uuid	Внешний ключ, ссылающийся на фильм
GenreId	uuid	Внешний ключ, ссылающийся на жанр

Таблица «MovieFrame» отвечает за хранение информации о моментах из фильма. Перечень и описание ее столбцов приведены в таблице 2.12.

Таблица 2.9 – Описание таблицы MovieFrame

Название	Тип данных	Описание
Id	uuid	Уникальный идентификатор момента из фильма
MovieId	varchar(50)	Внешний ключ, ссылающийся на фильм
FrameName	varchar(255)	Название формата показа для получения из внешних сервисов
Order	integer	Порядок отображения изображения

2.2.3 Проектирование базы данных сервиса Бронирований

Для реализации сервиса бронирований в рамках данного курсового проекта была выбрана документо-ориентированная система управления базами данных MongoDB. Выбор MongoDB обусловлен спецификой задач сервиса бронирований, который должен эффективно работать с комплексными структурами данных, включающими информацию о местах в зале, их статусах и взаимосвязях. MongoDB предоставляет естественную возможность хранения JSON-подобных документов, что идеально подходит для представления сложных структур мест в кинозалах и информации о бронированиях.

Логическая модель базы данных представлена на рисунке 2.3.

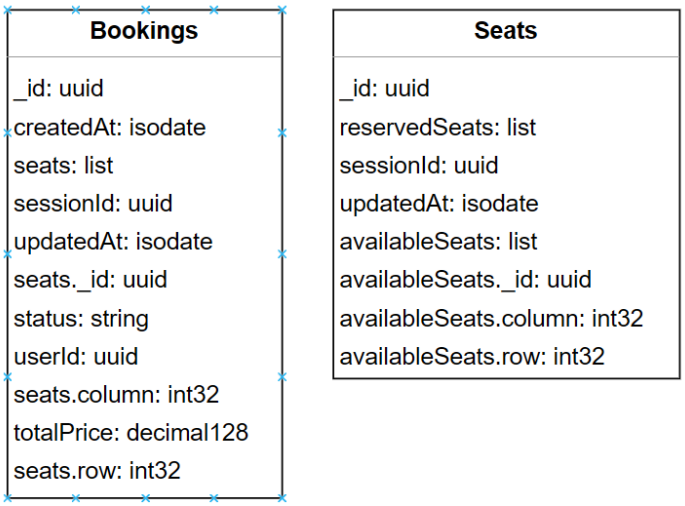


Рисунок 2.3 – Модель БД сервиса Бронирований

Коллекция «Bookings» будет содержать информацию о бронированиях пользователей. Структура представлена в таблице 2.13.

Таблица 2.13 – Описание коллекции Bookings

Название	Тип данных	Описание
id	ObjectId	Уникальный идентификатор бронирования
createdAt	ISODate	Дата и время создания бронирования
seats	Array	Массив забронированных мест
seats._id	ObjectId	Уникальный идентификатор места
seats.column	Int32	Номер места в ряду
seats.row	Int32	Номер ряда
sessionId	ObjectId	Идентификатор сеанса
updatedAt	ISODate	Дата и время последнего обновления
status	String	Статус бронирования
userId	ObjectId	Идентификатор пользователя
totalPrice	Decimal128	Общая стоимость бронирования

Коллекция «Seats» отвечает за хранение информации о состоянии мест для каждого сеанса. Перечень и описание ее полей приведены в таблице 2.14.

Таблица 2.13 – Описание коллекции Seats

Название	Тип данных	Описание
id	ObjectId	Уникальный идентификатор документа мест
reservedSeats	Array	Массив зарезервированных мест
sessionId	ObjectId	Идентификатор сеанса
updatedAt	ISODate	Дата и время последнего обновления
availableSeats	Array	Массив доступных мест
availableSeats._id	ObjectId	Уникальный идентификатор доступного места
availableSeats.column	Int32	Номер доступного места в ряду
availableSeats.row	Int32	Номер ряда доступного места

2.3 Проектирование API

Для эффективной синхронизации работы локальной и глобальной БД на Android-клиенте было выбрано спроектировать серверное приложение. Оно должно обладать WEB API, который обеспечивает доступ к каждой сущности базы данных. Чтобы обмен данных между клиентами и сервером был успешным, необходимо передавать информацию в формате JSON.

Архитектуру приложения можно представить на рисунке 2.4 в виде диаграммы компонентов.

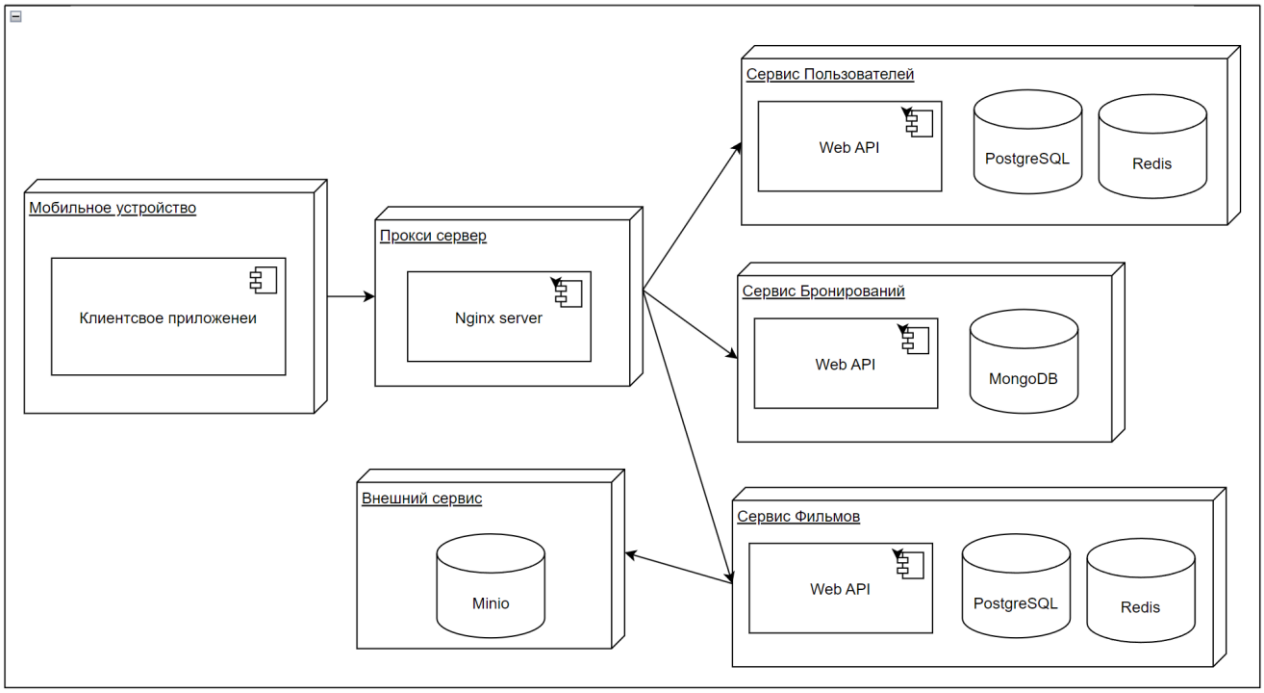


Рисунок 2.4 – Диаграмма компонентов

Для обеспечения гибкости и масштабируемости а также устойчивости сервер был разбит на 3 микросервиса:

– Сервис Пользователей;

– Сервис Фильмов;

– Сервис Бронирований;

Между собой сервисы общаются посредством очередей RabbitMQ или gRPC.

При обращении к серверу с заданным маршрутом и описанным HTTP методом, сервер определяет, с какой сущностью базы данных необходимо взаимодействовать и какие действия должны быть выполнены.

Для обеспечения высокой производительности и снижения нагрузки на базы данных используется Redis – высокоскоростное хранилище ключ-значение, выполняющее следующие функции.

Мы жестко разделяем уровни ответственности нашего приложения на 5 слоев, чтобы добиться расширяемости, лучшей тестируемости и гибкости приложения:

– Уровень доступа к данным;

– Бизнес логика приложения;

– Доменная область приложения;

– Уровень доступа к внешним сервисам;

– Уровень представления. В нашем случае это URL, по которому будут обращаться Android и Web-клиенты, чтобы взаимодействовать с данными.

2.4 Проектирование мобильного приложения

Клиентская часть приложения не менее важна, чем серверная. Это то, с чем будет контактировать пользователь, то, от чего в первую очередь зависят хорошие или плохие впечатления при использовании приложения. Здесь стоит понимать, что исходя из специфики проекта, в первую очередь разрабатывается мобильное приложение и самые строгие требования со стороны пользователя будут предъявляться именно к нему.

Навигация – очень важный для мобильного приложения элемент, при её проектировании необходимо поддерживать логическую связь между экранами.

При первом запуске приложения пользователь попадает на Activity авторизации, с которой он может попасть в Activity регистрации. После успешной авторизации/регистрации пользователь попадет в Activity списка.

В Activity подробностей фильма пользователь сможет увидеть всю информацию о выбранном фильме, а также выбрать дату и подходящий сеанс по времени и типу зала и далее перейти в Activity выбор мест.

В Activity выбора мест пользователь может выбрать список мест по типам и рассадке и далее создать бронирование этих мест.

В Activity профиля пользователь может просмотреть свои персональные данные и поменять их, выйти или удалить аккаунт.

Activity бронирований позволяет увидеть всю историю бронирований, их статус, если статус бронирования активен, то может либо отменить либо оплатить бронирование.

Общая схема взаимодействия между страницами Android-клиента представлена на рисунке 2.5.

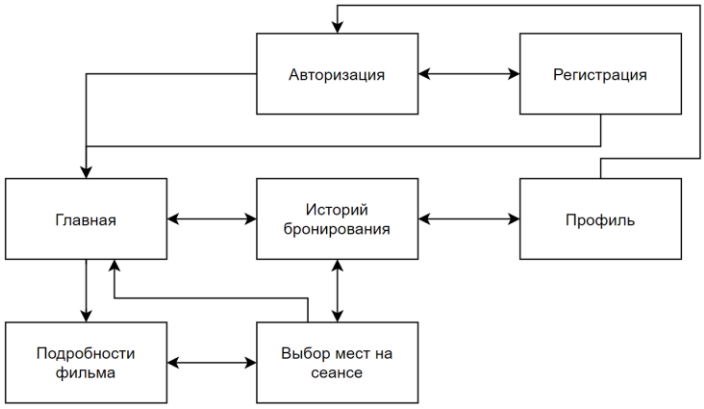


Рисунок 2.5 – Схема навигации для пользователя

Таким образом, клиентская часть приложения играет ключевую роль в формировании пользовательского опыта, особенно в мобильных приложениях. Для

обеспечения удобства и логичности взаимодействия пользователя с приложением, важно тщательно продумать навигацию и разбить логику на независимые части, что позволит последовательно и эффективно реализовывать каждую из них.

2.5 Вывод

В данной главе было рассмотрено проектирование каждого отдельного компонента приложения. По части мобильного приложения был определен основной функционал приложения, также были разработаны диаграмма вариантов использования приложения. Была также спроектирована база данных, где были определены необходимые таблицы, охарактеризованы поля каждой из них. Также были установлены связи между таблицами, заключающиеся в создании первичных и внешних ключей. При проектировании серверного приложения был выбран архитектурный стиль взаимодействия между клиентом и сервером – клиент-серверное взаимодействие, а также сервер был разбит на микросервисы. Это означает, что сервер предоставляет определенные услуги и ресурсы, которые клиент может запросить и использовать.

3. Программная реализация приложения

3.1 Технические средства для разработки

Для разработки глобальной базы данных были использованы следующие средства:

– PostgreSQL – система управления реляционными БД для хранения структурированных данных пользователей и фильмов;

– MongoDB – документо-ориентированная система управления базами данных для сервиса бронирований, обеспечивающая гибкое хранение JSON-структур с информацией о местах и их статусах;

– MinIO – высокопроизводительное объектное хранилище, используемое для хранения изображений (постеры фильмов, схемы залов, афиши).

Средства для разработки Web API:

– Rider – среда для разработки;

– .Net – это среда выполнения C# от Microsoft.

– ASP.NET – это фреймворк для создания веб-приложений с использованием C#.

Средства для разработки Android-клиента:

– Android Studio – среда для разработки Android приложений, с хорошим статическим анализатором кода;

– Flutter – это UI фреймворк, разработанный компанией Google, который позволяет создавать кроссплатформенные мобильные приложения.

– Dart – это язык программирования, разработанный компанией Google, который используется для создания мобильных, веб- и настольных приложений.

Инфраструктурные средства:

– Nginx – высокопроизводительный HTTP-сервер и обратный прокси-сервер, используемый для балансировки нагрузки между микросервисами и обеспечения безопасности API;

– RabbitMQ – брокер сообщений, обеспечивающий асинхронное взаимодействие между микросервисами и надежную доставку уведомлений пользователям о статусе бронирований.

3.2 Разработка баз данных

Для хранения данных будет использоваться PostgreSQL и MongoDB.

Основные обращения к базе данных будут осуществляется через ORM Entity Framework Core, что предоставит удобный интерфейс для взаимодействия с БД

3.3 Разработка API

Для данного курсового проекта был разработан сервер, который служит важной основой для функционирования системы. Серверная часть обеспечивает управление данными, взаимодействие между клиентскими приложениями и внешними сервисами, а также выполнение бизнес-логики. В этом проекте сервер

был разработан в среде разработки Rider с использованием технологии .Net и фреймворка ASP.NET.

Платформа .NET (C#) была выбрана для разработки сервера благодаря высокой производительности, многопоточности и асинхронной модели, что позволяет эффективно обрабатывать множество запросов, а также за счёт богатой экосистемы с NuGet-пакетами и мощными фреймворками вроде ASP.NET Core, которые ускоряют разработку и упрощают интеграцию сложных функций.

Фреймворк ASP.NET Core был выбран для разработки серверного приложения благодаря его мощным встроенным возможностям для обработки HTTP-запросов, гибкой маршрутизации и удобной системе промежуточного ПО .

Основная задача сервера заключается в обеспечении безопасного и эффективного управления данными. Например, сервер обрабатывает запросы на создание бронирования на определенные места для сеанса, для этого сервер проверяет не существует ли в бд такой записи, если есть то с каким статусом. Если он может создать новую запись, то он помещает бронирование в брокер сообщений.

Важной частью функциональности сервера является управление пользователями. Сервер позволяет пользователям регистрироваться, входить в систему, а также управлять своим аккаунтом.

Пример реализации одного из маршрута для авторизации представлен на листинге 3.1

```
[HttpPost("/users/login")]
[SwaggerRequestExample(typeof(CreateLoginRequest),
typeof(CreateLoginRequestExample))]
public async Task<IActionResult> Login(
    [FromBody] CreateLoginRequest request,
    CancellationToken cancellationToken)
{
    var existUser = await mediator.Send(
   new LoginQuery(request.Email,request.Password),cancellationToken);

    var authResultDto = await mediator.Send(
       new GenerateTokensCommand(existUser.Id, existUser.Role),
       cancellationToken);

    HttpContext.Response.Cookies.Append(
       JwtConstants.REFRESH_COOKIE_NAME,
       authResultDto.RefreshToken);

    return Ok(new{
          accessToken = authResultDto.AccessToken,
          refreshToken = authResultDto.RefreshToken});
}
```

Листинг 3.1 – Обработка входа пользователя

Пример реализации маршрута для создания бронирования представлен на листинге 3.2

```
[HttpPost("/users/login")]
[SwaggerRequestExample(typeof(CreateLoginRequest),
typeof(CreateLoginRequestExample))]
public async Task<IActionResult> Login(
    [FromBody] CreateLoginRequest request,
    CancellationToken cancellationToken)
{
    var existUser = await mediator.Send(
       new LoginQuery(request.Email, request.Password),
       cancellationToken);

    var authResultDto = await mediator.Send(
       new GenerateTokensCommand(existUser.Id, existUser.Role),
       cancellationToken);

    HttpContext.Response.Cookies.Append(
       JwtConstants.REFRESH_COOKIE_NAME,
       authResultDto.RefreshToken);

    return Ok(new {
          accessToken = authResultDto.AccessToken,
          refreshToken = authResultDto.RefreshToken }); }
```

Листинг 3.2 – Обработка создания бронирования

Структура архитектуры сервиса Пользователей представлена на рисунке 3.1

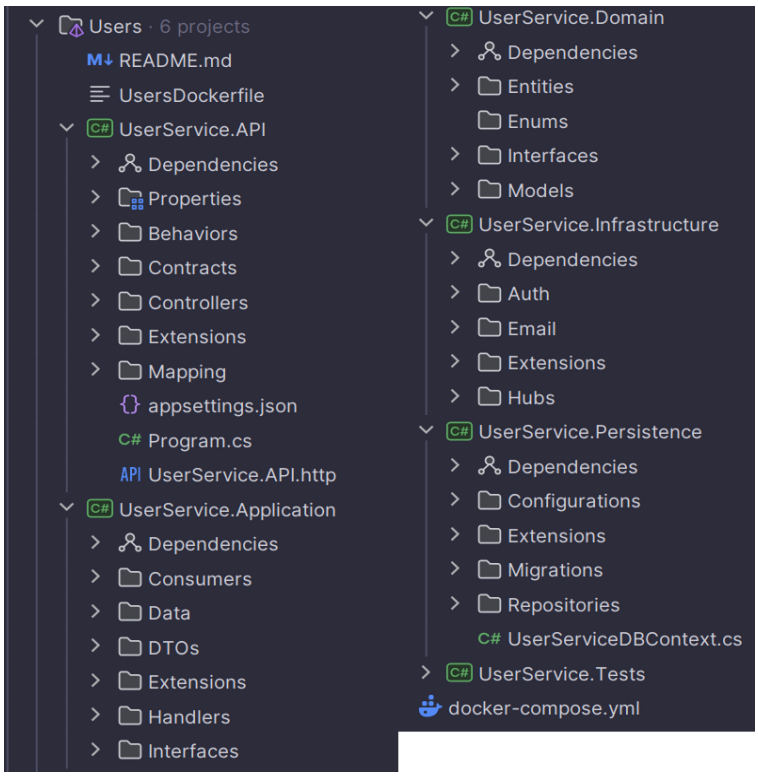


Рисунок 3.1 – Структура архитектуры сервиса Пользователей

В данном курсовом проекте был разработан сервер на платформе .NET с использованием фреймворка ASP.NET Core, что обеспечивает высокую

производительность и эффективное управление данными. Сервер выполняет ключевые функции, такие как обработка запросов на получение фильмов, создании бронирования со сложной логикой и использовании очереди и управление пользователями, включая регистрацию и авторизацию.

3.4Разработка мобильного приложения

– Мобильное приложение построено с использованием архитектурного паттерна Clean Architecture, который обеспечивает четкое разделение ответственности между различными слоями приложения и создает высокомасштабируемую, тестируемую и поддерживаемую кодовую базу. Данная архитектура основана на принципах SOLID и способствует созданию слабосвязанной системы, где каждый компонент имеет четко определенную область ответственности.

Архитектурное решение обеспечивает инверсию зависимостей, при которой высокоуровневые модули не зависят от низкоуровневых модулей, а оба типа модулей зависят от абстракций. Это позволяет легко заменять реализации отдельных компонентов без влияния на остальную систему и значительно упрощает процесс модульного тестирования каждого слоя приложения.

– Core содержит базовые компоненты приложения, включая конфигурацию, константы, обработку ошибок, сетевые утилиты и общие сервисы. В папке network реализованы HTTP-клиенты для взаимодействия с различными микросервисами, а services включает вспомогательные сервисы для работы с уведомлениями и кэшированием данных.

– Data отвечает за слой данных и включает три основные подпапки: datasources для взаимодействия с внешними источниками данных, models с моделями данных для сериализации/десериализации JSON, и repositories с реализацией репозиториев для управления данными из различных источников.

– Domain представляет бизнес-логику приложения и содержит entities с основными сущностями предметной области, repositories с абстракциями репозиториев и usecases с вариантами использования, которые инкапсулируют бизнес-правила приложения.

– Presentation управляет пользовательским интерфейсом и состоит из трех подпапок: providers с классами для управления состоянием приложения, screens содержащими основные экраны приложения, и widgets с переиспользуемыми UI-компонентами.

– DI (Dependency Injection) содержит конфигурацию контейнера зависимостей, который обеспечивает слабую связанность между компонентами и упрощает тестирование приложения.

– Config включает общие настройки приложения, такие как темы оформления и цветовые схемы, обеспечивая единообразный внешний вид интерфейса.

Реализация асихронной функции, которая выполняет запрос к серверу на авторизацию пользователя, представлена в листинге 3.2.

```
@override
Future<TokenModel> login({
  required String email,
  required String password,
}) async {
  try {
    final response = await client.post(
      ApiConstants.login,
      data: {'email': email, 'password': password},
    );

    final tokenData = {
      'access_token': response['accessToken'],
      'refresh_token': response['refreshToken'],
    };

    final TokenModel tokenModel = TokenModel.fromJson(tokenData);

    _saveTokens(tokenModel);

    return tokenModel;
  } catch (e) {
    throw AuthException(e.toString());
  }
}
```

Листинг 3.2 – Установка основного города пользователя

Пример реализации функции функции создания бронирования представлен в листинге, который находится в приложении Б.

Функция отправляет HTTP POST-запрос к серверу, передавая идентификаторы пользователя, сессии и список мест.

Если сервер успешно выполняет удаление (ответ с кодом состояния 200), функция создает новое бронирование, переводя статус этих мест в статут «Reserved» .

В случае ошибки функция показывает уведомление с сообщением об ошибке.

3.5 Вывод

В данной главе подробно рассмотрен процесс разработки всех компонентов приложения. Вначале было проведено исследование и отбор необходимых технологий для реализации курсового проекта. После выбора технологий приступили к разработке структуры базы данных, определив ключевые сервисы, таблицы и связи между ними для обеспечения целостности и эффективности работы с данными. Были разработаны запросы, необходимые для взаимодействия с базой данных, включая запросы на получение, добавление, обновление и удаление данных.

Далее было разработано серверное приложение, которое играет ключевую роль в обработке данных и взаимодействии между клиентской частью и базой данных. В этой части работы была представлена графическая структура серверного приложения, описаны все используемые классы и методы, а также их функциональность. Обсуждение включало описание логики обработки запросов, маршрутизации, аутентификации пользователей и других важных аспектов работы сервера.

После завершения серверной части проекта, было преступлено к разработке мобильного приложения. В этом разделе также была представлена графическая структура приложения, рассмотрены все классы и методы, их роль и взаимодействие. Было уделено внимание интерфейсу пользователя, взаимодействию с сервером, обработке данных и обеспечению удобства использования приложения. Подробно рассматривались компоненты приложения, такие как модели данных, сервисы, утилиты и представления, что позволило получить полное представление о работе мобильного приложения.

Затем перешли к разработке веб-приложения. Здесь также была представлена его графическая структура и основные файлы. Обсуждение включало описание архитектуры веб-приложения, взаимодействие с сервером и базой данных, а также реализацию пользовательского интерфейса. Были рассмотрены классы и методы, используемые для обеспечения функциональности веб-приложения, включая обработку пользовательских запросов, отображение данных и другие важные аспекты.

Таким образом, в данной главе детально описан весь процесс разработки приложения, начиная с выбора технологий и заканчивая реализацией всех компонентов – от базы данных до мобильного и веб-приложения.

4 Анализ информационной безопасности приложения

При разработке мобильного приложения ключевым аспектом является обеспечение информационной безопасности. Приложение должно защищать данные пользователей от несанкционированного доступа и взлома. Необходимо реализовать меры защиты данных пользователей, включая использование пароля для доступа к приложению и ограничение прав доступа. Важно точно определить и ограничить возможности каждой роли в приложении.

4.1 Защита API

Одним из критически важных аспектов обеспечения информационной безопасности мобильного приложения является защита программного интерфейса приложения (API) от несанкционированного доступа. Современные веб-приложения и мобильные клиенты активно используют RESTful API для обмена данными с сервером, что создает потенциальные точки уязвимости, требующие комплексного подхода к защите.

В разработанном приложении для заказа билетов в кинотеатре реализована система аутентификации и авторизации на основе JSON Web Token (JWT), которая обеспечивает безопасный обмен информацией между клиентом и сервером. JWT представляет собой компактный, самодостаточный способ безопасной передачи информации между сторонами в виде JSON-объекта, который может быть проверен и которому можно доверять благодаря цифровой подписи.

Архитектура безопасности API построена на принципе двухуровневой системы токенов, включающей Access Token и Refresh Token. Access Token представляет собой краткосрочный JWT-токен с ограниченным временем жизни (обычно 15-30 минут), который содержит информацию о пользователе, его ролях и правах доступа. Данный токен передается с каждым API-запросом в заголовке Authorization и используется сервером для валидации прав пользователя на выполнение конкретных операций.

Refresh Token является долгосрочным токеном (срок действия 7-30 дней), который хранится в защищенном хранилище на клиентском устройстве и используется исключительно для получения новых Access Token'ов при истечении их срока действия. Такой подход минимизирует риски компрометации системы, поскольку даже в случае перехвата Access Token злоумышленник получает доступ лишь на ограниченное время, после чего токен автоматически становится недействительным.

Процесс аутентификации включает несколько этапов валидации. При получении запроса сервер извлекает JWT-токен из заголовка Authorization, проверяет его цифровую подпись с использованием секретного ключа, валидирует срок действия токена и извлекает информацию о пользователе для последующей авторизации. В случае обнаружения недействительного или истекшего токена сервер возвращает HTTP-статус 401 (Unauthorized), что инициирует процедуру обновления токена на стороне клиента.

Система ролевого доступа (Role-Based Access Control, RBAC) интегрирована в структуру JWT-токенов, где каждый токен содержит информацию о роли

пользователя (User или Admin). Это позволяет серверу принимать решения об авторизации на уровне отдельных API-эндпоинтов, ограничивая доступ к административным функциям управления фильмами, залами и сеансами только для пользователей с соответствующими правами.

Код для валидации токена, продемонстрирован на листинге 4.1.

```
public class ActiveAdminHandler :
AuthorizationHandler<ActiveAdminRequirement>
{
    protected override Task HandleRequirementAsync(
       AuthorizationHandlerContext context,
       ActiveAdminRequirement requirement)
    {
       var userIdClaim =
context.User.FindFirst(ClaimTypes.NameIdentifier);
       var userRole = context.User.FindFirst(ClaimTypes.Role);

       if (userIdClaim == null || userRole == null)
       {
          context.Fail();

          return Task.CompletedTask;
       }

       if (!context.User.IsInRole(Role.Admin.GetDescription()) &&
          !context.User.IsInRole(Role.User.GetDescription()))
       {
          context.Fail();
          return Task.CompletedTask;
       }

       context.Succeed(requirement);
       return Task.CompletedTask;
    }
}

public class ActiveAdminRequirement : IAuthorizationRequirement;
```

4.2 Защита пользовательских данных

Самым нуждающимся в защите параметром системы является пароль. Пароль препятствует получению данных пользователя другими пользователями системы. Поэтому с помощью технологий хеширование был создан класс на стороне сервера да бы защитить пароль от его раскрытия.

Hash-функция работает только в одну сторону, что обеспечивает безопасность пароля, так как восстановить исходный пароль по hash невозможно. При вводе пароля система преобразует его в hash и сравнивает с сохраненным в базе данных. Если значения совпадают, доступ предоставляется. Хранение паролей в виде хешей добавляет дополнительный уровень защиты. Даже если

злоумышленник получит доступ к базе данных, он увидит лишь непонятные строки хешей, а не сами пароли.

Код, реализующий хеширование пароля, продемонстрирован на листинге 4.2.

```
public class PasswordHasher: IPasswordHasher
{
    public string Generate(string password)
    {
        return BCrypt.Net.BCrypt.EnhancedHashPassword(password);
    }

    public bool Verify(string password, string hashPassword)
    {
        return BCrypt.Net.BCrypt.EnhancedVerify(password,hashPassword);
    }
}
```

Листинг 4.2 – Код, реализующий хеширование верифицирование пароля

Хеширование с помощью BCrypt позволяет надёжно защитить пароли благодаря использованию соли и адаптивности алгоритма

4.2 Вывод

В данном разделе был проведен анализ информационной безопасности разрабатываемого мобильного приложения. Были выявлены ключевые аспекты: защита данных, ограничение прав доступа пользователей. Были представлены методы защиты API с ипользованием JWT токенов и пользовательских данных с использованием алгоритма bcrypt, который обеспечивает надежное хеширование паролей пользователей.

5 Тестирование приложения

Тестирование приложений является важнейшей частью разработки программного обеспечения. Оно позволяет выявлять ошибки, проверять функциональность и качество приложения перед его запуском.

Тестирование играет важную роль в снижении рисков, связанных с выпуском некачественного программного обеспечения. Это позволяет избежать финансовых потерь, рисков и недовольства пользователей

5.1 Тестирование валидации данных

Для начала протестируем страницу авторизации. Если пользователь не зарегистрирован, ввел неверные данные или не ввел какое-либо поле, то он получит предупреждение, показанное на рисунке 5.1.

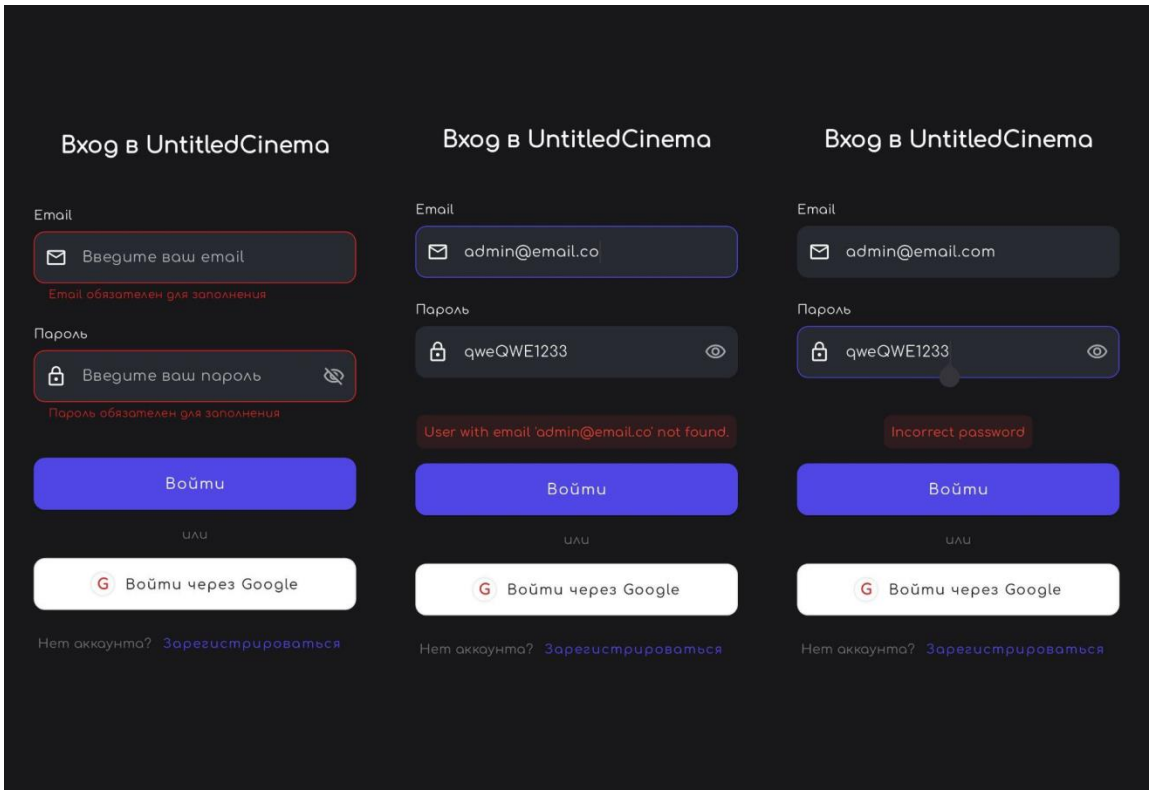


Рисунок 5.1 – Тестирование авторизации

Далее протестируем страницу регистрации. Пользователь может зарегистрироваться с уникальным email и именем, также ему необходимо повторить введённый пароль. Роль пользователя автоматически считается User. Кроме того, есть проверка на корректный email и пароль. Тестирование продемонстрировано на рисунке 5.2.

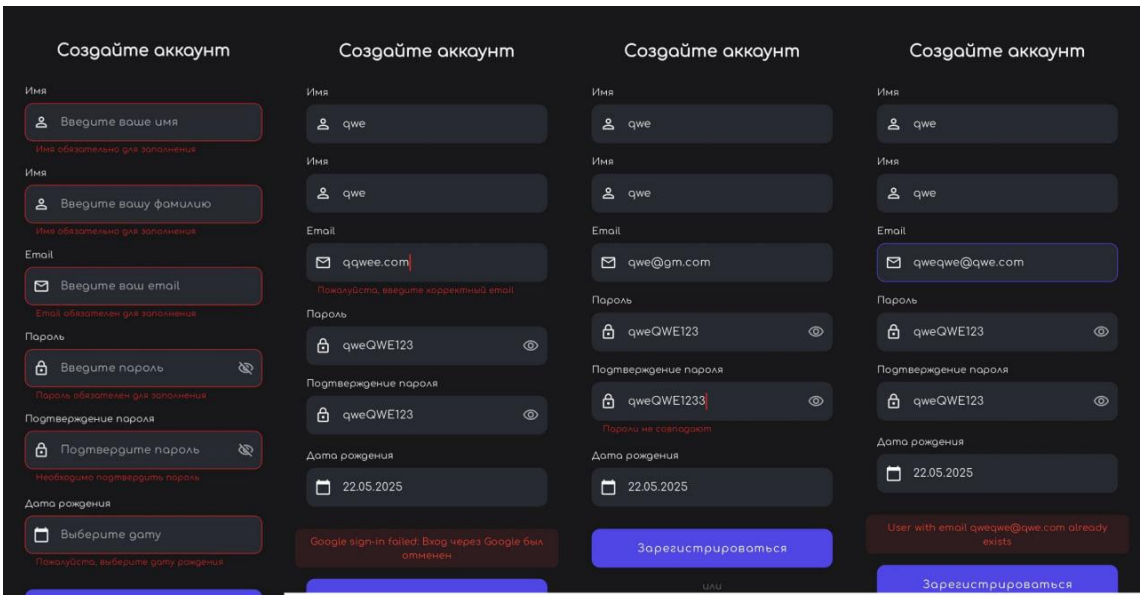


Рисунок 5.2 – Тестирование регистрации

Кроме рассмотренных возможностей приложения, обработаны возможные ошибки при редактировании профиля. Например, нельзя сохранить пользователя с пустым именем и фамилией, как минимум.Продемонстрировано на рисунке 5.4.

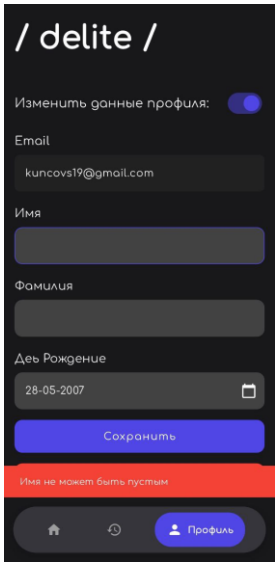


Рисунок 5.4 – Обработка ошибки при изменении профиля

5.2 Вывод

Тестирование приложения является неотъемлемой частью процесса разработки, позволяющей обеспечить высокое качество продукта и удовлетворенность пользователей. В рамках данного проекта, благодаря тщательной валидации и обработке ошибок, было создано надежное и удобное в использовании мобильное приложение для бронирования билетов в кинотеатре, которое успешно выполняет свои функции и отвечает потребностям целевой аудитории.

6. Руководство пользователя

В данном разделе будет описано руководство по использованию для гостя, пользователя и администратора.

6.1 Руководство администратора

Чтобы продолжить работу в приложении как администратор, пользователь должен иметь специальную роль и войти в приложение под соответствующей учетной записью для выполнения функций как администратор. Для него доступно только 2 страницы – «Управление» и «Профиль» где есть только почта администратора и кнопка чтобы выйти из учетной записи. При входе в приложение как администратора будет открыта страница, показанная на рисунке 6.1.

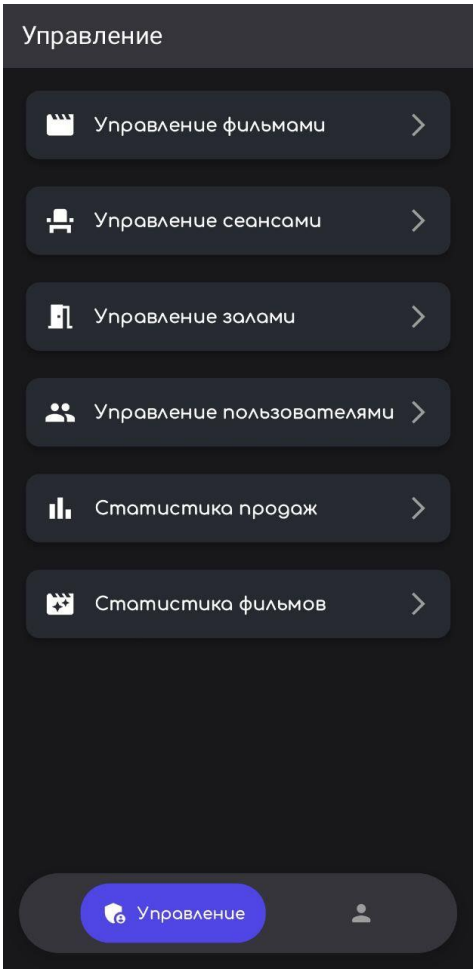


Рисунок 6.1 – Страница «Управление» для администратора

Если перейти по пункту Управление пользователями то можно будет видеть всхе пользователей, их имя, фамилию и почту и удалять их. Показано на рисунке 6.2.

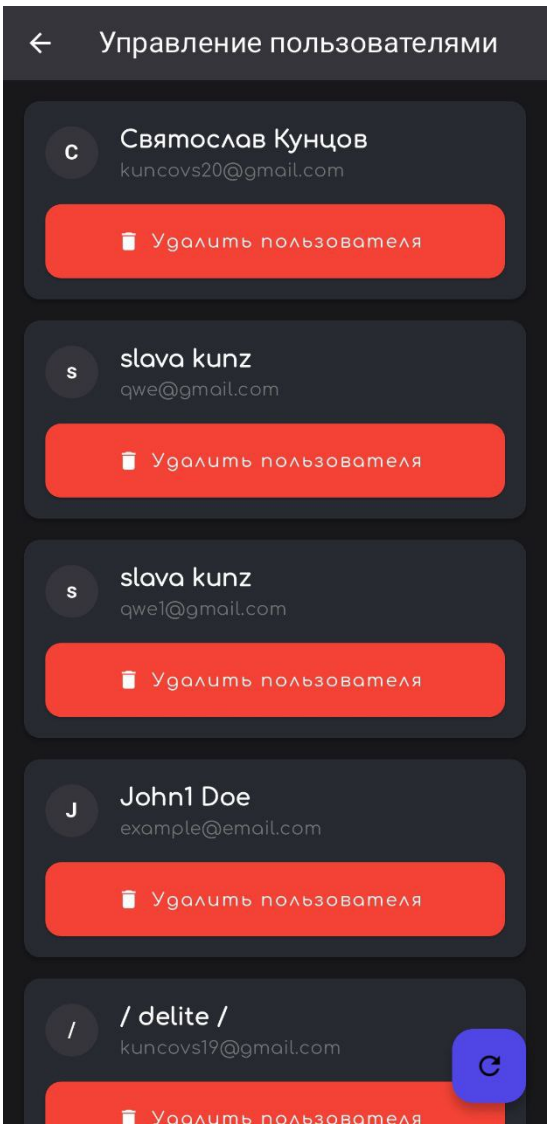


Рисунок 6.2 – Страница «Управление пользователями» для администратора

Если выбрать пункт Управление залами страницы «Управление» то можно увидеть список всех залов. Администратор может как отредактировать или удалить зал, так и создать новый. Есть 2 вида залов – с простой рассадкой и сложной, если простую рассадку мы можем создать указав количество рядом и мест в ряде. Для создания сложного зала требуется вручную задать JSON-схему, которая представляет собой двумерный массив чисел.

В этой схеме число:

– -1 означает отсутствие места (проход или техническая зона).

– 1 стандартное место.

– 2 и более уже зависят от создаваемых администратором типов мест.

Каждый вложенный массив соответствует одному ряду в зале, а элементы массива обозначают места в этом ряду.

Например, схема на [[-1, 1, 1, -1], [1, 1, 1, 1]] описывает зал с двумя рядами: в первом ряду 2 места между проходами, во втором - 4 места подряд. Такая система позволяет гибко конфигурировать залы с разным количеством мест в рядах, выделять VIP-зоны и обозначать проходы. Показано на рисунке 6.3.

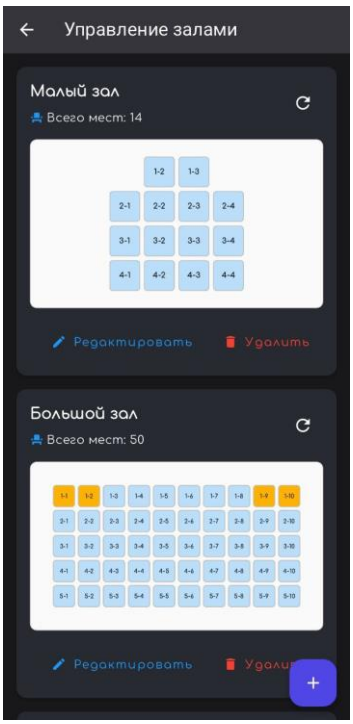


Рисунок 6.3 – Страница «Управление залами» для администратора

Если выбрать пункт Управление сеансами страницы «Управление» то можно увидеть список всех рабочих дней которые админ может удалить или создать новый. Показано на рисунке 6.4.

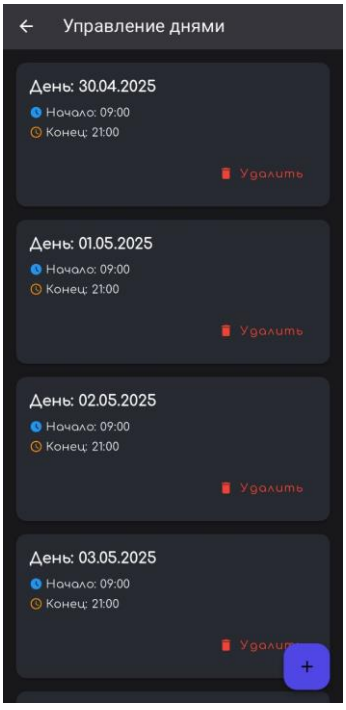


Рисунок 6.4 – Страница «Управление сеансами» для администратора

Если перейти по пункту Управление фильмами то можно будет увидеть список всех фильмов. Администратор может как отредактировать сам фильм, так и поменять постер и кадры фильма. Показано на рисунке 6.5.

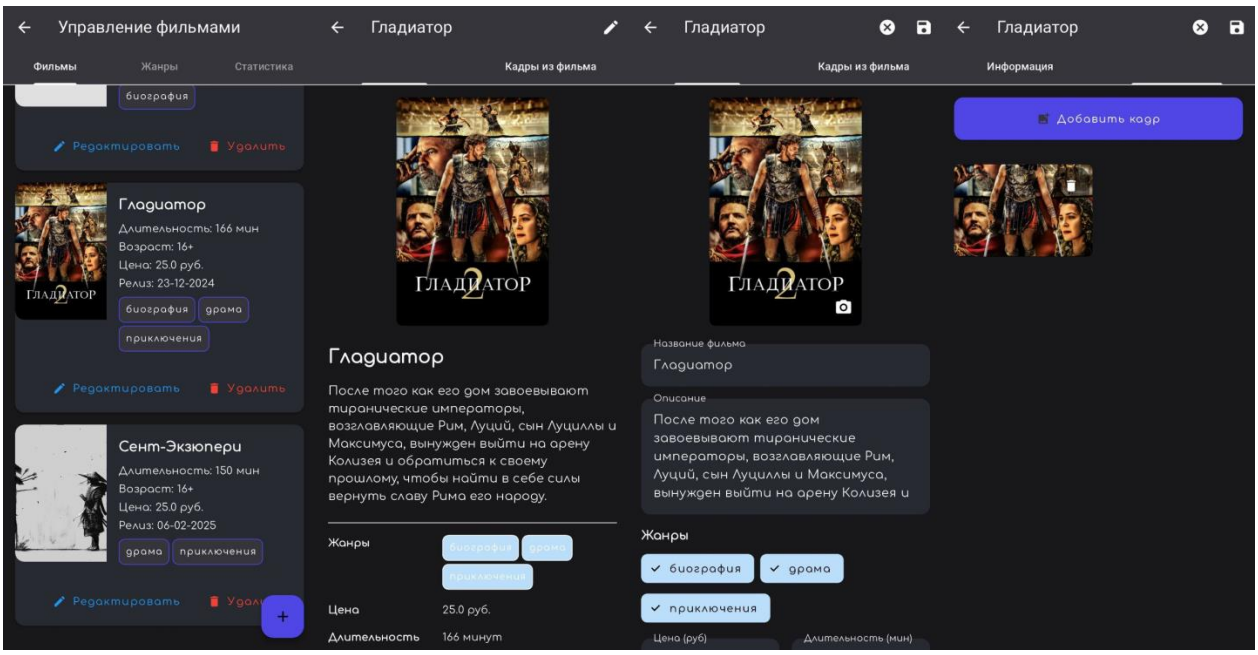


Рисунок 6.5 – Страница «Управление фильмами» для администратора

Если перейти по пункту Статистика продаж то мы видим график и статистику по продажам. Представлено на рисунке 6.6

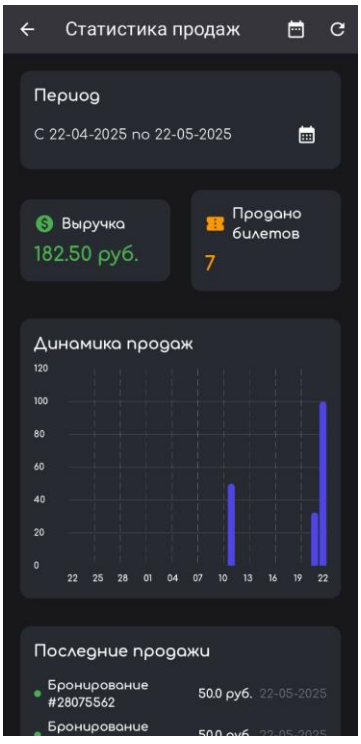


Рисунок 6.6 – Страница «Статистика продаж» для администратора

Если перейти по пункту Статистика фильмов то мы видим график и статистику по фильмам и их выручке. Представлено на рисунке 6.7

Рисунок 6.7 – Страница «Статистика фильмов» для администратора

6.2 Руководство пользователя

У пользователей, ранее не зарегистрированных, есть возможность пройти регистрацию. Он может ввести свой email и имя, а также придумать пароль для учетной записи. Страница регистрации показана на рисунке 6.8.

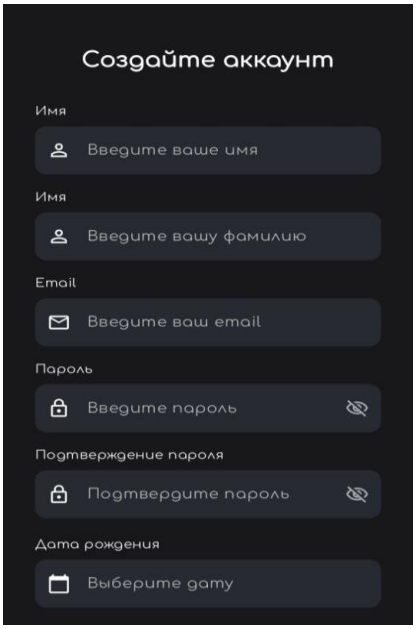


Рисунок 6.8 – Регистрация пользователя

После регистрации и авторизации пользователь переходит в основное окно приложения. Там он может просмотреть список фильмов, перейти к их

подробностям, выполнить поиск, сортировку и фильтрацию фильмов а также использовать пагинацию для перехода по списку фильмов. Также он может перейти к списку его прошлых бронирований, и конечно в свой профиль. Можно увидеть на рисунке 6.9

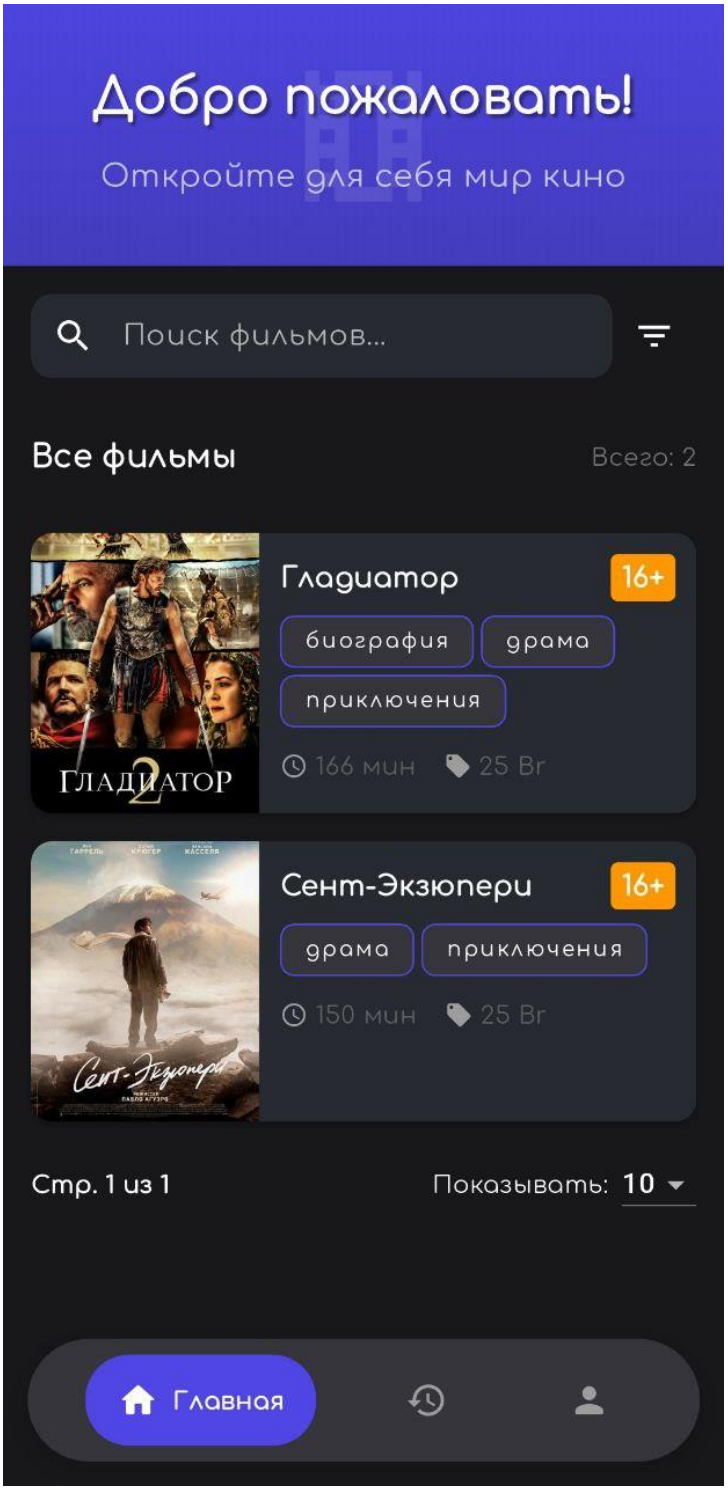


Рисунок 6.9 – Главная страница для пользователя

Если выбрать определенный фильм то мы переходим к его подробносям, а также может увидеть кадры из фильма. Также мы можем выбрать день сеанса и доступный зал. Можно увидеть на рисунке 6.10.

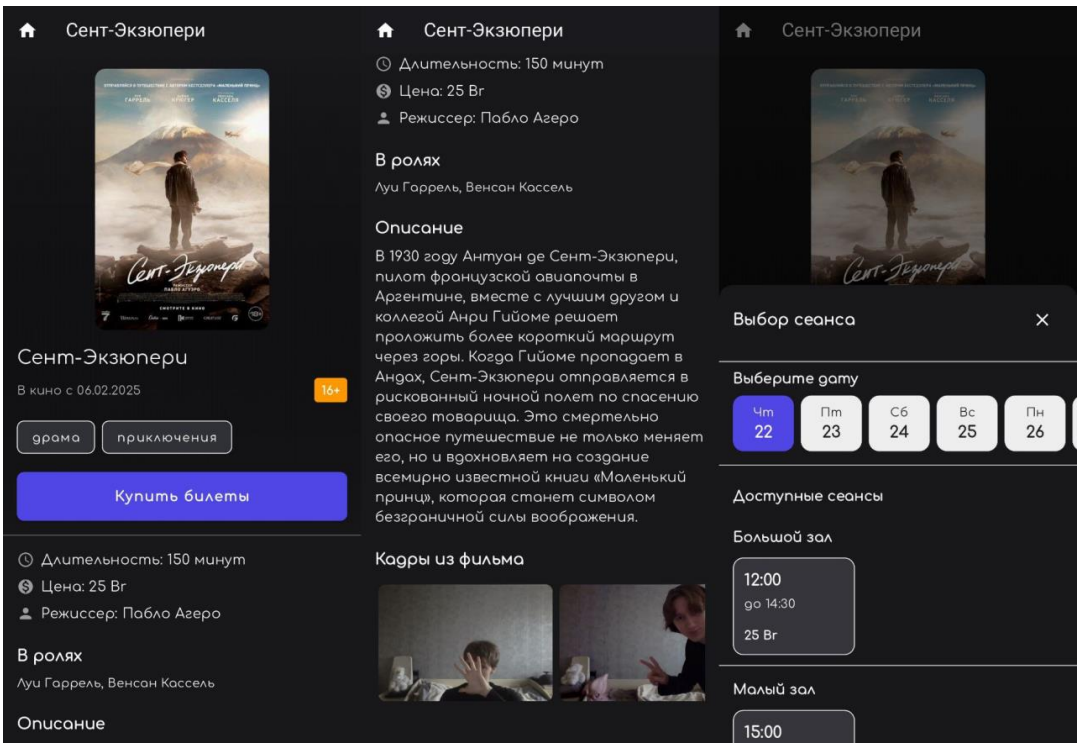


Рисунок 6.10 – Подробности выбранного фильма

Если пользователь выбран сеанс, то он переходит на страницу выбор мест. Показано на рисунке 6.11.

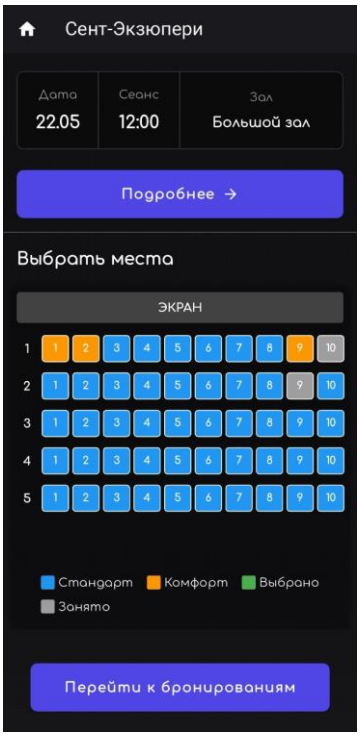


Рисунок 6.11 – Страница выбора места

Также пользователь может перейти на страницу истории бронирований и если есть бронирование в статусе Reserved то может оплатить либо отменить данное бронирование. Продемонстрировано на рисунке 6.11.

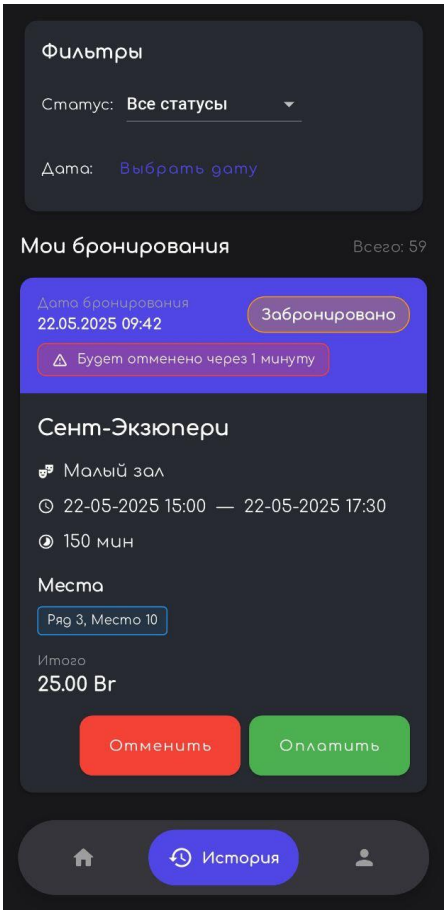


Рисунок 6.11 – Страница истории бронирований

6.4 Вывод

Таким образом, при первом запуске приложения пользователь попадает на экран Авторизация. Здесь он может ввести свои учетные данные (почту и пароль) для входа в систему.

Если у пользователя еще нет учетной записи, он может перейти на экран Регистрация. Для этого достаточно нажать на соответствующую кнопку. На экране регистрации пользователь вводит необходимую информацию для создания новой учетной записи.

Также пользователь может использовать свою учетную запись Google для входа.

После успешной авторизации пользователь попадает на Главную страницу. На этой странице отображается основная информация о фильмах.

С Главной страницы пользователь может перейти на страницу Истории бронирований.. Здесь он может отменить или оплатить активную бронь.

Также с Главной страницы доступен переход на страницу Подробностей выбранного фильма, где отображается вся информация. А также далее может выбрать сеанс и места.

На странице Профиль информация о профиле пользователя, где он может изменить свои учетные данные, удалить учётную запись или выйти из учетной записи.

Заключение

В ходе выполнения данной курсовой работы было успешно разработано современное мобильное приложение для бронирования билетов в кинотеатре. Разработанное программное решение представляет собой полнофункциональную систему, обеспечивающую пользователей актуальной информацией о киносеансах и возможностью удобного заказа билетов с учетом различных ролей и уровней доступа.

Разработанное программное средство предоставляет пользователю следующие функциональные возможности:

- регистрация и авторизация пользователей;
- просмотр актуального расписания киносеансов;
- детальная информация о фильмах с постерами и описанием;
- бронирование билетов с выбором конкретных мест в зале;
- просмотр истории бронирований и управление заказами;
- изменение данных личного профиля.

Администратору предоставляет следующие функциональные возможности:

- авторизация с расширенными правами доступа;
- управление каталогом фильмов и их характеристиками;
- настройка кинозалов и схем расположения мест;
- создание и редактирование расписания сеансов.

Гость имеет возможность:

- регистрация в системе.

Разработанное приложение обладает рядом дополнительных преимуществ, включая микросервисную архитектуру, которая обеспечивает простоту масштабирования и интеграции новых функций, оптимизированный программный код с использованием современных технологий Flutter и .NET, гарантирующий высокую производительность и стабильную работу системы бронирования, а также адаптивный пользовательский интерфейс, который обеспечивает удобство взаимодействия на различных мобильных устройствах при выборе мест в зале и оформлении заказов.

Реализованная система безопасности с использованием JWT-токенов и хэширования паролей обеспечивает надежную защиту пользовательских данных и финансовой информации при проведении транзакций по покупке билетов

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная система бронирования билетов функционирует корректно, обеспечивает все необходимые операции для эффективного управления продажами билетов в кинотеатре, а требования технического задания выполнены в полном объеме.

Список используемых источников

1 Flutter [Электронный ресурс]. – Режим доступа: https://docs.flutter.dev/tools/devtools/overview – Дата доступа: 16.04.2024

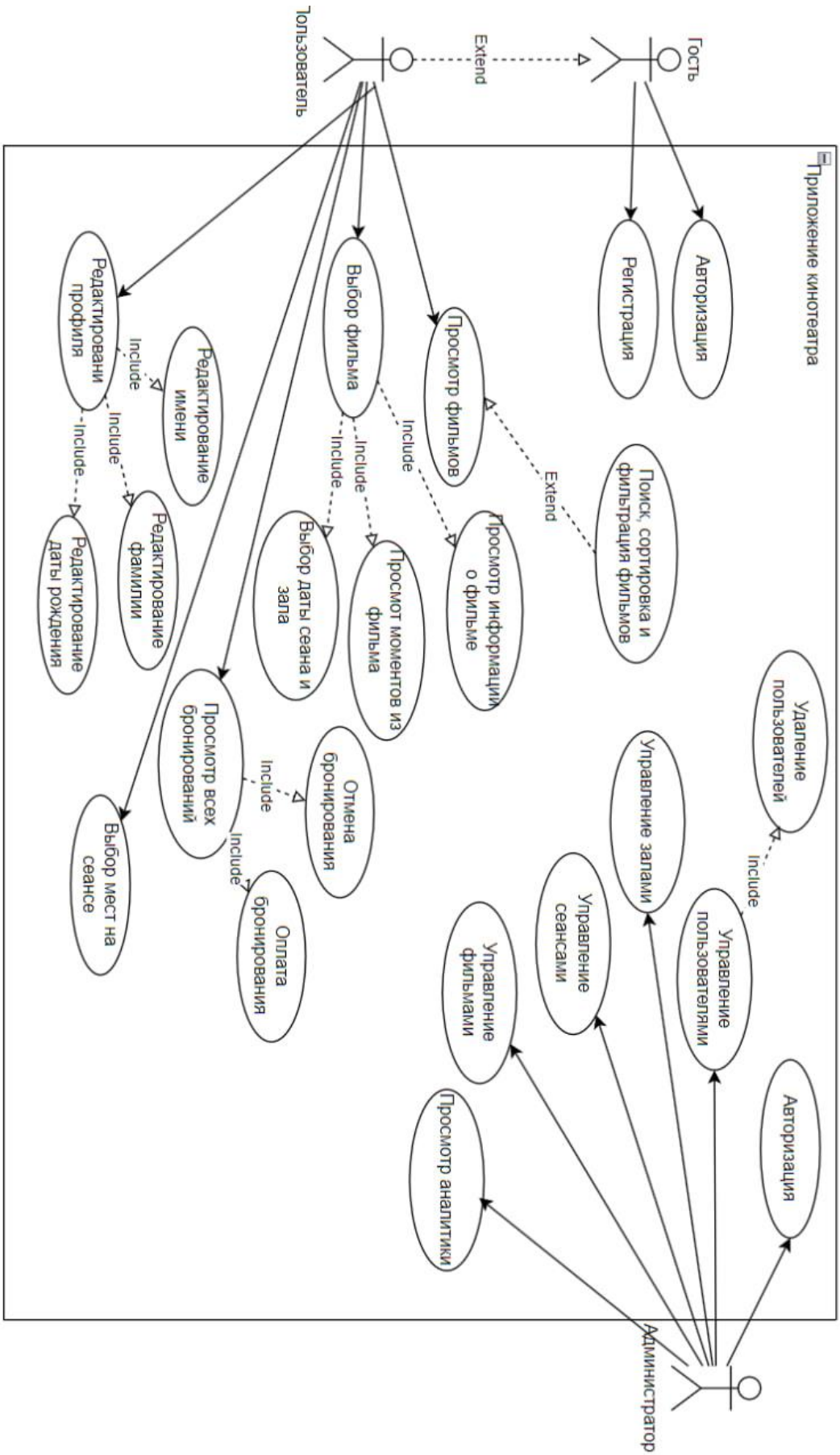
2 Solutions on Stackoverflow [Электронный ресурс]. – Электронные данные. – Режим доступа: https://stackoverflow.com. – Дата доступа: 20.04.2024;

3 Чернышев, С.А. Основы Dart. 2–е издание / С.А. Чернышев. – Москва: Издательство "Наука", 2023. – 350 с.

4 Flutterfor [Электронный ресурс]. – Режим доступа: https://flutterfor.dev/ – Дата доступа: 20.03.2024

5 Dartflutter [Электронный ресурс]. – Режим доступа: https://dartflutter.ru/ – Дата доступа: 25.03.2024

ПРИЛОЖЕНИЕ А: Диаграмма вариантов использования



ПРИЛОЖЕНИЕ Б: Функция создания бронирования

```
@override
Future<bool> createBooking({
  required String userId,
  required String sessionId,
  required List<Seat> seats,
}) async {
  try {
    final List<Map<String, dynamic>> seatMaps =
        seats
            .map(
              (seat) => {
                'id': seat.id,
                'row': seat.row,
                'column': seat.column,
              },
            )
            .toList();

    final body = {
      'userId': userId,
      'sessionId': sessionId,
      'seats': seatMaps,
    };

    final response = await client.post(ApiConstants.bookings, data:
body);

    if (response != null) {
      return true;
    } else {
      throw ServerException(
        'Ошибка при создании бронирования: ${response.statusCode}',
      );
    }
  } catch (e) {
    throw ServerException(
      'Ошибка при создании бронирования: ${e.toString()}',
    );
  }
}
```