СОДЕРЖАНИЕ

[Введение 5](#_Toc121437792)

[1 Анализ и моделирование предметной области программного средства 6](#_Toc121437793)

[1.1 Описание предметной области 6](#_Toc121437794)

[1.2 Разработка функциональной модели предметной области 6](#_Toc121437795)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 10](#_Toc121437796)

[1.4 Разработка информационной модели предметной области 11](#_Toc121437797)

[1.5 Модели представления программного средства и их описание 13](#_Toc121437798)

[2 Проектирование и конструирование программного средства 17](#_Toc121437799)

[2.1 Постановка задачи 17](#_Toc121437800)

[2.2 Архитектурные решения 17](#_Toc121437801)

[2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства 19](#_Toc121437802)

[2.4 Проектирование пользовательского интерфейса 22](#_Toc121437803)

[3 Тестирование и проверка работоспособности программного средства 27](#_Toc121437804)

[4 Руководство по развертыванию и использованию программного средства 30](#_Toc121437805)

[Заключение 39](#_Toc121437806)

[Список использованных источников 40](#_Toc121437807)

[Приложение А 42](#_Toc121437808)

[(обязательное) 42](#_Toc121437809)

[Отчет о проверке на заимствования в системе «Антиплагиат» 42](#_Toc121437810)

[Приложение Б 43](#_Toc121437811)

[(обязательное) 43](#_Toc121437812)

[Листинг кода алгоритмов, реализующих бизнес-логику 43](#_Toc121437813)

[Приложение В 50](#_Toc121437814)

[(обязательное) 50](#_Toc121437815)

[Листинг скрипта генерации базы данных 50](#_Toc121437816)

# Перечень условных обозначений, символов и терминов

|  |  |
| --- | --- |
| БД (база данных) | – представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины |
| Информационно-коммуникационные технологии | –процессы, методы поиска, сбора, хранения, обработки, предоставления, распространения информации и способы осуществления таких процессов и методов |
| Информационная система | –система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые), которые обеспечивают и распространяют информацию |
| Нормальная форма | –свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных |
| Среда выполнения | –вычислительное окружение, необходимое для выполнения компьютерной программы и доступное во время выполнения компьютерной программы |
| СУБД (система управления базами данных) | – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных |
| ТЗ (техническое задание) | – документ, содержащий требования заказчика к объекту разработки, определяющий порядок и условия её проведения |
| *API*(*applicationprogramminginterface*) | –описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой |
| *HTML*(*HyperTextMarkupLanguage*) | – стандартизированный язык разметки документов во Всемирной паутине |
| *HTTP* (*HyperTextTransferProtocol*) | –протокол прикладного уровня передачи данных изначально – в виде гипертекстовых документов в формате *html*, в настоящий момент используется для передачи произвольных данных. |
| *IDE*(*Integrateddevelopmentenvironment*) | –комплекс программных средств, используемый программистамидля разработки программного обеспечения |
| *IDEF* | – методология функционального моделирования (англ. *functionmodeling*) и графическая нотация, предназначенная для формализации и описания бизнес-процессов |
| *Java* | – строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией *SunMicrosystems* |
| *MicrosoftSQLServer* | – система управления реляционными базами данных (СУБД), разработанная корпорацией *Microsoft* |
| *SQL*(*structuredquerylanguage*) | –язык структурированных запросов, декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных |
| *MySQLServer* | – свободная реляционная система управления базами данных |
| *SybaseASE*(*AdaptiveServerEnterprise*) | – реляционная система управления базами данных компании *SAP*, одна из СУБД, использующая в качестве основного процедурного *SQL*-расширения язык *Transact-SQL* |
| *UML*(*UnifiedModelingLanguage*) | –язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур |
| *URL*(*UniformResourceLocator*) | – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса |

# Введение

Наверное, каждый современный человек любит кино. Кинематограф стал не только видом изобразительного искусства, но и занял свою нишу в сфере развлечений.

Современный кинематограф отражает социальные проблемы общества, освещает исторические события, иллюстрирует биографии известных людей, дает свою интерпретацию классическим произведениям, реализует самые смелые фантастические задумки, позволяет отвлечься от реальности и погрузится в вымышленный мир или просто служит развлечением.

С каждым годом увеличиваются бюджеты кинокартин, качество съемки, масштабность спецэффектов, тщательность подбора саундтреков, остроумность сценариев для того, чтобы соответствовать запросам кинозрителей, которые становятся все искушеннее.

Фильмы поражают своим многообразием: десятки жанров, стран производства и киностудий, тысячи режиссеров и сценаристов и еще больше актеров. Каждую неделю презентуются кинопремьеры и анонсируются будущие проекты.

Целью данной курсовой работы является разработка, продвижение и сопровождение автоматизированной системы, которая будет включать в себя базу данных и приложение, обладающее интерфейсом просмотра и правки, а также средствами обработки хранящейся в базе данных информации.

Разработка такого программного средства и станет целью данного курсового проекта. Для достижения поставленной цели необходимо выполнить ряд задач:

* изучить принцип работы кинотеатров;
* определить список функций, необходимых для эффективной работы программного продукта;
* составить полную схему алгоритма работы программы;
* спроектировать базу данных хранимой информации;
* реализовать серверную часть приложения, которая будет реализовывать бизнес логику, и будет выполнять работу с базой данных;
* реализовать клиентскую часть приложения, с удобным интерфейсом для пользователя;
* протестировать полученное программное средство и убедиться, что оно корректно реализует свою бизнес логику.

# Анализ и моделирование предметной области программного средства

## 1.1Описание предметной области

**1.1.1** На сегодняшний день кино, как особый вид развлечения невозможно искоренить из общества. Одни любят сидеть перед телевизором, а другие предпочитают просмотр фильмов в кинотеатрах. Ежегодно выпускается огромное количество фильмов, что помогает каждому человеку найти то, что приходится ему по душе. Большой выбор жанров помогает людям смотреть мелодрамы, детективы, ужасы или исторические фильмы.

В настоящее время можно проследить тенденции к расширению рынка предоставления всевозможных развлекательных услуг. Сюда, конечно же, стоит отнести и кинотеатры. Можно заметить, что количество кинотеатров неумолимо увеличивается как в больших городах, население которых переваливает за миллион, так и в городах поменьше.

Очень легко отслеживать и вести учет записей и бизнес-процессов на любом предприятии благодаря современным технологиям. Они позволяют автоматизировать и ускорить процесс работы с информацией, ведь самому человеку было бы трудно фиксировать данные и сортировать их вручную.

Так, например, кинотеатр, представляющий собой целый организм различных задач и потоков выполнения, требует глобальной автоматизации. Например, составление расписания показа фильмов, организация залов, распределение комфортного количества клиентов в каждом из них, предоставление онлайн и офлайн покупки билетов для посетителей, чтобы извлечь максимальную выгоду и грамотно распределить все процессы.

Автоматизация управления кинотеатра позволяет решать организационные вопросы, вести учет рабочего времени, контроль и отчетность выручки в едином комплексном режиме.

## 1.2 Разработка функциональной модели предметной области

Рассматриваемые в данном курсовом проекте бизнес-процессы кинотеатра, связанные с обслуживанием клиентов и принятием управленческих решений, можно представить в виде системы, спроектированной в *CASE*-средстве *AllFusionProcessModeler* с точки зрения стандарта *IDEF*0.

В качестве входных параметров для системы выступают данные пользователей системы, база данных фильмов, база данных кинозалов. В качестве механизма осуществления главной функции выступают администратор кинотеатра и программное обеспечение. Выходным параметром для данной системы является автоматизированный процесс работы с клиентами. Функциональная модель рассматриваемой системы представлена на рисунке 1.1.

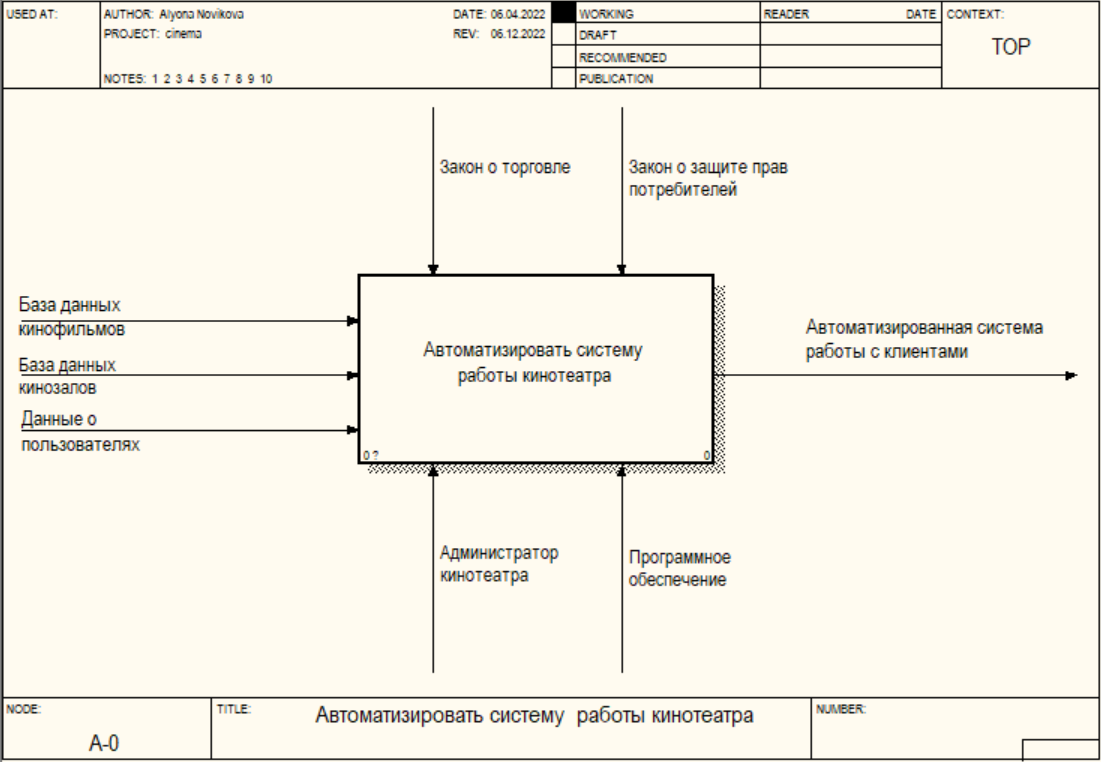


Рисунок 1.1 – Верхний уровень функциональной модели

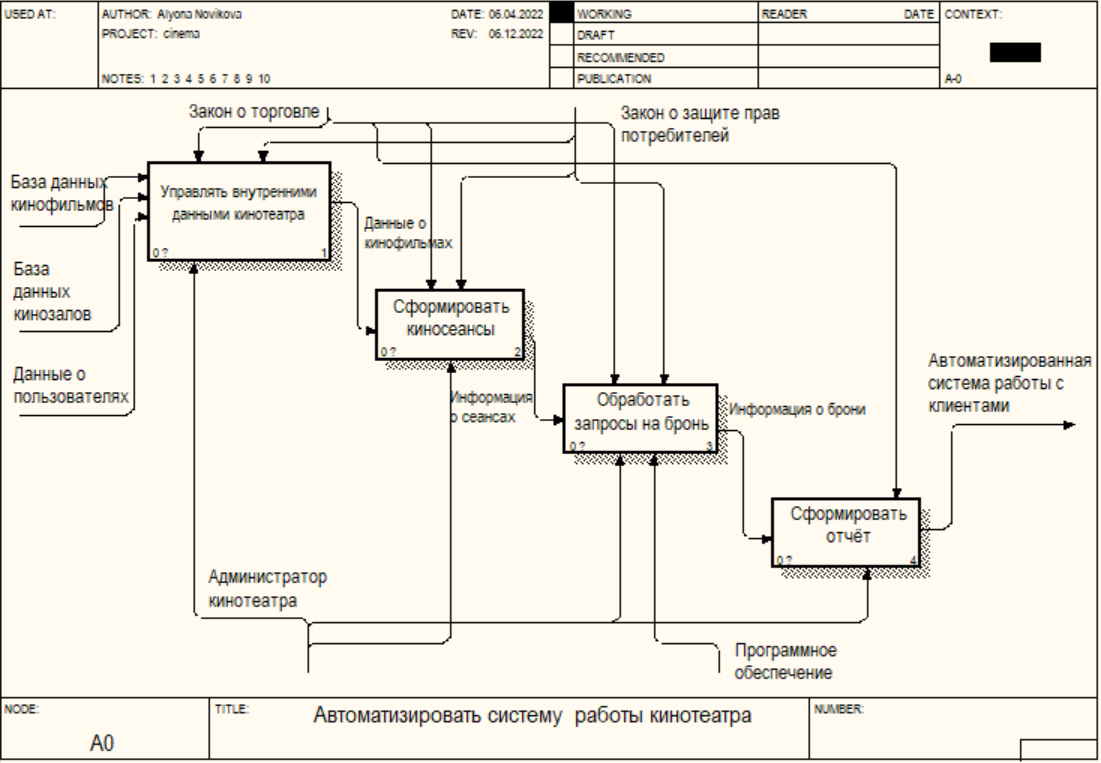


Рисунок 1.2 – Декомпозиция контекстной диаграммы

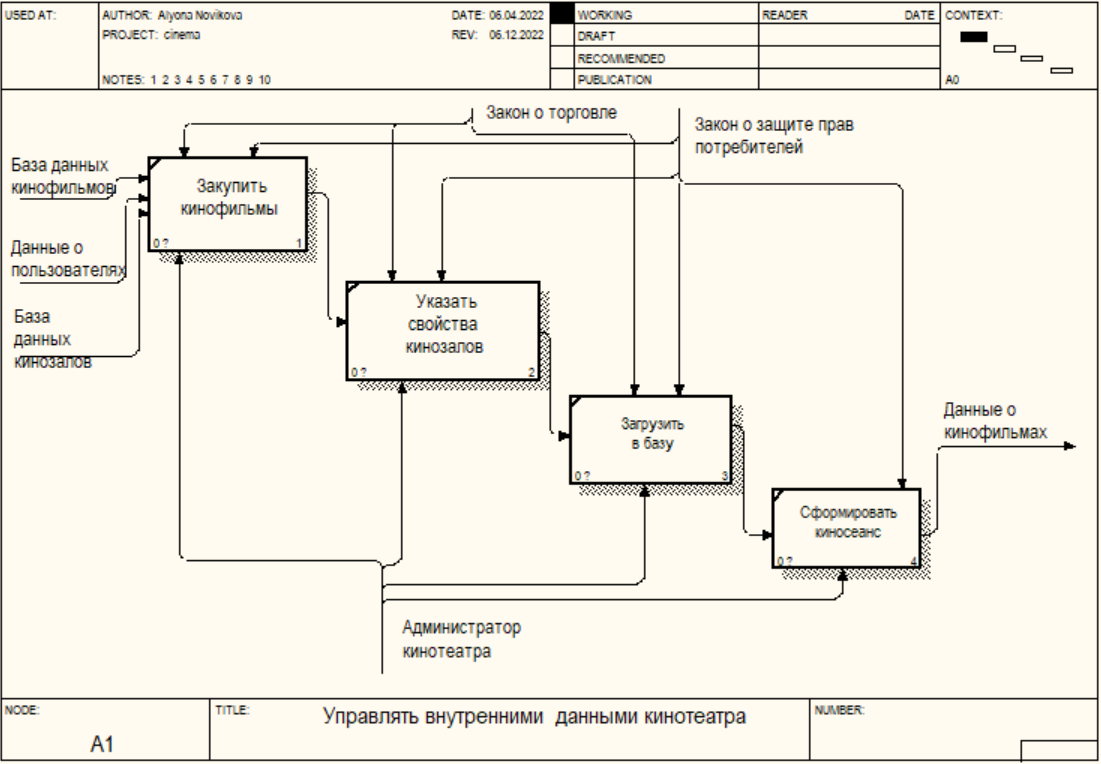


Рисунок 1.3 – Детализация блока «Управлять внутренними данными кинотеатра»

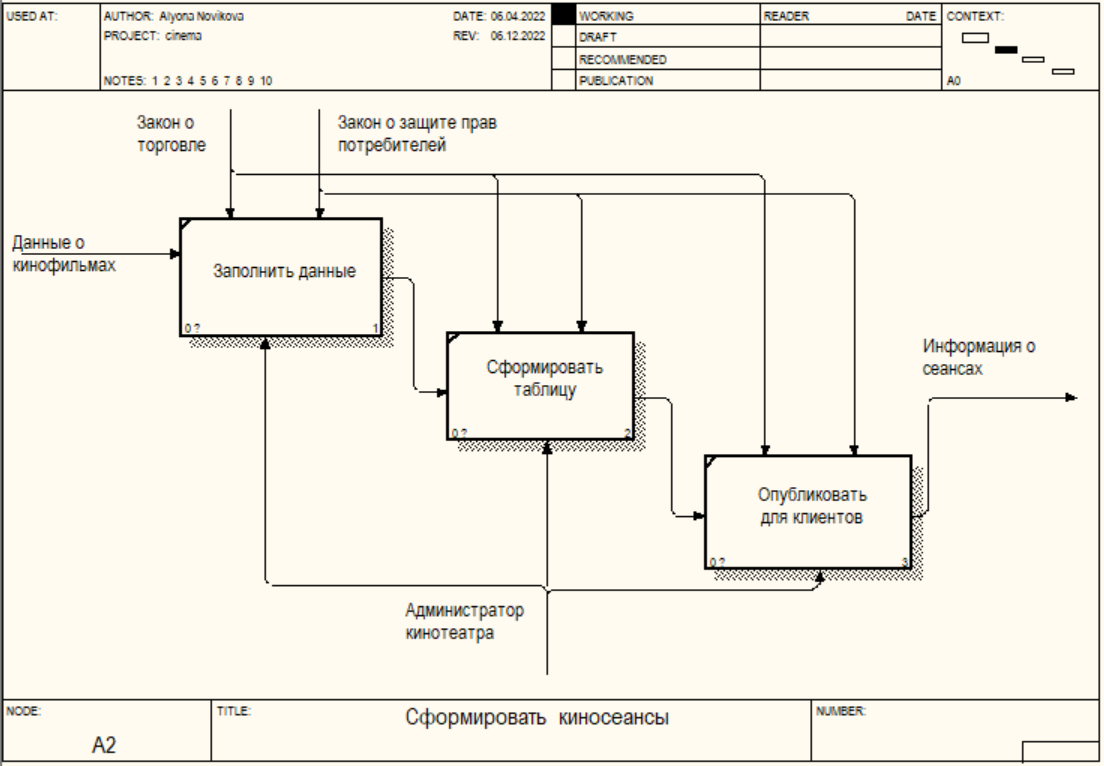


Рисунок 1.4 – Детализация блока «Сформировать киносеансы»

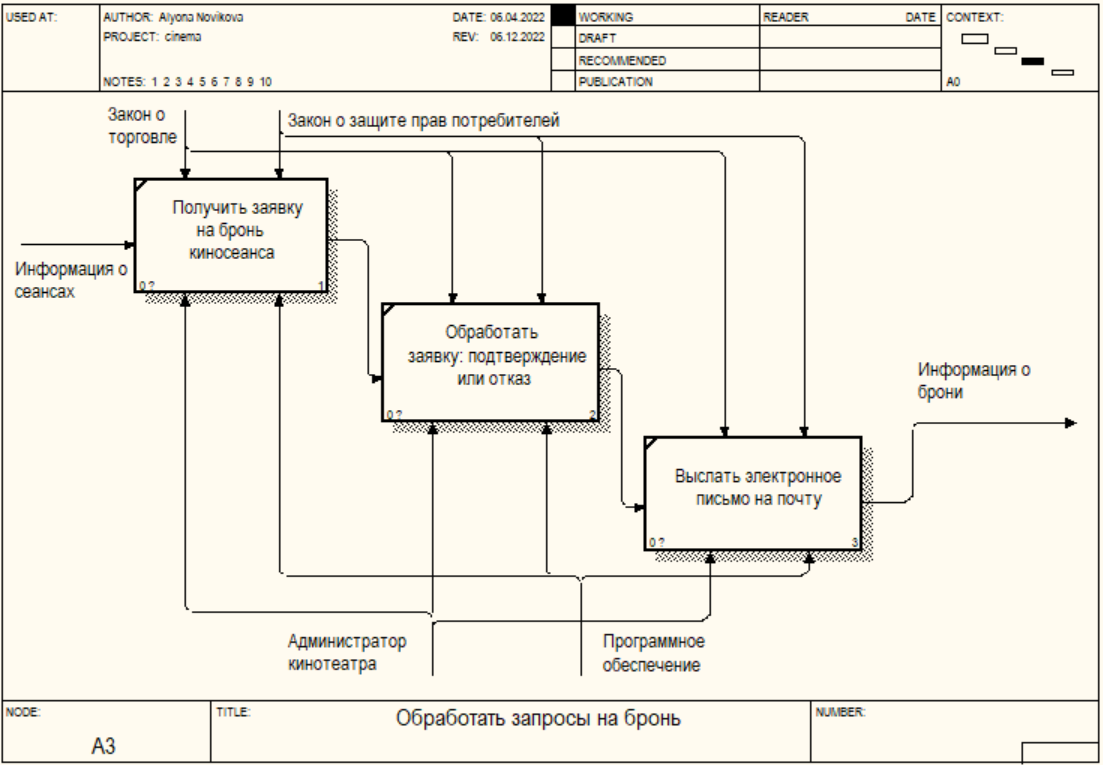


Рисунок 1.5 – Детализация блока «Обработать запросы на бронь»

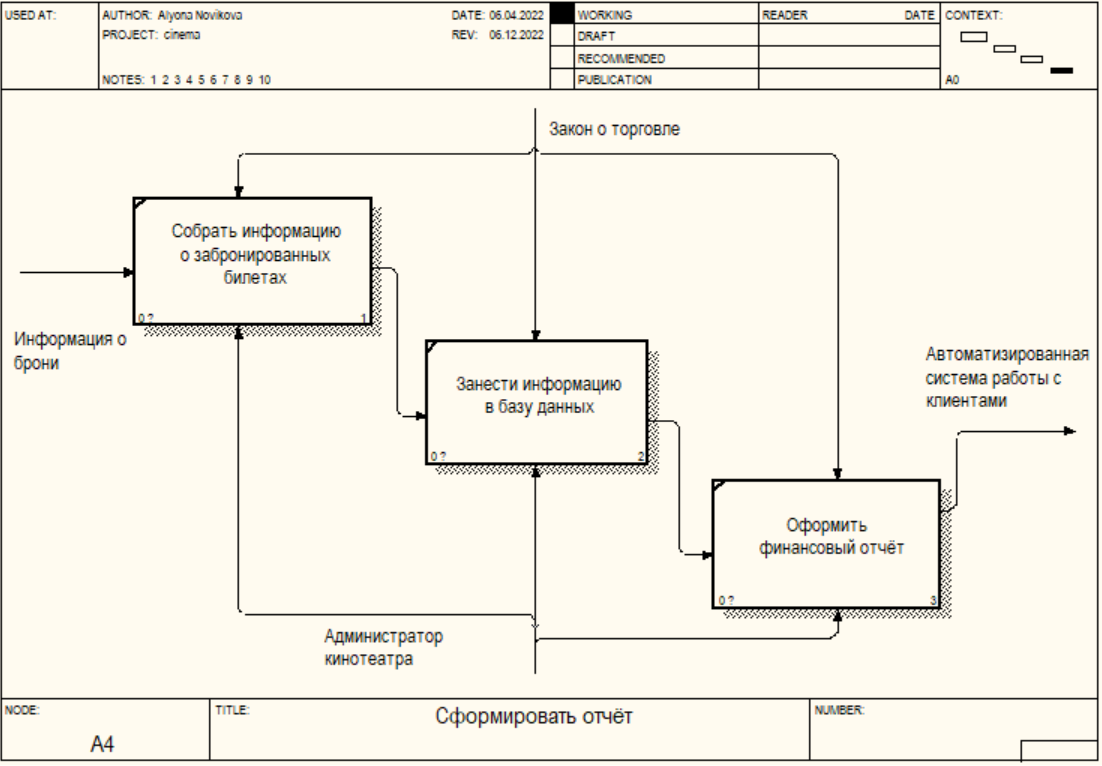


Рисунок 1.6 – Детализация блока «Сформировать отчёт»

Перед проектированием программного средства следует определить, какие требования должны быть предъявлены к нему, так как невыполнение некоторых требований, которые были сформированы еще на ранней стадии, говорит о том, что разработанный сервис не сможет эффективно и с максимальной пользой использоваться, как было задумано изначально.

## 1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Для разработки программного средства необходимо понять принцип работы кинотеатра.

Так как в системе предполагается хранение данных о кинофильмах и залах, а также информации о клиентах, необходимо спроектировать базу данных, которая осуществляла бы их хранение в электронном виде. Это позволит облегчить поиск информации для администратора. Для корректной работы необходимо предусмотреть возможность добавления, удаления и редактирования информации в базе данных. Для хранения информации будет использован MYSQLServer. Подключение к нему будет осуществляться при авторизации и в зависимости от полученных прав пользователь, будет получать определённый набор возможностей для выполнения своей работы.

Регистрация клиента в систему будет осуществляться после получения данных (ФИО, логин, пароль, электронная почта). По итогу регистрации к пользователю прикрепляется идентификационный номер клиента (генерируется автоматически), а также номер статуса (1 – Администратор, 2 – Пользователь). Номер статуса позволяет регулировать набор функций для того или иного пользователя.

Администратор имеет полномочия формировать киносеансы, вносить информацию о залах в кинотеатре, просматривать заявки на бронь, а также изучать информацию профиля и, в случае надобности, менять пароль.

Пользователь имеет возможность просматривать информацию о киносеансах, оставлять заявку на бронь и просмотреть данные аккаунта.

Суть диаграммы вариантов использования состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью вариантов использования. Для взаимодействия с программным средством существует 2 актера: администратор и пользователь. Как видно из диаграммы, набор доступных им возможностей отличается. Администратор имеет полномочия формировать киносеансы, вносить информацию о залах в кинотеатре, просматривать заявки на бронь, а также изучать информацию профиля и, в случае надобности, менять пароль. Пользователь имеет возможность просматривать информацию о киносеансах, оставлять заявку на бронь и просмотреть данные аккаунта.

Диаграмма использования программного средства, реализованного в данном курсовом проекте, представлена на рисунке 1.7.

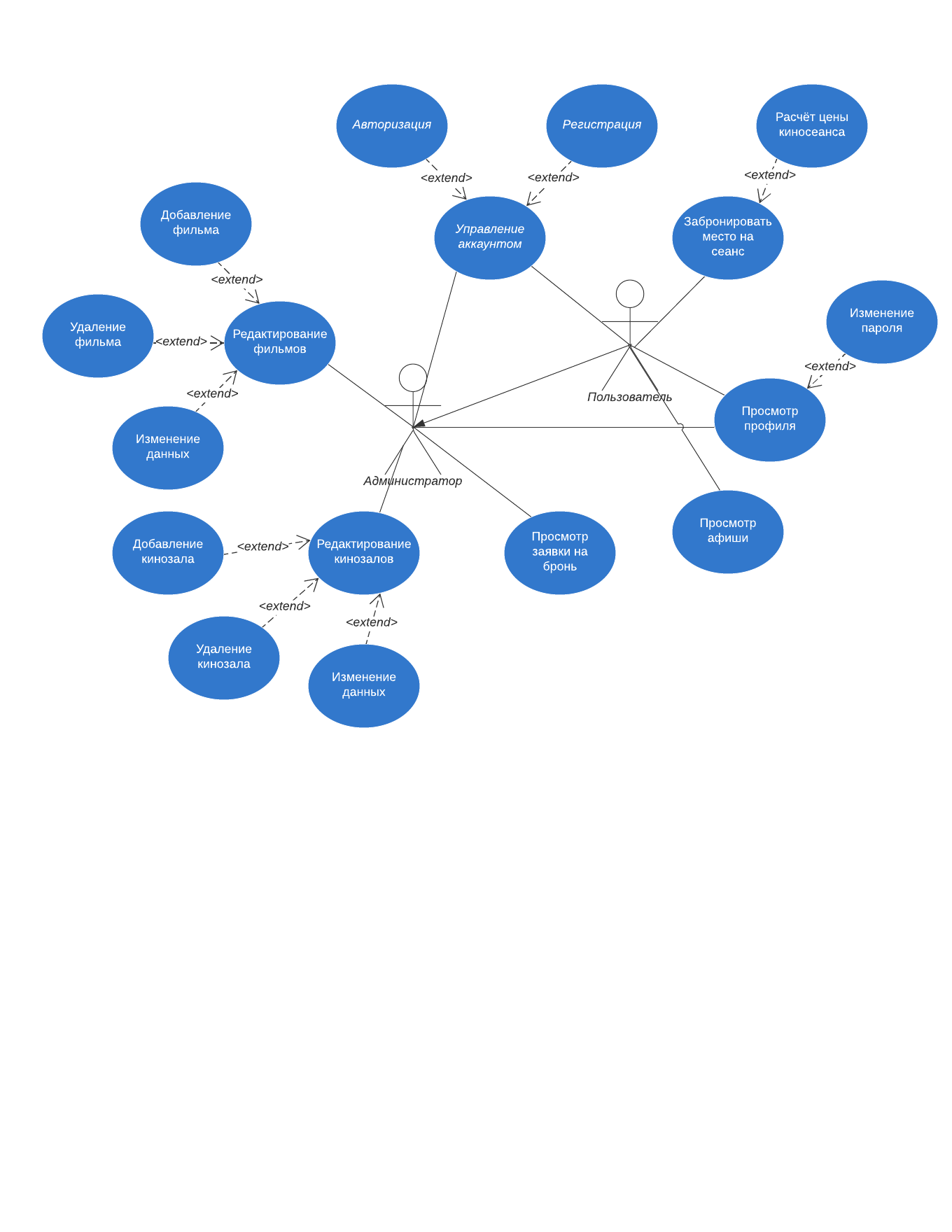


Рисунок 1.7 – Диаграмма вариантов использования

## 1.4 Разработка информационной модели предметной области

При проектировании системы было принято решение использовать следующие сущности:

* *User*;
* *CinemaHall*;
* *Movie*;
* *UserStatus*;
* *Booking*.



Рисунок 1.8 – Информационная модель системы

Ниже представлено подробное описание всех сущностей, входящих в модель.

Сущность *User* содержит в себе следующие атрибуты:

* *userId* – хранит уникальный идентификационный номер авторизованного пользователя;
* *username* – отвечает за хранение логина пользователя;
* *firstname* – хранит имя;
* *lastname* – хранит фамилию;
* *statusId (FK)* – хранит идентификационный номер статуса;
* *email* – хранит электронную почту.

Сущность *UserStatus* содержит в себе следующие атрибуты:

* *statusId* – хранит уникальный идентификационный номер статуса;
* *statusName* – хранит имя статуса (Администратор/Пользователя).

Сущность *Movie* содержит в себе следующие атрибуты:

* *movieId* – хранит уникальный идентификационный номер фильма;
* *movieName –* хранит название фильма;
* *movieGenre –* хранит жанр фильма;
* *movieCountry –* хранит страну производства фильма;
* *movieDuration –* хранит длительность фильма;
* *moviePrice –* хранит цену за сеанс;
* *hallId (FK) –* хранит идентификационный номер кинозала.

Сущность *CinemaHall* содержит в себе следующие атрибуты:

* *hallId* – хранит уникальный идентификационный номер фильма;
* *hallType –* хранит тип кинозала;
* *hallSeatsNumber –* хранит количество мест в зале.

Сущность *Booking* содержит в себе следующие атрибуты:

* *bookingId* – хранит уникальный идентификационный номер брони;
* *movieId (FK) –* хранит идентификационный номер сеанса;
* *userId (FK) –* хранит идентификационный номер пользователя.

## 1.5 Модели представления программного средства и их описание

**1.5.1** Диаграмма состояний. Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

Диаграмма состояний для разработанного программного приложения отображена на рисунке 1.9.

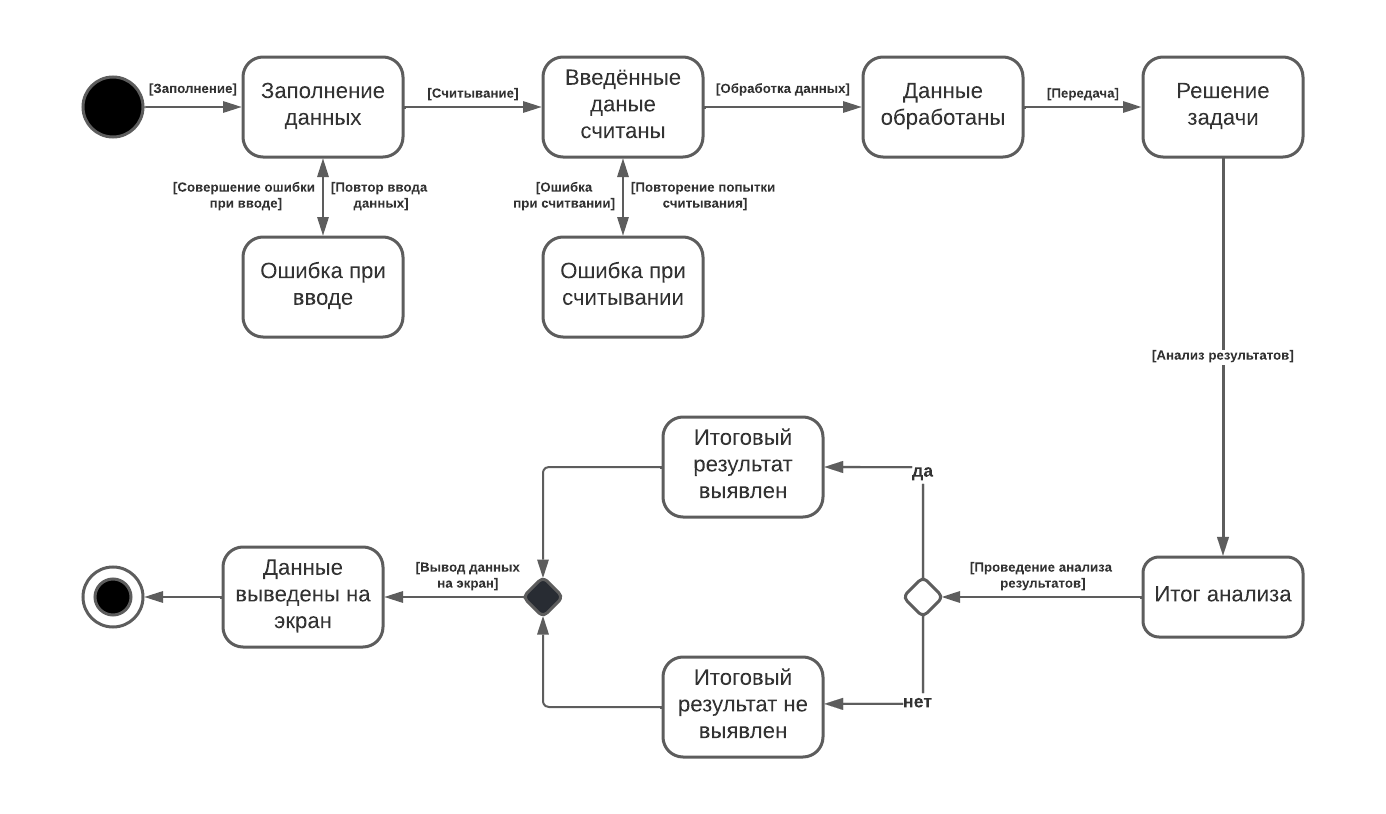


Рисунок 1.9 – Диаграмма состояний

Диаграммы состояний описывают динамическое представление объекта. Они особенно важны для моделирования поведения интерфейсов, классов или коопераций и подчеркивают событийно-зависимое поведение объекта, что особенно удобно для моделирования реактивных систем.

**1.5.2** Диаграмма последовательности. Диаграмма последовательности, представленная на рисунке 1.10, описывает процесс передачи данных серверу для дальнейшей обработки и записи их в таблицы баз данных. Сначала клиент посылает сигнал с запросом действия на сервер. Сервер после этого обращается к соответствующей функции, а она – к данным, хранящимся в объектах некоторого класса. При наличии необходимых данных в системе клиент получает результат работы функции, в противном случае – сообщение об отказе в действии.

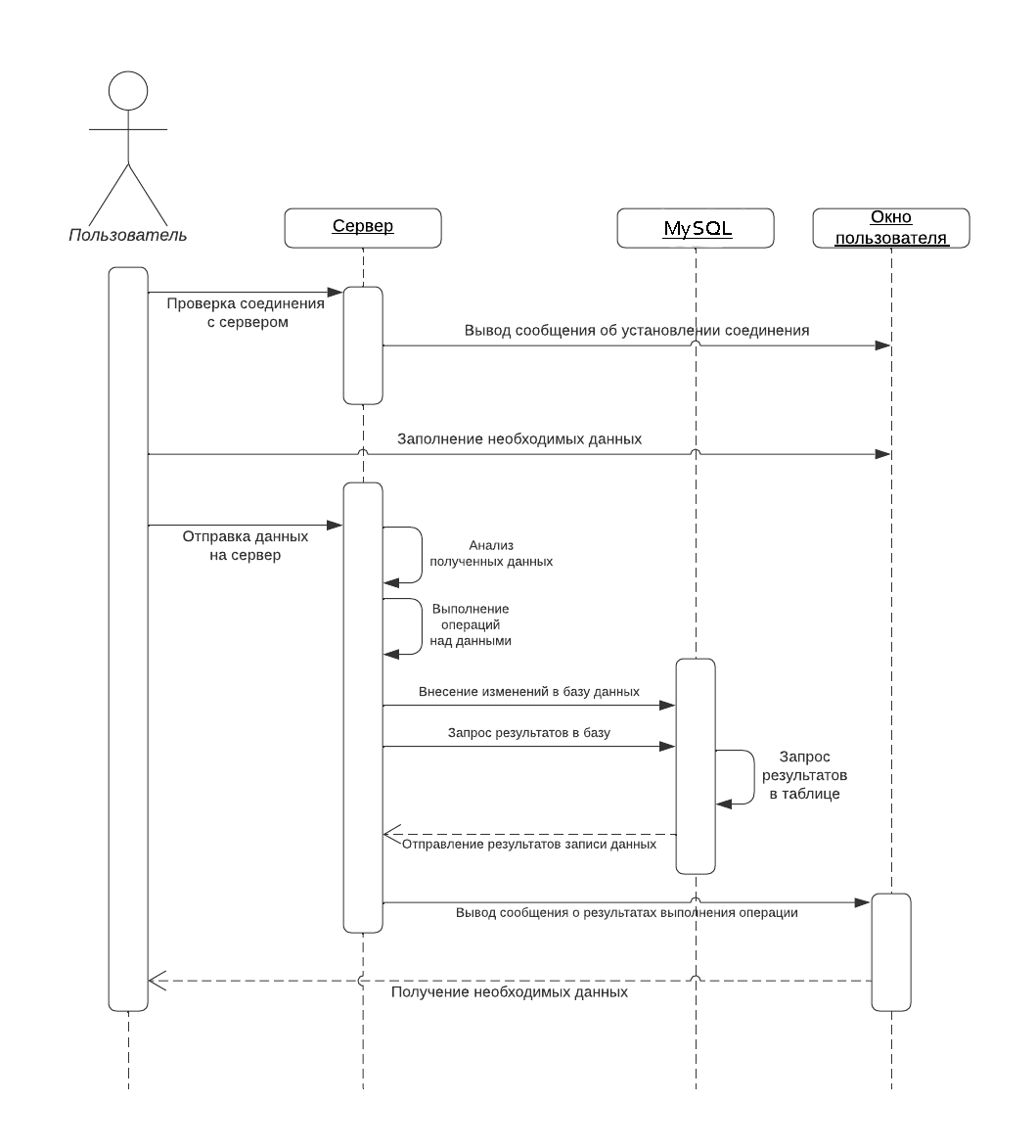


Рисунок 1.10 – Диаграмма последовательности передачи данных серверу

**1.5.3**Диаграмма развёртывания. С помощью диаграммы развертывания отобразим элементы и компоненты системы, существующие на этапе ее исполнения.

Диаграмму развертывания системы приведена на рисунке 1.11.

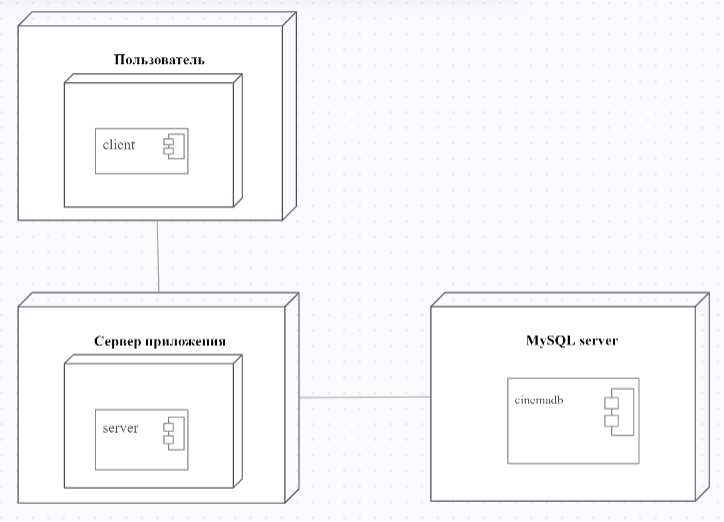


Рисунок 1.11 – Диаграмма развертывания

Таким образом, диаграмма развертывания предназначена для визуализации элементов и компонентов системы, существующих лишь на этапе ее исполнения (*runtime*), к которым относятся исполнимые файлы, динамические библиотеки, таблицы базы данных. Те компоненты, которые не используются на этапе исполнения (например, исходные тексты программ), на диаграмме не показываются.

Таким образом, в данной главе курсового проекта представлены моделирование предметной области и разработка требований к программному средству, анализ и формализация бизнес-процессов службы управления персоналом, разработаны спецификация функциональных требований, информационная модель программного средства, а также модели представления программного средства на основе языка *UML*.

# Проектирование и конструирование программного средства

## 2.1 Постановка задачи

Для разработки программного средства необходимо понять принцип работы кинотеатра.

Так как в системе предполагается хранение данных о кинофильмах и залах, а также информации о клиентах, необходимо спроектировать базу данных, которая осуществляла бы их хранение в электронном виде. Это позволит облегчить поиск информации для администратора. Для корректной работы необходимо предусмотреть возможность добавления, удаления и редактирования информации в базе данных. Для хранения информации будет использован MYSQLServer. Подключение к нему будет осуществляться при авторизации и в зависимости от полученных прав пользователь, будет получать определённый набор возможностей для выполнения своей работы.

Регистрация клиента в систему будет осуществляться после получения данных (ФИО, логин, пароль, электронная почта). По итогу регистрации к пользователю прикрепляется идентификационный номер клиента (генерируется автоматически), а также номер статуса (1 – Администратор, 2 – Пользователь). Номер статуса позволяет регулировать набор функций для того или иного пользователя.

Администратор имеет полномочия формировать киносеансы, вносить информацию о залах в кинотеатре, просматривать заявки на бронь, а также изучать информацию профиля и, в случае надобности, менять пароль.

Пользователь имеет возможность просматривать информацию о киносеансах, оставлять заявку на бронь и просмотреть данные аккаунта.

## 2.2 Архитектурные решения

Диаграммы классов используются при моделировании программных средств наиболее часто. На диаграммах классов показываются классы, интерфейсы и отношения между ними. В данном курсовом проекте были реализованы классы: *User*, *UserStatus, Movie, CinemaHall, Booking*. Все они используются для хранения информации о сущностях, присутствующих в системе. Класс *User* хранит в себе информацию о пользователе: поля *username, firstName, lastName, email*, которые содержат логин, имя, фамилию и электронную почту соответственно. Класс *UserStatus*, кроме уникального идентификационного номера (*statusId*), содержит поле *statusName* – тип статуса пользователя. Класс *Movie* содержит поля, хранящие информацию о киносеансе. Поля *movieName, movieGenre, movieCountry, movieDuration* содержат информацию о названии фильма, жанре, стране производства и длительности сеанса. Поле *moviePrice* хранит информацию о цене сеанса. Класс *CinemaHall* хранит поле *hallType* – тип кинозала и поле *hallSeatsNumber* – количество мест в зале. Класс *Booking* содержит поле *bookingId*, которое хранит уникальный идентификационный номер брони.

Диаграмма классов показана на рисунке 2.1.



Рисунок 2.1 – Диаграмма классов

Разрабатываемый программный продукт представляет собой клиент-серверное приложение для работы с СУБД *MySQLServer* через веб-сервис. Предполагается развертывание в организации, имеющей локальную вычислительную сеть с выделенным сервером. В таком случае на выделенном сервере устанавливается СУБД, на которой развертывается база данных системы, и создается директория для хранения документов. Экземпляры клиентских приложений размещаются на машинах сотрудников организации, на которых предварительно должен быть установлен *JDK* 1.8 и выше.

## 2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства

Рассмотрим алгоритмы реализации следующих методов, реализованных в данном курсовом проекте: авторизация, работа с программой, получение соединения с базой данных.

При запуске программы открывается главное окно с авторизацией. При успешной авторизации, система распознает идентификационный номер статуса пользователя и в зависимости от его значения предоставляет разный набор возможностей для работы в системе (см. рисунок 2.2).



Рисунок 2.2 – Схема алгоритма авторизации

Если выполнен вход под администратором, то пользователь имеет доступ к базам данных киносеансов, кинозалов, пользователей, заявок на бронь и имеет право добавлять, удалять и редактировать данные. А также имеет возможность просматривать личный профиль и изменять пароль аккаунта. В случае авторизации в качестве пользователя, то функционал программы предоставляет возможность просматривать базы данных киносеансов и залов, создавать заявку на бронь сеанса и просматривать профиль с возможностью изменения пароля (см. рисунок 2.3).



Рисунок 2.3 – Схема алгоритма работы с программой

Блок-схема алгоритма получения соединения с базой данных представлена на рисунке 2.4.

Для начала мы получаем соединение с БД. Далее если соединение получено, то происходит формирование запроса, потом установка параметра id в запрос. После этого запрос выполняется и если он выполнен, то происходит получение и возврат результата.

Если соединение не было установлено, то выбрасывается исключение.

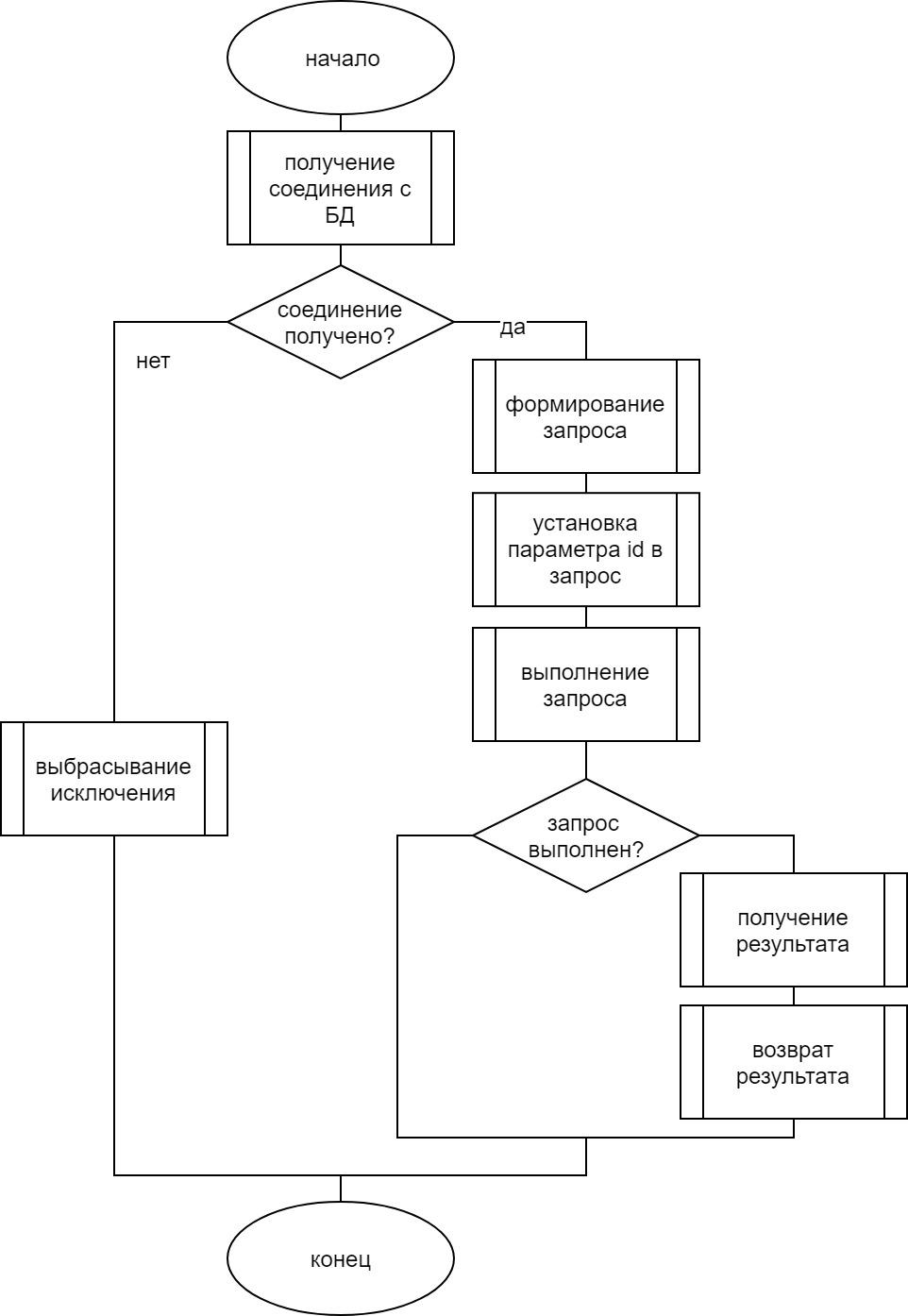


Рисунок 2.4 – Схема алгоритма соединения с БД

## 2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс – это набор программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи. Каждый диалог состоит из отдельных процессов ввода/вывода, которые физически обеспечивают связь пользователя и компьютера. Обмен информацией осуществляется передачей сообщения.

Приложение имеет оконный интерфейс, разработанный с использованием *JavaFX*. JavaFX предоставляет богатый набор графических и мультимедийных API интерфейсов, а также использует современный графический процессор с помощью аппаратно-ускоренной графики. JavaFX также предоставляет интерфейсы, с помощью которых разработчики могут комбинировать графическую анимацию и управление пользовательским интерфейсом.

Главное окно администратора содержит вкладки, предоставляющие доступ к различным категориям сущностей, участвующих в бизнес-процессе – данные фильмов, кинозалов, пользователей, заявки на бронь и профиль аккаунта (см.рисунок 2.5).

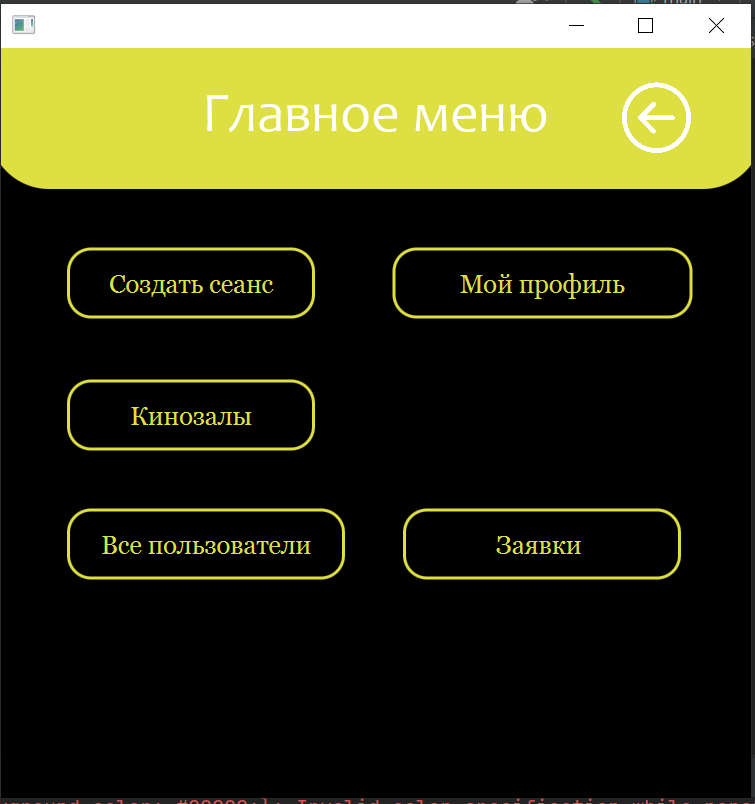


Рисунок 2.5 – Главное окно администратора

Окно с таблицей кинозалов предоставляет таблицу с данными и кнопками «Добавление» и «Удаление», которые реализуют соответствующие функции. При двойном клике левой кнопкой мыши по строке таблицы – администратор имеет возможность изменить данные.

Окно с кинофильмами имеет те же функции, а также есть возможность добавления информации в текстовый файл (см. рисунки 2.6 – 2.7).

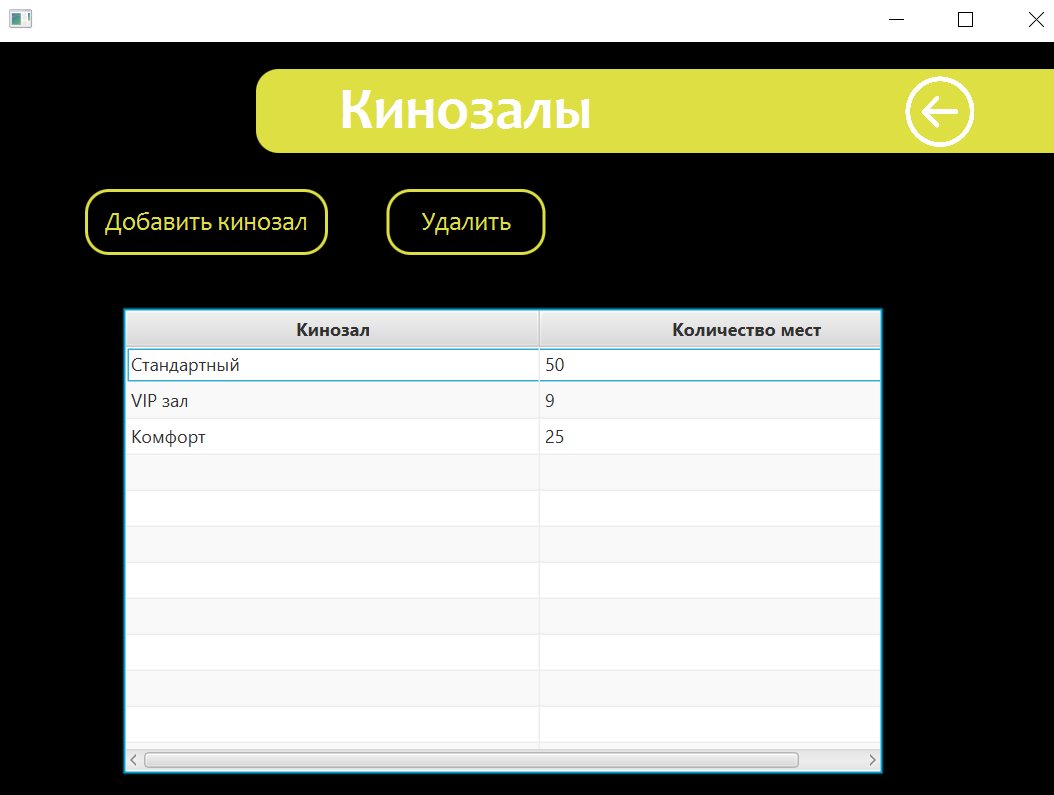


Рисунок 2.6 – Окно кинозалов

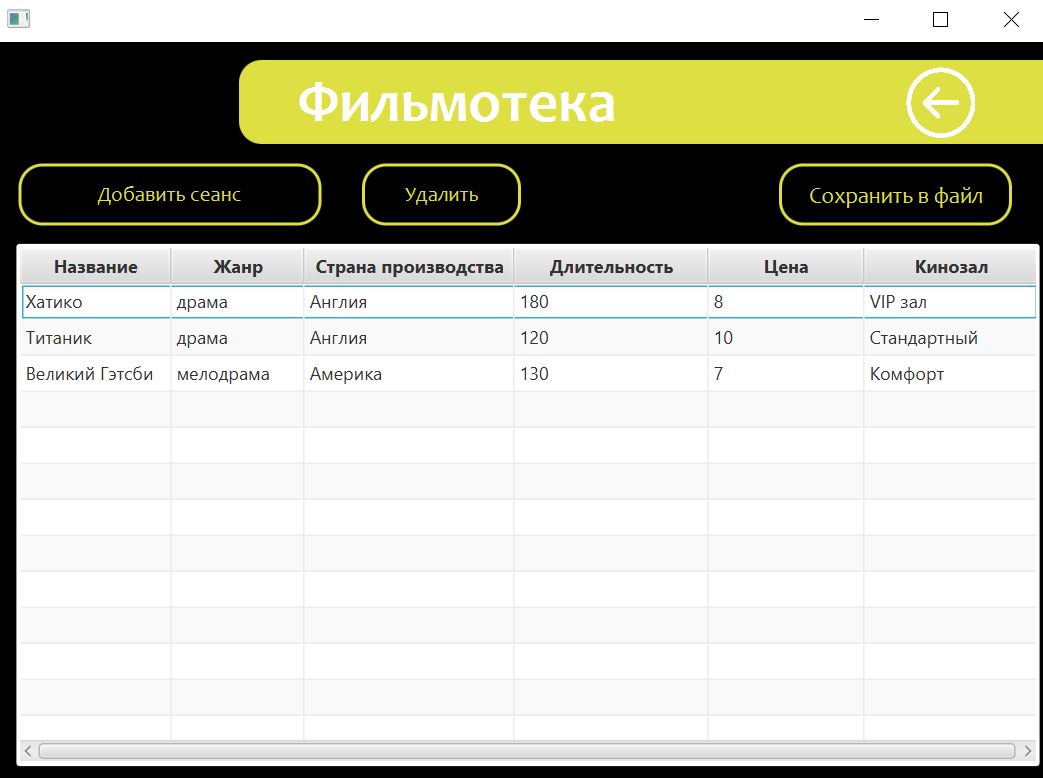


Рисунок 2.7 – Окно киносеансов

Окно «Профиль» выводит личные данные аккаунта и кнопку для изменения пароля пользователя.

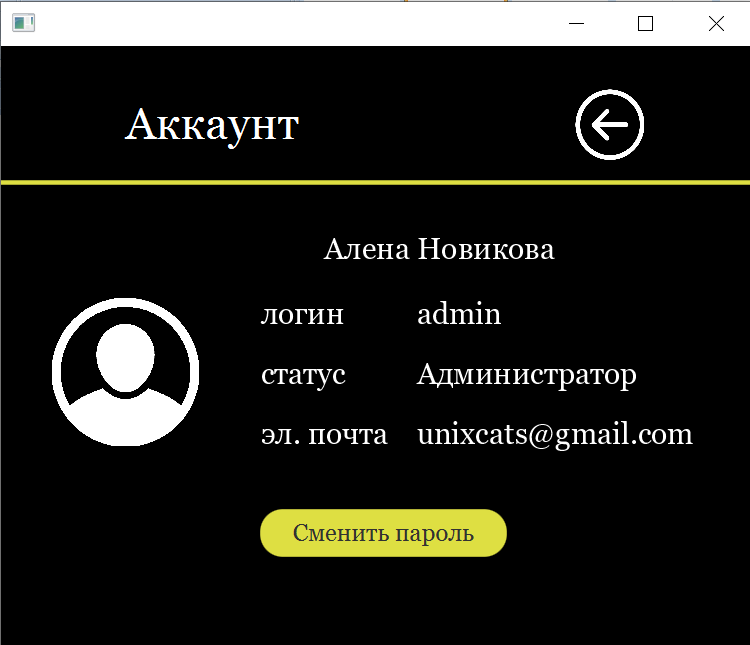


Рисунок 2.8 – Окно профиля

Все компоненты взаимодействуют друг с другом и связаны. Так же могут работать по отдельности. Для каждой части разработана таблица в которой содержатся и хранятся данные.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

**2.5 Обоснование выбора компонентов и технологий для реализации программного средства**

Данное программное приложение реализовано на языке Java в среде разработки IntelliJ IDEA.

Программное обеспечение JetBrainsIntelliJ IDEA – это ведущая среда быстрой разработки на языке Java. IntelliJ IDEA представляет собой высокотехнологичный комплекс тесно интегрированных инструментов программирования, включающий интеллектуальный редактор исходных текстов с развитыми средствами автоматизации, мощные инструменты рефакторинга кода, встроенную поддержку технологий J2EE, а также инновационный визуальный конструктор графических интерфейсов.

Уникальные возможности IntelliJ IDEA избавляют программиста от груза рутинной работы, помогают своевременно устранить ошибки и повысить качество кода, поднимая продуктивность разработчика на новую высоту.

Java – это объектно-ориентированный [язык программирования](https://web-creator.ru/articles/server_side_languages), разработанный компанией SunMicrosystems (в последующем приобретённой компанией Oracle).

Программы на Java транслируются в байт-код, который затем выполняется виртуальной машиной Java (JVM). JVM – это программа, которая обрабатывает байтовый код и передает инструкции оборудованию как интерпретатор. Достоинством подобной реализации является независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует JVM.

Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Для того, чтобы хранить данные мы используем СУБД MySQL.

MySQL– это реляционная система управления базами данных с открытым исходным кодом. В настоящее время эта СУБД одна из наиболее популярных в веб-приложениях  – подавляющее большинство CMS использует именно MySQL (часто только её, без альтернатив), а почти все веб-фреймворки поддерживают MySQL уже на уровне базовой конфигурации (без дополнительных модулей).

Из преимуществ СУБД MySQL стоит отметить простоту использования, гибкость, низкую стоимость владения (относительно платных СУБД), а также масштабируемость и производительность.

MySQL позволяет хранить целочисленные значения со знаком и беззнаковые, длиной в 1, 2, 3, 4 и 8 байтов, работает со строковыми и текстовыми данными фиксированной и переменной длины, позволяет осуществлять SQL-команды select, delete, insert, replace и update, обеспечивает полную поддержку операторов и функций в select- и where- частях запросов, работает с group by и order by, поддерживает групповые функции count(), avg(), std(), sum(), max() и min(), позволяет использовать join в запросах, поддерживает репликацию, транзакции, работу с внешними ключами и каскадные изменения на их основе, а также обеспечивает многие другие функциональные возможности.

# Тестирование и проверка работоспособности программного средства

В результате создания программного средства для оптимизации работы кинотеатра, стоит убедиться, что оно может корректно выполнять работу и будет устойчивым к ошибкам пользователя. Было проведено тестирование программы, отражающее возможные исключительные ситуации.

Для начала представлена некорректная авторизация пользователя. Введён неверный логин или пароль, программа реагирует на ситуацию и возвращает пользователю окно с указанием ошибки. Иллюстрация работы программы приведена на рисунке 3.1.

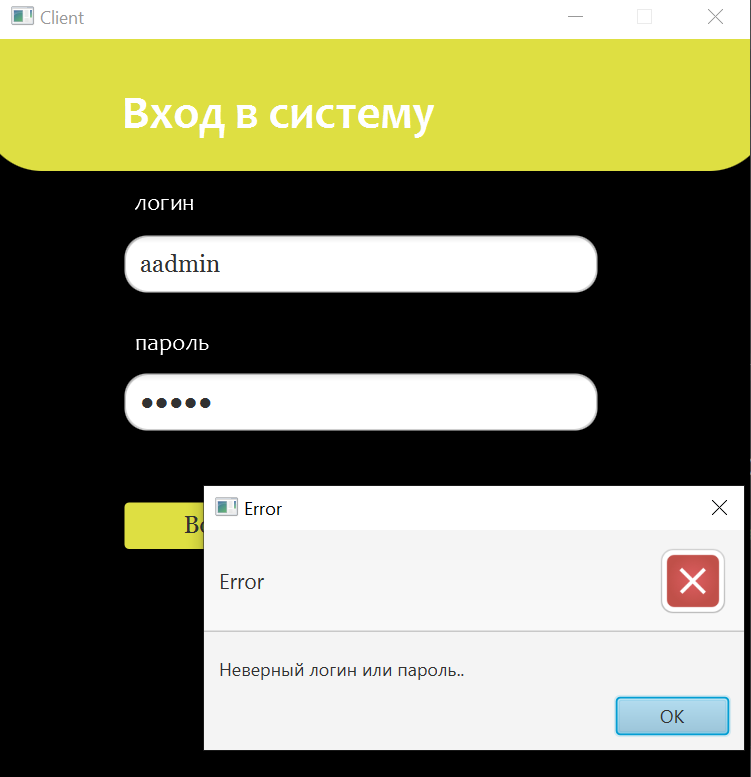


Рисунок 3.1 – Попытка входа с указанием неверных данных

Далее была протестирована команда изменения пароля в личном профиле пользователя. Программа реагирует на три ситуации:

* текущий пароль введён неверно;
* несовпадение введённых паролей;
* новый пароль не проходит валидацию.

Действие программы отражено на рисунках 3.2 – 3.4.

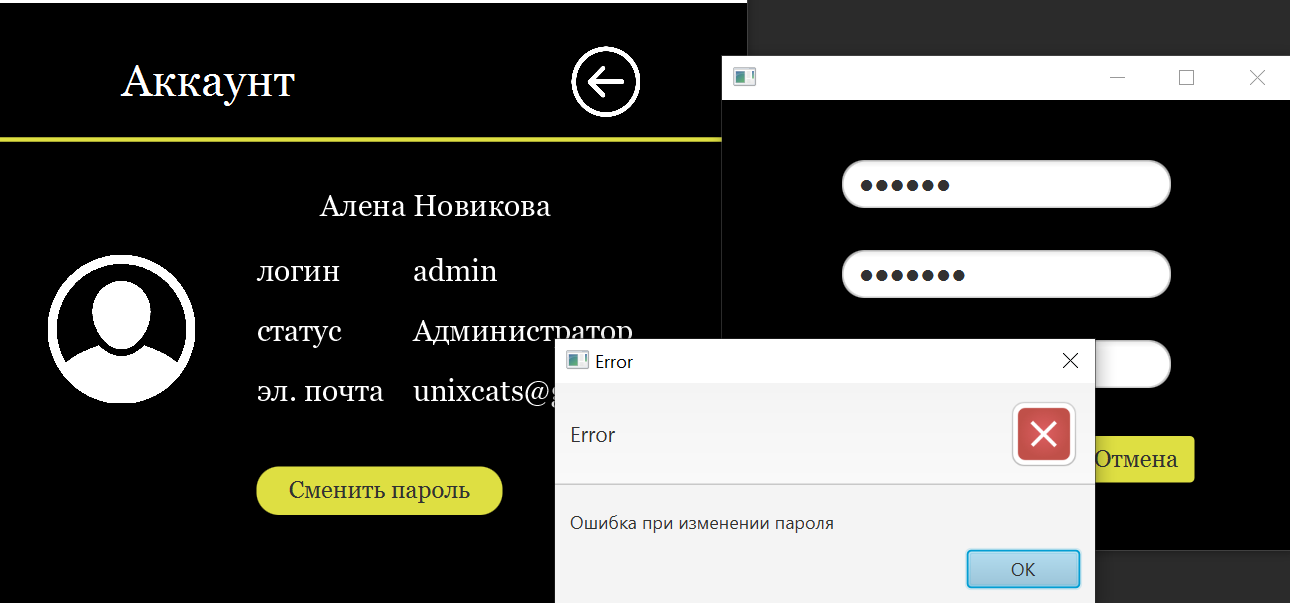


Рисунок 3.2 – Попытка ввода неверного текущего пароля

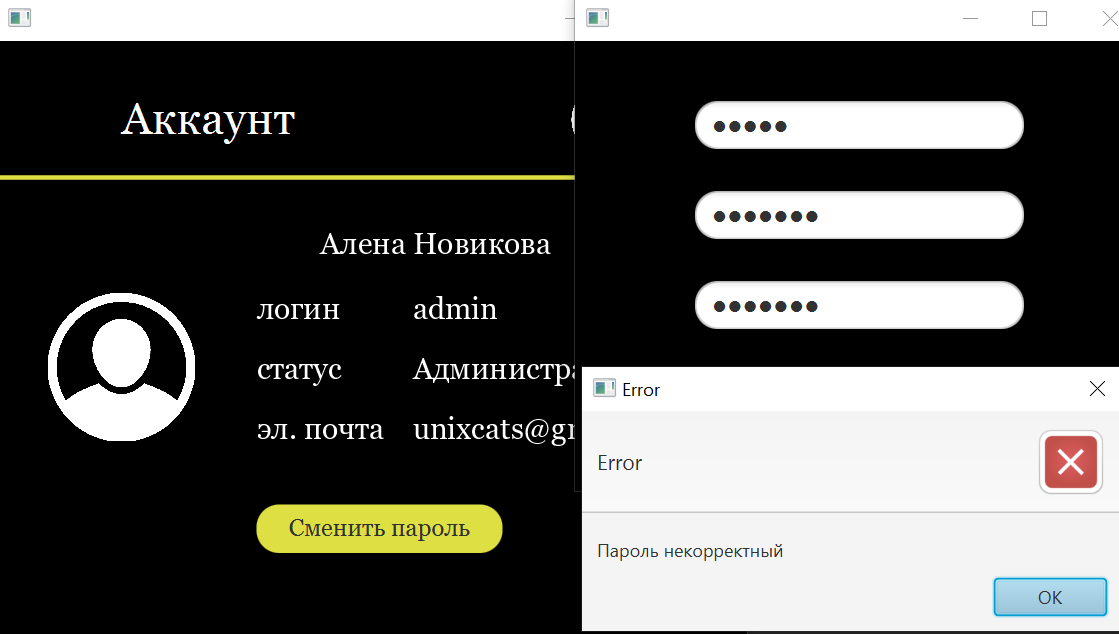


Рисунок 3.3 – Попытка ввода несовпадающих паролей

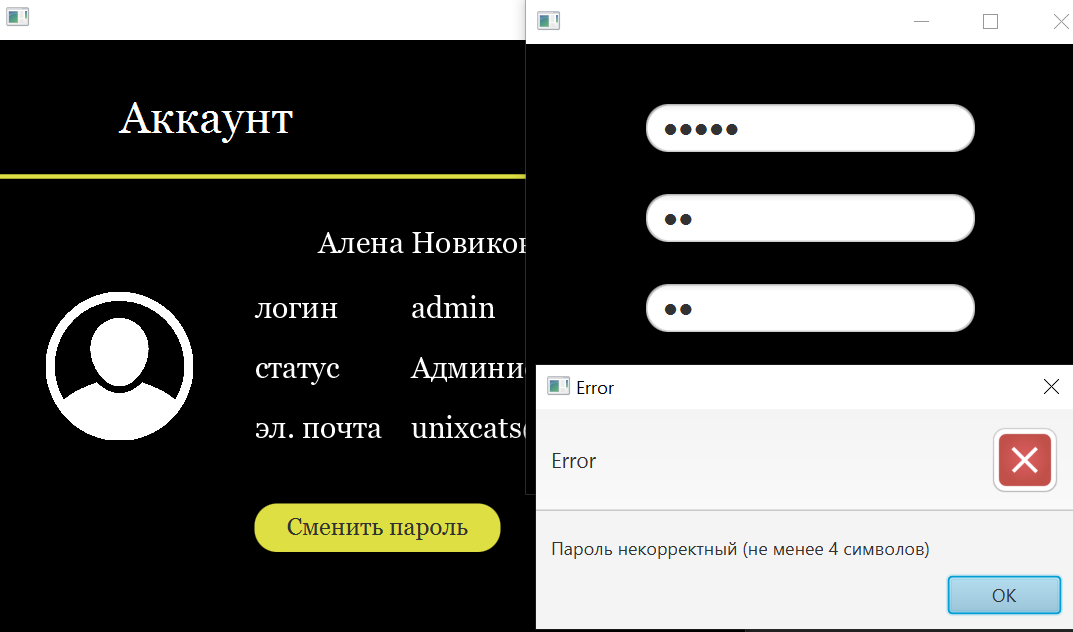


Рисунок 3.4 – Попытка ввода некорректного нового пароля

Далее было проведено тестирование программы при регистрации нового пользователя. При введении некорректных данных, программа обрабатывает несколько исключительных ситуаций:

* логин короче 4 символов или состоит не из латинских букв;
* имя и фамилия короче 2 символов;
* пароль короче 4 символов или состоит не из латинских букв;
* электронная почта не соответствует корректному формату;
* введённые пароли не совпадают.

Система выводит окно с ошибкой и объяснением возможных ошибок пользователя (см. рис. 3.5).

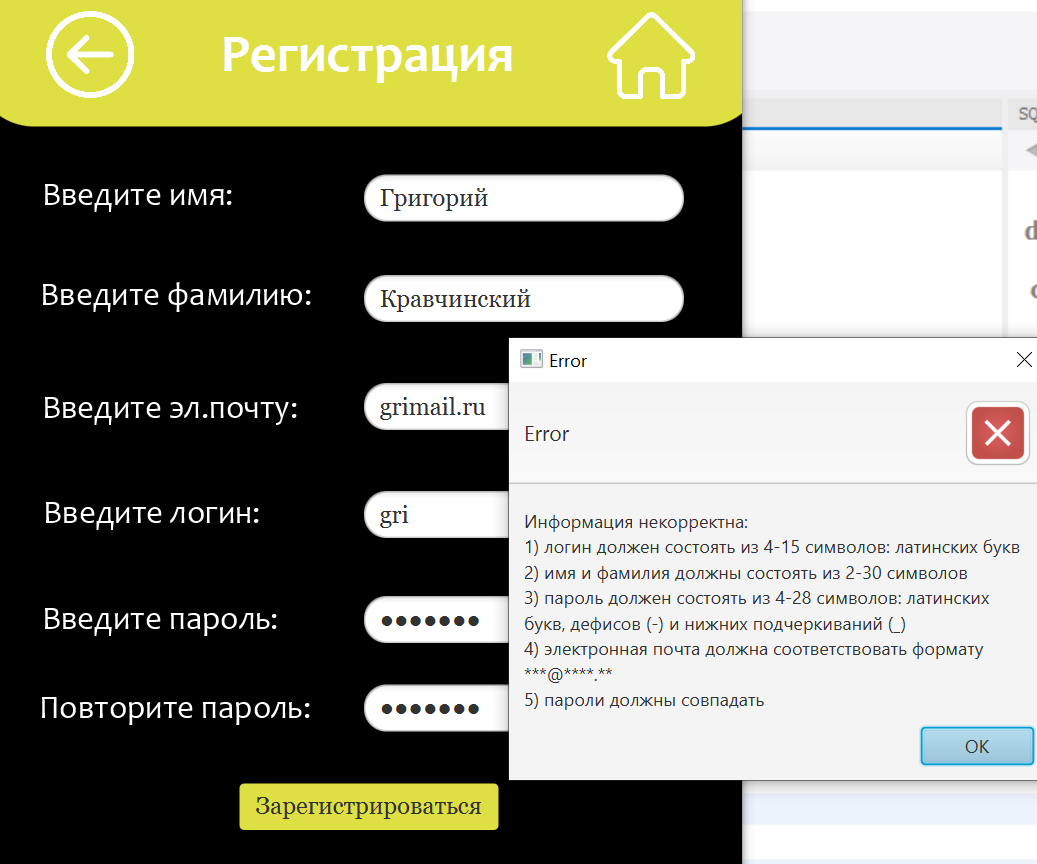


Рисунок 3.5 – Попытка ввода некорректных данных при регистрации

# Руководство по развертыванию и использованию программного средства

В первую очередь необходимо запустить сервер проекта. Для этого пользователь должен запустить проект Server. На экране отобразится экран с текущими подключениями и количеством клиентов в системе (см.рис. 4.1).

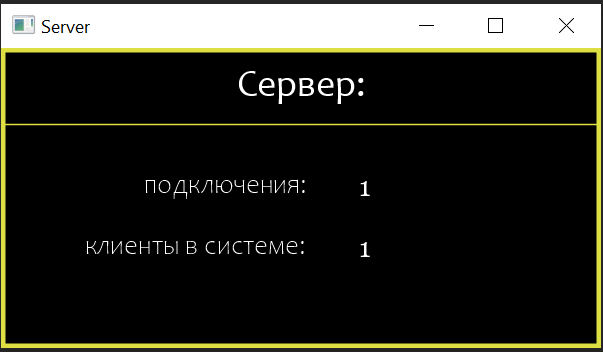


Рисунок 4.1 – Запуск сервера

После этого требуется запустить проект Client. Перед пользователем отобразится экран входа в систему, где предоставляется возможность пройти авторизацию, введя логин и пароль, или регистрацию пользователя, с помощью нажатия на соответствующую кнопку (см.рис. 4.2).

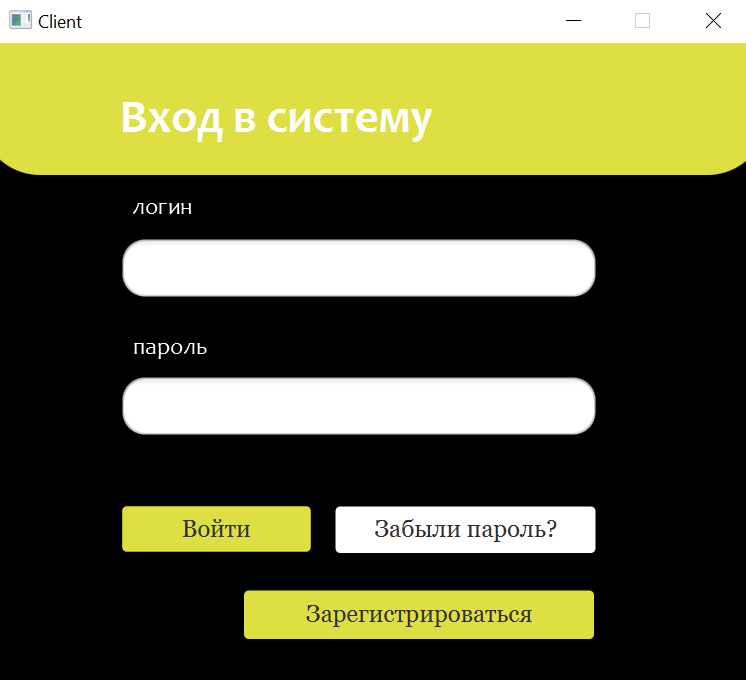


Рисунок 4.2 – Экран авторизации

При нажатии на кнопку «Зарегистрироваться», пользователь переходит на экран регистрации нового пользователя. В данном случае представлены поля для введения соответствующих данных и кнопка их подтверждения (см.рис. 4.3).

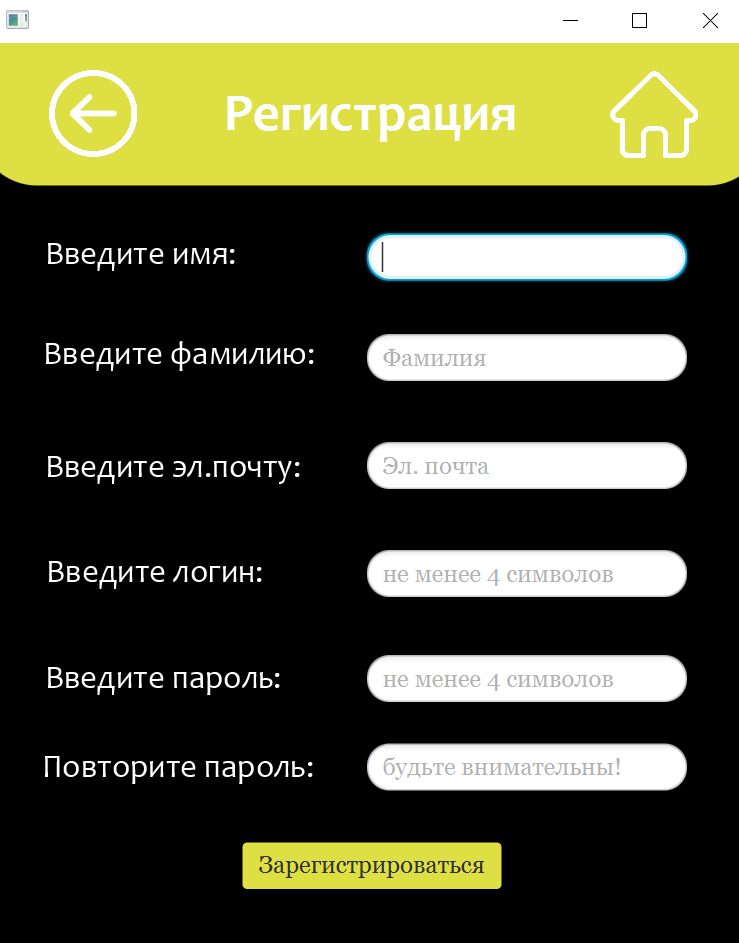


Рисунок 4.3 – Экран регистрации

При корректном введении данных и совпадении логина и пароля с зарегистрированным в базе пользователем, система выведет сообщение об успешном входе в систему и предоставит экран с соответствующим функционалом, в зависимости от статуса пользователя (см.рис. 4.4).

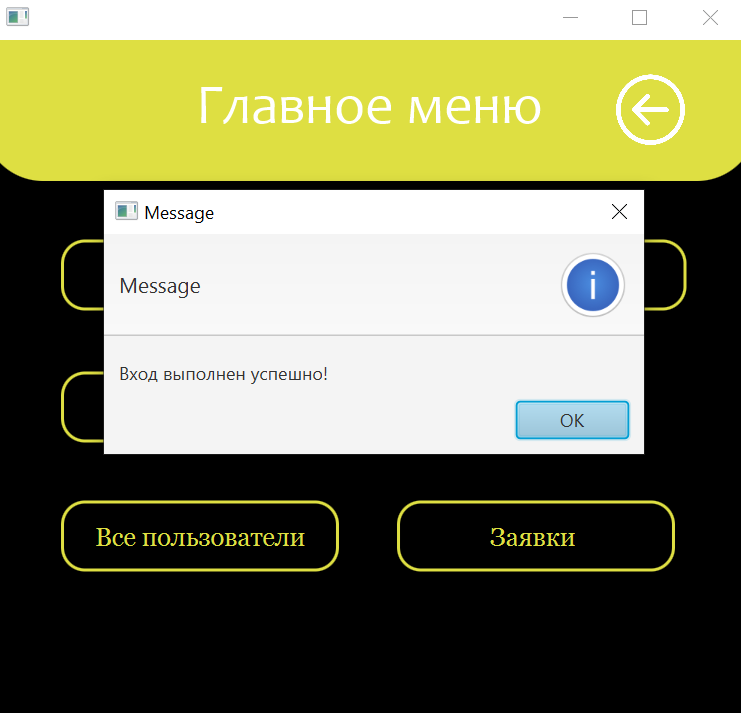


Рисунок 4.3 – Успешный вход в систему

При входе в систему в качестве администратора, функционал главного экрана будет представлен соответствующим функционалом в виде кнопок: «Создать сеанс», «Кинозалы», «Все пользователи», «Мой профиль», «Заявки» (см.рис. 4.4).

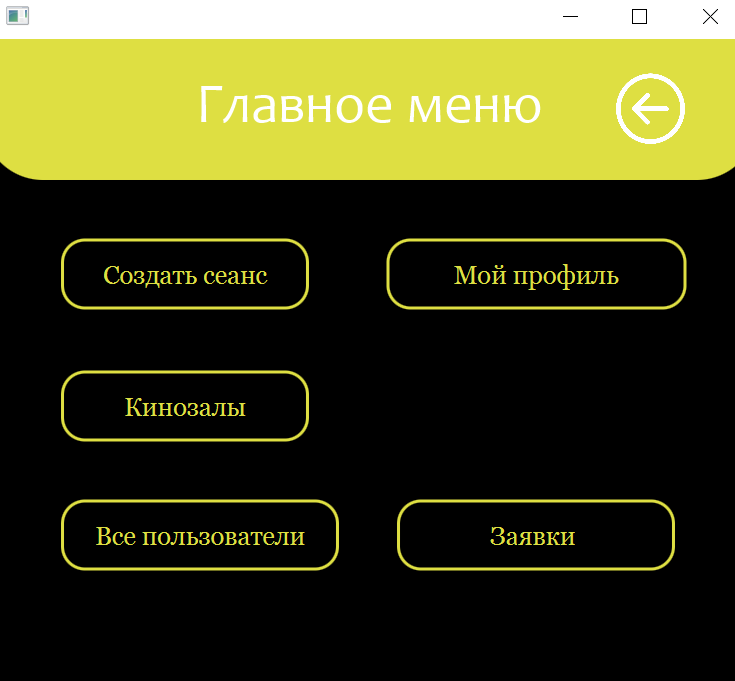


Рисунок 4.4 – Главный экран администратора

При нажатии на кнопку «Создать сеанс» администратору предоставляется окно с таблицей актуальных сеансов и возможность добавить сеанс, удалить любой из таблицы, а также сохранить данные в текстовый файл (см.рис. 4.5).

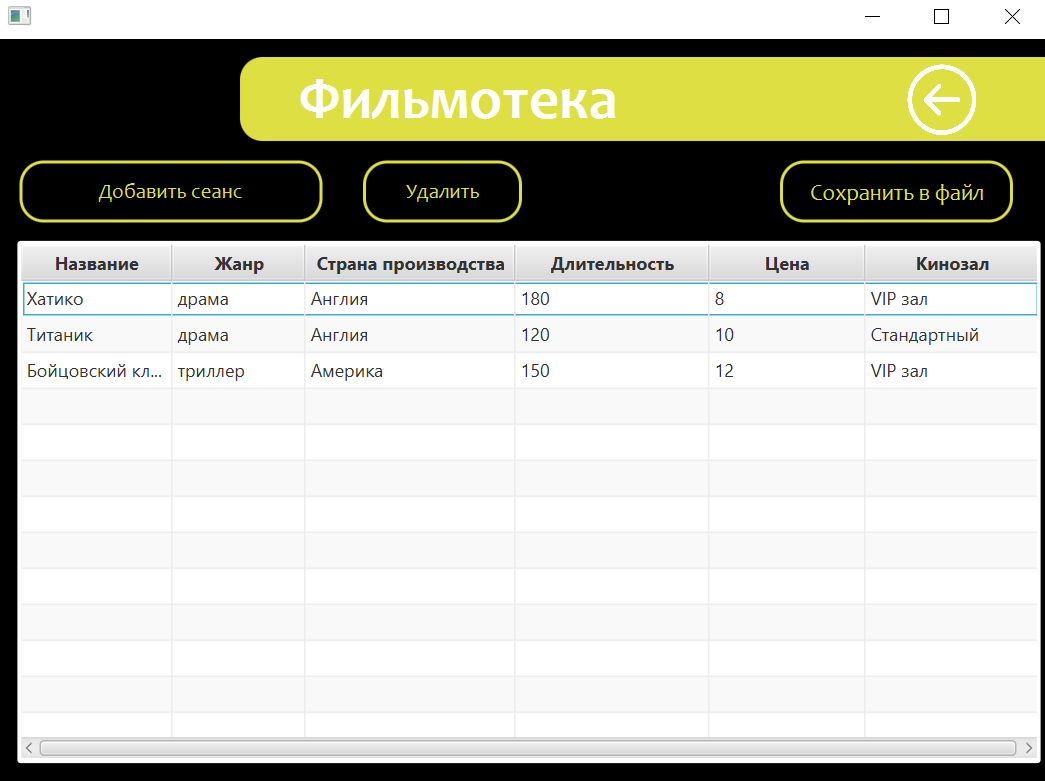


Рисунок 4.5 – Экран киносеансов

Администратор может сформировать киносеанс. После нажатия на кнопку «Добавить сеанс» система отобразит окно с полями, которые необходимо заполнить данными (см.рис. 4.6).

После сохранения, данные сразу будут добавлены в таблицу (см.рис. 4.7).

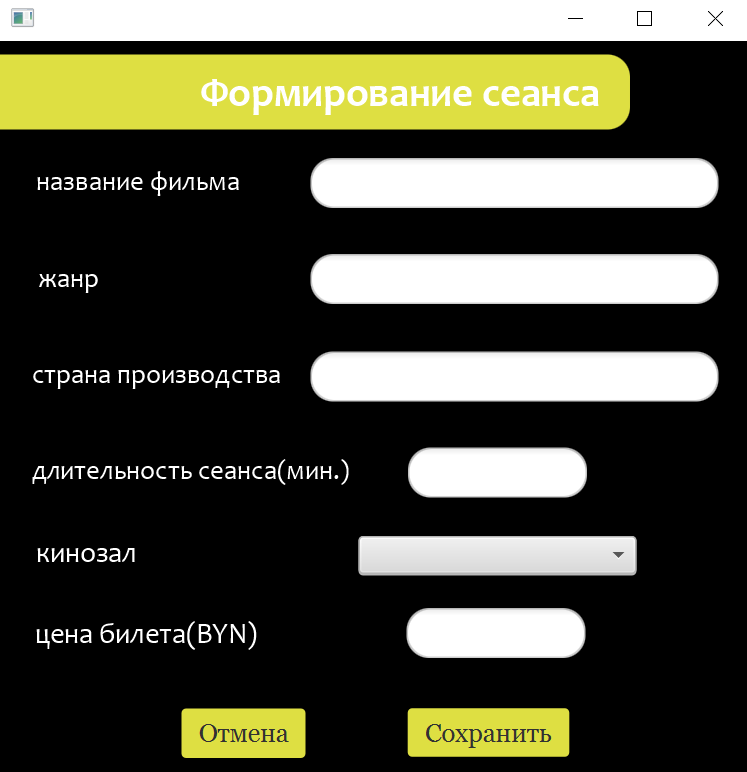


Рисунок 4.6 – Добавление киносеансов

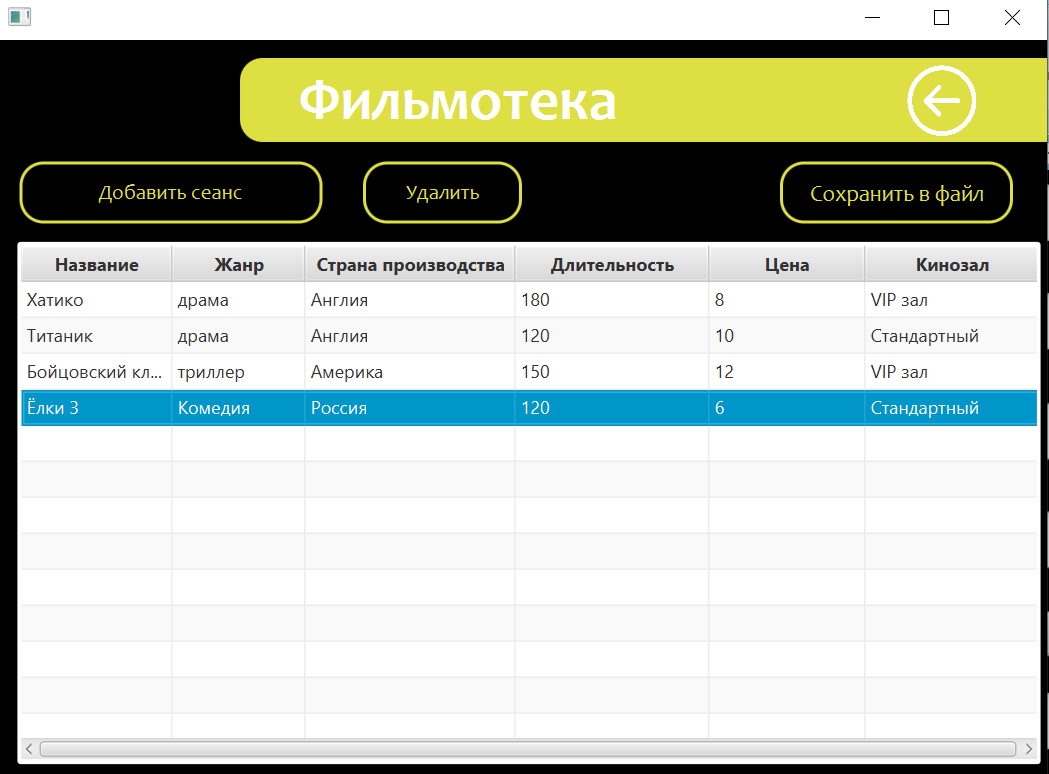


Рисунок 4.7 – Экран киносеансов

Администратору доступна функция работы с базой данных залов кинотеатра. При открытии окна, программа предоставляет таблицу с данными, а также кнопки добавления и удаления полей (см.рис. 4.8). Окно добавления кинозала представлено на рисунке 4.9.

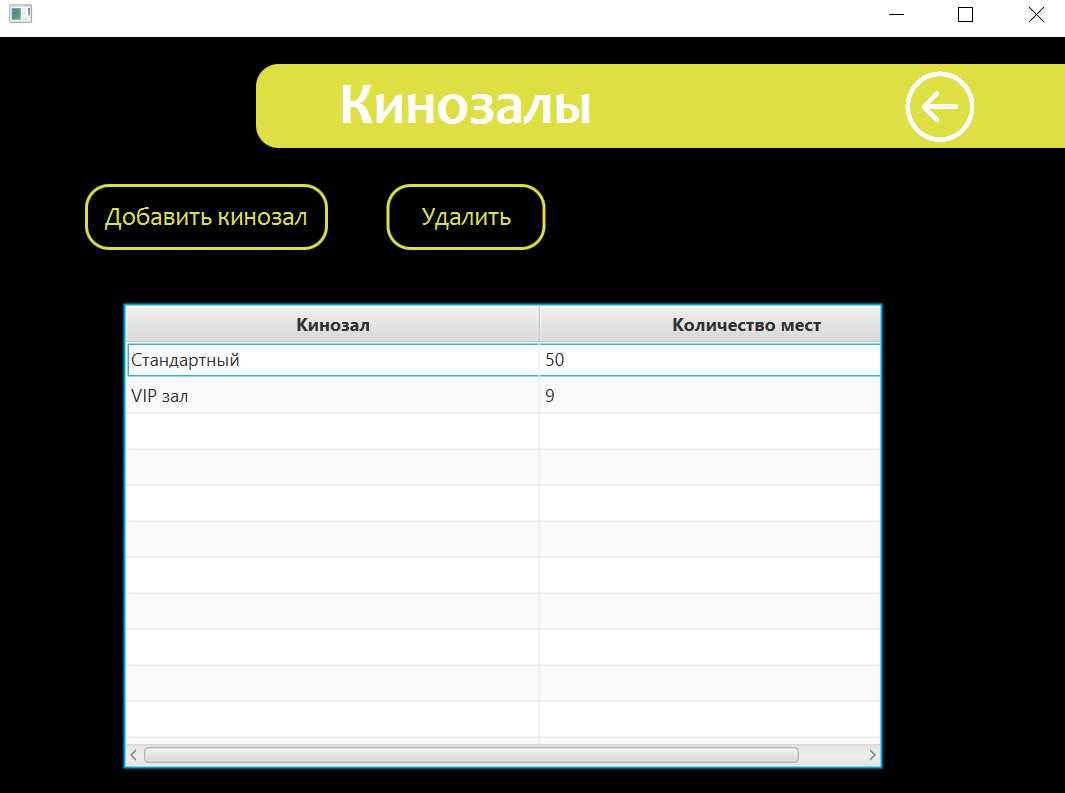


Рисунок 4.8 – Экран кинозалов

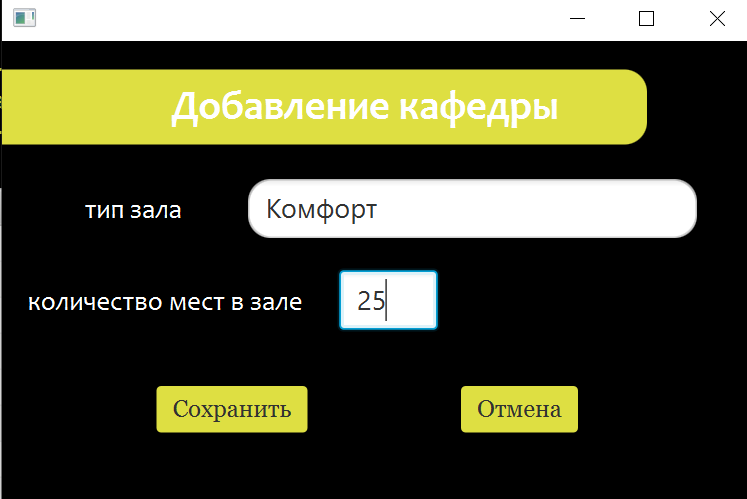


Рисунок 4.9 – Добавление кинозала

Вход в систему в качестве администратора предоставляет возможность работы с базой данных всех пользователей системы (рис. 4.10).



Рисунок 4.10 – Окно пользователей системы

Любой пользователь системы, независимо от статуса, может просматривать личную информацию профиля (см.рис. 4.11) и изменять пароль (см.рис. 4.12).

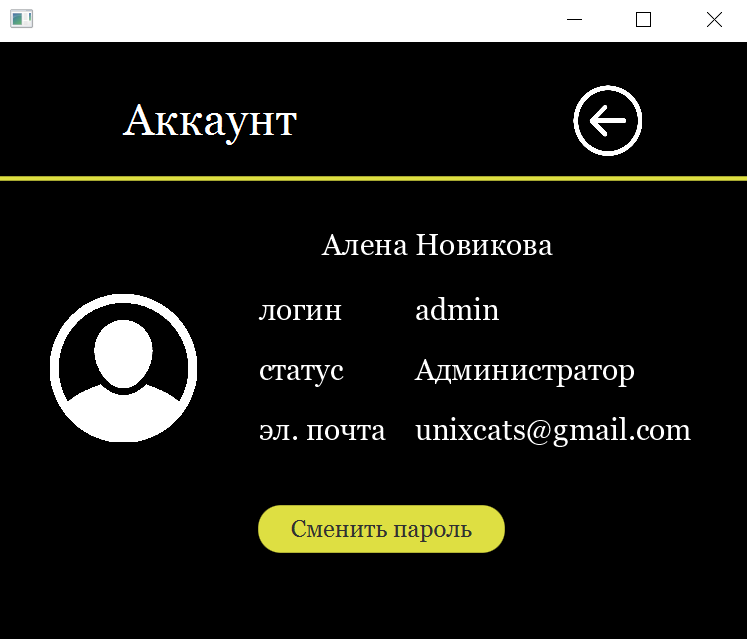


Рисунок 4.11 – Данные аккаунта

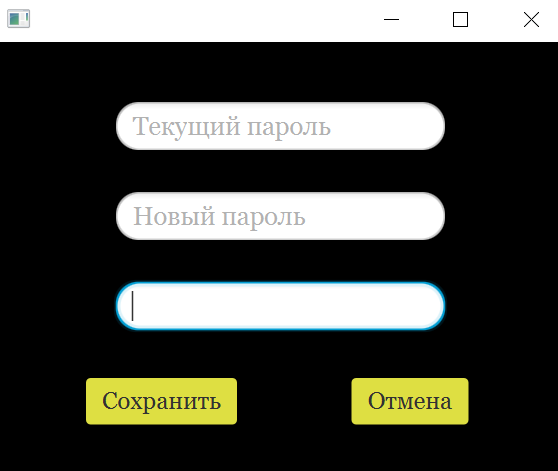


Рисунок 4.12 – Изменение пароля

При входе в систему в качестве клиента, главное меню выглядит следующим образом (см.рис. 4.13).

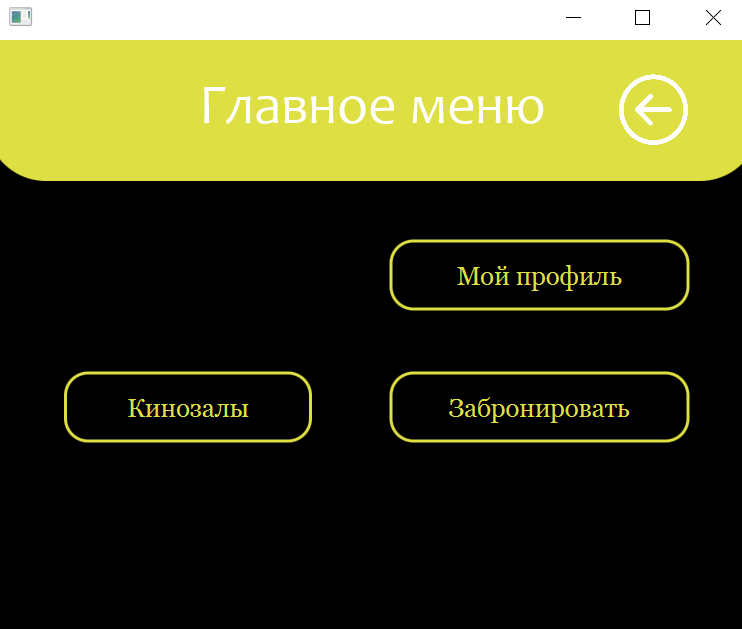


Рисунок 4.13 – Главное меню клиента

Клиент программы, в отличие от администратора, имеет возможность забронировать билет на определённый сеанс кинотеатра. Система представляет пользователю афишу. Выбор интересующего сеанса и нажатие на кнопку формирует заявку на бронь, которая в последующем будет отображена администратору (см.рис. 4.14).

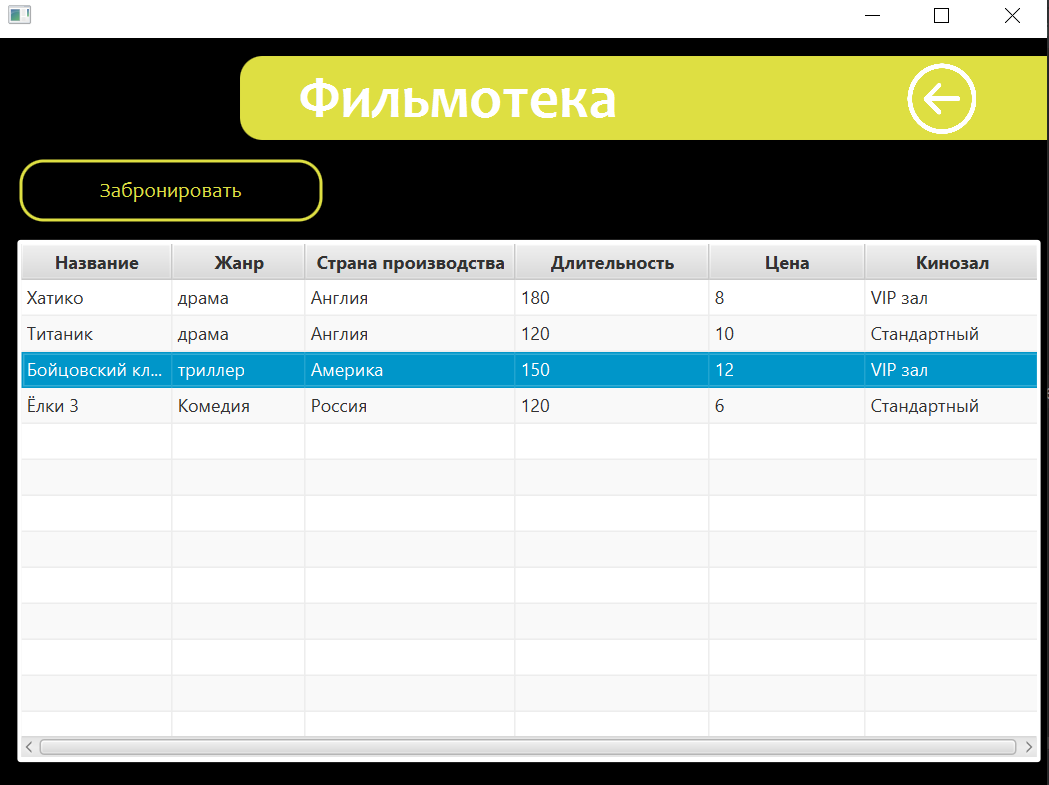


Рисунок 4.14 – Создание заявки на бронь сеанса

# Заключение

Целью данного курсового проекта являлось создание автоматизированной системы работы кинотеатра, использование которого обеспечит удобное хранение информации о фильмах, кинозалов, пользователей системы и заявок на бронь от клиентов.

Приложение удовлетворяет основным характеристикам, которые были заявлены. Оно удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области.

Использование данной программы в реальном кинотеатре сделает работу автоматизированной, более быстрой и упрощенной. В дальнейшем данная программа может редактироваться и совершенствоваться в соответствии с требованиями предметной области. Подводя итог, можно сказать, что цели и задачи, поставленные перед данной работой, успешно достигнуты и выполнены.

Таким образом цель курсового проекта достигнута, разработанный сервис выполняет поставленные требования и удобен в использовании.

# Список использованных источников

1. Ерохина, Т. В. Государственное управление здравоохранением / Т. В. Ерохина. – СПб.: BHV, 2017. – 286 с.
2. Князюк, Н. Ф. Методология построения интегрированной системы менеджмента медицинских организаций / Н. Ф. Князюк, И. С. Кицул. – Москва: Менеджер здравоохранения, 2016. – 312 с.
3. Стародубов, В. И. Методологические технологии и руководство по управлению качеством медицинской помощи / В. И. Стародубов, Г. И. Галанова. – Москва: Менеджер здравоохранения, 2015. – 208 с.
4. Модернизация здравоохранения: 100 ответов на актуальные вопросы / под ред. Ф. Н. Кадырова. – Москва: Менеджер здравоохранения, 2014. – 319 с.
5. Заборовский, Г. И. Общественное здоровье и здравоохранение / Г. И. Заборовский, Е. М. Тищенко. – Минск: Асвета, 2015. – 204 с.
6. Киммел, П. Основы визуального анализа и проектирования. / П. Киммел. – Москва: Нолидж, 2013. – 237 с.
7. Князюк, Н. Ф. Управление медицинским учреждением: справочник / Н. Ф. Князюк. – Москва: Менеджер здравоохранения, 2015. – 660 с.
8. Журнал Медицинские новости [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.mednovosti.by.
9. Национальный правой интернет-портал Республики Беларусь [Электронный ресурс]. – Электронные данные. – Режим доступа: http://pravo.by/main.aspx?guid=1011.
10. Межгосударственный статистический комитет содружества независимых государств [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.cisstat.com.
11. Министерство Здравоохранения Республики Беларусь [Электронный ресурс]. – Электронные данные. – Режим доступа: http://minzdrav.gov.by/ru/static/activities/stat\_otchetnost.
12. Маклаков, С. В. Книга моделирования бизнес процессов при помощи BpWin 4.0: справ. пособие / С. В. Маклов. – Москва: Диалог-Мифи, 2002. – 74 с.
13. Дубейковский, В. И. Эффективное моделирование с AllFusion Process Modeler 4.1.4 и AllFusion PM / В. И. Дубейковский. – Москва: Нолидж, 2011. – 284 с.
14. Леоненков, А. В. Самоучитель UML / А. В. Леоненков. – СПб.: BHV, 2014. – 304 с.
15. Сергеев, В. И., Григорьев, М. Н., Уваров, С. А. Логистика: информационные системы и технологии / В. И. Сергеев, М. Н. Григорьев, С. А. Уваров. – М.: Альфа-Пресс, 2018 – 355 с.
16. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: Вильямс, 2016 – 544 с.
17. UML. Классика CS. 2-у изд./Пер. с англ.; Под общей редакцией проф. С. Орлова - СПб.: Питер, 2006. - 736 с.: ил.
18. Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление – М: БХВ-Петербург, 2004. – 1040 с.
19. Харрингтон Джен Л. Проектирование реляционных баз данных – М: Лори, 2016. – 514 с.
20. BPWin [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.bpwin.ru/.
21. UML. Классика CS. 2-е изд./Пер. с англ.; Под общей редакцией проф. С.Орлова – СПб.: Питер, 2006. – 736 с.
22. Басс, Л., Архитектура программного обеспечения на практике. 2-е издание / Л. Басс, П. Клементс, Р. Кацман.– СПб.: Питер, 2006. – 575 с.
23. Фаулер, М. Архитектура корпоративных программных приложений.: Пер. с англ. / М. Файлер.– М.: Издательский дом «Вильямс», 2006. – 544с.

# Приложение А

# (обязательное)

## Отчет о проверке на заимствования в системе «Антиплагиат»

# Приложение Б

# (обязательное)

## Листинг кода алгоритмов, реализующих бизнес-логику

Файл *SignInCommand*.*java*

package command.implement.user;  
  
import command.Command;  
import command.exception.CommandException;  
import entities.User;  
import service.UserService;  
import service.implement.UserServiceImplement;  
import usage.cooper.ClientRequest;  
import usage.cooper.ServerResponse;  
  
import service.exception.ServiceException;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class SignInCommand implements Command {  
  
 private UserService service;  
 private ClientRequest request;  
 private ServerResponse response;  
  
 public SignInCommand(ClientRequest request, ServerResponse response) {  
 this.service = UserServiceImplement.*getInstance*();  
 this.request = request;  
 this.response = response;  
 }  
  
 @Override  
 public ServerResponse execute() throws CommandException {  
 Map<String, Object> data = request.getData();  
 String username = (String) data.get("username");  
 String password = (String) data.get("password");  
 try {  
 User user = service.signIn(username, password);  
 Map<String, Object> dataSend = new HashMap<>();  
 dataSend.put("user", user);  
  
 response.setData(dataSend);  
 } catch (ServiceException e) {  
 throw new CommandException(e);  
 }  
 return response;  
 }  
}

Файл *MovieListController*.*java*

package control.movie;  
  
import cooper.ClientRequest;  
import cooper.ServerResponse;  
import entities.attribute.MovieAttribute;  
import entities.Movie;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.collections.FXCollections;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import main.Runner;  
import usage.MapParser;  
import usage.confirm.SceneChanger;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.\*;  
  
import static javafx.scene.control.Alert.AlertType.*ERROR*;  
import static javafx.scene.control.Alert.AlertType.*INFORMATION*;  
  
public class MovieListController {  
  
 @FXML  
 private TableView<MovieAttribute> moviesTable;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieNameColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieGenreColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieCountryColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieDurationColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> cinemaHallColumn;  
 @FXML  
 private TableColumn<MovieAttribute, Integer> moviePriceColumn;  
 @FXML  
 private Button addMovie;  
  
 @FXML  
 private Button deleteMovie;  
  
 @FXML  
 private Button back;  
  
 @FXML  
 private Button main;  
  
 @FXML  
 private Button saveToFile;  
  
  
 private SceneChanger sceneChanger = SceneChanger.*getInstance*();  
 private List<Movie> movies;  
 private List<String> cinemaHalls = new ArrayList<>();  
 private MapParser parser = MapParser.*getInstance*();  
  
 @FXML  
 private void initialize() {  
 if (!"Администратор".equals(Runner.*getStatus*().getStatusName())) {  
 addMovie.setVisible(false);  
 deleteMovie.setVisible(false);  
 }  
  
 fillMoviesTable();  
  
 addMovie.setOnAction(event -> {  
 sceneChanger.changeSceneAndWait("/fxml/add\_movie.fxml");  
 fillMoviesTable();  
 });  
  
 deleteMovie.setOnAction(event -> {  
 deleteMovie();  
 fillMoviesTable();  
 });  
  
 moviesTable.setOnMouseClicked(event -> {  
 if (event.getClickCount() == 2) {  
 MovieAttribute movieAttribute = moviesTable.getSelectionModel().getSelectedItem();  
 sceneChanger.setDataId(movieAttribute.getMovieId().getValue());  
 sceneChanger.changeSceneAndWait("/fxml/edit\_movie.fxml");  
 fillMoviesTable();  
 }  
 });  
  
 back.setOnAction(event -> {  
 back.getScene().getWindow().hide();  
 SceneChanger.*getInstance*().changeScene("/fxml/main.fxml");  
 });  
 main.setOnAction(event -> {  
 main.getScene().getWindow().hide();  
 SceneChanger.*getInstance*().changeScene("/fxml/main.fxml");  
 });  
  
  
 this.saveToFile.setOnAction(event -> {  
  
 try {  
 FileWriter writer = new FileWriter("src/main/resources/movies" + ".txt");  
  
 for (Movie movie : this.movies) {  
 String movieName = movie.getMovieName();  
 String movieGenre = movie.getMovieGenre();  
 String movieCountry = movie.getMovieCountry();  
 String movieDuration = movie.getMovieDuration();  
 String hallType = movie.getCinemaHall().getHallType();  
  
 String text =  
 " Название фильма: " + movieName  
 + " Жанр: " + movieGenre  
 + " Страна производства " + movieCountry  
 + " Длительность " + movieDuration  
 + " Тип зала " + hallType;  
  
 writer.write(text + System.*getProperty*("line.separator"));  
 }  
 writer.close();  
  
 Alert alert = new Alert(*INFORMATION*, "Отчет успешно сохранен в файл!");  
 alert.show();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 Alert alert = new Alert(*ERROR*, "Ошибка при записи в файл");  
 alert.show();  
 }  
  
  
 });  
 }  
  
 private void getMovies() {  
 Runner.*sendData*(new ClientRequest("getAllMovies", new HashMap<>()));  
 ServerResponse response = Runner.*getData*();  
 if (!response.isError()) {  
 Map<String, Object> productMap = response.getData();  
 List productData = (List) productMap.get("movies");  
 movies = parser.movies(productData);  
 }  
 }  
  
 private void fillMoviesTable() {  
 getMovies();  
 List<MovieAttribute> movieAttributes = new ArrayList<>();  
 movies.forEach(movie -> movieAttributes.add(new MovieAttribute(movie)));  
 moviesTable.setItems(FXCollections.*observableArrayList*(movieAttributes));  
 movieNameColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieName());  
 movieGenreColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieGenre());  
 movieCountryColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieCountry());  
 movieDurationColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieDuration());  
 moviePriceColumn.setCellValueFactory(cellData -> {  
 int moviePriceColumn = cellData.getValue().getMoviePrice().getValue().intValue();  
 return new SimpleIntegerProperty(moviePriceColumn).asObject();  
 });  
 cinemaHallColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getCinemaHall().getHallType()));  
 }  
  
 private void deleteMovie() {  
 int movieId = moviesTable.getSelectionModel()  
 .selectedItemProperty()  
 .getValue()  
 .getMovieId()  
 .getValue();  
 Map<String, Object> map = new HashMap<>();  
 map.put("movieId", movieId);  
 Runner.*sendData*(new ClientRequest("deleteMovie", map));  
 ServerResponse response = Runner.*getData*();  
 if (response.isError()) {  
 Alert alert = new Alert(*ERROR*, "Ошибка удаления фильма");  
 alert.show();  
 }  
 }  
  
}

Файл *BookingController*.*java*

package control.user;  
  
import cooper.ClientRequest;  
import cooper.ServerResponse;  
import entities.Movie;  
import entities.attribute.MovieAttribute;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.collections.FXCollections;  
import javafx.fxml.FXML;  
import javafx.scene.control.Alert;  
import javafx.scene.control.Button;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import main.Runner;  
import usage.MapParser;  
import usage.confirm.SceneChanger;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
import static javafx.scene.control.Alert.AlertType.*ERROR*;  
import static javafx.scene.control.Alert.AlertType.*INFORMATION*;  
  
public class BookingController {  
  
 @FXML  
 private TableView<MovieAttribute> moviesTable;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieNameColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieGenreColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieCountryColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> movieDurationColumn;  
  
 @FXML  
 private TableColumn<MovieAttribute, String> cinemaHallColumn;  
 @FXML  
 private TableColumn<MovieAttribute, Integer> moviePriceColumn;  
 @FXML  
 private Button booking;  
 @FXML  
 private Button back;  
  
 @FXML  
 private Button main;  
  
  
 private SceneChanger sceneChanger = SceneChanger.*getInstance*();  
 private List<Movie> movies;  
 private List<String> cinemaHalls = new ArrayList<>();  
 private MapParser parser = MapParser.*getInstance*();  
  
 @FXML  
 private void initialize() {  
  
 fillMoviesTable();  
  
 booking.setOnAction(event -> {  
 bookingMovie();  
 });  
  
 back.setOnAction(event -> {  
 back.getScene().getWindow().hide();  
 SceneChanger.*getInstance*().changeScene("/fxml/main.fxml");  
 });  
 main.setOnAction(event -> {  
 main.getScene().getWindow().hide();  
 SceneChanger.*getInstance*().changeScene("/fxml/main.fxml");  
 });  
  
 }  
  
 private void getMovies() {  
 Runner.*sendData*(new ClientRequest("getAllMovies", new HashMap<>()));  
 ServerResponse response = Runner.*getData*();  
 if (!response.isError()) {  
 Map<String, Object> productMap = response.getData();  
 List productData = (List) productMap.get("movies");  
 movies = parser.movies(productData);  
 }  
 }  
private void bookingMovie(){  
 int userId = Runner.*getUserId*();  
 int movieId = moviesTable.getSelectionModel().getSelectedItem().getMovieId().intValue();  
 Map<String, Object> data = new HashMap<>();  
 data.put("userId", userId);  
 data.put("movieId", movieId);  
 Runner.*sendData*(new ClientRequest("booking", data));  
 ServerResponse response = Runner.*getData*();  
 if (!response.isError()) {  
 booking.getScene().getWindow().hide();  
 SceneChanger.*getInstance*().changeScene("/fxml/main.fxml");  
 } else {  
 Alert alert = new Alert(*ERROR*, "Ошибка бронирования");  
 alert.show();  
 }  
  
}  
 private void fillMoviesTable() {  
 getMovies();  
 List<MovieAttribute> movieAttributes = new ArrayList<>();  
 movies.forEach(movie -> movieAttributes.add(new MovieAttribute(movie)));  
 moviesTable.setItems(FXCollections.*observableArrayList*(movieAttributes));  
 movieNameColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieName());  
 movieGenreColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieGenre());  
 movieCountryColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieCountry());  
 movieDurationColumn.setCellValueFactory(cellData -> cellData.getValue().getMovieDuration());  
 moviePriceColumn.setCellValueFactory(cellData -> {  
 int moviePriceColumn = cellData.getValue().getMoviePrice().getValue().intValue();  
 return new SimpleIntegerProperty(moviePriceColumn).asObject();  
 });  
 cinemaHallColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getCinemaHall().getHallType()));  
 }  
  
}

# Приложение В

# (обязательное)

# Листинг скрипта генерации базы данных

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema cinemadb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema cinemadb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `cinemadb` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `cinemadb` ;

-- -----------------------------------------------------

-- Table `cinemadb`.`cinemahalls`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `cinemadb`.`cinemahalls` (

`hall\_id` INT NOT NULL AUTO\_INCREMENT,

`hall\_seatsnumber` INT NULL DEFAULT NULL,

`hall\_type` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`hall\_id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 7

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `cinemadb`.`movies`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `cinemadb`.`movies` (

`movie\_id` INT NOT NULL AUTO\_INCREMENT,

`movie\_country` VARCHAR(255) NULL DEFAULT NULL,

`movie\_duration` VARCHAR(255) NULL DEFAULT NULL,

`movie\_genre` VARCHAR(255) NULL DEFAULT NULL,

`movie\_name` VARCHAR(255) NULL DEFAULT NULL,

`movie\_price` INT NULL DEFAULT NULL,

`cinemahall\_id` INT NULL DEFAULT NULL,

PRIMARY KEY (`movie\_id`),

INDEX `FK8qhgrv8j027vg5f3k5t6xg4mw` (`cinemahall\_id` ASC) VISIBLE,

CONSTRAINT `FK8qhgrv8j027vg5f3k5t6xg4mw`

FOREIGN KEY (`cinemahall\_id`)

REFERENCES `cinemadb`.`cinemahalls` (`hall\_id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 8

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `cinemadb`.`user\_statuses`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `cinemadb`.`user\_statuses` (

`status\_id` INT NOT NULL AUTO\_INCREMENT,

`status\_name` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`status\_id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `cinemadb`.`users`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `cinemadb`.`users` (

`user\_id` INT NOT NULL AUTO\_INCREMENT,

`email` VARCHAR(255) NULL DEFAULT NULL,

`first\_name` VARCHAR(255) NULL DEFAULT NULL,

`last\_name` VARCHAR(255) NULL DEFAULT NULL,

`password` VARCHAR(255) NULL DEFAULT NULL,

`username` VARCHAR(255) NULL DEFAULT NULL,

`status\_id` INT NULL DEFAULT NULL,

PRIMARY KEY (`user\_id`),

INDEX `FK1lqs7qsjk62rrrt1rogyasqaf` (`status\_id` ASC) VISIBLE,

CONSTRAINT `FK1lqs7qsjk62rrrt1rogyasqaf`

FOREIGN KEY (`status\_id`)

REFERENCES `cinemadb`.`user\_statuses` (`status\_id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `cinemadb`.`booking`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `cinemadb`.`booking` (

`booking\_id` INT NOT NULL AUTO\_INCREMENT,

`movie\_id` INT NULL DEFAULT NULL,

`user\_id` INT NULL DEFAULT NULL,

PRIMARY KEY (`booking\_id`),

INDEX `FK5aq97ew6hc9ev61dk26g605a4` (`movie\_id` ASC) VISIBLE,

INDEX `FK7udbel7q86k041591kj6lfmvw` (`user\_id` ASC) VISIBLE,

CONSTRAINT `FK5aq97ew6hc9ev61dk26g605a4`

FOREIGN KEY (`movie\_id`)

REFERENCES `cinemadb`.`movies` (`movie\_id`),

CONSTRAINT `FK7udbel7q86k041591kj6lfmvw`

FOREIGN KEY (`user\_id`)

REFERENCES `cinemadb`.`users` (`user\_id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;