

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

Факультет _____ информационных технологий
Кафедра _____ программной инженерии
Специальность _____ 1-40 05 01 Информационные системы и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и
стандарты проектирования»

Тема Программное средство «Продажа билетов в кинотеатры»

Исполнитель
Студентка 3 курса 1 группы _____ Водчиц Анастасия Витальевна
(Ф.И.О.)

Руководитель работы _____ асс. Гончар Егор Андреевич
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____
Председатель _____
(подпись)

Минск 2024

Содержание

Введение	4
1 Аналитический разбор аналогов	5
1.1 ByCard	5
1.2 Skyline	7
1.3 Moon	8
1.4 Выводы по разделу	10
2 Анализ требований к программному средству и разработка функциональных требований	11
2.1 Определение требований к программному средству	11
2.2 Описание средств разработки	12
2.3 Описание функциональности программного средства	13
2.4 Выводы по разделу	14
3 Проектирование программного средства	15
3.1 Архитектура проектируемого программного средства	15
3.2 Подключение к базе данных	18
3.3 Разработка базы данных	20
3.4 Проектирование диаграммы последовательностей	22
3.5 Выводы по разделу	22
4 Реализация программного средства	23
4.1 Реализация паттерна Singleton	23
4.2 Использование конвертеров	24
4.3 Паттерн Repository	25
4.4 Паттерн UnitOfWork	26
4.5 Реализация переключения языков	27
4.6 Реализация паттерна MVVM	28
4.6.1 Модель	28
4.6.2 Представление	29
4.6.3 Модель представления	30
4.7 Реализация паттерна ICommand	31
4.8 Выводы по разделу	32
5 Тестирование, проверка работоспособности и анализ полученных результатов	33
5.1 Обработка ошибок	33
5.2 Выводы по разделу	38
6 Руководство по использованию	39
Заключение	45
Список используемых источников	46
Приложение А	47
Приложение Б	48
Приложение В	49
Приложение Г	50
Приложение Д	51
Приложение Е	53

Введение

Современный мир характеризуется стремительным развитием технологий, которые неизменно влияют на все сферы человеческой жизни. В условиях цифровой трансформации общества автоматизация процессов стала неотъемлемой частью успешной деятельности компаний и организаций. Это особенно актуально для сферы развлечений, где высокий уровень конкуренции и постоянные изменения в предпочтениях аудитории требуют внедрения инновационных решений, направленных на улучшение качества сервиса.

Одним из ключевых аспектов работы современных кинотеатров является организация процесса продажи билетов. Для посетителей важно, чтобы покупка билетов была быстрой, удобной и доступной, а для администрации кинотеатра автоматизация позволяет эффективно управлять сеансами, распределением мест в залах и учетными операциями. Устаревшие методы работы, такие как ручное оформление билетов или использование несогласованных систем, больше не соответствуют требованиям времени. В таких условиях возникает необходимость в создании специализированного программного обеспечения, которое объединяет удобство использования, высокую производительность и широкую функциональность.

Данная курсовая работа посвящена разработке программного средства «Продажа билетов в кинотеатры». Создание подобной системы имеет важное значение как с точки зрения улучшения взаимодействия клиентов с кинотеатром, так и с позиции оптимизации работы сотрудников. Она должна стать инструментом, который позволит не только ускорить процесс покупки билетов, но и повысить удовлетворенность посетителей, предоставив им современный способ планирования своего досуга.

В работе уделяется внимание основным этапам проектирования и реализации программного продукта, включая разработку концепции системы, построение модели данных, создание пользовательского интерфейса и тестирование готового решения. Основной акцент делается на обеспечении простоты и удобства использования системы как для сотрудников кинотеатра, так и для его клиентов. Внедрение такого решения в повседневную практику позволит сократить время обработки заказов, минимизировать человеческие ошибки и создать благоприятную среду для взаимодействия между всеми участниками процесса.

Таким образом, цель работы состоит в создании функционального программного обеспечения, способного повысить эффективность работы кинотеатров и удовлетворенность посетителей, а также продемонстрировать возможности современных технологий в области автоматизации услуг.

1 Аналитический разбор аналогов

Разработка любого программного обеспечения, в том числе системы «Продажа билетов в кинотеатры», требует глубокого изучения уже существующих решений на рынке. Это позволяет определить их сильные и слабые стороны, а также учесть особенности, которые могут быть полезны в реализации собственного проекта. Анализ аналогов служит основой для формирования требований к будущему приложению, помогает выбрать оптимальные технологии и подходы для реализации функционала и избежать ошибок, допущенных конкурентами.

В данном разделе подробно рассмотрены три популярных сервиса для продажи билетов на киносеансы: ByCard, Skyline и Moon. Анализ включает разбор их функциональных возможностей, а также выявление аспектов, которые могут быть использованы в рамках курсового проекта. Особое внимание уделено тем аспектам, которые связаны с удобством пользователя, функциональностью фильтров, процессом оформления заказа и управления данными о мероприятиях.

1.1 ByCard

Сервис ByCard известен своей гибкостью и широкими возможностями. Конечно, на данном веб-сайте продаются билеты и на другие виды мероприятий, не только кино, но в рамках темы проекта, будет рассмотрен раздел продажи билетов в кино. На главной странице, представленной на рисунке 1.1, пользователям сразу предлагается доступ к афише, которая содержит перечень фильмов. Здесь указаны привлекающие внимание постеры, рейтинги и стоимость билетов.

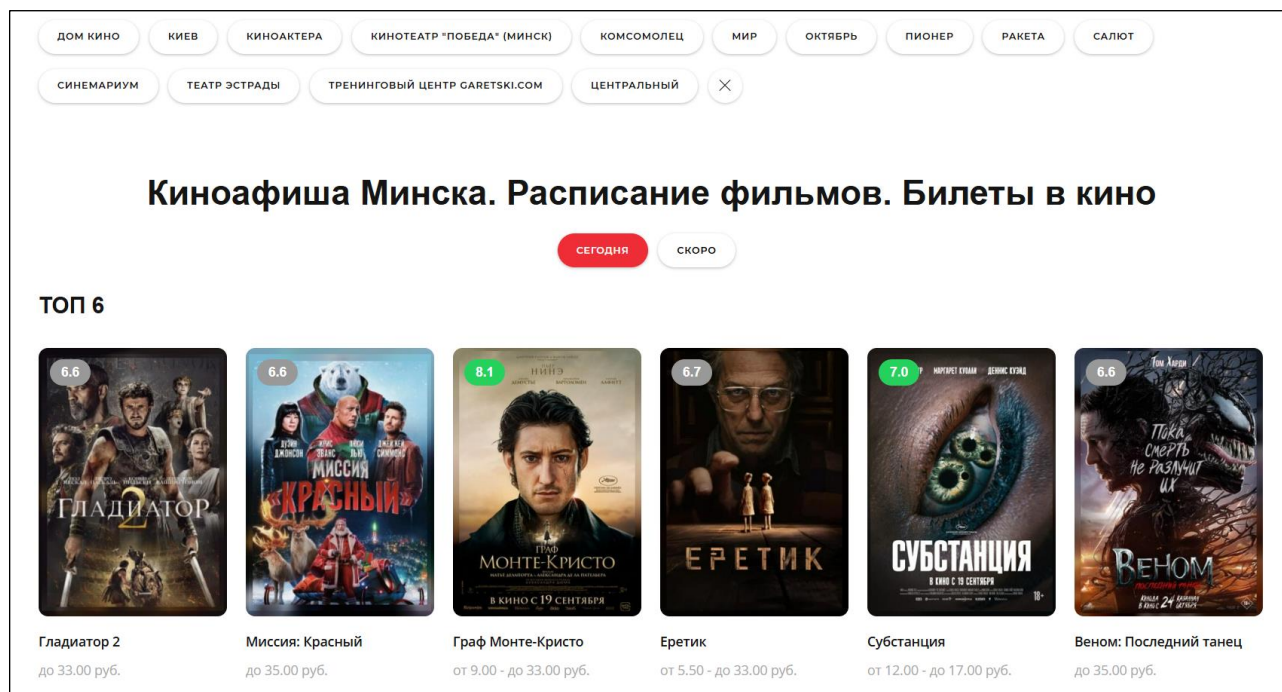


Рисунок 1.1 – Главная страница веб-сайта ByCard

Пользователь может фильтровать список фильмов по дате и конкретному кинотеатру, как показано на рисунке 1.2. Фильтрация по другим важным критериям,

таким как жанр или цена, отсутствует, что усложняет процесс выбора кинофильма для пользователей.

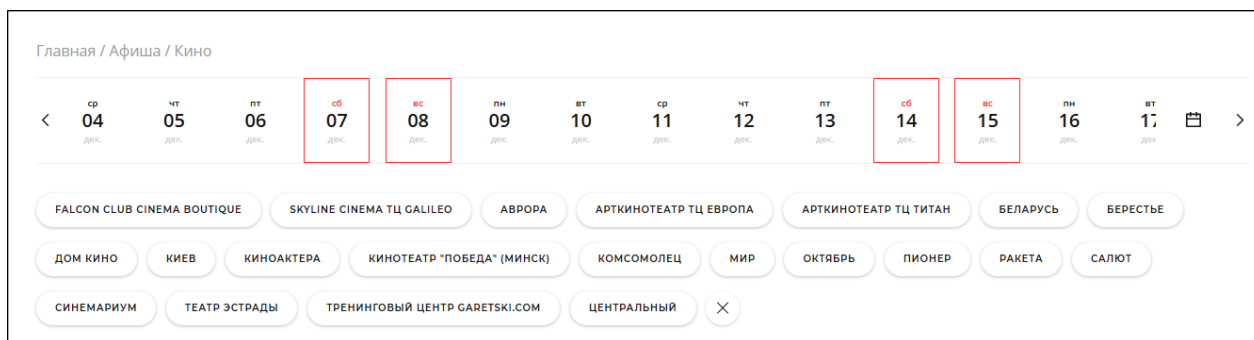


Рисунок 1.2 – Фильтрация веб-сайта VyCard

На отдельной странице фильма представлены описание, отзывы, трейлер, расписание.

Важным функциональным элементом VyCard является возможность выбора мест с помощью интерактивной схемы зала, как на рисунке 1.3. Это решение позволяет пользователям видеть расположение своих мест относительно экрана, что особенно важно для тех, кто предпочитает сидеть в определённых секторах. После выбора мест пользователи могут быстро оформить заказ через интегрированную систему онлайн-оплаты, которая поддерживает различные способы оплаты, включая банковские карты и электронные кошельки. Оплата подтверждается моментально, а билеты отправляются на электронную почту или предоставляются в виде QR-кода.

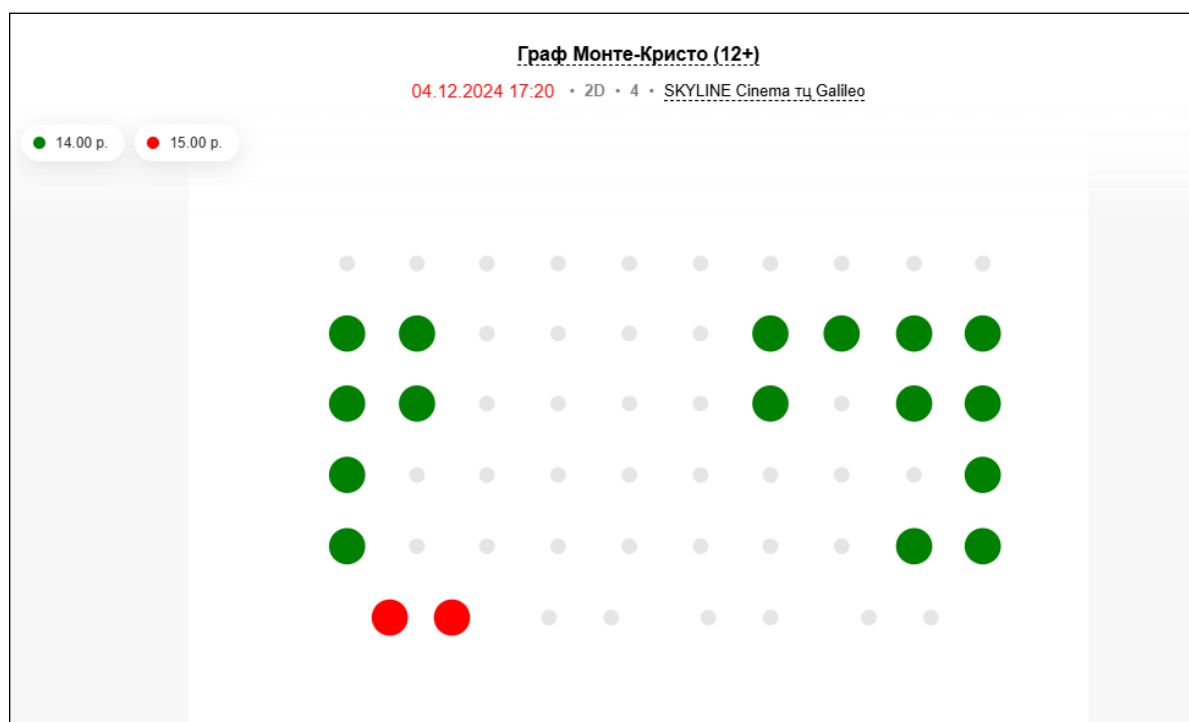


Рисунок 1.3 – Схема зала веб-сайта VyCard

Не менее важной функцией является личный кабинет, где пользователи могут просматривать историю своих покупок и управлять бронированием. Это полезно для

тех, кто часто посещает кино, так как позволяет сохранять удобные настройки и следить за накопленными бонусами.

1.2 Skyline

Платформа Skyline предлагает несколько иной подход. Она ориентирована на минимализм и удобство, предоставляя пользователям простой доступ к расписанию фильмов. Тут присутствует сортировка по дате, времени сеансов, жанрам и форматам. При наведении на фильм, пользователь получает информацию о времени проведения, формате, возрастном ограничении, что продемонстрировано на рисунке 1.4.

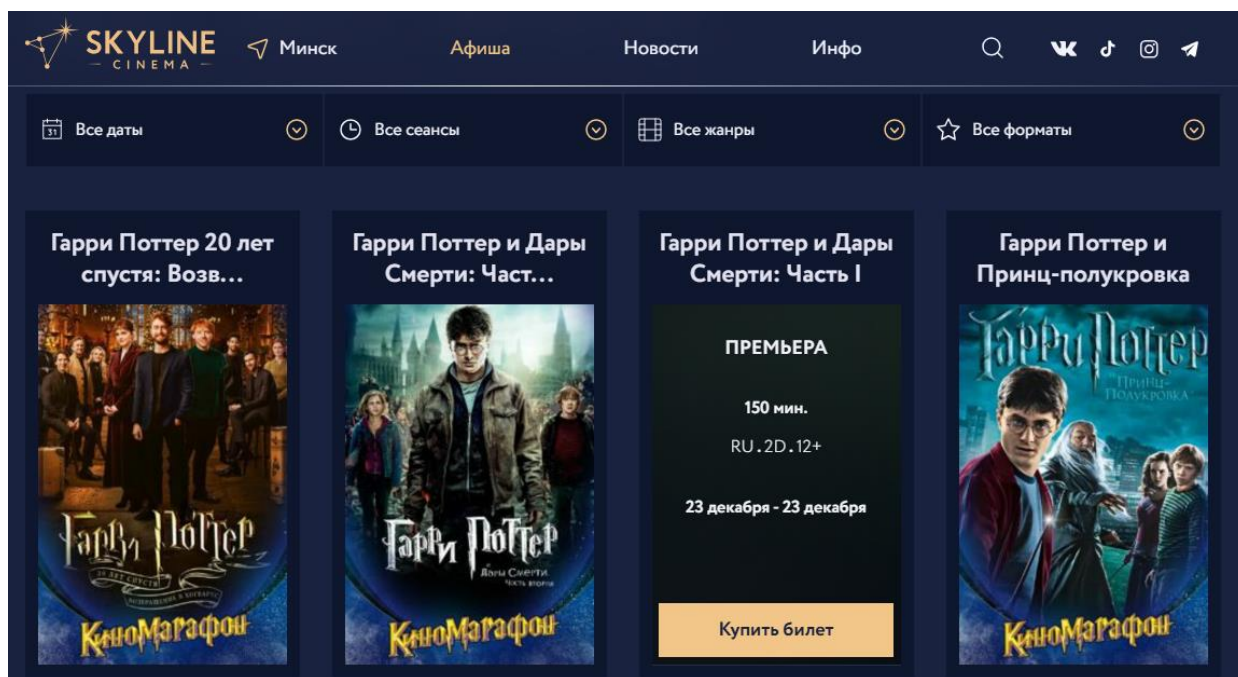


Рисунок 1.4 – Главная страница веб-сайта ByCard

На отдельной странице кинофильма расположено описание, даты проведения, возрастное ограничение. Трейлера и комментариев нет. Однако, внутри кинофильма также можно применить сортировку, как показано на рисунке 1.5.

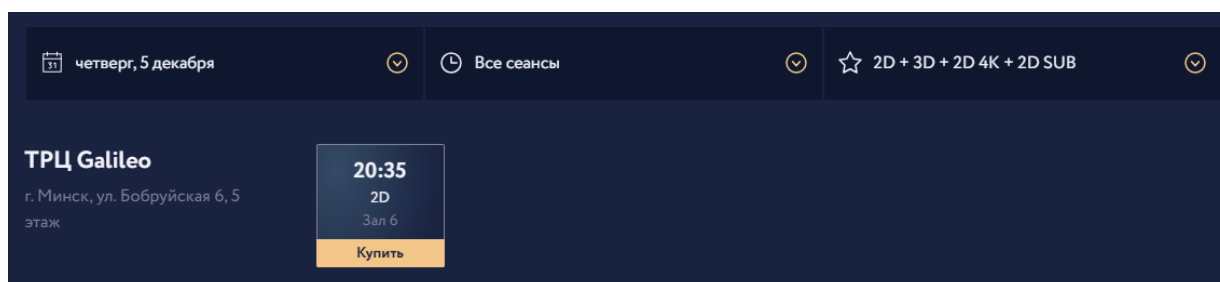


Рисунок 1.5 – Сортировка по отдельному кинофильму веб-сайта ByCard

Всплывающее окно с выбором места, которое продемонстрировано на рисунке 1.6, в кинозале оформлено удобно и интуитивно понятно. Выбор места в зале на платформе Skyline более интерактивен по сравнению с другими сайтами. Свободные

и занятые места отображаются разными цветами. При выборе места, пользователь сталкивается с вторым всплывающим окном, уточняющим, нуждается ли он в льготных билетах (пенсионеры, студенты), что очень удобно и не требует дополнительных действий для получения льготной цены.

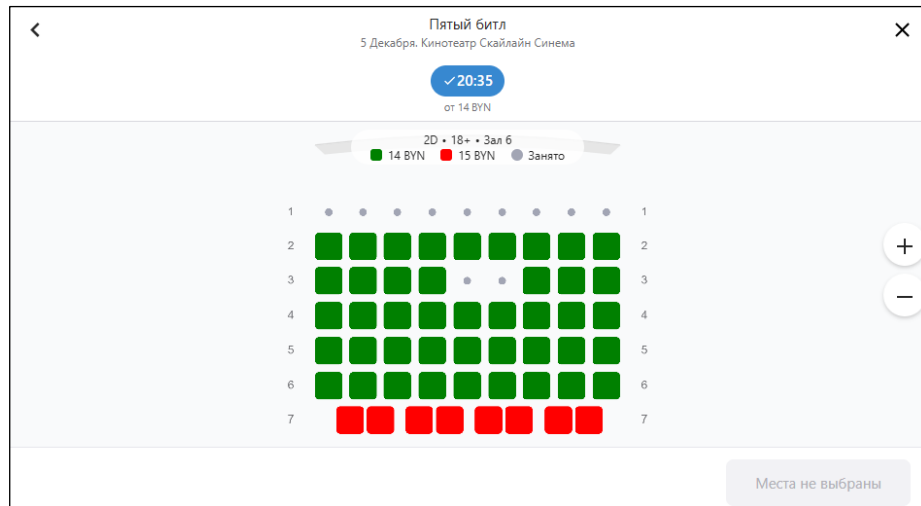


Рисунок 1.6 – Схема зала веб-сайта ByCard

На сайте Skyline функционал личного кабинета сосредоточен на удобстве управления бронированиями и минималистичном интерфейсе. В личном кабинете пользователь может увидеть историю своих заказов, что особенно важно для регулярных посетителей кинотеатров.

1.3 Moon

Платформа Moon расширяет границы стандартного представления о сайтах для продажи билетов. Особое внимание уделено системам фильтрации, продемонстрированной на рисунке 1.7. Пользователи могут искать фильмы по дате, жанру, месту проведения, времени, виду зала, технологии, языку, жанрам.

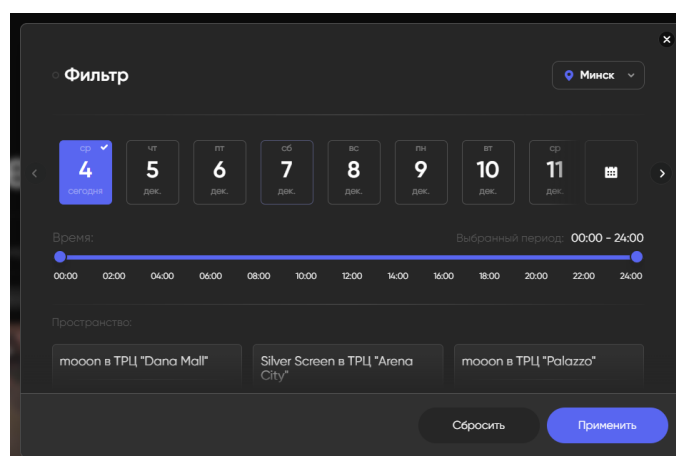


Рисунок 1.7 – Фильтрация веб-сайта Moon

На самой главной странице расположены фильмы с перечислением жанров, времени продолжительности, языка, возрастного ограничения.

Этот веб-сайт имеет невероятно подробное описание кинофильма. В отличие от других сервисов, тут пользователь может ознакомиться с режиссёром, актёрами, галереей, трейлером. Однако на веб-сайте отсутствует система оценок фильмов и комментирования. Отдельная страница кинофильма представлена на рисунке 1.8.

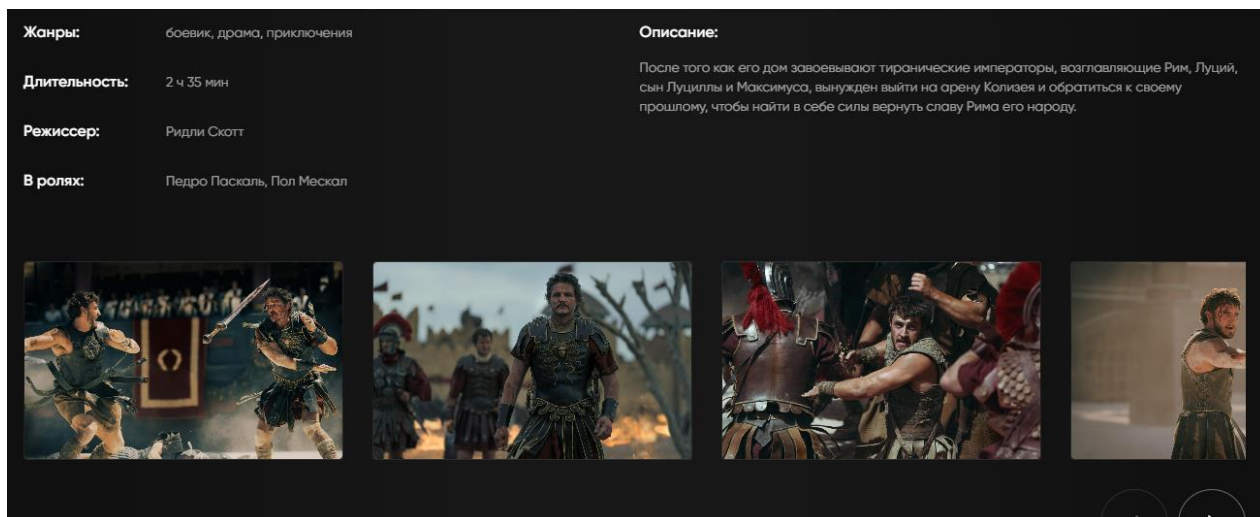


Рисунок 1.8 – Отдельная страница кинофильма веб-сайта Моон

Выбор места в зале на Моон реализован с использованием современной интерактивной схемы, как показано на рисунке 1.9. Пользователь видит подробную визуализацию зала, включая расположение экрана, выходов и других важных деталей. Это позволяет более точно выбрать место, которое подходит пользователю. Свободные и занятые места обозначены цветовой индикацией, что делает процесс выбора быстрым и интуитивным.

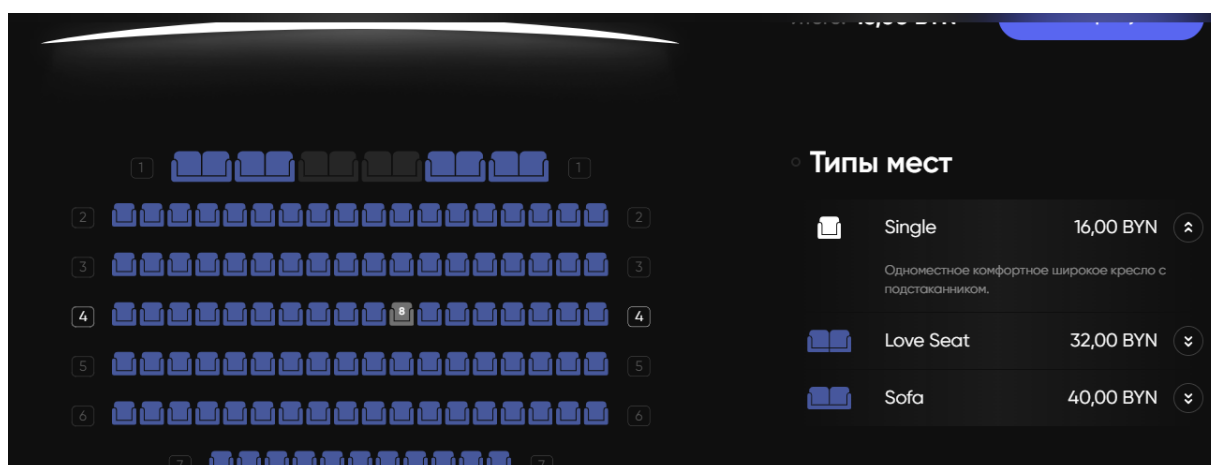


Рисунок 1.9 – Схема зала веб-сайта Моон

Платформа Моон предлагает более расширенные функции личного кабинета, направленные на повышение пользовательского опыта. Здесь пользователь может не только просматривать историю заказов, но и получать персонализированные рекомендации на основе своих предпочтений.

1.4 Выводы по разделу

В ходе анализа трех популярных сервисов для продажи билетов в кинотеатры ByCard, Skyline и Moon были выявлены их ключевые особенности, сильные и слабые стороны с точки зрения функциональности и удобства для пользователя.

Обобщая результаты анализа, можно сделать следующие выводы:

- наиболее развитая система фильтрации реализована в Moon, позволяя пользователю быстро находить интересующие фильмы по множеству параметров. ByCard ограничивается фильтрацией по дате и кинотеатру, что является существенным недостатком. Skyline предлагает средний уровень фильтрации, достаточный для базового поиска;

- все три сервиса предоставляют интерактивную схему зала. Skyline и Moon предлагают более удобные и наглядные схемы с цветовой индикацией и дополнительными функциями (льготные билеты в Skyline, детальная визуализация в Moon);

- наиболее полную информацию о фильме предоставляет Moon, включая данные о режиссере, актерах, галерею и трейлер. ByCard и Skyline ограничиваются базовым описанием;

- все три сервиса предлагают личный кабинет с базовыми функциями. Moon выделяется возможностью получения персонализированных рекомендаций;

- Skyline делает акцент на минимализме и простоте, что делает его удобным для быстрого бронирования. Moon, несмотря на обширный функционал, также удобен в использовании благодаря продуманному интерфейсу. ByCard может показаться несколько перегруженным информацией.

В рамках курсового проекта целесообразно учесть следующие аспекты:

- разработать гибкую и многофункциональную систему фильтрации, аналогичную Moon, для обеспечения удобного поиска фильмов по различным критериям;

- реализовать наглядную и интуитивно понятную интерактивную схему зала с цветовой индикацией свободных и занятых мест, возможностью выбора места относительно экрана;

- предоставить подробную информацию о каждом фильме;

- разработать личный кабинет с возможностью просмотра истории заказов.

Данный анализ позволил сформировать представление о лучших практиках в сфере онлайн-продажи билетов в кинотеатры и определить ключевые направления для разработки собственного приложения в рамках курсового проекта. Учет сильных сторон рассмотренных аналогов и устранение их недостатков позволит создать конкурентоспособный и удобный для пользователей сервис.

2 Анализ требований к программному средству и разработка функциональных требований

Начало разработки любого программного обеспечения лежит в плоскости четкого понимания его назначения и функциональности. Необходимо разработать требования к программному средству для продажи билетов в кинотеатры, выбрать оптимальные инструменты для его реализации и погрузиться в процесс формирования функциональных требований, исходя из потребностей будущих пользователей системы. Этот процесс критически важен, так как от точности и полноты собранных требований напрямую зависит успешность всего проекта.

Помимо анализа требований, необходимо уделить внимание выбору технологического стека. Правильный подбор инструментов разработки – это залог не только успешной реализации, но и обеспечения надежности, производительности и масштабируемости будущей системы. Этот выбор определит архитектуру приложения и подходы к решению поставленных задач.

2.1 Определение требований к программному средству

Разработка программного средства для продажи билетов в кинотеатры требует тщательного определения требований, которые обеспечат эффективное взаимодействие между различными ролями пользователей системы.

Программное средство предназначено для автоматизации процесса продажи билетов в кинотеатрах. Оно предоставляет удобный интерфейс для клиентов, позволяющий им выбирать фильмы, просматривать расписание сеансов, выбирать места в зале и приобретать билеты. Для сотрудников кинотеатров система облегчает управление расписанием, добавление новых фильмов и анализ продаж.

В данном разделе подробно рассматриваются требования к проекту, разделённые по трем основным ролям: менеджер, букер и клиент. Каждая из этих ролей выполняет уникальные функции, необходимые для бесперебойной работы системы, управления данными и обеспечения высокого уровня обслуживания пользователей. Понимание и точное формулирование этих требований является ключевым этапом в процессе разработки, позволяющим создать функционально полноценное и удобное приложение.

В системе предусмотрено три основные роли пользователей:

- пользователь может просматривать каталог кинофильмов, искать кинофильмы, добавлять билеты в корзину, оформлять заказ, а также оставлять комментарии и обращаться в службу поддержки;

- букер может создавать, редактировать и удалять кинофильмы, управлять расписанием кинофильмов, просматривать свои кинофильмы и обращаться в службу поддержки;

- менеджер – это пользователь с расширенными правами, который управляет категориями кинофильмов, просматривает заявки на возврат билетов, просматривает списки кинофильмов и их расписаний, управляет службой поддержки, а также управляет кинотеатрами.

Система должна быть интуитивно понятной, надёжной и производительной. Поддержка мультиязычности необходима для удобства пользователей из разных

регионов. Данные о клиентах, билетах и расписаниях должны храниться в защищённой базе данных с минимизацией риска потери информации.

2.2 Описание средств разработки

Для реализации программного средства «Продажа билетов в кинотеатры» используются современные инструменты разработки, обеспечивающие высокую производительность, удобство создания пользовательских интерфейсов и работы с данными. Эти инструменты выбираются с учётом их функциональных возможностей, соответствующих требованиям проекта.

В качестве основы для разработки была выбрана платформа .NET 8, представляющая собой современную, кроссплатформенную среду с открытым исходным кодом.

Центральным инструментом в процессе разработки выступает интегрированная среда разработки (IDE) Microsoft Visual Studio 2022. Она предоставляет разработчикам полный набор функций для написания кода, отладки, тестирования и развертывания приложений. Visual Studio 2022 также обеспечивает полную поддержку .NET 6 и WPF, предоставляя соответствующие шаблоны проектов и инструменты, необходимые для работы с этими технологиями.

Для создания пользовательского интерфейса настольного приложения была выбрана технология WPF (Windows Presentation Foundation), являющаяся частью платформы .NET. WPF, предлагает разработчикам мощные средства для построения современных, интерактивных и визуально привлекательных интерфейсов. Одним из ключевых преимуществ WPF является поощрение использования шаблона проектирования MVVM (Model-View-ViewModel), который способствует четкому разделению UI и логики приложения.

Описание внешнего вида и структуры пользовательского интерфейса осуществляется с помощью декларативного языка разметки XAML, основанного на XML. XAML позволяет разработчикам описывать структуру окон, расположение элементов управления, их внешний вид и поведение, отделяя эти аспекты от логики приложения, реализованной на языке C#. Такой подход делает код более читаемым и упрощает его поддержку. Visual Studio 2022, в свою очередь, предоставляет визуальный редактор XAML и другие инструменты, упрощающие работу с разметкой.

Язык программирования C# выступает в роли основного инструмента для реализации бизнес-логики приложения. На C# пишутся классы, представляющие сущности предметной области (фильмы, сеансы, пользователи, билеты), обрабатываются события пользовательского интерфейса и осуществляется взаимодействие с базой данных.

Для управления данными и упрощения работы с ними используется Entity Framework Core – легкий, расширяемый и кроссплатформенный ORM (Object-Relational Mapper) для .NET. В проекте применяется подход Code First, при котором модель базы данных описывается с помощью C#-классов. Entity Framework Core берет на себя генерацию и обновление схемы базы данных на основе C#-модели, а также предоставляет механизм миграций для управления изменениями в структуре базы данных.

В качестве системы управления базами данных (СУБД) была выбрана Microsoft SQL Server – реляционная СУБД, разработанная компанией Microsoft. SQL Server обеспечивает надежное хранение данных, поддерживает высокую производительность и масштабируемость, что особенно важно для приложений, работающих с большими объемами данных и высокими нагрузками. Тесная интеграция SQL Server с платформой .NET и средой разработки Visual Studio делает его естественным выбором для данного проекта.

Такой комплексный подход к выбору средств разработки позволяет создать современное, надежное и производительное приложение, полностью соответствующее требованиям, предъявляемым к системе продажи билетов в кинотеатры.

2.3 Описание функциональности программного средства

Программное средство состоит из нескольких взаимосвязанных модулей, обеспечивающих выполнение функций для различных категорий пользователей. Клиент имеет доступ к главному экрану с каталогом фильмов, фильтрацией и сортировкой, личному кабинету, а также к разделу для покупки билетов с выбором мест. Букер работает через личный кабинет, где управляет расписанием и каталогом фильмов. Менеджер использует административную панель для модерации контента.

Система поддерживает несколько языков интерфейса, что делает её удобной для пользователей из разных стран. Язык интерфейса можно выбрать при регистрации или изменить в настройках профиля. Вся текстовая информация интерфейса хранится в специализированных файлах локализации, что позволяет легко добавлять новые языки.

Букер получает доступ к интерфейсу, который позволяет добавлять новые фильмы, изменять их описание и удалять устаревшие записи. В разделе управления расписанием букер может назначать фильмы на конкретные сеансы, выбирая дату, время и зал. В системе исключена возможность пересечений сеансов.

Менеджер управляет пользователями и модерацией контента через административную панель. Он может отвечать на обращения и редактировать кинофильмы, управлять содержанием базы данных, управлять возвратами билетов.

Схема зала визуально отображает доступные места, которые пользователь может выбрать с помощью кликов. Свободные места подсвечиваются зелёным, купленные – серым, забронированные – жёлтым, а выбранные пользователем – оранжевым.

Функциональность системы проиллюстрирована с помощью UML-диаграммы прецедентов, представленной в Приложении Б. Диаграмма прецедентов (Use Case Diagram) является одним из ключевых элементов языка UML (Unified Modeling Language) и служит для описания функциональных требований к системе с точки зрения внешнего наблюдателя. Она показывает, как акторы (действующие лица) взаимодействуют с системой для достижения определенных целей.

Функции менеджера сервиса:

- редактирование личных данных пользователей;
- редактирование личных данные букеров;

- просмотр и управление обращениями (имеется возможность закрыть сообщение или добавить к нему ответ);
- контроль содержимого базы данных.

Функции букера:

- добавление новых фильмов с указанием основной информации;
- удаление фильмов из каталога;
- обновление информации о фильмах;
- назначение фильмов на конкретные сеансы, установка времени, даты и зала;
- регистрация и авторизация.

Функции клиента:

- регистрация и авторизация;
- просмотр каталога кинофильмов;
- отображение фильмов с полной информацией;
- фильтрация фильмов по жанру, дате выхода, доступности в кинотеатрах;
- сортировка по дате и цене;
- возможность оценивать фильмы по шкале (1-5 звёзд) и писать отзывы;
- просмотр отзывов других пользователей;
- покупка билетов;
- оформление возвратов билетов;
- просмотр доступных сеансов и интерактивный выбор мест в зале.

2.4 Выводы по разделу

Раздел посвящен разработке системы продажи билетов в кинотеатры, включая функциональные требования, выбор средств и описание функциональности.

Определены задачи и разграничены обязанности трёх ролей: менеджера, букера и клиента. Менеджер управляет данными и обращениями, букер – информацией о фильмах и сеансах, клиент – просматривает каталог, ищет фильмы, покупает и возвращает билеты, выбирает места.

Выбор средств разработки обоснован созданием надежного и производительного приложения: платформа .NET 8, среда Visual Studio 2022, технология WPF для интерфейса с MVVM и XAML, язык C#, Entity Framework Core для работы с данными и SQL Server для их хранения.

Функциональность описана с применением UML-диаграммы прецедентов, демонстрирующей взаимодействие пользователей (Пользователь, Букер, Менеджер) с системой.

В итоге, раздел формирует понимание задач приложения, его функционирования и используемых технологий. Анализ требований, выбор инструментов и наглядное представление функциональности создают основу для разработки и гарантируют соответствие продукта целям и ожиданиям.

3 Проектирование программного средства

3.1 Архитектура проектируемого программного средства

Папка Data на рисунке 3.1 представляет собой слой доступа к данным в приложении, построенном с использованием паттерна Repository и Unit of Work.

1. Репозитории: классы, отвечающие за работу с конкретными типами данных (сущностями) в базе данных.
2. Unit of Work: класс, координирующий работу репозитория и обеспечивающий целостность данных.
3. UserSession: класс, хранящий информацию о текущем пользователе.

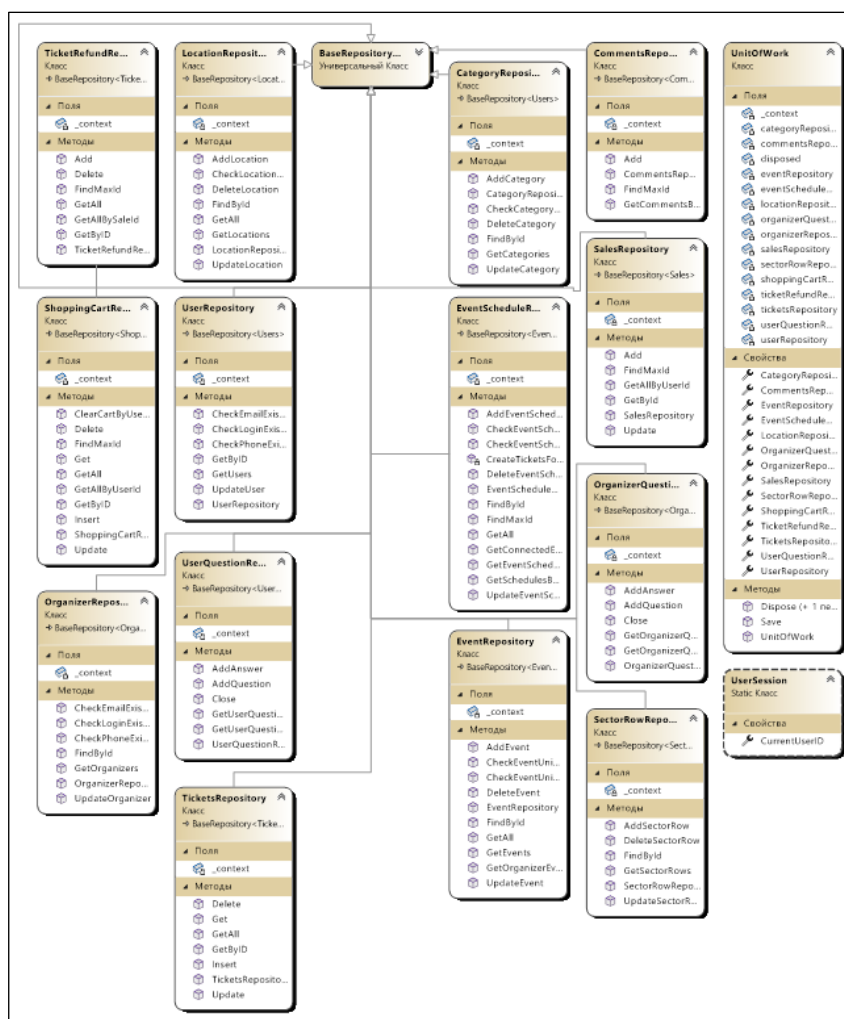


Рисунок 3.1 – UML диаграмма классов для папки Data

Папка Models, чья диаграмма представлена на рисунке 3.2, содержит классы, представляющие сущности предметной области проектируемого приложения. Эти классы описывают структуру данных, с которыми работает приложение. Папка Models является ядром модели данных приложения, определяющим его основные сущности и их взаимосвязи.

Выделяется интерфейс IQuestion, который демонстрирует использование полиморфизма для работы с различными типами вопросов. Благодаря этому

интерфейсу, в будущем можно работать с различными типами вопросов через одну коллекцию.



Рисунок 3.2 – UML диаграмма классов для папки Models

Папка Dictionaries визуализирована на рисунке 3.3 и предназначена для локализации приложения, то есть для перевода интерфейса на разные языки. В ней содержатся файлы ресурсов, которые представляют собой словари с переводами текстовых элементов интерфейса. Приложение использует эти файлы для динамической подстановки текста на нужном языке. Когда пользователь меняет язык интерфейса, приложение загружает соответствующий файл ресурсов и отображает текстовые элементы, используя значения из этого файла.

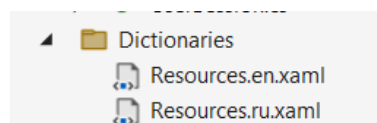


Рисунок 3.3 – Папка с переводами для локализации приложения

На рисунке 3.4. изображена папка Command, которая содержит только класс RelayCommand. Этот класс реализует интерфейс ICommand и предназначен для создания команд в MVVM (Model-View-ViewModel) архитектуре. Команды

инкапсулируют действия, которые могут быть выполнены в ответ на действия пользователя (например, нажатие кнопки).

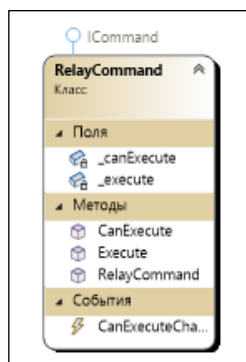


Рисунок 3.4 – UML диаграмма классов для папки Command

Папка Context содержит класс `ApplicationDbContext`, что видно из рисунка 3.5. Этот класс является контекстом базы данных и наследуется от `DbContext`. Он отвечает за взаимодействие с базой данных и управление сущностями.

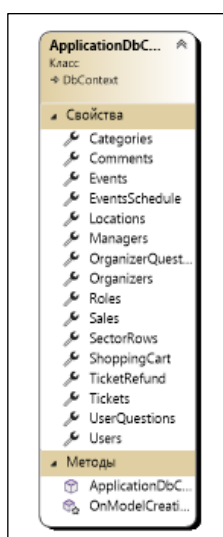


Рисунок 3.5 – UML диаграмма классов для папки Context

Папка `ViewModels`, представленная в приложении А, содержит классы, реализующие `ViewModel` в рамках паттерна MVVM (Model-View-ViewModel). Эти классы выступают в роли посредников между представлениями (`View`) и моделями (`Model`), подготавливая данные для отображения и обрабатывая команды пользователя. папка `ViewModels` является центральным элементом в управлении данными и логикой пользовательского интерфейса в приложениях, построенных с использованием паттерна MVVM. `ViewModels` обеспечивают чистое разделение ответственности между представлением данных и бизнес-логикой, упрощая разработку, тестирование и поддержку приложения.

Папка `Views`, продемонстрированная на рисунке 3.6, является важной частью архитектуры приложения, построенного на основе паттерна MVVM (Model-View-ViewModel). Она содержит классы, отвечающие за визуальное представление данных пользователю и обработку его взаимодействия с интерфейсом. Эти классы,

называемые «Представлениями» (Views), формируют пользовательский интерфейс приложения.

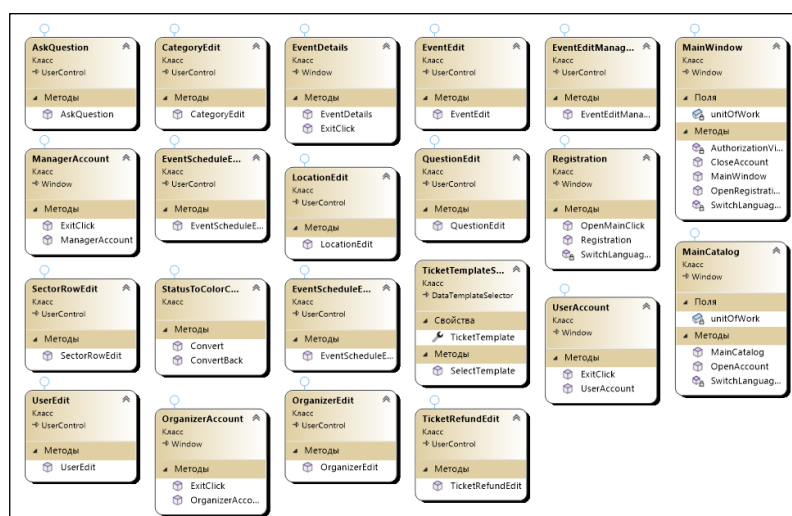


Рисунок 3.6 – UML диаграмма классов для папки Views

Папка Utilities содержит вспомогательные классы, как показано на рисунке 3.7, которые предоставляют различные конвертеры, используемые в разных частях приложения. Эти классы обычно не относятся к бизнес-логике, а реализуют общие механизмы, упрощающие разработку.

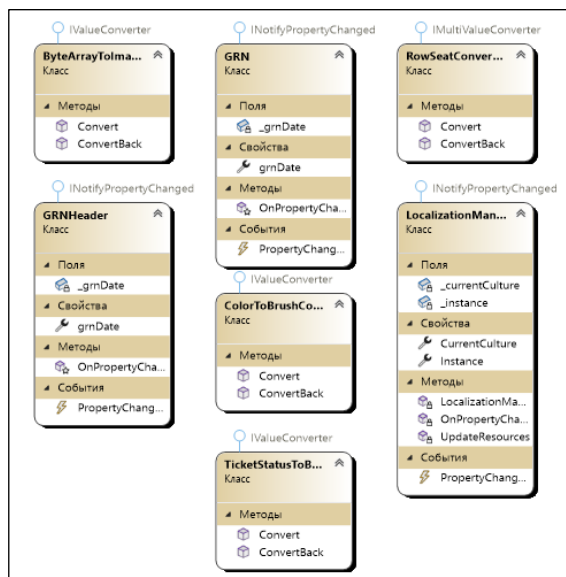


Рисунок 3.7 – UML диаграмма классов для папки Utilities

Такой подход обеспечивает разделение ответственности, слабую связанность компонентов и, как следствие, высокую расширяемость приложения.

3.2 Подключение к базе данных

Для работы с базой данных используется Entity Framework Core – объектно-реляционный преобразователь (ORM) с открытым исходным кодом, разработанный компанией Microsoft.

Строка подключения к базе данных продемонстрирована на листинге 3.1 и хранится в файле конфигурации App.config в секции connectionStrings:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="TicketSalesDb"
connectionString="Server=VODCHYTS;Database=CourseWork;Trusted_Connec
tion=True;TrustServerCertificate=True;"
        providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

Листинг 3.1 – Строка подключения

В данной строке подключения указаны следующие параметры:

1. Server: имя сервера, на котором расположена база данных.
2. Database: имя базы данных.
3. Trusted_Connection: указывает на использование аутентификации Windows.
4. TrustServerCertificate: указывает на доверие к сертификату сервера.
5. providerName: имя провайдера данных для подключения к SQL Server.

Для получения строки подключения используется класс ConfigurationManager из пространства имен System.Configuration.

Инициализация контекста базы данных ApplicationDbContext и управление репозиториями осуществляется в классе UnitOfWork, что продемонстрировано на листинге 3.2.

```
public UnitOfWork()
{
    string connectionString =
ConfigurationManager.ConnectionStrings["TicketSalesDb"].ConnectionSt
ring;
    var optionsBuilder = new
DbContextOptionsBuilder<ApplicationDbContext>();
    optionsBuilder.UseSqlServer(connectionString);
    _context = new ApplicationDbContext(optionsBuilder.Options);
}
```

Листинг 3.2 – Подключение к базе данных

В конструкторе UnitOfWork происходит следующее:

1. Из файла конфигурации считывается строка подключения к базе данных с именем TicketSalesDb.
2. Создается объект DbContextOptionsBuilder для настройки параметров подключения.
3. С помощью метода UseSqlServer указывается, что в качестве СУБД используется SQL Server, и передается строка подключения.
4. Создается экземпляр ApplicationDbContext с использованием настроенных параметров.

Таким образом, подключение к базе данных в данном приложении осуществляется с использованием Entity Framework Core, строка подключения хранится в файле конфигурации, а управление подключением и репозиториями реализовано в классе UnitOfWork.

3.3 Разработка базы данных

Диаграмма, представленная в Приложении Г, иллюстрирует схему базы данных, разработанную для программного средства. Схема включает в себя 16 взаимосвязанных сущностей:

1. UserQuestions: вопросы пользователей.
2. Organizers: букеры.
3. OrganizerQuestions: вопросы букеров.
4. Categories: категории кинофильмов.
5. Events: кинофильмы.
6. TicketRefund: возврат билетов.
7. Tickets: билеты.
8. Users: пользователи.
9. Sales: продажи билетов.
- 10.EventSchedule: расписание кинофильмов.
- 11.Locations: места проведения.
- 12.Roles: роли пользователей.
- 13.SectorRows: ряды секторов в местах проведения.
- 14.Managers: менеджеры.
- 15.ShoppingCart: корзина покупок.
- 16.Comments: комментарии.

В Entity Framework Core (EF Core) широко применяется подход Code First. Это методология разработки, при которой первоначально создаются классы на языке C# (или другом .NET языке), представляющие собой модели данных приложения, а затем, на их основе, EF Core автоматически генерирует схему и структуру базы данных. Этот подход противоположен подходу Database First, где сначала проектируется база данных, а затем на ее основе генерируются классы-модели.

Каждый класс, который хранится в базе данных, называется сущностью. Свойства сущности соответствуют столбцам в таблице базы данных. Каждая сущность должна иметь первичный ключ, который уникально идентифицирует каждую запись в таблице. По умолчанию, EF Core ищет свойство с именем «Id» или «[ИмяКласса]Id».

EF Core использует набор соглашений для автоматического сопоставления классов и свойств с таблицами и столбцами в базе данных. Классы обычно сопоставляются с таблицами во множественном числе. Свойства сопоставляются со столбцами с тем же именем.

Можно изменять поведение EF Core, используя атрибуты данных или Fluent API. Атрибуты добавляются непосредственно к классам и свойствам, а Fluent API используется в методе OnModelCreating класса DbContext для более тонкой настройки.

Навигационные свойства – это свойства сущности, которые ссылаются на другие связанные сущности. Они не хранят данные напрямую, а предоставляют способ доступа к связанным объектам. Навигационные свойства обычно связаны с внешними ключами. Внешний ключ – это свойство в одной сущности, которое ссылается на первичный ключ другой сущности. EF Core может автоматически определить внешние ключи на основе соглашений об именовании.

Рассмотрим, как пример класс `Comments`, представленный на листинге 3.3.

```
public class Comments
{
    [Key]
    public int CommentID { get; set; }
    public int UserID { get; set; }
    public int EventID { get; set; }
    public required string CommentText { get; set; }
    public DateTime CommentDate { get; set; }
    public int FivePointRating { get; set; }
    public Users User { get; set; }
    public Events Event { get; set; }
}
```

Листинг 3.3 – Пример определения сущности

`Comments` – это сущность, которая будет сопоставлена с таблицей `Comments` в базе данных. Свойства:

1. `CommentID` – первичный ключ (определено атрибутом `[Key]`).
2. `UserID` – внешний ключ, ссылающийся на таблицу `Users`.
3. `EventID` – внешний ключ, ссылающийся на таблицу `Events`.
4. `CommentText` – текст комментария.
5. `CommentDate` – дата и время создания комментария.
6. `FivePointRating` – оценка по пятибалльной шкале.

`User` – это навигационное свойство, представляющее связь «многие-к-одному» с сущностью `Users` и позволяет получить доступ к объекту `Users`, связанному с данным комментарием. Свойство `Event` аналогично является навигационным.

В EF Core связи между сущностями определяют, как объекты соотносятся друг с другом.

`DbContext` – это центральный класс в EF Core, представляющий сеанс работы с базой данных. Внутри метода `OnModelCreating` разработчик получает доступ к объекту `ModelBuilder`. Этот объект предоставляет Fluent API. Рассмотрим пример связи на листинге 3.3.

```
modelBuilder.Entity<Comments>()
    .HasOne(c => c.User)
    .WithMany(u => u.Comments)
    .HasForeignKey(c => c.UserID);
```

Листинг 3.3 – Пример связи между сущностями

Определяется связь между сущностями `Comments` и `Users`. Эта связь описывается как «один-ко-многим» со стороны сущности `Users` к

сущности Comments. В терминах реляционной модели это означает, что одному пользователю может соответствовать несколько записей комментариев, в то время как каждый комментарий связан только с одним пользователем.

3.4 Проектирование диаграммы последовательностей

Диаграмма последовательностей UML – это мощный инструмент для моделирования взаимодействия между различными частями системы. Она наглядно отображает, как объекты обмениваются сообщениями во времени, демонстрируя порядок выполнения операций. В отличие от других диаграмм UML, которые фокусируются на структуре системы, диаграмма последовательностей делает акцент на динамическом поведении. Это делает ее незаменимой при проектировании, документировании и анализе сложных систем, где важно понимать последовательность действий и взаимосвязь между компонентами.

В рассматриваемой в Приложении В диаграмме последовательностей ключевыми элементами выступают линии жизни и сообщения, которые в совокупности формируют наглядное представление о динамике взаимодействия между участниками системы.

Линии жизни изображаются вертикальными пунктирными линиями и представляют собой временную ось существования каждого участника. В данном случае, представлены три линии жизни (клиент, приложение и база данных).

Сообщения отображаются горизонтальными стрелками, соединяющими линии жизни. Они показывают передачу данных или управляющих сигналов между участниками. Направление стрелки указывает на инициатора и получателя сообщения.

Клиент взаимодействует с приложением, чтобы получить информацию о кинофильмах и купить билеты. Приложение, в свою очередь, выступает посредником между клиентом и базой данных. Когда клиенту нужен список фильмов, приложение запрашивает его у базы данных и отображает клиенту. Если клиент хочет увидеть подробную информацию о фильме, приложение снова обращается к базе данных, получает нужные сведения и показывает их клиенту на отдельной странице.

3.5 Выводы по разделу

В результате проектирования была разработана архитектура программного средства, основанная на современных паттернах и принципах разработки, что обеспечивает её гибкость, расширяемость и простоту сопровождения. Детально описаны механизмы взаимодействия с базой данных с использованием Entity Framework Core, а также спроектирована структура базы данных, отвечающая требованиям предметной области.

4 Реализация программного средства

Опираясь на концептуальные решения, принятые на предыдущих этапах проектирования и анализа, происходит переход к практическому воплощению этих идей в виде работающего программного продукта.

Целью является предоставление полного описания процесса реализации программного средства, позволяющего понять, как из теоретических концепций и требований, сформулированных на ранних этапах, был создан конечный продукт. По итогам необходимо получить четкое представление о внутренней организации приложения, принципах его работы и технологиях, использованных при его создании.

4.1 Реализация паттерна Singleton

Singleton (Одиночка) – это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру.

LocalizationManager – это класс, отвечающий за управление локализацией (переводом) приложения на разные языки.

- свойство CurrentCulture хранит информацию о том, какой язык сейчас используется в приложении;

- метод UpdateResources ищет XAML-файлы ресурсов в папке Dictionaries с именами вида «Resources.язык.xaml»;

- событие PropertyChanged уведомляет другие части приложения об изменении значения CurrentCulture. Это позволяет автоматически обновить интерфейс при смене языка.

Имея один объект LocalizationManager, проще управлять локализацией. Не нужно передавать экземпляр LocalizationManager в каждый класс, которому нужна информация о текущем языке.

Приватное статическое поле «_instance» хранит единственный экземпляр класса LocalizationManager, как продемонстрировано на листинге 4.1. Ключевое слово static означает, что поле принадлежит классу, а не конкретному объекту.

Публичное статическое свойство Instance предоставляет глобальную точку доступа к единственному экземпляру класса.

LocalizationManager использует null-coalescing assignment. Если экземпляр еще не создан, то создается новый объект LocalizationManager и присваивается «_instance».

```
private static LocalizationManager? _instance;
public static LocalizationManager Instance => _instance ??= new
LocalizationManager();
```

Листинг 4.1 – Отложенная реализация Singleton

Конструктор класса объявлен как private. Это предотвращает создание экземпляров класса извне, используя оператор new.

В данном классе реализован классический Singleton с отложенной инициализацией (Lazy Initialization). Экземпляр класса создается только при первом обращении к свойству Instance.

4.2 Использование конвертеров

Converter (конвертер) – это специальный класс, который преобразует данные из одного типа в другой.

Основное назначение конвертеров:

- адаптация данных для отображения;
- конвертеры также могут выполнять обратное преобразование данных;
- конвертеры могут форматировать данные для отображения, например, преобразовывать дату в определенный строковый формат или число в денежный формат.

Конвертер реализует IValueConverter, который содержит два метода:

1. Convert преобразует значение из исходного типа в целевой тип.
2. ConvertBack выполняет обратное преобразование из целевого типа в исходный тип.

Пример создания конвертера можно увидеть на листинге 4.2.

```
public class ByteArrayToImageConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object
parameter, CultureInfo culture)
    {
        if (value is byte[] bytes && bytes.Length > 0)
        {
            using (var stream = new MemoryStream(bytes))
            {
                var bitmap = new BitmapImage();
                bitmap.BeginInit();
                bitmap.CacheOption = BitmapCacheOption.OnLoad;
                bitmap.StreamSource = stream;
                bitmap.EndInit();
                return bitmap;
            }
        }
        return null;
    }
    public object ConvertBack(object value, Type targetType, object
parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Листинг 4.2 – Пример создания конвертера

ByteArrayToImageConverter – это простой, но полезный конвертер, который позволяет отображать изображения, хранящиеся в виде массива байтов, в WPF-

приложении. Он использует `MemoryStream` и `BitmapImage` для преобразования данных из одного формата в другой. Обратное преобразование в данном примере не реализовано. Конвертеры обычно объявляются как ресурсы в XAML и затем используются в привязках данных (`Data Binding`).

Пример на листинге 4.3. демонстрирует правильное использование конвертера `ByteArrayToImageConverter` для отображения изображения, хранящегося в виде массива байтов в `ViewModel`. Обнаружив, что для привязки указан конвертер, WPF находит конвертер `ByteArrayToImageConverter` по ключу `ByteArrayToImageConverter` в ресурсах. WPF вызывает метод `Convert` конвертера `ByteArrayToImageConverter`, передавая ему значение свойства `EventDetails.Event.Image` (массив байтов) в качестве параметра `value`. В конце конвертер преобразует массив байтов в объект `BitmapImage`.

```
<Window.Resources>
<utilities:ByteArrayToImageConverter
x:Key="ByteArrayToImageConverter" />
</Window.Resources>
//...
<Image Source="{Binding EventDetails.Event.Image,
Converter={StaticResource ByteArrayToImageConverter}}" Height="250"
Margin="0,10" />
```

Листинг 4.3 – Пример использования конвертера

Кроме этого, в приложении используются конвертеры для окрашивания места в плане зала кинотеатра, форматирования строки для подсказки мест в плане зала кинотеатра, управления видимостью свободных мест.

4.3 Паттерн Repository

Паттерн `Repository` (Репозиторий) – это слой абстракции между уровнем доступа к данным (например, базой данных) и бизнес-логикой приложения. Он предоставляет унифицированный интерфейс для работы с данными, скрывая детали реализации доступа к данным от остальной части приложения.

- репозиторий инкапсулирует всю логику, связанную с получением и сохранением данных;
- поскольку репозиторий является абстракцией, его легко заменить заглушкой при модульном тестировании;
- централизованное управление данными;
- репозиторий скрывает детали реализации доступа к данным. Это позволяет изменить способ хранения данных без необходимости изменения бизнес-логики.

В данном приложении созданы репозитории для работы со всеми существующими сущностями. Рассмотрим пример репозитория `CategoryRepository` для управления сущностью каталога на Листинге Д.

Прежде всего, `CategoryRepository` хранит в себе ссылку на контекст базы данных. Этот контекст, предоставляемый `Entity Framework Core`, является ключом к взаимодействию с базой данных.

CategoryRepository предоставляет набор методов, каждый из которых отвечает за определенную операцию с категориями:

1. **GetCategories**. Этот метод извлекает из базы данных все существующие категории и возвращает их в виде удобной коллекции **ObservableCollection**. Эта коллекция особенно полезна в WPF-приложениях, так как автоматически обновляет пользовательский интерфейс при любых изменениях в своем содержимом.

2. **FindById**. Данный метод позволяет найти категорию по ее уникальному идентификатору. Он обращается к базе данных, ищет запись с указанным **CategoryID** и возвращает соответствующий объект **Categories**.

3. **UpdateCategory**. Этот метод предназначен для обновления информации о категории. Он находит категорию по **id**, изменяет её **CategoryName** на указанное в параметрах **name** и сохраняет изменения в базе данных.

4. **AddCategory**. Этот метод отвечает за добавление новой категории в базу данных. Он проверяет уникальность **name**. Если имя уникально, то он находит максимальное значение **CategoryID** в базе данных, добавляет к нему единицу, чтобы сгенерировать новый уникальный идентификатор. Затем создает новый объект **Categories**, добавляет его в контекст и сохраняет изменения в базе данных, возвращая созданный объект.

5. **DeleteCategory**. Этот метод удаляет категорию из базы данных по ее идентификатору. Он находит соответствующую запись и помечает ее для удаления, а затем сохраняет изменения в базе, физически удаляя запись.

6. **CheckCategoryUnique**. Этот вспомогательный метод проверяет, существует ли уже категория с указанным именем в базе данных. Используется, чтобы каждая категория имела уникальное имя.

Кроме **CategoryRepository** в приложении реализовано ещё 14 репозитория, один из которых – **BaseRepository** – репозиторий с общими для некоторых сущностей методами.

4.4 Паттерн UnitOfWork

Unit of Work (UoW) – это паттерн проектирования, который выступает в роли посредника между бизнес-логикой приложения и базой данных. Он управляет транзакциями и отслеживает изменения объектов в памяти, гарантируя, что все изменения будут сохранены в базе данных как единая атомарная операция.

Часто UoW используется вместе с паттерном **Repository**. UoW предоставляет репозиториям доступ к контексту базы данных и управляет сохранением изменений, сделанных через репозитории. Частичный код класса представлен на листинге в приложении Е.

В классе **UnitOfWork** код начинается с объявлений приватных полей для хранения экземпляров репозитория. Каждый репозиторий отвечает за работу с определенным типом сущностей.

Далее располагаются публичные свойства, предоставляющие доступ к репозиториям. Это гарантирует, что экземпляр репозитория будет создан только при первом обращении к нему, а не при создании объекта **UnitOfWork**. Все репозитории получают текущий **_context** при создании.

В конструкторе считывается строка подключения к базе данных из файла конфигурации.

Создается строитель опций `DbContextOptionsBuilder` и настраивается для использования SQL Server с указанной строкой подключения.

Создается экземпляр `ApplicationDbContext` с настроенными опциями и сохраняется в поле `_context`.

Метод `Save` сохраняет все изменения, отслеживаемые контекстом `_context`, в базе данных.

Методы `Dispose(bool disposing)` и `Dispose()` реализуют паттерн `Disposable` для освобождения ресурсов, занимаемых объектом `UnitOfWork`, в частности, контекстом базы данных `_context`.

`UnitOfWork` изолирует бизнес-логику от деталей работы с базой данных и упрощает тестирование приложения.

4.5 Реализация переключения языков

В WPF-приложении реализован удобный механизм переключения языков интерфейса, позволяющий пользователю легко выбирать между русской и английской локализациями. Сердцем этого механизма является класс `LocalizationManager`, спроектированный как `Singleton`. Реализация данного паттерна в `LocalizationManager` уже рассматривалась в главе 4.1.

`LocalizationManager` хранит в себе текущую культуру, представленную объектом `CultureInfo`, в свойстве `CurrentCulture`. Когда это свойство изменяется, `LocalizationManager` инициирует процесс обновления ресурсов приложения, вызывая приватный метод `UpdateResources`.

Метод `UpdateResources`, представленный на листинге 4.4 – это ключевое звено в механизме смены языка. Он конструирует новый словарь ресурсов `ResourceDictionary`, динамически формируя путь к файлу ресурсов на основе текущего значения `CurrentCulture`. Путь этот строится по шаблону: «`Dictionaries/Resources.{currentCulture.Name}.xaml`». Затем `LocalizationManager` очищает текущую коллекцию словарей ресурсов и добавляет в нее новый.

```
private void UpdateResources()
{
    var dictionary = new ResourceDictionary
    {
        Source = new
Uri($"Dictionaries/Resources.{_currentCulture.Name}.xaml",
UriKind.Relative)
    };
    Application.Current.Resources.MergedDictionaries.Clear();
    Application.Current.Resources.MergedDictionaries.Add(dictionary);
}
```

Листинг 4.4 – Обновление языка через метод `UpdateResources`

За инициирование процесса смены языка отвечает метод `SwitchLanguage_Click`, продемонстрированный на листинге 4.5, который

привязан к событию нажатия на кнопку в интерфейсе. Этот метод обращается к единственному экземпляру `LocalizationManager` через его статическое свойство `Instance`, определяет текущую культуру и меняет её на противоположную: с «ru» на «en» или наоборот.

```
private void SwitchLanguage_Click(object sender, RoutedEventArgs e)
{
    var currentCulture =
LocalizationManager.Instance.CurrentCulture.Name;
    if (currentCulture == "ru")
    {
        LocalizationManager.Instance.CurrentCulture = new
CultureInfo("en");
    }
    else
    {
        LocalizationManager.Instance.CurrentCulture = new
CultureInfo("ru");
    }
}
```

Листинг 4.5 – Инициирование процесса смены языка через метод `UpdateResources`

Сами же локализованные строки хранятся в файлах-словарях ресурсов, имеющих формат XAML. Каждый такой файл содержит набор пар «ключ-значение», где ключ – это уникальный идентификатор строки, а значение – её перевод на соответствующий язык.

Чтобы отобразить локализованную строку в интерфейсе, в XAML-разметке используется специальная конструкция привязки данных – `DynamicResource`.

4.6 Реализация паттерна MVVM

Model-View-ViewModel (MVVM) – это архитектурный паттерн, разделяющий приложение на три взаимодействующих компонента: Модель (Model), Представление (View) и Модель представления (ViewModel).

4.6.1 Модель

Модель – это фундамент приложения, содержащий данные и бизнес-логику. По сути, это представление предметной области в коде. Модель полностью автономна и независима от пользовательского интерфейса. Она не имеет представления о том, каким образом и где данные будут отображаться. Её главная задача – управление данными, обеспечение их целостности, корректности и актуальности.

- хранение данных;
- модель содержит логику, определяющую, как данные могут быть созданы, изменены и использованы. Сюда входят правила валидации данных, вычисления, операции над данными;

– взаимодействие с источниками данных (например, база данных, веб-сервис).

Рассмотрим в качестве примера класс `Categories` в листинге 4.6, который является частью Модели.

```
public class Categories
{
    [Key]
    public int CategoryID { get; set; }
    public required string CategoryName { get; set; }
    public ICollection<Events> Events { get; set; }
}
```

Листинг 4.6 – Модель `Categories`

Этот класс представляет сущность «Категория» и хранит её свойства: уникальный идентификатор и название. Свойство `Events` типа `ICollection` указывает на связь между категориями и кинофильмами, предполагая, что одна категория может содержать множество кинофильмов.

4.6.2 Представление

`View` – это лицо приложения, то, что видит пользователь. `View` отвечает за отображение данных из `ViewModel` и обработку пользовательского ввода.

В качестве примера рассмотрим файл `CategoryEdit.xaml`, часть которого представлена на листинге 4.7.

```
<UserControl x:Class="CouseWork.ViewModels.CategoryEdit" ...>
    ...
    <UserControl.DataContext>
        <viewModel:CategoryEditViewModel/>
    </UserControl.DataContext>
    <Grid ...>
        <DataGrid ItemsSource="{Binding Categories}"
                    SelectedItem="{Binding SelectedCategory}" ...>
            <DataGrid.Columns>
                <DataGridTextColumn Header="ID" Binding="{Binding
CategoryID}" />
                <DataGridTextColumn Header="Name" Binding="{Binding
CategoryName}" />
            </DataGrid.Columns>
        </DataGrid>
        <StackPanel ...>
            <TextBox
                Text="{Binding
UpdateSourceTrigger=PropertyChanged}" ... />
        </StackPanel>
    </Grid>
</UserControl>
```

Листинг 4.7 – View `CategoryEdit`

`UserControl.DataContext` создает новый экземпляр `CategoryEditViewModel` и устанавливает его как `DataContext` для всего `UserControl`. Теперь все элементы

управления внутри UserControl знают, где искать данные и команды – в CategoryEditViewModel.

Центральное место в отображении занимает DataGrid, который отображает список категорий. ItemsSource связывается со свойством Categories в CategoryEditViewModel. Categories – это коллекция объектов Categories, представляющих категории. DataGrid автоматически отображает каждую категорию в отдельной строке. Внутри DataGrid определены колонки «DataGrid.Columns», каждая из которых привязана к определенному свойству объекта Categories.

SelectedItem связывается со свойством SelectedCategory в CategoryEditViewModel. Когда пользователь выбирает строку в DataGrid, SelectedCategory в ViewModel обновляется.

View не должна напрямую обращаться к Модели или базе данных. Она получает данные и отправляет команды только через ViewModel.

4.6.3 Модель представления

ViewModel – это связующее звено между View и Model. Это абстракция View, предоставляющая ей данные и команды в удобном формате. ViewModel не зависит от конкретного View и может использоваться с разными View. Рассмотрим на примере содержимого класса CategoryEditViewModel основные обязанности ViewModel с листинга 4.8.

```
private ObservableCollection<Categories> _categories;
public ObservableCollection<Categories> Categories
{
    get => _categories;
    set {_categories = value; OnPropertyChanged(nameof(Categories));}
}
public class CategoryEditViewModel()
{
    unitOfWork = new UnitOfWork();
    Categories = unitOfWork.CategoryRepository.GetCategories(); ...
}
private bool ValidateFieldsUpdate()
{
    bool isValid = true;
    if (string.IsNullOrEmpty(Id))
    {IdError = "Выберите категорию"; isValid = false;}
    return isValid;
}
```

Листинг 4.8 – CategoryEditViewModel

При создании экземпляра CategoryEditViewModel происходит его инициализация. В конструкторе создается объект UnitOfWork, предоставляющий доступ к репозиториям, в том числе и к CategoryRepository. С помощью последнего из базы данных извлекается список всех категорий и сохраняется в свойстве Categories типа ObservableCollection.

Свойство `Categories` имеет тип `ObservableCollection`. Эта коллекция автоматически уведомляет `View` об изменениях, что приводит к обновлению `DataGrid`.

Методы `ValidateFieldsUpdate`, `ValidateFieldsAdd` и `ValidateFieldsDelete` проверяют корректность данных перед выполнением операций обновления, добавления и удаления соответственно. Результаты валидации сохраняются в свойствах ошибок, которые затем отображаются в представлении.

`CategoryEditViewModel` реализует интерфейс `INotifyPropertyChanged`, что показано на листинге 4.9 и позволяет ему уведомлять представление об изменениях в своих свойствах.

```
public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName) =>
    PropertyChanged?.Invoke(this, new
    PropertyChangedEventArgs(propertyName));
```

Листинг 4.9 – Реализация `INotifyPropertyChanged`

При изменении любого свойства, предназначенного для отображения в представлении, вызывается метод `OnPropertyChanged`, который, в свою очередь, инициирует событие `PropertyChanged`. Представление, подписанное на это событие, автоматически обновляет соответствующие элементы интерфейса.

4.7 Реализация паттерна `ICommand`

Паттерн `ICommand` (Команда) инкапсулирует запрос в виде объекта. Этот объект содержит всю необходимую информацию для выполнения действия.

В приложении реализуется механизм обработки действия пользователя, а именно нажатия на кнопку обновления ответа в пользовательском интерфейсе, что иллюстрируется на листинге 4.10.

```
public ICommand AnswerCommand { get; }
public QuestionEditViewModel() { AnswerCommand = new
    RelayCommand(AddAnswer, CanChangeQuestion); }
private void AddAnswer(object parameter)
{
    _unitOfWork.OrganizerQuestionRepository.AddAnswer(int.Parse(Question
    ID), AnswerText);
    RefreshAnswers();
}
private bool CanChangeQuestion(object parameter)
{ return !string.IsNullOrEmpty(AnswerText) && Status == "In
    processing"; }
```

Листинг 4.10 – Реализация `ICommand`

В классе определено свойство `AnswerCommand` типа `ICommand`. Это свойство представляет собой команду, которая будет связана с кнопкой в пользовательском интерфейсе.

Инициализация команды `AnswerCommand` происходит в конструкторе `QuestionEditViewModel`. Для этого используется вспомогательный класс `RelayCommand`, реализующий интерфейс `ICommand`. В конструктор `RelayCommand` передаются два делегата: `AddAnswer` и `CanChangeQuestion`.

Делегат `AddAnswer` представляет собой метод, содержащий основную логику, которая должна выполняться при вызове команды, то есть при нажатии пользователем на кнопку.

Делегат `CanChangeQuestion` представляет собой метод, определяющий, доступна ли команда для выполнения в текущий момент. В данном случае метод `CanChangeQuestion` проверяет, что текст ответа не является пустым и статус вопроса равен «In processing». Только при выполнении обоих этих условий команда считается доступной для выполнения.

В коде XAML, определяющем разметку пользовательского интерфейса, кнопка `Button` с помощью свойства `Command` привязывается к команде `AnswerCommand`. Это означает, что при нажатии на эту кнопку будет инициирован вызов команды.

В итоге, при взаимодействии пользователя с кнопкой будет вызвана команда `AnswerCommand`. Сначала будет проверено, доступна ли команда для выполнения, путем вызова метода `CanChangeQuestion`. Если команда доступна, то будет выполнен метод `AddAnswer`, который добавит ответ в базу данных и обновит отображение. Если же команда недоступна (например, текст ответа пустой), то кнопка будет визуально неактивна, и действие не будет выполнено.

4.8 Выводы по разделу

В данном разделе представлена реализация программного средства, построенного на основе современных архитектурных решений и паттернов проектирования. Для обеспечения единой точки доступа к языковым настройкам и ресурсам, а также для управления локализацией, был применен паттерн `Singleton` в классе `LocalizationManager`. Для преобразования и форматирования данных между Моделью/ViewModel и Представлением используются конвертеры, реализующие интерфейс `IValueConverter`, например, `ByteArrayToImageConverter` для отображения изображений из массива байтов. Абстрагирование логики доступа к данным от бизнес-логики достигается с помощью паттерна `Repository`, как показано на примере `CategoryRepository`. Паттерн `Unit of Work` обеспечивает управление транзакциями и консистентность данных, предоставляя доступ к репозиториям и управляя сохранением изменений в базе данных. Реализован удобный механизм переключения языков на основе `LocalizationManager`, `CultureInfo`, динамической загрузки словарей ресурсов (`ResourceDictionary`) и привязки данных `DynamicResource`. Все приложение построено с использованием паттерна MVVM, разделяющего Модель (данные и бизнес-логика), Представление (пользовательский интерфейс) и Модель представления (связующее звено), что обеспечивает слабую связанность компонентов и упрощает тестирование. Для инкапсуляции действий пользователя, таких как нажатие кнопки, используется паттерн `ICommand`, реализованный в классе `RelayCommand` и продемонстрированный на примере команды `AnswerCommand`.

5 Тестирование, проверка работоспособности и анализ полученных результатов

5.1 Обработка ошибок

При вводе данных в форму авторизации, приложение требует заполнения всех полей и выводит соответствующие ошибки, если этого не сделать, что продемонстрировано на рисунке 5.1.

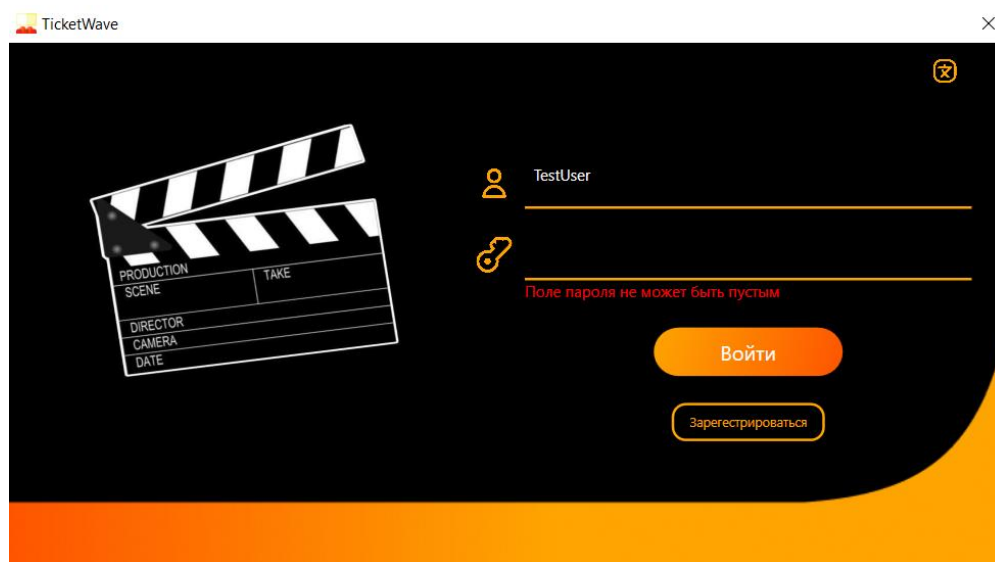


Рисунок 5.1 – Ошибки ввода при авторизации

На скриншоте показан результат попытки авторизации с неверными учетными данными. Приложение корректно отреагировало на ввод неверного имени пользователя или пароля, отобразив соответствующее сообщение об ошибке, как показано на рисунке 5.2.

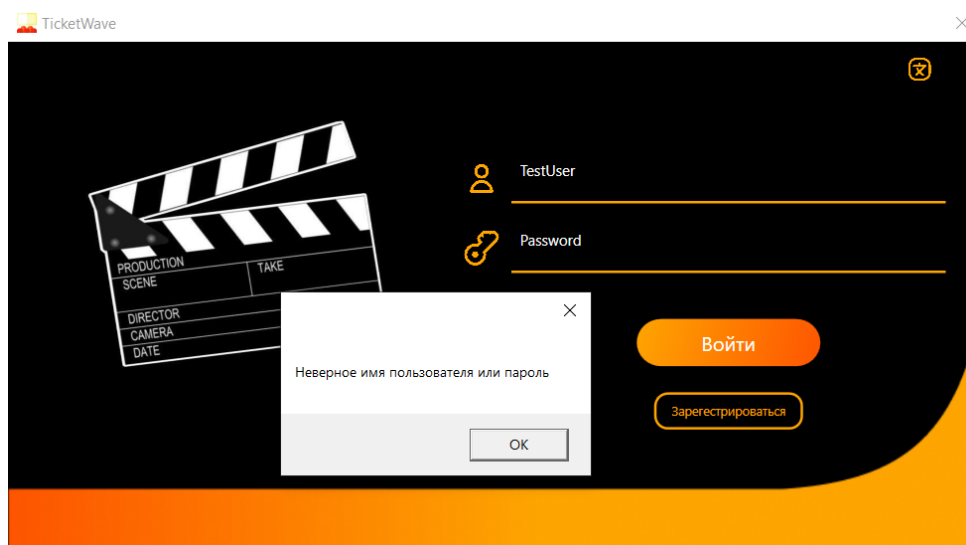


Рисунок 5.2 – Авторизация с неверными учетными данными

На рисунке 5.3 представлена форма регистрации. Форма содержит ряд полей для ввода данных пользователя: Логин, Пароль, Имя, Фамилия, Почта и Телефон.

Под каждым полем размещены подсказки о неправильности введенных данных. Форма реализует механизм валидации введенных данных. Об этом свидетельствуют сообщения об ошибках, отображаемые под полями. Текущее состояние формы говорит о том, что валидация работает корректно, указывая на занятость логина и email, некорректную длину пароля и телефона, а также на необходимость заполнения полей «Имя» и «Фамилия».

The screenshot shows a registration form for 'TicketWave'. The form fields and their validation messages are as follows:

- Логин:** TestUser. Message: Этот логин занят.
- Пароль:** Test. Message: Длина от 8 (заглавные, цифры, символы).
- Имя:** (empty). Message: Поле имени не может быть пустым.
- Фамилия:** (empty). Message: Поле фамилии не может быть пустым.
- Почта:** TestUser@TestUser. Message: Этот email занят.
- Телефон:** 234. Message: Телефон должен содержать 10 - 15 цифр.

Buttons on the right: Зарегистрироваться, Перейти к авторизации, and Регистрация букера.

Рисунок 5.3 – Регистрация пользователя с не валидными данными

На рисунке 5.4 можно наблюдать, что после регистрации пользователей их пароли хранятся в хешированном виде. Это видно по столбцу «Password», где вместо явного пароля находится длинная последовательность символов. Такой подход обеспечивает безопасность данных, так как в случае утечки базы данных злоумышленники не смогут получить доступ к реальным паролям пользователей. Хеширование паролей является стандартной практикой в разработке приложений для защиты конфиденциальной информации.

Результаты		Сообщения						
	UserID	Login	Password	FirstName	LastName	Email	Phone	RoleID
1	15	TestUser	6c96d432989ec7bca1c21cd532bad3e11b81c20a8ccb5e49...	TestUser	TestUser	TestUser@TestUser	802917685394	1

Рисунок 5.4 – Хеширование пароля

На рисунке 5.5 представлена страница каталога, являющаяся важной частью пользовательского интерфейса приложения. Данная страница предоставляет пользователю удобные инструменты для навигации и поиска интересующих его кинофильмов. Интерфейс страницы интуитивно понятен и позволяет пользователю фильтровать каталог кинофильмов по различным параметрам, что существенно облегчает процесс выбора.

Рассмотрим подробнее доступные фильтры. Пользователь может выбрать интересующий его жанр, сузив тем самым область поиска до определенной категории фильмов. Также доступен фильтр по сеансам, позволяющий отобрать фильмы, демонстрируемые в удобное для пользователя время. Имеется возможность

фильтрации по кинотеатрам и по дате проведения, которую можно отключить. Кроме этого, присутствует сортировка по дате проведения и цене, как по убыванию, так и по возрастанию.

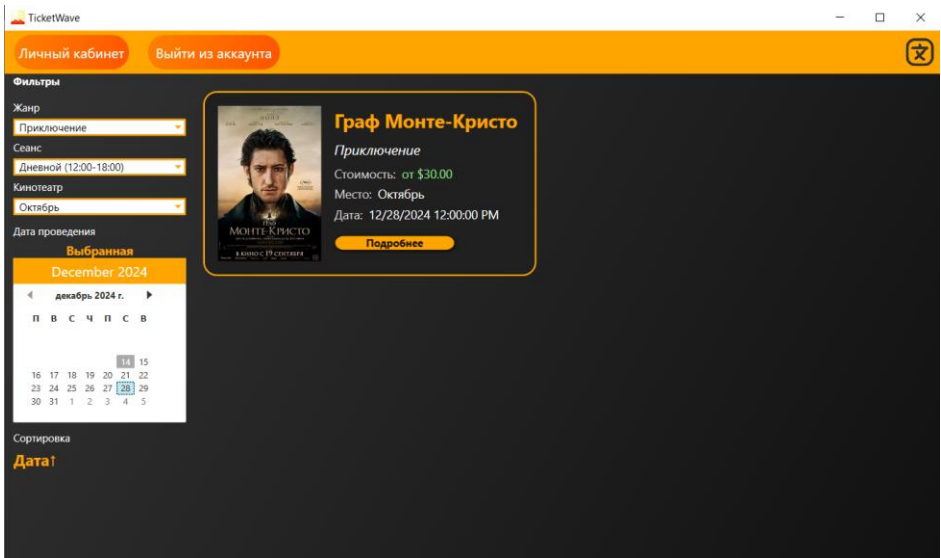


Рисунок 5.5 – Фильтрация каталога

Ошибка при создании кинотеатра, продемонстрированная на рисунке 5.6, заключается в том, что кинотеатр с названием «Moon» уже существует. Это видно из сообщения, которое отображается красным цветом справа от поля ввода названия. Чтобы исправить ошибку, пользователю необходимо ввести уникальное название локации в поле «Название».

Также если в поле «Количество рядов» ввести некорректное количество, система может выдать соответствующую ошибку.

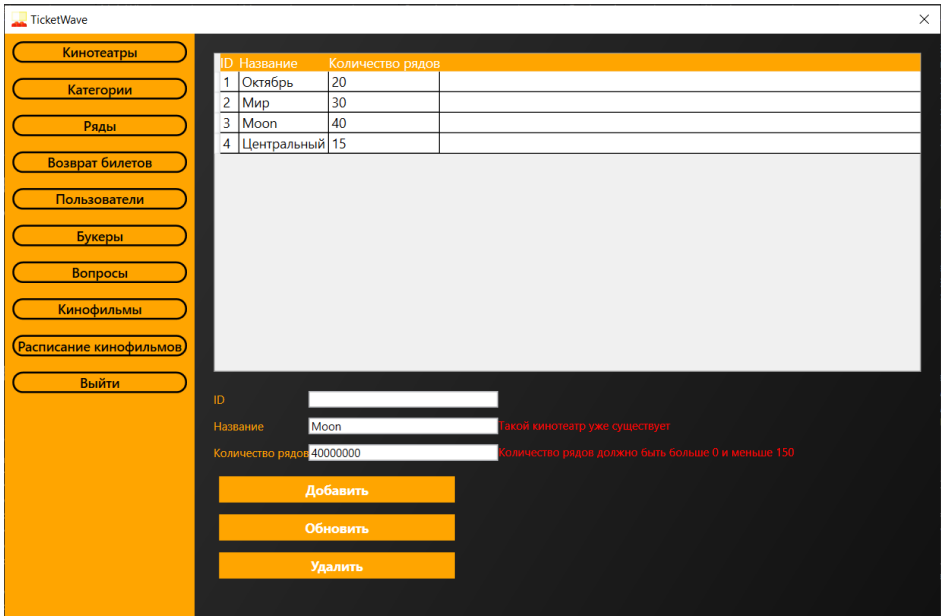


Рисунок 5.6 – Ошибка при создании кинотеатра

Ошибка при попытке удалить категорию «Приключение», продемонстрированная на рисунке 5.7, заключается в том, что эта категория в

данный момент используется в системе (связана с активными билетами). Система не позволяет удалить категорию, так как это нарушит целостность данных.

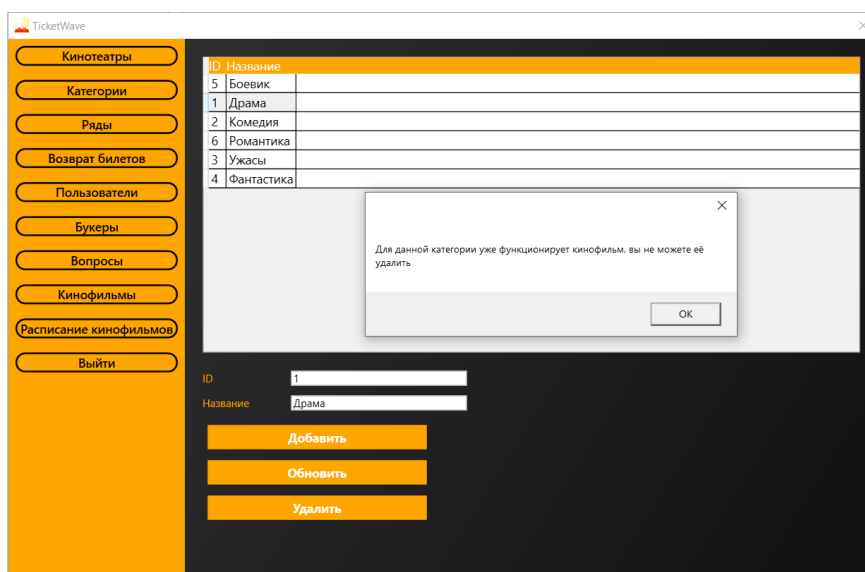


Рисунок 5.7 – Ошибка при удалении категории

Валидация данных при создании кинофильма является важным аспектом, обеспечивающим корректность и полноту информации, и продемонстрирована на рисунке 5.8. Система проверяет длительность, цену, дату, не допуская отрицательных значений, выхода за пределы допустимых диапазонов. Календарь для выбора даты начала показа настроен отображаться с текущего дня и на год вперёд, также система запретит создавать кинофильмы с одинаковыми названиями.

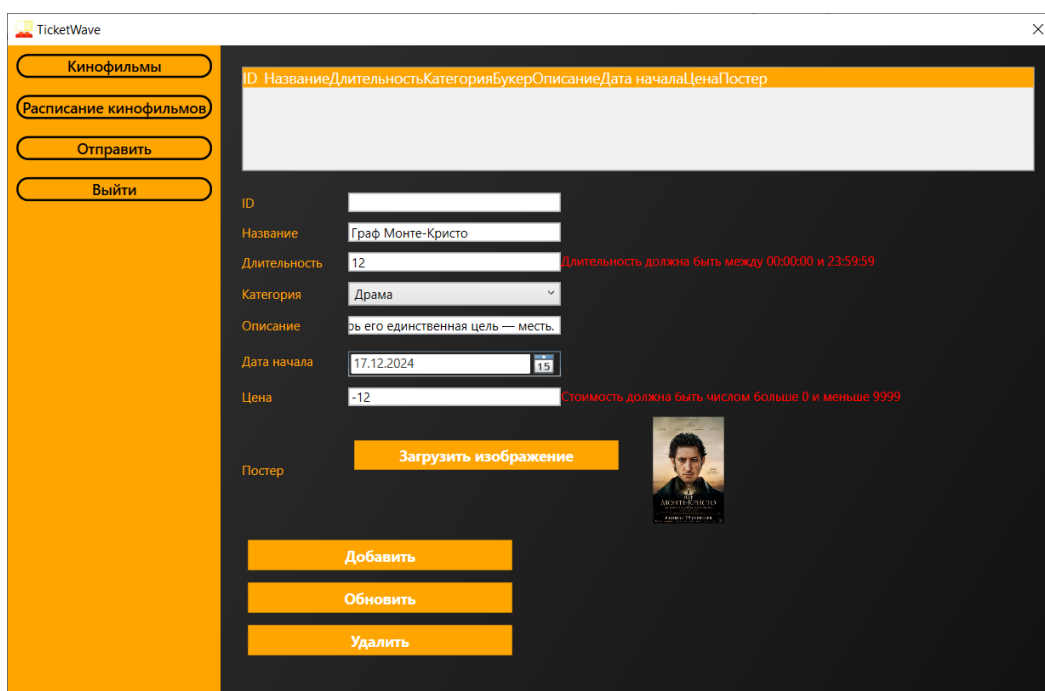


Рисунок 5.8 – Валидация при создании кинофильма

При создании расписания показа кинофильма проверяется «Дата показа». Оно должно быть заполнено корректной датой. Приложение автоматически подставляет

текущую дату, но пользователь может изменить её на любую другую, начиная с текущей.

«Время» должно быть заполнено корректным временем в формате ЧЧ:ММ. Поле «Название» представляет собой выпадающий список с доступными для выбора фильмами. Поле «Кинотеатр» также является выпадающим списком, из которого пользователь выбирает кинотеатр.

Самое главное – приложение проверяет, не занят ли выбранный кинотеатр в указанное время. Если кинотеатр в это время занят, то на экран выводится сообщение «Кинотеатр в это время занят», что продемонстрировано на рисунке 5.9. Также дата показа не может быть раньше даты начала, иначе система вновь выдаст ошибку.

ID	Дата показа	Дата начала	Название	Кинотеатр
1	12/18/2024 12:00:00 PM	12/17/2024 12:00:00 AM	Граф Монте-Кристо	Октябрь

ID: 1

Дата показа: 18.12.2024 Кинотеатр в это время занят

Время: 13:00

Название: Граф Монте-Кристо

Кинотеатр: Октябрь

Добавить

Обновить

Удалить

Рисунок 5.9 – Валидация при создании расписания кинофильма

Управление рядами является важной частью работы менеджера в системе TicketWave, поскольку именно на основе рядов формируются все билеты. Рассмотрим подробнее возможности и ограничения, связанные с управлением рядами.

Менеджер может добавлять новые ряды в локацию (добавление всегда происходит в конец), удалять выбранные ряды (при этом будет происходить смещение оставшихся рядов), изменять информацию о рядах (множитель цены и количество мест в ряду). Несмотря на широкие возможности управления рядами, существуют определенные ограничения, направленные на обеспечение целостности данных и стабильности работы системы, а именно запрещено переносить ряды из локации в локацию, то есть изменять информацию об локации ряда, и проводить какие-либо манипуляции с активными билетами, то есть с теми, которые куплены или забронированы.

Пример ситуации, в которой происходит попытка удаления записи о мероприятии, для которого уже были приобретены билеты, продемонстрирован на

рисунке 5.10. В данном случае система выдаёт ошибку из-за существующих зависимостей между таблицами.

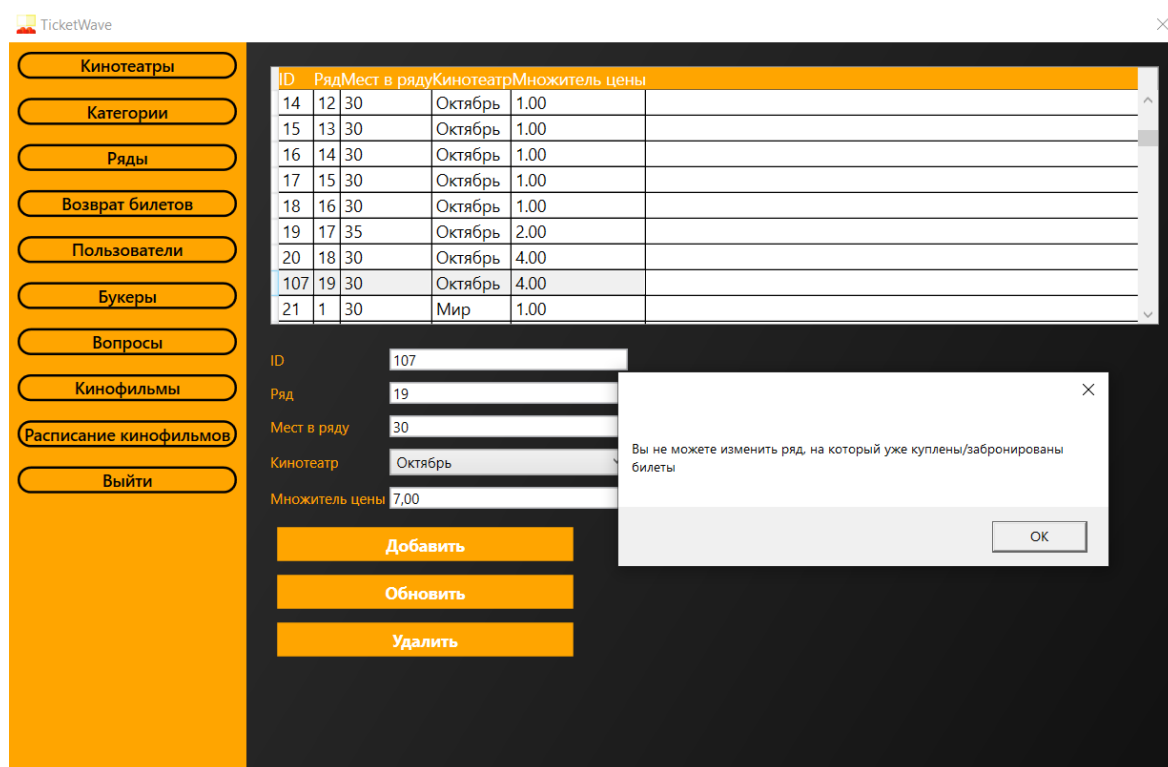


Рисунок 5.10 – Валидация при работе с рядами

Важно понимать, что любые изменения в конфигурации рядов, будь то добавление, удаление или редактирование, приводят к автоматическому перестроению всех билетов на все мероприятия в данной локации. Это гарантирует актуальность информации о доступных местах и ценах на билеты.

5.2 Выводы по разделу

Были проведены испытания разработанного приложения, включающие проверку функциональности форм авторизации, регистрации, создания и редактирования данных, а также работы пользовательского интерфейса каталога кинофильмов.

Проведенное в рамках данного раздела тестирование подтвердило высокую работоспособность и эффективность разработанного приложения, а также его соответствие заявленным требованиям и ожиданиям. Система, как показали проведённые испытания, обеспечивает не только корректную и бесперебойную обработку пользовательских данных, но и гарантирует надежную защиту конфиденциальной информации, в частности, паролей. Пользовательский интерфейс предоставляет пользователям удобные и эффективные инструменты для взаимодействия с приложением, значительно облегчая процесс поиска и выбора интересующих киносеансов. В свою очередь, менеджеры получают в свое распоряжение гибкие и мощные инструменты для управления данными о кинотеатрах, фильмах, расписании и рядах, что позволяет им оперативно и эффективно решать поставленные задачи.

6 Руководство по использованию

Процесс взаимодействия пользователя с системой начинается с регистрации, которая представлена на рисунке 6.1. Вам необходимо заполнить шесть обязательных полей, каждое из которых обозначено соответствующей надписью. Логин, почта и телефон должны быть уникальными.

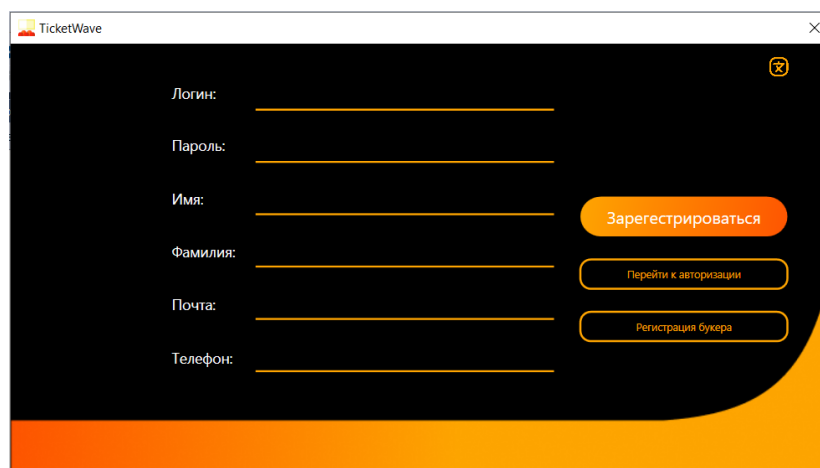
The screenshot shows a web browser window titled "TicketWave". The registration form is on a dark background with orange accents at the bottom and right. It contains six input fields with labels: "Логин:", "Пароль:", "Имя:", "Фамилия:", "Почта:", and "Телефон:". To the right of the fields are three buttons: "Зарегистрироваться" (orange), "Перейти к авторизации" (yellow), and "Регистрация букера" (yellow).

Рисунок 6.1 – Регистрация пользователя

Необходимо нажать на кнопку «Зарегистрироваться», чтобы завершить процесс регистрации и создать свой аккаунт.

Кнопка «Перейти к авторизации» предназначена для тех, кто уже зарегистрировался в системе. Нажав на неё, произойдёт переход в окно авторизации, где можно в свой аккаунт, используя логин и пароль.

Кнопка «Регистрация букера» предназначена для пользователей, которые хотят зарегистрироваться в системе в качестве букера. Нажав на неё, произойдёт переход к форме регистрации, предназначенной специально для букмекеров.

На рисунке 6.2 продемонстрировано окно авторизации.

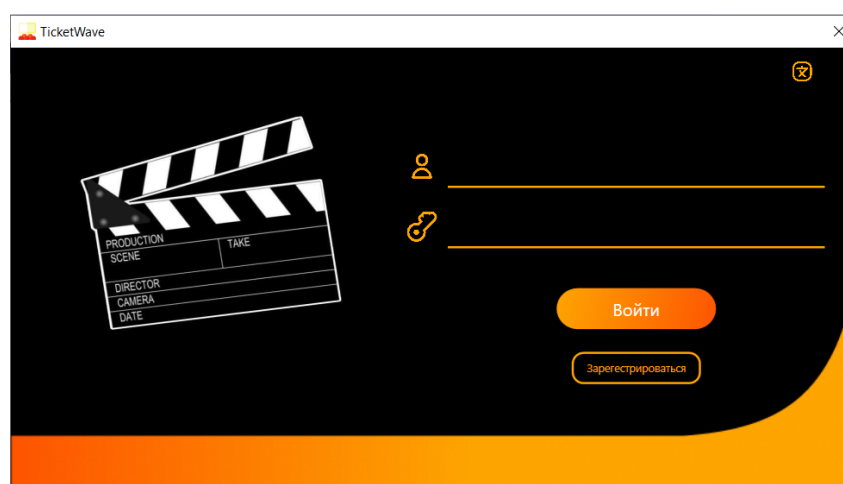
The screenshot shows the login page of the TicketWave system. It features a clapperboard graphic on the left with fields for "PRODUCTION", "SCENE", "TAKE", "DIRECTOR", "CAMERA", and "DATE". On the right, there are two input fields with user and password icons, followed by "Войти" (orange) and "Зарегистрироваться" (yellow) buttons. The layout is consistent with the registration page, with a dark background and orange accents.

Рисунок 6.2 – Авторизации пользователя

В окне располагается два поля. Верхнее поле предназначено для ввода логина, указанного при регистрации. Нижнее поле предназначено для ввода пароля.

При нажатии на кнопку «Войти» происходит авторизация пользователя в системе. При нажатии на кнопку «Зарегистрироваться» пользователь перенаправляется в окно регистрации, описанное ранее. В правом верхнем углу расположена кнопка для переключения языка интерфейса на английский.

После авторизации пользователь попадает в каталог фильмов, соответствующий рисунку 6.3. Окно каталога разделено на две основные части: панель фильтров слева и область отображения результатов справа.

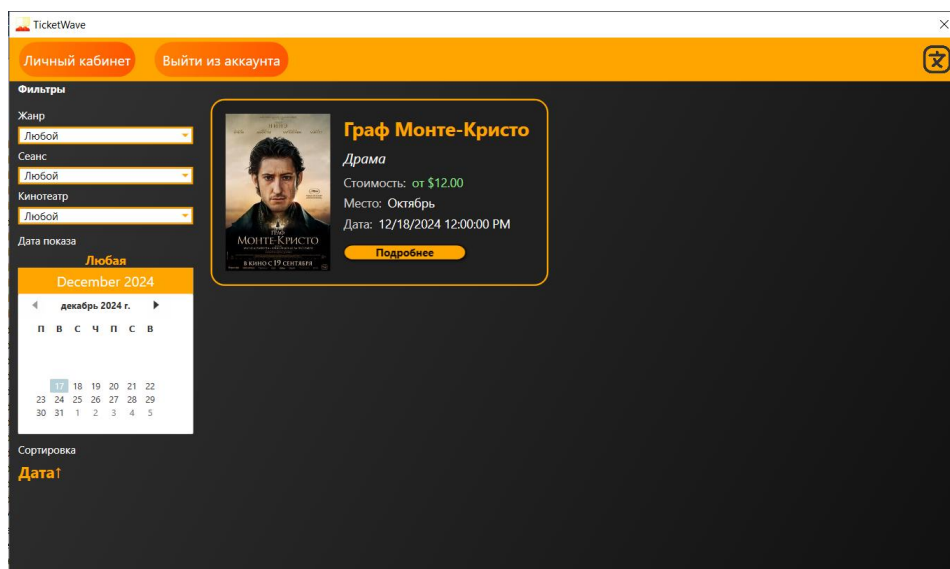


Рисунок 6.3 – Каталог кинофильмов

В верхней части панели расположена надпись «Фильтры», выделенная оранжевым цветом. Система позволяет выбрать интересующий жанр фильма, определенный сеанс и конкретный кинотеатр.

Присутствует календарь с возможностью выбора конкретной даты показа. Календарь может находиться в двух режимах:

1. Любая дата показа, то есть выбранная на календаре дата не влияет на отображение кинофильмов.
2. Выбранная дата показа, то есть будут отображаться только те кинофильмы, чьи показы настроены на выбранный день.

Система позволяет сортировать результаты по дате и по цене, чтобы поменять критерий сортировки, необходимо нажать на неё. Рядом располагается стрелка, указывающая направление сортировки, её можно поменять.

Карточка фильма включает постер, название фильма, жанр, стоимость, название кинотеатра, дата и время сеанса. При нажатии на кнопку «Подробнее» открывается страница с подробной информацией о фильме и сеансе.

При нажатии на кнопку «Личный кабинет» осуществляется переход в личный кабинет пользователя. При нажатии на кнопку «Выйти из аккаунта» происходит выход из учетной записи. Также в верхней панели расположена кнопка смены языка интерфейса.

При переходе на подробную информацию о кинофильме перед пользователем открывается отдельная страница. Основную часть экрана занимает схема зала, представленная в виде рядов с пронумерованными местами, что показано на

рисунке 6.4. Ряды и места в них пронумерованы, можно навести курсором на место и получить справку. Цветовая индикация мест позволяет различать свободные (зеленый), купленные (серые), забронированные (жёлтый) и выбранные пользователем (оранжевый) места.

В нижней части окна находится область «Выбранные билеты». Здесь отображаются места с указанием ряда, номера места и стоимости. Рядом с каждым выбранным билетом есть красный крестик, при нажатии на который можно отменить выбор данного места. Под списком выбранных билетов расположена кнопка «Добавить в корзину» для перехода к следующему этапу покупки.

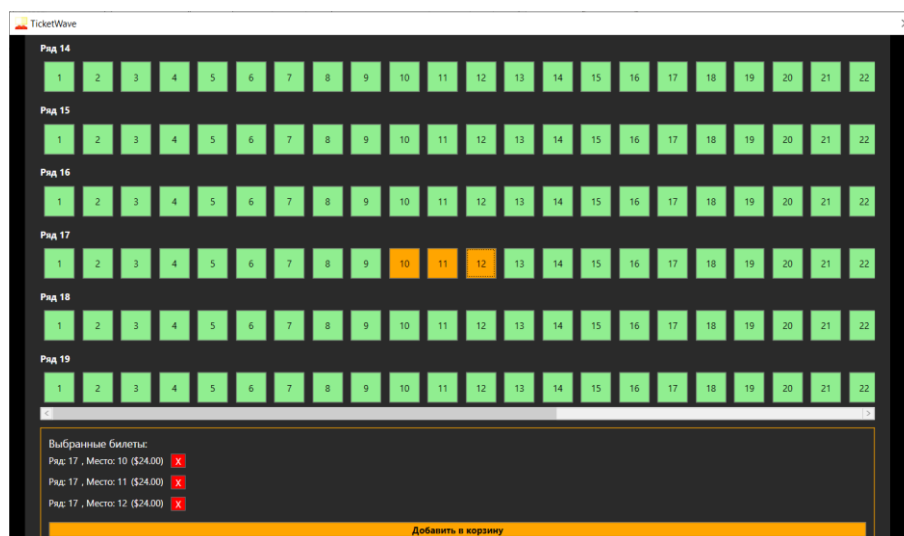


Рисунок 6.4 – Схема выбора билетов

После выбора мест и нажатия кнопки «Добавить в корзину» пользователь может перейти в свой личный кабинет, а именно в раздел «Ваша корзина», продемонстрированная на рисунке 6.5. В верхней части окна расположена кнопка «Выйти», позволяющая выйти из личного кабинета.

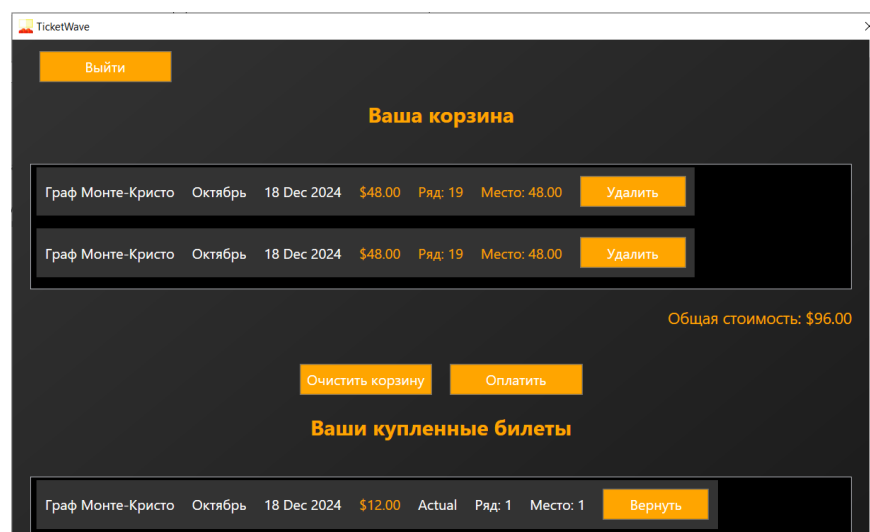


Рисунок 6.5 – Личный кабинет пользователя

Здесь отображаются выбранные пользователем билеты. Каждый билет представлен в отдельном блоке с дополнительной информацией. Справа находится

кнопка «Удалить», при нажатии на которую соответствующий билет удаляется из корзины.

Ниже расположена информация об общей стоимости билетов в корзине.

Под информацией об общей стоимости находятся две кнопки. При нажатии на кнопку «Очистить корзину» все билеты удаляются из корзины. При нажатии на кнопку «Оплатить» пользователь переходит к оплате выбранных билетов.

Ниже раздела «Ваша корзина» находится раздел «Ваши купленные билеты». Здесь отображаются уже купленные пользователем билеты. Информация о билете представлена так же, как и в разделе «Ваша корзина». Отличие состоит в том, что вместо кнопки «Удалить» у купленных билетов присутствует кнопка «Вернуть». При нажатии на эту кнопку отправляется запрос на возврат билета, который позже будет одобряться администратором.

У пользователей есть возможность оставлять комментарии к фильмам, а также просматривать расписание сеансов на другие дни, что показано на рисунке 6.6.

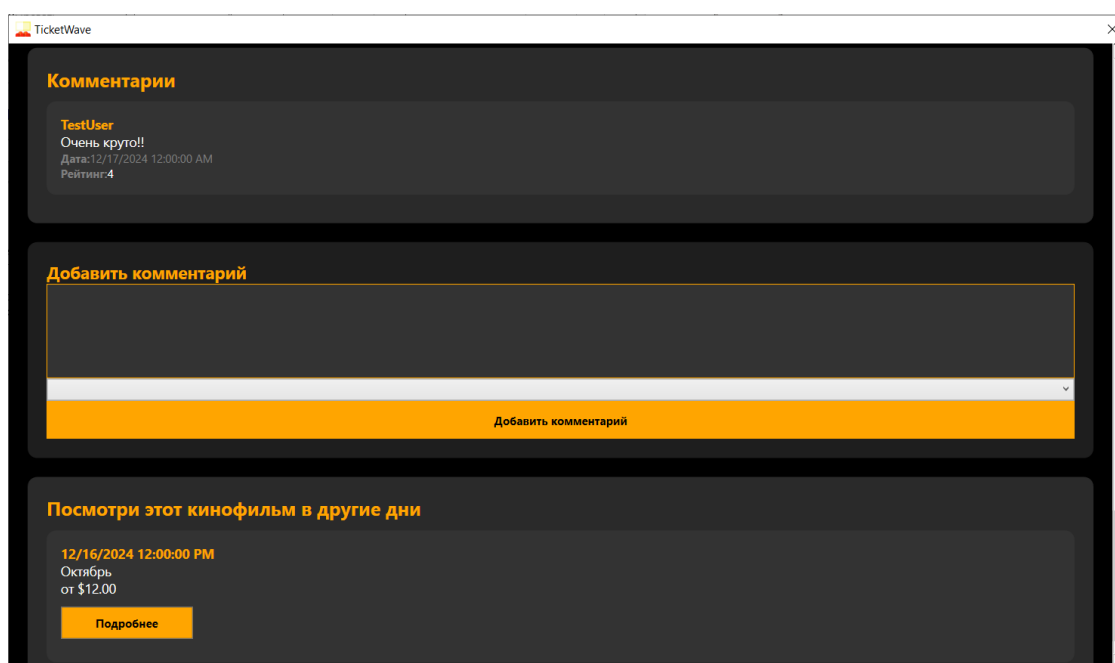


Рисунок 6.6 – Комментарии к кинофильму

В верхней части отображаются уже оставленные комментарии. Каждый комментарий содержит имя пользователя, текст комментария, дату и выставленный рейтинг. Ниже находится раздел «Добавить комментарий». Здесь пользователь может ввести свой комментарий в текстовое поле. Однако, важно учитывать, что возможность добавить комментарий доступна только тем пользователям, которые ранее посещали данное мероприятие. Под полем для ввода комментария находится кнопка «Добавить комментарий».

В нижней части окна расположена секция «Посмотри этот кинофильм в другие дни». Здесь отображается информация о сеансах этого же фильма на другие даты. Для каждого сеанса указана дата, время, кинотеатр и стоимость билета. Также присутствует кнопка «Подробнее», при нажатии на которую пользователь перенаправляется на страницу с детальной информацией.

Основной функционал менеджеров и организаторов рассмотрен в разделе 5. Также в интерфейсе менеджера присутствует кнопка «Возврат билетов». При нажатии на кнопку «Возврат билетов» открывается таблица со списком запросов на возврат. Эта таблица содержит следующую информацию по каждому запросу. Раздел изображён на рисунке 6.7.

Под таблицей с запросами расположены две кнопки. При нажатии на кнопку «Принять» менеджер подтверждает возврат билета, при нажатии на кнопку «Отклонить» менеджер отклоняет запрос на возврат.

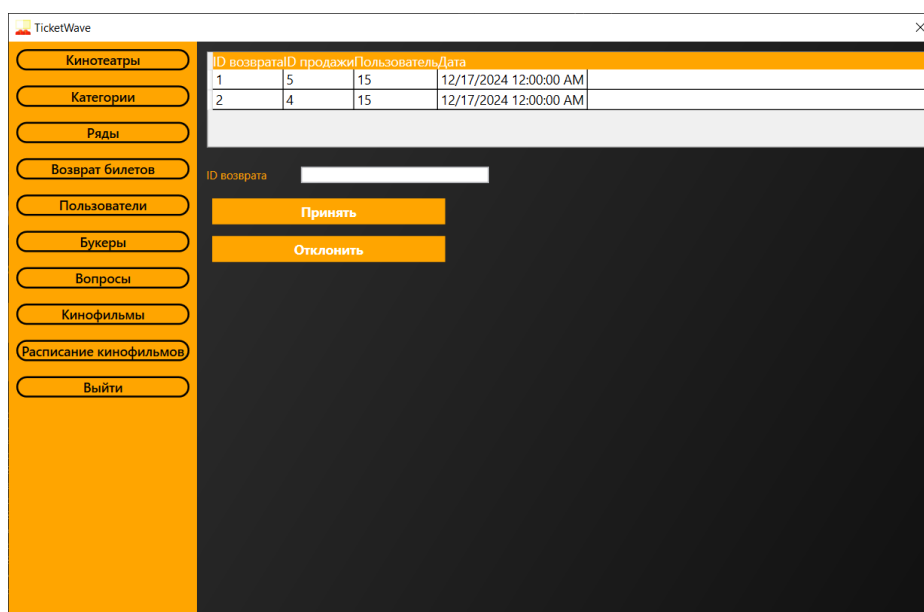


Рисунок 6.7 – Возврат билетов менеджером

В интерфейсе менеджера есть раздел, предназначенный для работы с вопросами пользователей, продемонстрированный на рисунке 6.8.

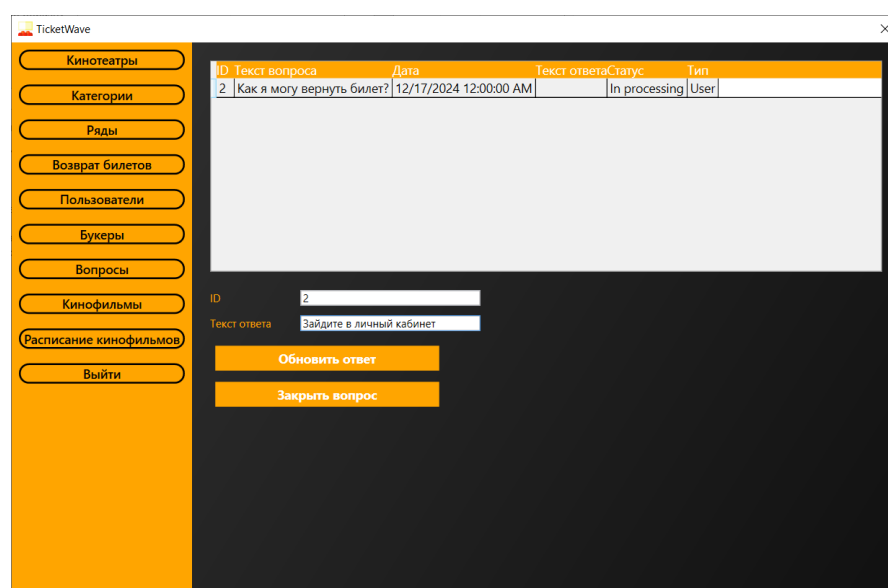


Рисунок 6.8 – Ответы на вопросы менеджером

При нажатии на кнопку «Вопросы» в левом меню, открывается окно, разделенное на две части. В верхней части находится таблица с перечнем вопросов,

включающая ID вопроса, текст вопроса, дату, текст ответа, статус и тип вопроса. Ниже расположены поля для ввода ID вопроса и текста ответа, а также кнопки «Обновить ответ» и «Закрыть вопрос».

Менеджер может просматривать список вопросов, выбирать интересующий его вопрос из таблицы, и вводить текст ответа в соответствующее поле. После ввода ответа, необходимо нажать кнопку «Обновить ответ», чтобы сохранить изменения. В таком случае, текст ответа добавится к уже существующему. Также предусмотрена возможность закрыть вопрос, нажав на кнопку «Закрыть вопрос». Это означает, что вопрос был обработан и больше не требует внимания. Важно, что при закрытии вопроса менеджер также должен вписать информацию в поле, которая добавится к общему ответу.

Менеджер, используя свой личный кабинет, обладает широкими полномочиями по управлению контентом. Он может добавлять в систему новые кинотеатры, а также редактировать уже существующие записи о кинотеатрах. Ему доступно управление категориями фильмов, то есть жанрами, что позволяет ему добавлять, изменять или удалять жанры для корректной работы фильтрации на пользовательской стороне.

Менеджер отслеживает наполнение базы данных фильмов букерами. Помимо этого, менеджер управляет учетными записями букеров и пользователей.

Можно с уверенностью заключить, что разработанная система предоставляет пользователям удобный, функциональный и интуитивно понятный интерфейс для поиска фильмов, выбора и покупки билетов, а также управления своими заказами. Грамотное разделение ролей на пользователей, букеров и менеджеров обеспечивает эффективное управление всеми аспектами работы системы. Продуманные и тщательно реализованные механизмы фильтрации, сортировки и отображения информации делают процесс взаимодействия с системой максимально комфортным и приятным для пользователя любого уровня подготовки.

Заключение

В результате выполнения курсовой работы был проведён всесторонний анализ предметной области, разработаны и реализованы ключевые компоненты программного средства «Продажа билетов в кинотеатры».

На этапе анализа были изучены существующие аналоги, такие как ByCard, Skyline и Moon, выявлены их сильные и слабые стороны, что позволило сформулировать требования к разрабатываемой системе. Определены основные роли пользователей (клиент, букер, менеджер) и их функциональные возможности. Обоснован выбор технологического стека: платформа .NET 8, среда разработки Visual Studio 2022, технология WPF с использованием MVVM и XAML, язык C#, Entity Framework Core для работы с данными и SQL Server в качестве СУБД.

Проектирование системы включало разработку архитектуры, основанной на паттернах Repository, Unit of Work и MVVM. Спроектирована структура базы данных из 16 взаимосвязанных сущностей, описан механизм взаимодействия с базой данных с помощью Entity Framework Core. Для наглядного представления функциональности и динамического взаимодействия компонентов системы были использованы UML-диаграммы прецедентов и последовательностей.

Реализация программного средства включила использование паттерна Singleton для управления локализацией, конвертеров для преобразования данных, паттернов Repository и Unit of Work для работы с данными. Реализован механизм переключения языков интерфейса. Приложение построено на основе паттерна MVVM, обеспечивающего разделение ответственности между моделью, представлением и моделью представления. Для инкапсуляции действий пользователя использован паттерн ICommand.

Тестирование подтвердило работоспособность и эффективность приложения. Были проверены формы авторизации, регистрации, создания и редактирования данных, а также работа пользовательского интерфейса. Система корректно обрабатывает ошибки ввода данных, обеспечивает защиту паролей пользователей с помощью хеширования. Пользовательский интерфейс предоставляет удобные инструменты для поиска и фильтрации кинофильмов, а менеджеры получают гибкие возможности для управления данными.

Разработанное программное средство «Продажа билетов в кинотеатры» отвечает поставленным целям и задачам. Оно обладает удобным и интуитивно понятным интерфейсом, обеспечивает эффективное взаимодействие пользователей с системой и предоставляет широкие функциональные возможности для каждой из ролей. Использование современных технологий и архитектурных решений обеспечивает надежность, масштабируемость и простоту сопровождения приложения. Полученные результаты демонстрируют успешное применение объектно-ориентированных технологий программирования и стандартов проектирования для решения задач автоматизации в сфере развлечений.

Список используемых источников

1. Албахарв, Джозеф, Албахари, Бен. С# 6.0. Справочник. Полное описание языка, 6-е изд. : Пер. с англ. – М. : ООО "И.Д. Вильяме", 2016. – 1040 с.
2. Metanit.com – Руководство по WPF [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/wpf/> – Дата доступа: 12.11.2024.
3. Леоненков А. В. Самоучитель UML / Леоненков Александр Васильевич – 2-е изд. – СПб.: БХВ-Петербург, 2004. – 432 с.
4. Professorweb.ru – Основы WPF [Электронный ресурс] – Режим доступа: <https://professorweb.ru/my/WPF/base/> – Дата доступа: 23.11.2024.
5. Котлов, А. Разработка приложений с использованием EntityFramework / А. Котлов. – М.: Бином, 2019. – 240 с.

Приложение Б

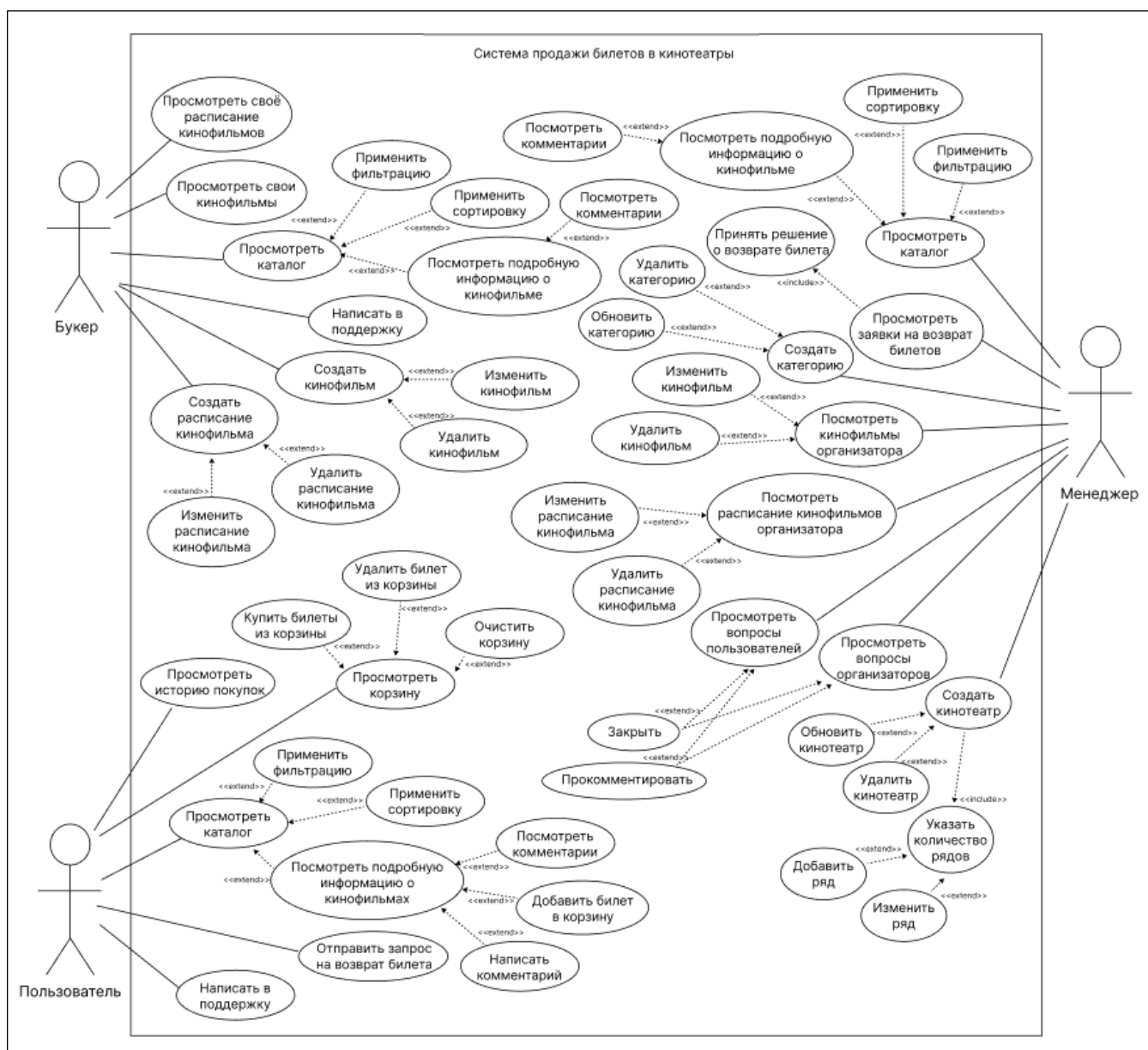


Рисунок Б.1 – Диаграмма прецедентов

Приложение В

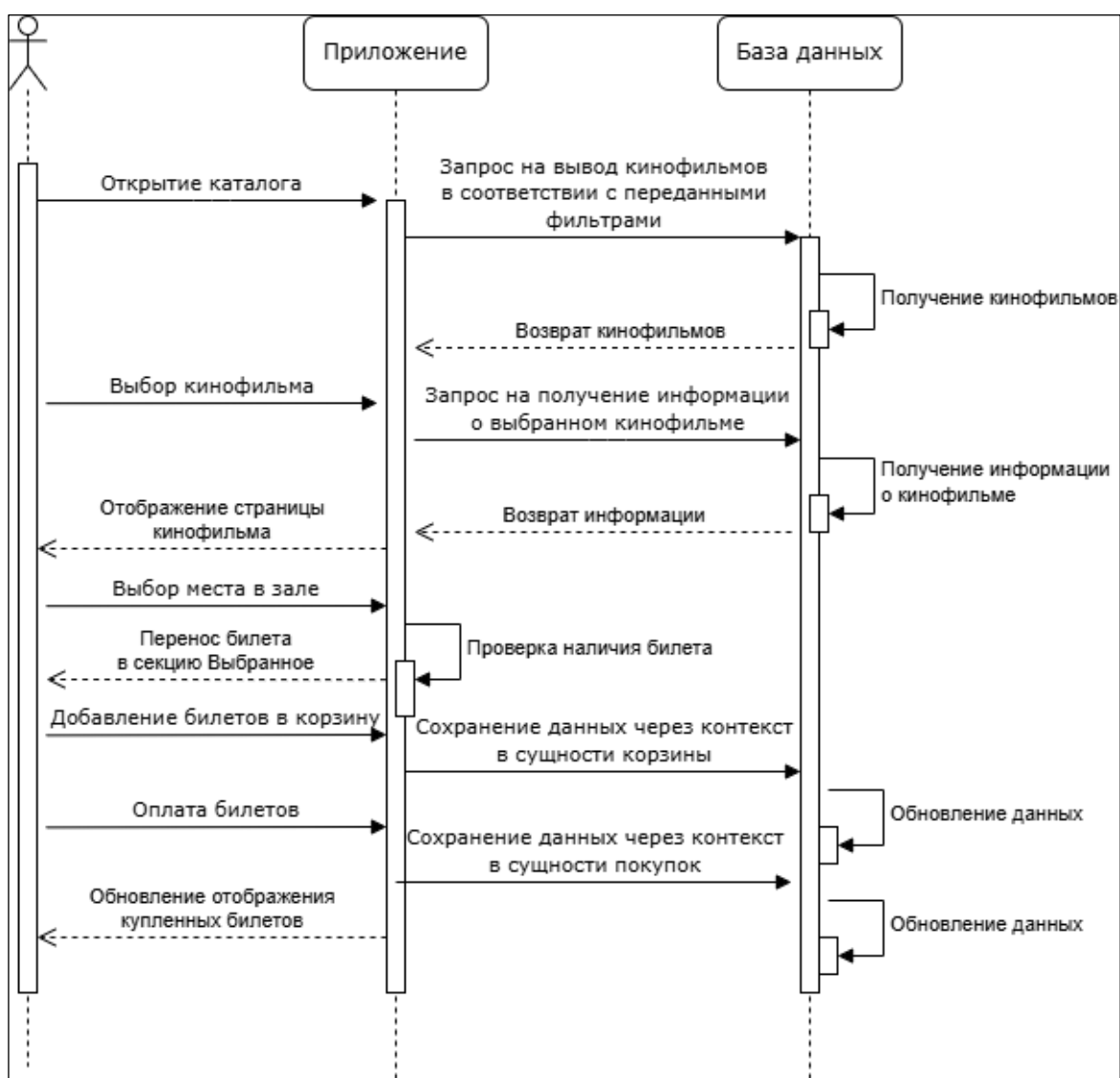


Рисунок В.1 – Диаграмма последовательности

Приложение Г

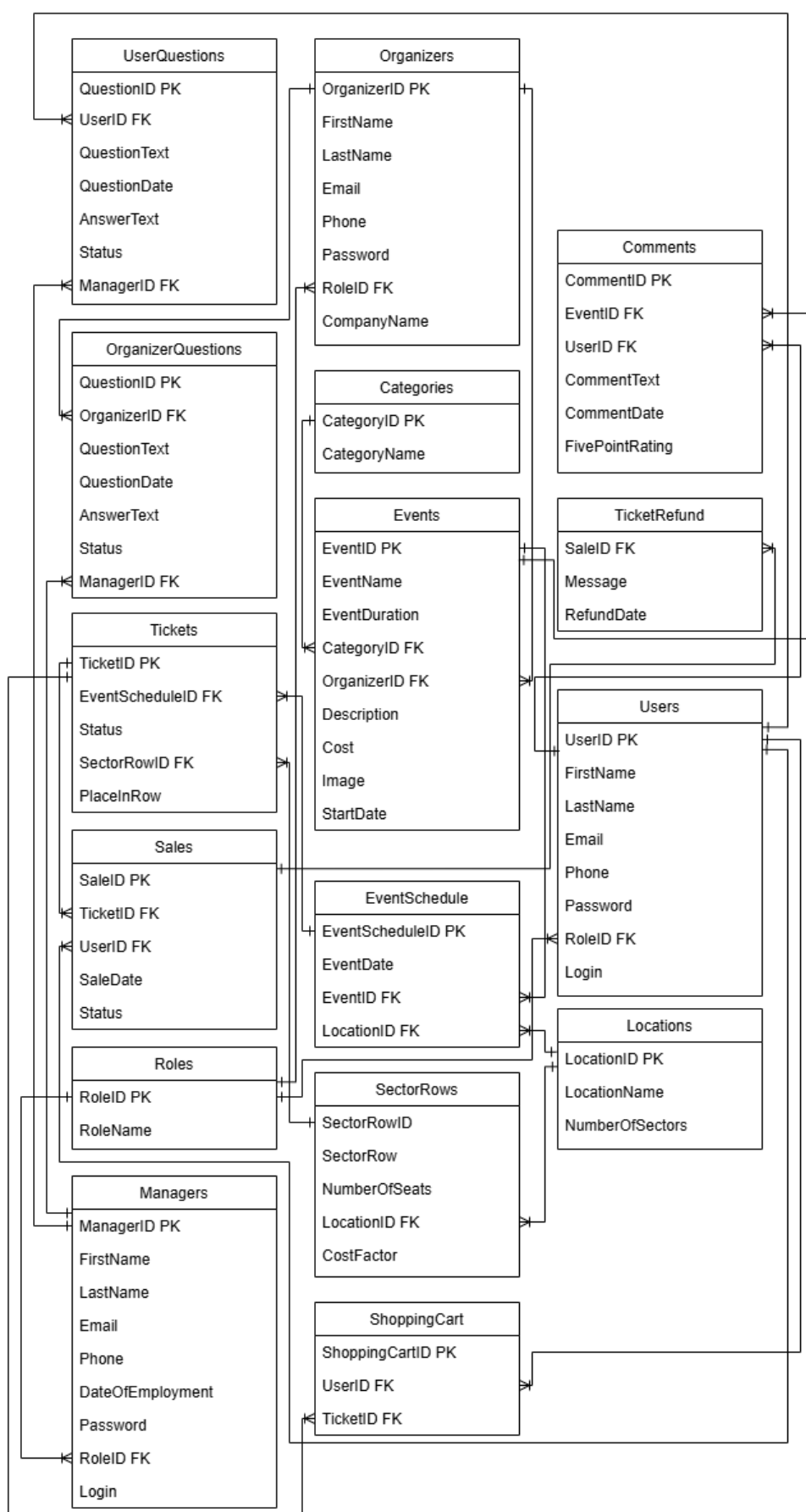


Рисунок Г.1 – Логическая схема базы данных

Приложение Д

```

public class CategoryRepository : BaseRepository<Users>
{
    ApplicationDbContext _context;
    public CategoryRepository(ApplicationDbContext context) :
base(context)
    {
        _context = context;
    }
    public ObservableCollection<Categories> GetCategories()
    {
        var categories = _context.Categories.ToList();

        return new ObservableCollection<Categories>(categories);
    }
    public Categories? FindById(int id)
    {
        return _context.Categories.FirstOrDefault(l => l.CategoryID
== id);
    }
    public Categories? UpdateCategory(int id, string name)
    {
        var category = FindById(id);

        if (category != null)
        {
            category.CategoryName = name;
            _context.SaveChanges();
            return category;
        }
        return null;
    }
    public Categories? AddCategory(string name)
    {
        if (!CheckCategoryUnique(name))
        {
            var maxCategoryId = _context.Categories
                .Max(l => (int?)l.CategoryID) + 1 ?? 1;
            var category = new Categories { CategoryID =
maxCategoryId, CategoryName = name };
            _context.Categories.Add(category);
            _context.SaveChanges();
            return category;
        }
        return null;
    }

    public Categories? DeleteCategory(int id)
    {
        var category = _context.Categories.FirstOrDefault(l =>
l.CategoryID == id);

```

```
if (category != null)
{
    _context.Categories.Remove(category);
    _context.SaveChanges();
    return category;
}
return null;
}
public bool CheckCategoryUnique(string name)
{
    return _context.Categories.Any(l => l.CategoryName == name);
}
}
```

Листинг Д.1 – Репозиторий категории

Приложение Е

```

public class UnitOfWork
{
    private ApplicationDbContext _context;
    private CategoryRepository categoryRepository;
    // ...
    private bool disposed = false;
    public CategoryRepository CategoryRepository
    {
        get
        {
            if (categoryRepository == null) categoryRepository = new
CategoryRepository(_context); return categoryRepository;
        }
    }

    public UnitOfWork()
    {
        string connectionString =
ConfigurationManager.ConnectionStrings["TicketSalesDb"].ConnectionSt
ring;
        var optionsBuilder = new
DbContextOptionsBuilder<ApplicationDbContext>();
        optionsBuilder.UseSqlServer(connectionString);

        _context = new ApplicationDbContext(optionsBuilder.Options);
    }
    public void Save()
    {
        _context.SaveChanges();
    }

    public virtual void Dispose(bool disposing)
    {
        if (!this.disposed)
        {
            if (disposing)
            {
                _context.Dispose();
            }
            this.disposed = true;
        }
    }
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }
}

```

Листинг Е.1 – Реализация паттерна Unit Of Work