**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное   
учреждение высшего образования   
«Санкт-Петербургский политехнический университет Петра Великого»  
ИНСТИТУТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

|  |  |
| --- | --- |
|  |  |
|  |

**курсовой проект**

Тема: «ПРИЛОЖЕНИЕ по работе с бд кинотеатра»

специальность 09.02.07 группа 32919/9

Студент (ка) Судник Е.А.  
 (подпись) (ФИО)  
Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (ФИО)

Санкт-Петербург  
2025

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc201343566)

[1 ОБЩАЯ ЧАСТЬ 5](#_Toc201343567)

[1.1 Описание предметной области 5](#_Toc201343568)

[1.2 Анализ методов решения 6](#_Toc201343569)

[1.3 Обзор средств программирования 7](#_Toc201343570)

[1.4 Описание языка C# 10](#_Toc201343571)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 14](#_Toc201343572)

[2.1 Постановка задачи 14](#_Toc201343573)

[2.1.1 Основания для разработки 14](#_Toc201343574)

[2.1.2 Назначение программы 14](#_Toc201343575)

[2.2 Проектирование приложения 14](#_Toc201343576)

[2.2.1 Диаграмма прецедентов 14](#_Toc201343577)

[2.2.2 Диаграмма активности 16](#_Toc201343578)

[2.2.3 Диаграмма состояний 17](#_Toc201343579)

[2.2.4 Диаграмма классов 18](#_Toc201343580)

[2.3 Текст программы 18](#_Toc201343581)

[2.4 Описание программы 19](#_Toc201343582)

[2.4.1 Общие сведения 19](#_Toc201343583)

[2.4.2 Функциональное назначение 19](#_Toc201343584)

[2.4.3 Описание логической структуры системы 19](#_Toc201343585)

[2.4.4 Используемые технические и программные средства 25](#_Toc201343586)

[2.4.5 Вызов и загрузка 25](#_Toc201343587)

[2.5 Руководство оператора 25](#_Toc201343588)

[2.5.1 Назначение программы 25](#_Toc201343589)

[2.5.2 Выполнение программы 26](#_Toc201343590)

[2.5.3 Сообщения оператору 27](#_Toc201343591)

[2.6 Программа и методика испытаний 28](#_Toc201343592)

[2.6.1 Модульное тестирование 28](#_Toc201343593)

[2.6.2 Ручное тестирование 29](#_Toc201343594)

[ЗАКЛЮЧЕНИЕ 30](#_Toc201343595)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31](#_Toc201343596)

[ПРИЛОЖЕНИЕ А 32](#_Toc201343597)

[ПРИЛОЖЕНИЕ Б 38](#_Toc201343598)

[ПРИЛОЖЕНИЕ В 44](#_Toc201343599)

[ПРИЛОЖЕНИЕ Г 82](#_Toc201343600)

ВВЕДЕНИЕ

В настоящее время компьютерные технологии широко применяются во многих сферах жизни, включая развлекательную индустрию и сферу обслуживания. Современное программное обеспечение позволяет автоматизировать рутинные процессы, повысить качество обслуживания клиентов и упростить управление бизнесом. В частности, в системе управления кинотеатром использование программных решений способствует эффективному контролю за фильмами, залами, сеансами и билетами, а также облегчает работу сотрудников.

Целью данной работы является разработка приложения для работы с базой данных кинотеатра, которое позволит автоматизировать основные бизнес-процессы кинотеатра.

Для достижения поставленной цели необходимо решить следующие задачи:

* проанализировать предметную область кинотеатра;
* спроектировать структуру базы данных и архитектуру приложения;
* разработать программное обеспечение для управления фильмами, залами, сеансами, билетами и сотрудниками;
* провести тестирование и отладку разработанного приложения.

# ОБЩАЯ ЧАСТЬ

## Описание предметной области

Компания «КиноМастер» — современный кинотеатр, предоставляющий услуги по организации кинопоказов для широкой аудитории. Основная задача кинотеатра — обеспечить комфортный просмотр фильмов в оборудованных залах и качественное обслуживание посетителей.

Кинотеатр состоит из нескольких ключевых компонентов:

* Фильмы — аудиовизуальные произведения, которые демонстрируются посетителям;
* Залы — помещения с определённым количеством мест, предназначенные для показа фильмов;
* Сеансы — конкретные показы фильмов в определённое время и в выбранном зале;
* Билеты — документы, дающие право занять конкретное место на сеансе;
* Сотрудники — персонал кинотеатра с разными ролями, управляющий процессами кинотеатра.

Основные бизнес-процессы:

* Управление фильмами. Добавление, редактирование и удаление фильмов для поддержания актуального каталога.
* Управление залами. Ведение информации о залах и количестве мест в каждом зале.
* Управление сеансами. Создание и редактирование расписания сеансов с указанием фильма, зала, времени начала и цены билета.
* Продажа и бронирование билетов. Отображение всех мест зала для выбранного сеанса со статусами «Свободен», «Забронирован» или «Продан». Изменение статуса билетов при продаже или бронировании.
* Управление сотрудниками. Учёт сотрудников, управление их ролями и правами доступа к системе.

Входные и выходные данные:

* Входные данные: информация о фильмах, залах, сеансах, билетах и сотрудниках, вводимая сотрудниками кинотеатра.
* Выходные данные: актуальное расписание сеансов, статус билетов.

Программа автоматизирует перечисленные процессы, обеспечивая удобный интерфейс для работы с базой данных кинотеатра. Система позволяет:

* Управлять каталогом фильмов;
* Вести учёт залов и мест;
* Планировать сеансы и назначать им фильмы и залы;
* Просматривать и изменять статусы билетов;
* Управлять учётными записями сотрудников с разграничением прав.

Таким образом, система повышает эффективность работы кинотеатра, снижает вероятность ошибок и улучшает качество обслуживания посетителей.

## Анализ методов решения

Для решения задачи автоматизации управления кинотеатром можно выделить несколько способов реализации:

1. Ручной способ: Все процессы — учёт фильмов, залов, сеансов и билетов — ведутся вручную на бумажных носителях или в простых таблицах. Такой способ требует значительных временных затрат, подвержен ошибкам и не обеспечивает оперативного контроля и анализа.
2. Автоматизированный способ: Исходные данные (информация о фильмах, залах, сеансах, билетах и сотрудниках) вводятся пользователями вручную через программный интерфейс. Программа обрабатывает данные, хранит их в базе данных и предоставляет удобные средства для просмотра, редактирования и отчётности. Такой способ значительно ускоряет работу, снижает количество ошибок и повышает качество обслуживания.
3. Автоматический способ: Данные автоматически собираются из внешних источников (интеграция с онлайн-сервисами, сканирование билетов, автоматическое обновление расписания). В рамках данного проекта автоматический способ не реализуется, так как основная задача — создать удобный инструмент для сотрудников кинотеатра с ручным вводом данных.

В нашем проекте выбран автоматизированный способ реализации, который оптимально сочетает удобство использования и эффективность работы. Программа обеспечивает ввод данных пользователями через графический интерфейс, хранение и обработку информации в базе данных MySQL, а также предоставляет функции управления и отчётности.

## Обзор средств программирования

Для реализации информационной системы управления кинотеатром рассматривались следующие языки программирования и среды разработки:

1. Java

Плюсы:

* Кроссплатформенность — приложения работают на разных операционных системах благодаря JVM.
* Большое количество библиотек и фреймворков для разработки графического интерфейса (например, JavaFX, Swing).
* Высокая производительность и масштабируемость.

Минусы:

* Сложное и громоздкое создание пользовательского интерфейса по сравнению с современными средствами.
* Требуется установка JVM на клиентских машинах.
* Меньшая интеграция с Windows-специфичными технологиями.

1. Python

Плюсы:

* Простота и лаконичность синтаксиса, быстрая разработка.
* Большое количество библиотек для работы с базами данных и GUI (PyQt, Tkinter).
* Хорошо подходит для прототипирования.

Минусы:

* Относительно низкая производительность по сравнению с компилируемыми языками.
* Меньшая поддержка сложных GUI-приложений под Windows.
* Требуется установка интерпретатора на клиентских машинах.

1. C# с использованием Windows Forms в Visual Studio 2022

Плюсы:

* Глубокая интеграция с платформой Windows, что упрощает разработку настольных приложений.
* Богатый набор визуальных компонентов для создания удобного и современного интерфейса. [2]
* Высокая производительность и поддержка объектно-ориентированного программирования.
* Поддержка работы с базами данных через ADO.NET и Entity Framework.
* Среда Visual Studio обеспечивает мощные инструменты отладки и разработки. [6]

Минусы:

* Привязанность к платформе Windows.

Таким образом, средством программирования данного курсового проекта был выбран язык С#, среда разработки Visual Studio 2022.

Для хранения и управления данными информационной системы управления кинотеатром рассматривались следующие СУБД:

1. MySQL

Плюсы:

* Бесплатная и с открытым исходным кодом (Community Edition).
* Широко распространённая и хорошо документированная СУБД.
* Высокая производительность и стабильность для приложений среднего масштаба.
* Поддержка стандартного SQL и множества инструментов для администрирования.
* Простая интеграция с C# через официальный драйвер MySQL Connector/NET. [3]
* Возможность размещения на удалённом сервере, что удобно для централизованного доступа.

Минусы:

* Отсутствие некоторых продвинутых функций, присущих коммерческим СУБД.
* Ограниченные возможности масштабирования для очень больших систем.

1. Microsoft SQL Server

Плюсы:

* Глубокая интеграция с платформой Windows и инструментами Microsoft.
* Богатый функционал для управления данными, безопасности и аналитики.
* Высокая производительность и масштабируемость.
* Хорошая поддержка ORM (Entity Framework) и инструментов разработки.

Минусы:

* Более сложная установка и администрирование по сравнению с MySQL.

1. PostgreSQL

Плюсы:

* Бесплатная и с открытым исходным кодом, с расширенными возможностями.
* Поддержка сложных типов данных, расширяемость и высокая надежность.
* Хорошая производительность и поддержка стандартов SQL.
* Поддержка транзакций и расширенных функций.

Минусы:

* Более высокая сложность настройки и администрирования.
* Меньшая распространённость в Windows-среде по сравнению с MySQL и MS SQL Server.

Для данного проекта выбрана СУБД MySQL, размещённая на удалённом сервере filess.io.

Таким образом, сочетание C# + Visual Studio 2022 и MySQL обеспечивает оптимальный баланс функциональности, удобства разработки и производительности для решения поставленной задачи по автоматизации работы кинотеатра. [4]

## Описание языка C#

C# — это современный объектно-ориентированный язык программирования, разработанный Microsoft в рамках платформы .NET. Он сочетает мощь и гибкость низкоуровневых языков с высокой степенью безопасности и удобством разработки, благодаря автоматическому управлению памятью и строгой типизации.

Он поддерживает все ключевые принципы ООП: инкапсуляцию, наследование, полиморфизм и абстракцию. Также предоставляет продвинутые возможности: LINQ, async/await для асинхронного программирования, делегаты, события, работу с коллекциями через System.Collections, пространства имён, свойства, методы расширения и многое другое. [1]

C# тесно интегрирован с Visual Studio — одной из самых мощных IDE, что делает его особенно популярным среди студентов и профессиональных разработчиков. [6]

Таблица 1 – Ключевые слова C#

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | extern | null | struct |
| as | false | object | switch |
| base | finally | operator | this |
| bool | fixed | out | throw |
| break | float | override | true |
| byte | for | params | try |
| case | foreach | private | typeof |
| catch | goto | protected | uint |
| const | if | public | ulong |
| continue | implicit | readonly | unchecked |
| decimal | in | ref | unsafe |
| default | int | return | ushort |
| delegate | interface | sbyte | using |
| do | internal | sealed | virtual |
| double | is | short | void |
| else | lock | sizeof | while |
| enum | long | stackalloc |  |
| event | namespace | static |  |
| explicit | new | string |  |

Таблица 2 – Контекстные ключевые слова C#

|  |  |  |  |
| --- | --- | --- | --- |
| add | equals | join | set |
| ascending | from | let | value |
| async | get | on | var |
| await | global | orderby | where |
| by | group | partial | yield |
| descending | in | remove |  |
| dynamic | into | select |  |

Таблица 3 – Основные алгоритмические конструкции C#:

|  |  |
| --- | --- |
| Оператор присваивания | string hello = "hello " + "world"; //результат равен "hello world"  int x1 = 2 + 4; // результат равен 6  int x2 = 10 - 6; //результат равен 4  int x3 = 10 \* 6; //результат равен 60  double x4 = 10.0 / 4.0; //результат равен 2.5  double x5 = 10.0 % 4.0; //результат равен 2  int y1 = 5;  int z1 = ++y1; // z1=6; y1=6  int y2 = 5;  int z2 = y2++; // z2=5; y2=6  int y3 = 5;  int z3 = --y3; // z3=4; y3=4  int y4 = 5;  int z4 = y4--; // z4=5; y4=4 |
| Условный оператор | if (условие) {(действие) } else {(альтернатива)} ;  int num1 = 8;  int num2 = 6;  if(num1 > num2)  {  Console.WriteLine("Число {0} больше числа {1}", num1, num2);  }  else  {  Console.WriteLine("Число {0} меньше числа {1}", num1, num2);  } |
| Оператор множественного ветвления | Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий:  Console.WriteLine("Нажмите Y или N");  string selection = Console.ReadLine();  switch (selection)  {  case "Y":  Console.WriteLine("Вы нажали букву Y");  break;  case "N":  Console.WriteLine("Вы нажали букву N");  break;  default:  Console.WriteLine("Вы нажали неизвестную букву");  break;  }  После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора сase. И если совпадение будет найдено, то будет выполняться определенный блок сase.  В конце блока сase ставится оператор break, чтобы избежать выполнения других блоков.  Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default,. |
| Арифметический цикл  (применяется, когда известно количество повторений цикла) | for ([инициализация счетчика]; [условие]; [изменение счетчика])  for (int i = 0; i < 10; i++)  {  Console.WriteLine("Квадрат числа {0} равен {1}", i, i \* i);  } |
| Цикл с предусловием  (применяется, когда неизвестно количество повторений цикла) | while (условие) { тело цикла };  Этот цикл будет выполняться до тех пор, пока истинно условие (логическое выражение, возвращающее значение типа Boolean). При этом если это выражение сразу равно false, тело цикла не будет выполнено ни разу. Нужно очень внимательно следить за написанием условия и контролем завершения цикла, так как в результате ошибки цикл while будет повторяться бесконечное количество раз, что приведёт к "зацикливанию" и "зависанию" программы. |
| Цикл с постусловием  (применяется, когда неизвестно количество повторений цикла) | do { тело цикла } while (условие);  Повторения сначала выполняет тело цикла, а затем уже проверяет выполнение условия. Таким образом, этот вариант цикла гарантирует, что тело цикла будет выполнено по крайней мере один раз. И будет выполняться до тех пор, пока условие не станет истинным (true). |

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Постановка задачи

### Основания для разработки

Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.

### Назначение программы

Программа с графическим интерфейсом предназначена для автоматизации управления кинотеатром. Сотрудники могут добавлять и редактировать фильмы, управлять залами и расписанием сеансов, а также оформлять продажу и бронирование билетов. Программа отображает все места в зале для выбранного сеанса и позволяет изменять статус каждого билета. Система обеспечивает удобный и быстрый доступ к основным функциям учёта и обслуживания посетителей кинотеатра.

Полный текст технического задания приведён в приложении А.

## Проектирование приложения

На этапе проектирования были разработаны диаграмма прецедентов, диаграмма последовательностей, диаграмма классов и диаграмма активности.

### Диаграмма прецедентов

Диаграмма прецедентов частично описывает use case — прецедент использования проектируемой системы, давая представление о взаимодействии внешних пользователей (акторов) с системой. Такая диаграмма фокусируется на том, что должна делать система по отношению к своему окружению, а не на деталях внутренней реализации. Это частичная спецификация поведения системы, отражающая основные сценарии использования с точки зрения пользователя.

На рисунке 1 представлена диаграмма вариантов использования для программы по работе с базой данных кинотеатра. На диаграмме показаны основные акторы системы — «Обычный сотрудник» и «Администратор», а также их взаимодействие с ключевыми функциями программы: авторизация, выход из аккаунта, управление фильмами, сеансами, залами, билетами и сотрудниками.

Администратор обладает расширенными правами по сравнению с обычным сотрудником и может управлять учетными записями сотрудников.



Рисунок 1 – Диаграмма прецедентов

### Диаграмма активности

На рисунке 2 показана диаграмма активности процесса авторизации пользователя в системе управления кинотеатром.

Диаграмма активности (activity diagram) — это UML-диаграмма, на которой отображаются действия, выполняемые участниками процесса, а также последовательность и условия перехода между этими действиями. Такая диаграмма позволяет наглядно представить логику работы системы, специфицировать исполняемое поведение в виде последовательного и/или параллельного выполнения операций, связанных между собой потоками управления.

На данной диаграмме активности отражён процесс входа пользователя в систему: пользователь вводит логин и пароль, система проверяет их корректность, происходит обращение к базе данных, и в зависимости от результата пользователю либо отображается сообщение об ошибке, либо открывается главная страница с приветствием.

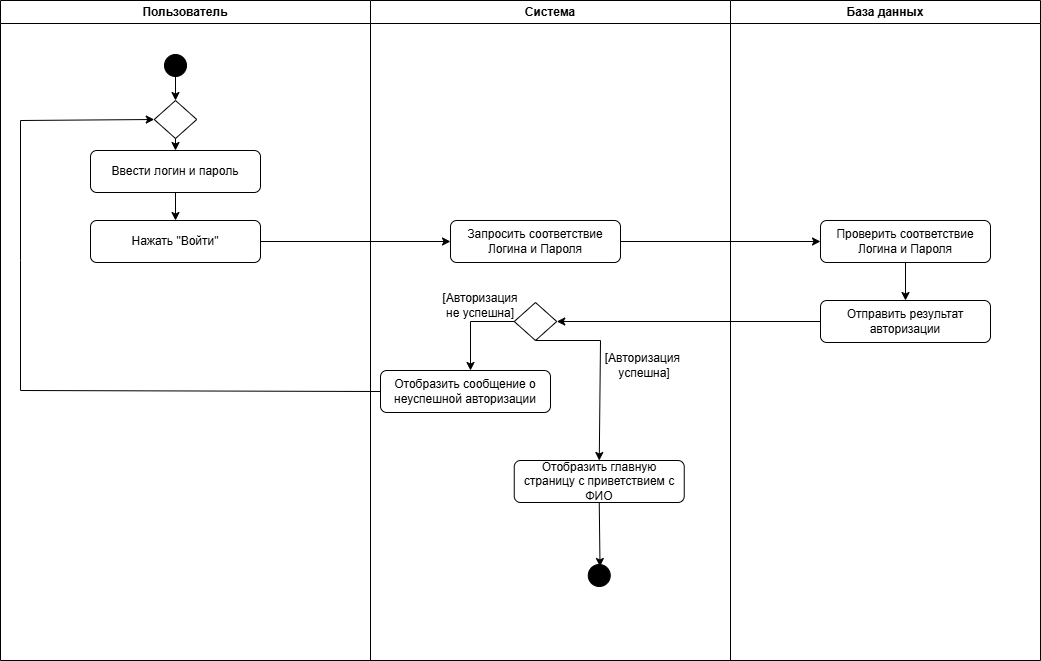


Рисунок 2 – Диаграмма активности

### Диаграмма состояний

На рисунке 3 показана диаграмма состояний для объекта «Билет» в системе управления кинотеатром.

Диаграмма состояний (state diagram) — это UML-диаграмма, которая отображает возможные состояния объекта в течение его жизненного цикла, а также события, вызывающие переходы между этими состояниями. Такая диаграмма позволяет наглядно представить, как объект изменяет своё состояние в ответ на действия пользователя или системы.

На данной диаграмме отражены основные состояния билета: «Свободен» (доступен для бронирования или продажи), «Забронирован» (временно закреплён за клиентом), «Продан» (билет оплачен и выдан клиенту).

Переходы между состояниями происходят по следующим событиям:

* «Забронировать» — переводит билет в состояние «Забронирован»;
* «Продать» — переводит билет в состояние «Продан» либо из состояния «Свободен», либо из состояния «Забронирован»;
* «Отменить бронь» — возвращает билет из состояния «Забронирован» в состояние «Свободен»;
* «Оформить возврат» — возвращает билет из состояния «Продан» в состояние «Свободен».

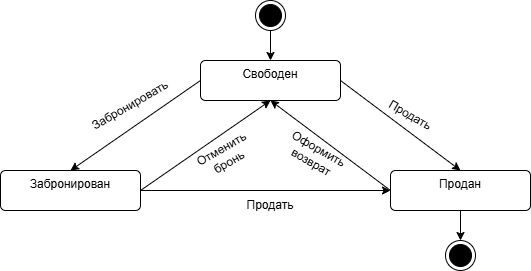


Рисунок 3 – Диаграмма состояний

### Диаграмма классов

На рисунке 4 представлена диаграмма классов программы по работе с БД кинотеатра. Диаграмма отражает структуру основных компонентов программы и их взаимодействие в рамках реализованной предметной области. В системе выделены ключевые сущности, такие как фильмы, залы, сеансы, билеты и сотрудники, а также соответствующие им классы, обеспечивающие работу с данными и отделение бизнес-логики от пользовательского интерфейса.

Диаграмма демонстрирует, как между собой связаны основные классы пользовательского интерфейса, а также какие методы и поля реализованы для выполнения ключевых функций программы.

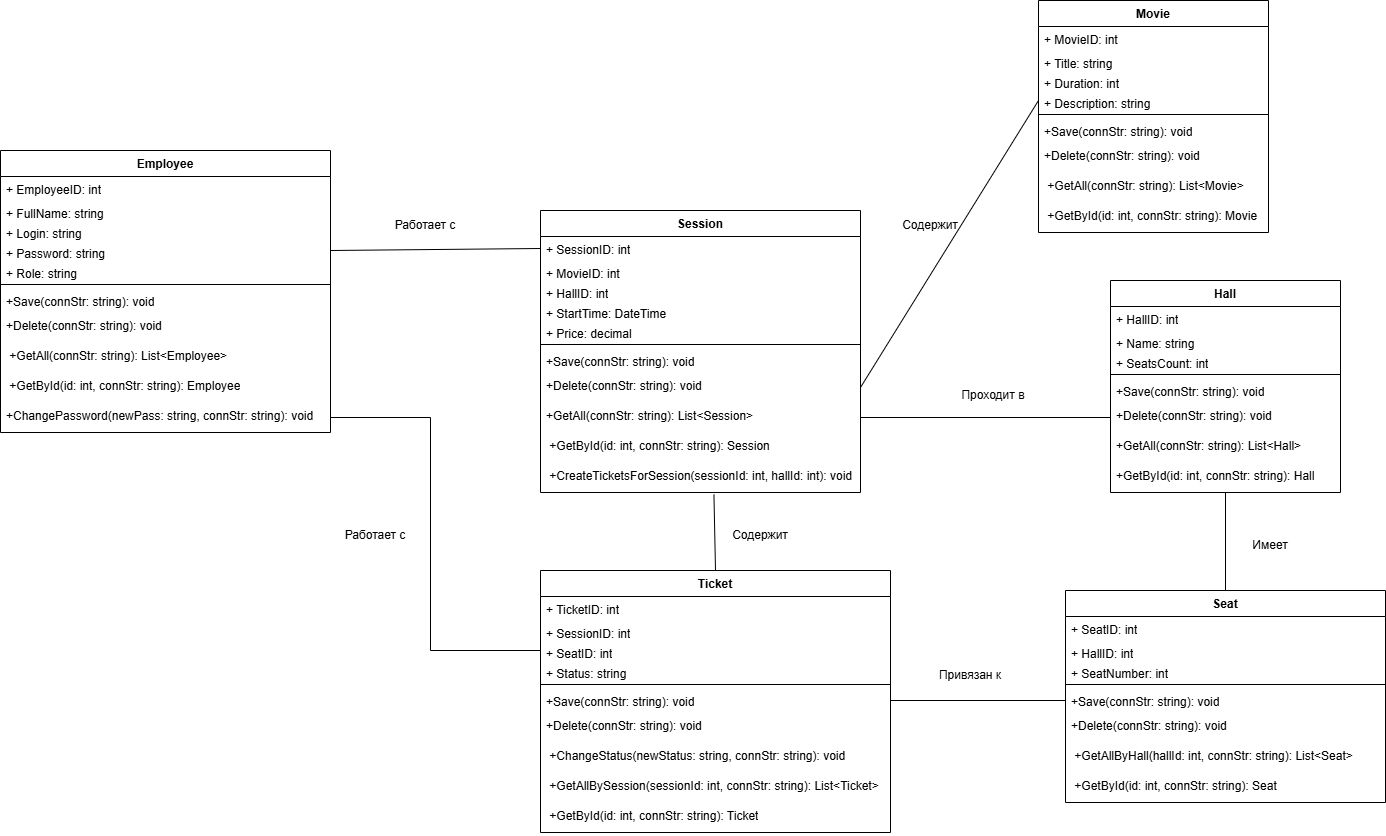


Рисунок 4 – Диаграмма классов

## Текст программы

Текст программы в соответствии с ГОСТ 19.101-77 (СТ СЭВ 1626-79) и ГОСТ 19.401-79 (СТ СЭВ 3746-82) представляет собой запись программы на исходном языке программирования с необходимыми комментариями. Текст программы представляет собой документ, выполненный машинным способом, и приведен в приложении В.

## Описание программы

### Общие сведения

Программное обеспечение предназначено для автоматизации управления кинотеатром с использованием базы данных. Система обеспечивает централизованный учёт фильмов, залов, расписания сеансов, а также оформление продажи и бронирования билетов через графический пользовательский интерфейс.

### Функциональное назначение

Основное назначение программного продукта: система автоматизации управления кинотеатром с использованием базы данных.

Эксплуатационное назначение программного продукта: программа предназначена для сотрудников кинотеатра и менеджеров, обеспечивая удобное и эффективное управление каталогом фильмов, залами, расписанием сеансов, а также процессами продажи и бронирования билетов.

### Описание логической структуры системы

Программа представляет собой систему управления кинотеатром, состоящую из нескольких модулей:

* MainForm.cs — главная форма приложения, содержащая функционал для работы с фильмами, залами, сеансами, билетами и сотрудниками. Этот модуль является центральным и обеспечивает взаимодействие между другими модулями.
* MovieEditForm.cs — форма для добавления и редактирования информации о фильмах.
* SessionEditForm.cs — форма для добавления и редактирования сеансов.
* EmployeeEditForm.cs — форма для добавления и редактирования данных сотрудников.
* HallEditForm.cs — форма для добавления и редактирования залов.
* LoginForm.cs — форма авторизации пользователей (сотрудников кинотеатра).

MainForm.cs использует модули MovieEditForm.cs, SessionEditForm.cs, EmployeeEditForm.cs, и HallEditForm.cs для добавления и редактирования данных. Сцепление по образцу (СЦ = 3), так как передаются структуры данных.

LoginForm.cs передает данные авторизованного пользователя (имя и роль) в MainForm.cs для отображения и управления доступом. Сцепление по образцу (СЦ = 3), так как передается объект с данными пользователя.

Программа использует функции следующих библиотек среды C#:

System, System.Windows.Forms, System.Data, MySql.Data.MySqlClient, System.ComponentModel, System.Drawing, System.Linq, System.Text, System.Threading.Tasks.

Исполняемы файл программы создан средствами среды C#, имеет имя CinemaApp.exe и размер 117 кб.

1. Описание методов класса LoginForm.cs

Класс LoginForm предназначен для авторизации сотрудников кинотеатра. Связность модуля функциональная (СС=10), так как он решает одну задачу – проверку учетных данных и вход в систему.

Поля:

* connectionString – строка подключения к базе данных MySQL.

Методы:

* btnLogin\_Click(object sender, EventArgs e) – метод, вызываемый при нажатии кнопки входа. Проверяет введенные данные, выполняет запрос к БД и, при успешной аутентификации, открывает главную форму (MainForm).

1. Описание методов класса MainForm.cs

Класс MainForm является основной формой приложения и обеспечивает управление фильмами, залами, сеансами, билетами и сотрудниками. Связность модуля информационная (СС=7), так как он работает с различными сущностями, но все они связаны с управлением кинотеатром.

Поля и свойства:

* connectionString – строка подключения к базе данных.
* \_currentUserRole – роль текущего пользователя (например, "Администратор").

25 методов:

* LoadSessionsForTickets() – загружает список сеансов для выпадающего списка билетов.
* LoadTickets(int sessionId) – загружает билеты для указанного сеанса.
* LoadMovies() – загружает список фильмов.
* LoadHalls() – загружает список залов.
* LoadEmployees() – загружает список сотрудников (доступно только администраторам).
* LoadSessions(string movieFilter = "") – загружает сеансы с возможностью фильтрации по названию фильма.
* btnAddMovie\_Click(object sender, EventArgs e) – открывает форму для добавления нового фильма.
* btnEditMovie\_Click(object sender, EventArgs e) – открывает форму для редактирования выбранного фильма.
* btnDeleteMovie\_Click(object sender, EventArgs e) – удаляет выбранный фильм после подтверждения.
* btnAddHall\_Click(object sender, EventArgs e) – открывает форму для добавления нового зала.
* btnEditHall\_Click(object sender, EventArgs e) – открывает форму для редактирования выбранного зала.
* btnDeleteHall\_Click(object sender, EventArgs e) – удаляет выбранный зал после подтверждения.
* btnAddSession\_Click(object sender, EventArgs e) – открывает форму для добавления нового сеанса.
* btnEditSession\_Click(object sender, EventArgs e) – открывает форму для редактирования выбранного сеанса.
* btnDeleteSession\_Click(object sender, EventArgs e) – удаляет выбранный сеанс после подтверждения.
* btnSellTicket\_Click(object sender, EventArgs e) – изменяет статус билета на "Продан".
* btnReserveTicket\_Click(object sender, EventArgs e) – изменяет статус билета на "Забронирован".
* btnCancelReservation\_Click(object sender, EventArgs e) – изменяет статус билета на "Свободен".
* ChangeTicketStatus(int seatId, int sessionId, string newStatus) – обновляет статус билета в базе данных.
* btnAddEmployee\_Click(object sender, EventArgs e) – открывает форму для добавления нового сотрудника.
* btnEditEmployee\_Click(object sender, EventArgs e) – открывает форму для редактирования выбранного сотрудника.
* btnDeleteEmployee\_Click(object sender, EventArgs e) – удаляет выбранного сотрудника после подтверждения.
* btnAddEmployee\_Click(object sender, EventArgs e) – открывает форму для добавления нового сотрудника.
* btnEditEmployee\_Click(object sender, EventArgs e) – открывает форму для редактирования выбранного сотрудника.
* btnDeleteEmployee\_Click(object sender, EventArgs e) – удаляет выбранного сотрудника после подтверждения.

1. Описание методов класса MovieEditForm.cs

Класс MovieEditForm отвечает за добавление и редактирование фильмов. Связность модуля функциональная (СС=10), так как он выполняет одну задачу – сохранение данных о фильме.

Поля и свойства:

* \_movieId – идентификатор фильма (0 – добавление, иначе – редактирование).
* connectionString – строка подключения к БД.

Методы:

* btnSave\_Click(object sender, EventArgs e) – сохраняет данные фильма в БД.
* btnCancel\_Click(object sender, EventArgs e) – закрывает форму без сохранения.

1. Описание методов класса SessionEditForm.cs

Класс SessionEditForm предназначен для добавления и редактирования сеансов. Связность модуля функциональная (СС=10).

Поля и свойства:

* \_sessionId – идентификатор сеанса.
* connectionString – строка подключения к БД.

Методы:

* LoadMovies() – загружает список фильмов в выпадающий список.
* LoadHalls() – загружает список залов.
* btnSave\_Click(object sender, EventArgs e) – сохраняет данные сеанса в БД.
* btnCancel\_Click(object sender, EventArgs e) – закрывает форму без сохранения.

1. Описание методов класса EmployeeEditForm.cs

Класс EmployeeEditForm отвечает за добавление и редактирование сотрудников. Связность модуля функциональная (СС=10).

Поля и свойства:

* \_employeeId – идентификатор сотрудника.
* connectionString – строка подключения к БД.

Методы:

* btnSave\_Click(object sender, EventArgs e) – сохраняет данные сотрудника в БД.
* btnCancel\_Click(object sender, EventArgs e) – закрывает форму без сохранения.

1. Описание методов класса HallEditForm.cs

Класс HallEditForm предназначен для добавления и редактирования залов. Связность модуля функциональная (СС=10).

Поля и свойства:

* \_hallId – идентификатор зала.
* connectionString – строка подключения к БД.

Методы:

* btnSave\_Click(object sender, EventArgs e) – сохраняет данные зала в БД.
* btnCancel\_Click(object sender, EventArgs e) – закрывает форму без сохранения.

В приложение Б продемонстрирована работа всей программы.

### Используемые технические и программные средства

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

* процессор Intel или другой совместимый;
* объем свободной оперативной памяти ~500 Кб;
* объем необходимой памяти на жестком диске ~3Мб;
* стандартный VGA-монитор или совместимый;
* стандартная клавиатура;
* манипулятор «мышь»;
* дополнительно: звуковая карты, колонки или наушники.

### Вызов и загрузка

Программа может быть загружена как с диска, так и с жесткого диска. В последнем случае требуется предварительно переписать программу с диска на жесткий диск.

Исполняемым файлом программы является файл CinemaApp.exe. Для его запуска необходимо дважды щелкнуть по исполняемому файлу левой кнопкой мышки.

## Руководство оператора

### Назначение программы

Основное назначение программного продукта заключается в автоматизации управления кинотеатром с помощью графического интерфейса. Программа позволяет сотрудникам добавлять и редактировать фильмы, управлять залами и расписанием сеансов, а также оформлять продажу и бронирование билетов. Система обеспечивает удобный доступ к информации, надёжное хранение данных в базе и эффективную организацию работы персонала кинотеатра.

### Выполнение программы

Для запуска программы дважды щёлкните левой кнопкой мыши по исполняемому файлу CinemaApp.exe.

Этапы работы приложения и основные действия пользователя проиллюстрированы в приложении Б.

1. Авторизация пользователя

После запуска программы загружается главное окно, на котором отображается форма для ввода логина и пароля. Пользователь должен ввести свои учётные данные. После успешной авторизации отображается приветственное сообщение, а также открывается доступ к основным функциям программы. Если пользователь не является администратором, вкладка "Сотрудники" скрывается.

2. Работа с фильмами

Пользователь может просматривать список фильмов, добавлять новые фильмы, редактировать или удалять существующие. Для добавления или редактирования фильма открывается отдельная форма, где необходимо ввести или изменить название, длительность и описание фильма. После подтверждения изменения автоматически обновляются данные на главной форме.

3. Управление залами

В разделе "Залы" сотрудник может просматривать список залов, добавлять новые, редактировать или удалять существующие. При добавлении или редактировании зала указываются его название и количество мест.

4. Управление сеансами

Пользователь может просматривать расписание сеансов, добавлять новые сеансы, редактировать или удалять их. Для каждого сеанса указывается фильм, зал, дата и время начала, а также цена билета. При необходимости можно фильтровать сеансы по названию фильма.

5. Продажа и бронирование билетов

В разделе "Билеты" сотрудник выбирает нужный сеанс из выпадающего списка. После этого отображается список всех мест в зале с указанием их статуса (свободно, забронировано, продано). Сотрудник может оформить продажу или бронирование билета, после чего статус соответствующего места обновляется.

6. Управление сотрудниками

Данная функция доступна только администраторам. В разделе "Сотрудники" можно просматривать, добавлять, редактировать и удалять учётные записи сотрудников, а также задавать им роли.

### Сообщения оператору

В процессе работы программы оператору регулярно выводятся всплывающие сообщения, информирующие о результатах выполненных действий, ошибках или необходимости дополнительных действий. Такие сообщения позволяют пользователю быстро ориентироваться в ситуации, понимать причину возникновения проблемы и корректно реагировать на неё. В таблице 4 приведён перечень основных сообщений, которые может увидеть оператор, а также описание ситуаций, при которых они появляются.

Таблица 4 - Сообщения оператору

|  |  |
| --- | --- |
| **Сообщение** | **Причина возникновения** |
| «Ошибка подключения к базе данных» | Не удалось установить соединение с сервером базы данных. |
| «Ошибка загрузки фильмов: ...» | Проблема при попытке загрузить список фильмов из базы данных. |
| «Ошибка загрузки залов: ...» | Проблема при попытке загрузить список залов из базы данных. |
| «Ошибка загрузки сотрудников: ...» | Проблема при попытке загрузить данные сотрудников из базы данных. |
| «Ошибка загрузки сеансов: ...» | Проблема при попытке загрузить расписание сеансов из базы данных. |
| «Ошибка загрузки билетов: ...» | Проблема при попытке загрузить билеты для выбранного сеанса. |
| «Неверный логин или пароль» | Введены некорректные данные при авторизации. |
| «Ошибка удаления записи» | Не удалось удалить выбранную запись из базы данных. |
| «Операция успешно выполнена» | Подтверждение успешного завершения действия (добавление, редактирование, удаление). |
| «Заполните все обязательные поля» | Не заполнены обязательные поля при добавлении или редактировании записи. |
| «Ошибка сохранения данных» | Ошибка при записи изменений в базу данных. |

## Программа и методика испытаний

### Модульное тестирование

Юнит-тесты, представленные в проекте, охватывают основные функциональные возможности ключевых классов приложения: Movie, Hall, Session, Ticket и Employee. Каждый тест проверяет корректность выполнения базовых операций CRUD (создание, чтение, обновление, удаление) для соответствующей сущности. Например, тест Test\_Movie\_CRUD добавляет новый фильм в базу данных, проверяет его сохранение и возможность получения по ID, редактирует описание фильма и убеждается, что изменения сохранились корректно, а затем удаляет фильм и проверяет, что он больше не доступен. Аналогичная логика применяется для залов, сеансов, сотрудников и билетов, чтобы убедиться в стабильности всех уровней работы с данными.

Особое внимание уделено тестированию каскадных операций — например, при удалении зала или фильма должны корректно удаляться связанные с ними сеансы и билеты. Тест Test\_Session\_Delete\_Removes\_Tickets подтверждает, что при удалении сеанса все связанные билеты также удаляются из базы данных. Результаты тестирования представлены на рисунке 5.

Код тестов представлен в приложение Г.

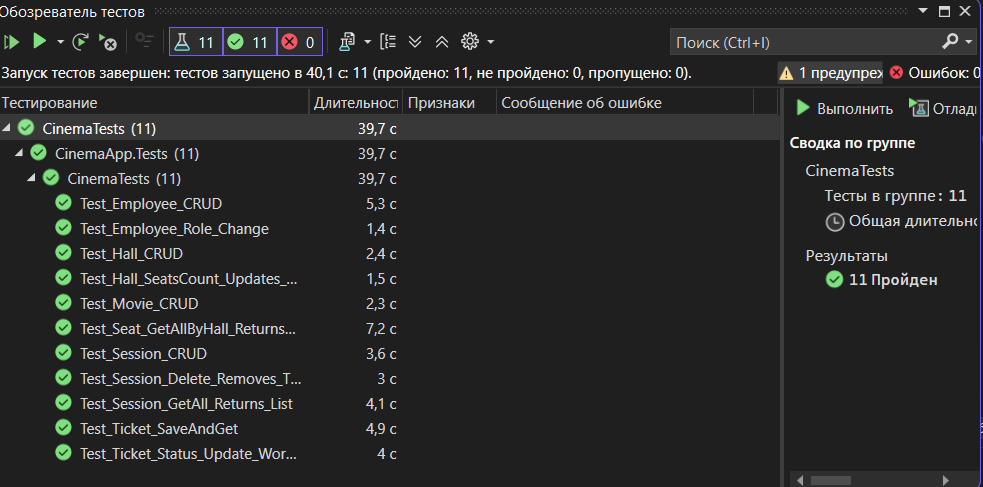


Рисунок 5 - Результат тестов

### Ручное тестирование

Целью испытаний является проверка соответствия программы требованиям Технического Задания.

Результаты испытаний программы представлены в таблице Б.1 приложения Б.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана программа по работе с БД кинотеатра, полностью соответствующая поставленному техническому заданию. Цель проекта успешно достигнута. Все поставленные задачи реализованы.

Программа позволяет авторизовать пользователей, загружать данные из базы и отображать их в удобном виде, выполнять операции добавления, изменения и удаления записей, менять статус билетов, автоматически создавать места и билеты при изменении параметров зала или сеанса.

Разработанное приложение полностью соответствует требованиям технического задания, обеспечивая логическую и физическую целостность данных, защиту информации и разграничение прав доступа между администратором и оператором. Вместе с тем, в дальнейшем программу можно улучшить, внедрив дополнительные функции, такие как шифрование паролей, поддержка печати билетов, интеграция с внешними системами, использование паттернов проектирования, например Repository или MVVM, а также переход на более современные платформы, такие как .NET MAUI или ASP.NET, для создания кроссплатформенного или веб-приложения. Эти доработки позволят повысить надежность, расширяемость и удобство использования системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Microsoft Learn — C# Guide [Электронный ресурс] — URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 05.06.2025)
  2. Windows Forms Documentation (Microsoft Learn) [Электронный ресурс] — URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/> (дата обращения: 07.06.2025)
  3. Официальная документация MySQL [Электронный ресурс] — URL: [https://dev.mysql.com/doc/](https://dev.mysql.com/doc/%20%20)  (дата обращения: 08.06.2025)
  4. Виды базы данных [Электронный ресурс] — URL: [https://habr.com/ru/companies/amvera/articles/754702/](https://habr.com/ru/companies/amvera/articles/754702/%20) (дата обращения: 08.06.2025)
  5. Microsoft Learn — Объектно-ориентированное программирование в C# [Электронный ресурс] — URL: [https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/object-oriented](https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/object-oriented?spm=a2ty_o01.29997173.0.0.6818c921bt2gql) (дата обращения: 09.06.2025)
  6. Visual Studio — Microsoft [Электронный ресурс] — URL: [https://visualstudio.microsoft.com/](https://visualstudio.microsoft.com/%20) (дата обращения: 09.06.2025)

ПРИЛОЖЕНИЕ А

Министерство ОБРАЗОВАНИЯ И науки Российской Федерации  
Федеральное автономное образовательное учреждение высшего профессионального образования  
«Санкт-Петербургский государственный политехнический университет Петра Великого»  
(ФГАОУ ВО «СПбПУ)  
**Институт среднего профессионального образования**

УТВЕРЖДАЮ  
Председатель ПЦК   
\_\_\_\_\_\_\_\_\_\_\_ Сынкова А.Д.  
\_\_ .\_\_. 2025

Приложение по работе с БД Кинотеатра  
**Техническое задание**  
Листов 6

ПРОВЕРИЛ  
Преподаватель   
\_\_\_\_\_\_\_\_\_\_\_ Челищева Л.Д.  
19.06.2025

ВЫПОЛНИЛ  
Студент группы 32919/9   
\_\_\_\_\_\_\_\_\_\_\_ Судник Е.А.  
19.06.2025

2025

1. **ВВЕДЕНИЕ**
   1. Полное наименование программной разработки: «Приложение по работе с БД кинотеатра».
   2. Приложение по работе с БД кинотеатра, позволяющая продавать билеты, добавлять и редактировать фильмы, залы, сеансы, сотрудников.
   3. В соответствии с заданием программный продукт состоит из теоретической и практической частей. Теоретическая часть включает описание предметной области задачи, анализ методов её решения, обзор и обоснование выбора средств программирования. Практическая часть включает подробное описание проектирования программного продукта, разработку и реализацию на языке программирования C# с использованием среды программирования Visual Studio программных модулей программного продукта.
2. **ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ**
   1. Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.
3. **НАЗНАЧЕНИЕ РАЗРАБОТКИ**
   1. Основное назначение программного продукта: программа предназначена для управления данными кинотеатра, включая информацию о фильмах, залах, сеансах, билетах и сотрудниках.
   2. Эксплуатационное назначение программного продукта: программа предназначена для программа предназначена для пользователей, работающих в сфере управления кинотеатром, для эффективного сбора, хранения, обработки и анализа данных о фильмах, залах, сеансах, билетах и сотрудниках.

**4 ТРЕБОВАНИЯ К РАЗРАБОТКЕ**

4.1. Требования к функциональным характеристикам

4.1.1 Состав выполняемых функций

Программа предоставляет следующие функции: авторизация сотрудников с разграничением прав доступа, просмотр, добавление, редактирование и удаление данных о фильмах, залах, сеансах и билетах, поиск фильмов по названию, обновление информации в интерфейсе, сохранение данных в базу данных и загрузка из неё. Для пользователей с ролью «Администратор» предусмотрена возможность управления таблицей «Сотрудники», включая добавление, редактирование и удаление учетных записей.

4.1.2 Организация входных и выходных данных

Входные: MovieID, HallID, SessionID, TicketID, EmployeeID, название фильма, длительность фильма, описание фильма, имя зала, количество мест в зале, дата и время сеанса, цена билета, номер места, статус билета, логин сотрудника, пароль, ФИО сотрудника, роль.

4.1.3 Временная характеристика

* + - * + время отклика системы должно быть не более 4 секунд;
        + система должна обеспечивать доступ к данным не более, чем за 5 секунды.

4.2 Требования к надежности:

* + - * + использование лицензированного программного обеспечения;
        + проверка программы на наличие вирусов;
        + организация бесперебойного питания.

4.3 Условия эксплуатации

4.3.1 Температура окружающего воздуха

Программное обеспечение предназначено для работы в стандартных климатических условиях офисного помещения.

4.3.2 Относительная влажность

Программное обеспечение рассчитано на работу в условиях умеренной влажности, характерной для офисных помещений.

4.4 Требования к составу и параметрам технических средств

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

* + - * + процессор Intel или другой совместимый;
        + объем свободной оперативной памяти ~50 Мб;
        + объем необходимой памяти на жестком диске ~150Мб;
        + стандартный VGA-монитор или совместимый.

4.5 Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы Microsoft Windows 7 или выше. Язык интерфейса – русский.

4.6 Требования к маркировке и упаковке

Программа должна поставляться в виде проекта, исполняемого (еxе) файла, установщика и документации.

4.7 Требования к транспортировке и хранению

Программа распространяется в электронном виде. Требования к транспортировке и хранению не предъявляются.

4.8 Требования к программной документации: техническое задание, пояснительная записка.

**5 ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ**

5.1. Предварительный состав программной документации:

- «Техническое задание»;

- разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии;

- разрабатываемое программное обеспечение должно включать справочную систему.

5.2. Перечень материалов пояснительной записки

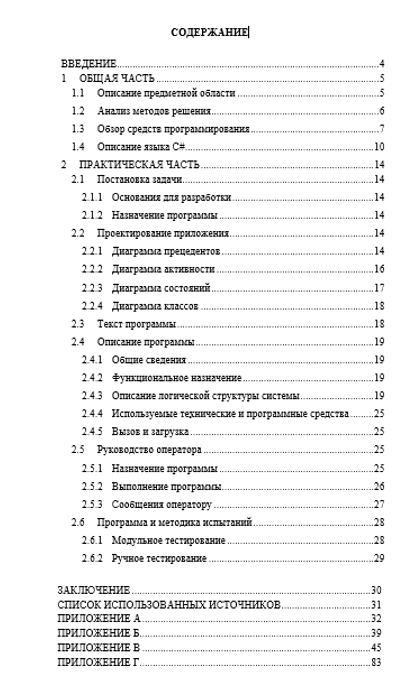


Рисунок 1 – Перечень материалов пояснительной записки

**6 ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ**

Технико-экономические показатели не рассчитываются.

**7 СТАДИИ И ЭТАПЫ РАЗРАБОТКИ**

Таблица 1 – Стадии и этапы разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Содержание стадии** | **Содержание этапа** | **Срок 2024 г.** | | **Форма  отчетности** |
| **начало** | **конец** |
| Техническое задание | Составление технического задания | 13.01 | 14.01 | Техническое задание |
| Эскизный проект | Проектирование программы | 15.01 | 15.02 | UML-диаграммы |
| Рабочий проект | Разработка спецификаций | 16.03 | 16.04 | Спецификации отдельных компонентов |
| Составление программы | 17.04 | 14.06 | Программная документация |
| Приёмо-сдаточные испытания | 15.06 | | Протокол испытаний (п. 2.7 пояснительной записки) |
| Приёмка | Защита курсового проекта | 19.06 | | Оценка за курсовой проект |

**8 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ**

8.1. Порядок контроля

Контроль выполнения должен осуществляться руководителем курсового проекта (преподавателем) в соответствие с п.7.

8.2. Порядок приемки

Приемка должна осуществляться с участием руководителя после проведения приемо-сдаточных испытаний. В результате защиты курсового проекта должна быть выставлена оценка за курсовой проект.

ПРИЛОЖЕНИЕ Б

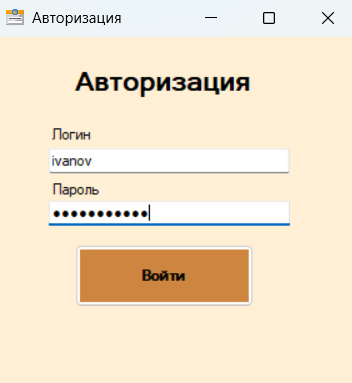


Рисунок Б. 1 – Авторизация пользователя

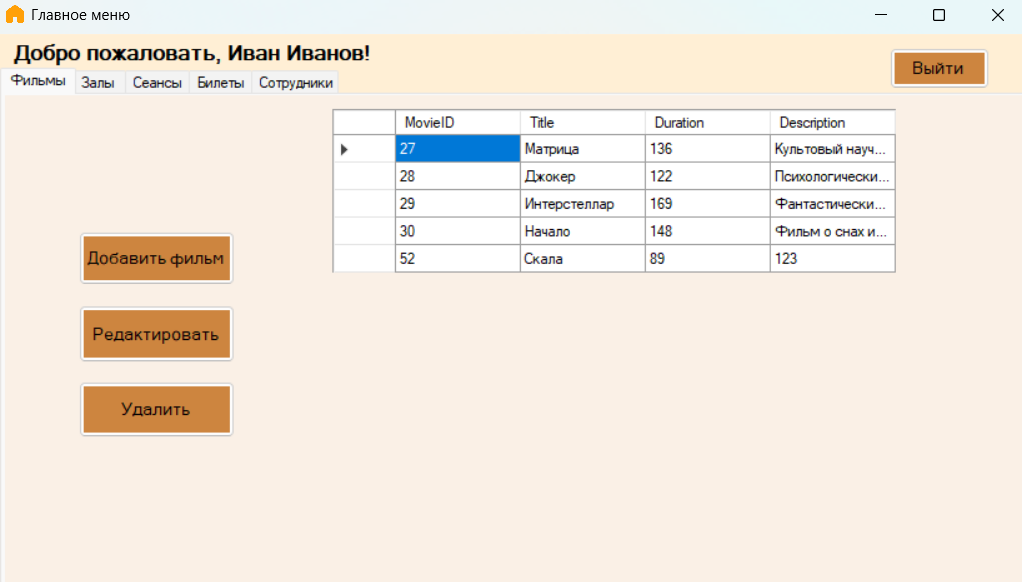


Рисунок Б. 2 – Вкладка фильмы

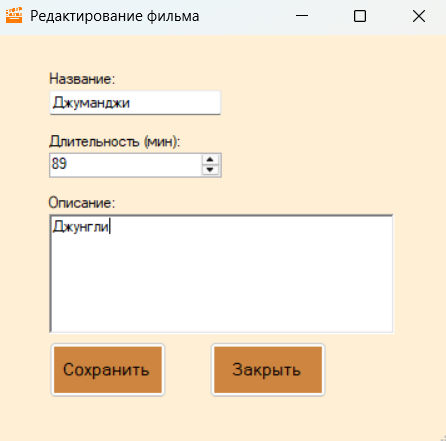


Рисунок Б. 3 – Форма добавления фильма

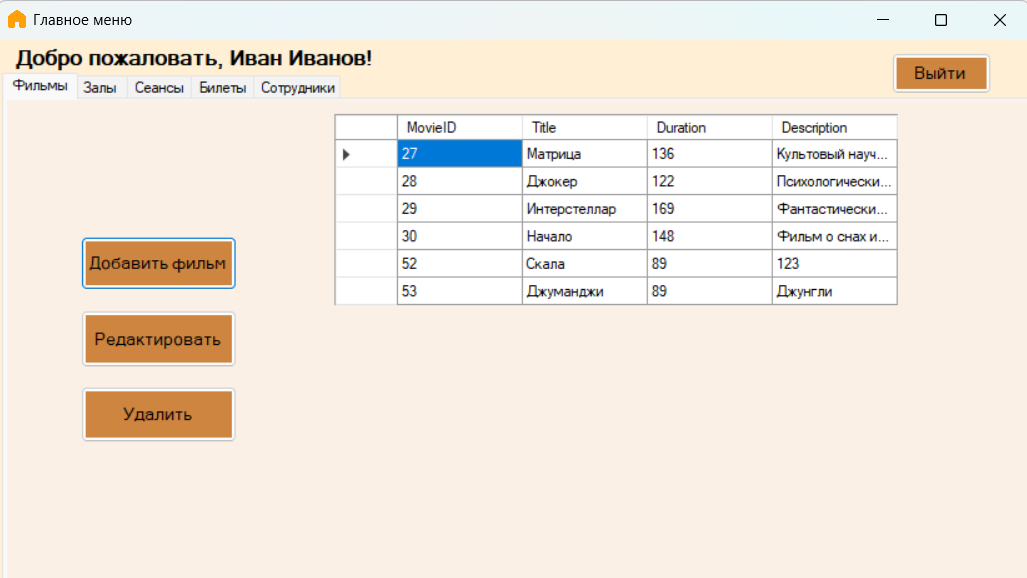


Рисунок Б. 4 – Результат добавления фильма

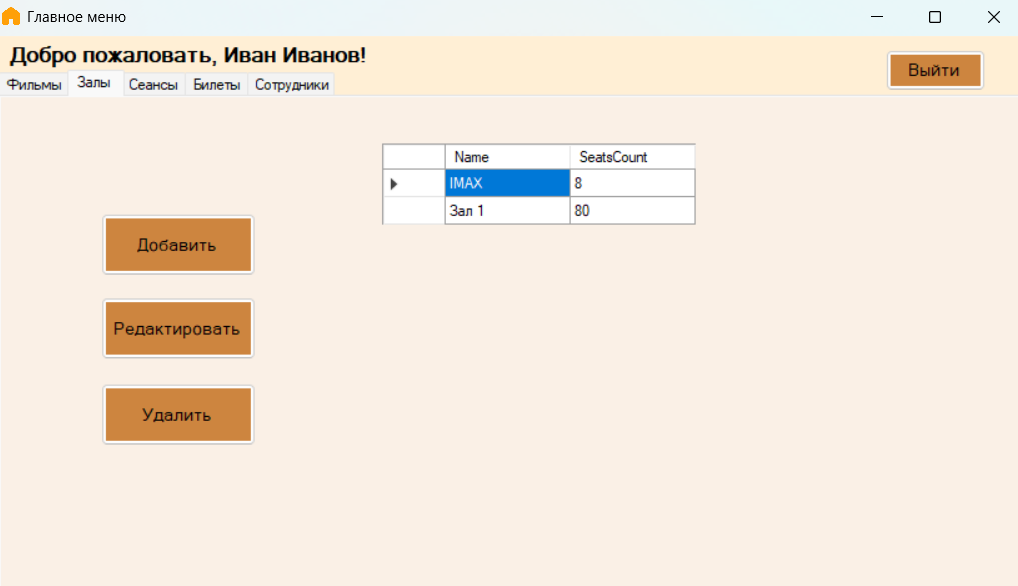


Рисунок Б. 5 - Вкладка Залы

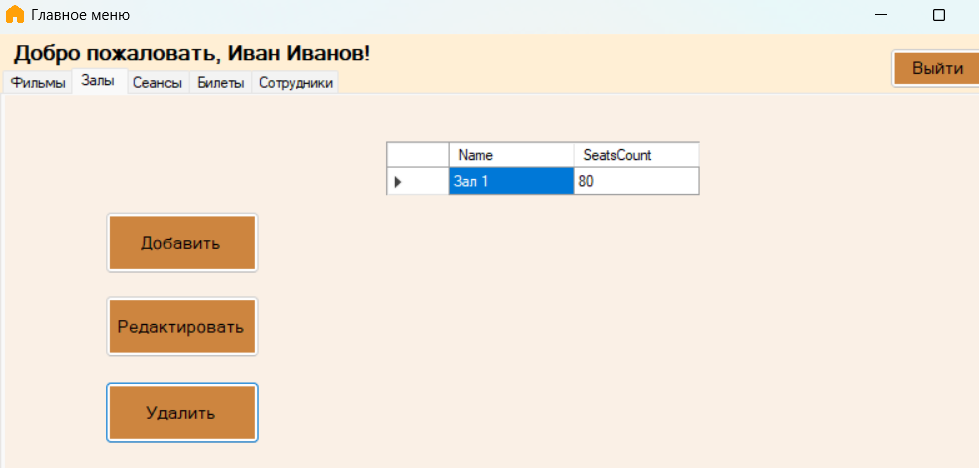


Рисунок Б. 6 – Таблица после удаления зала

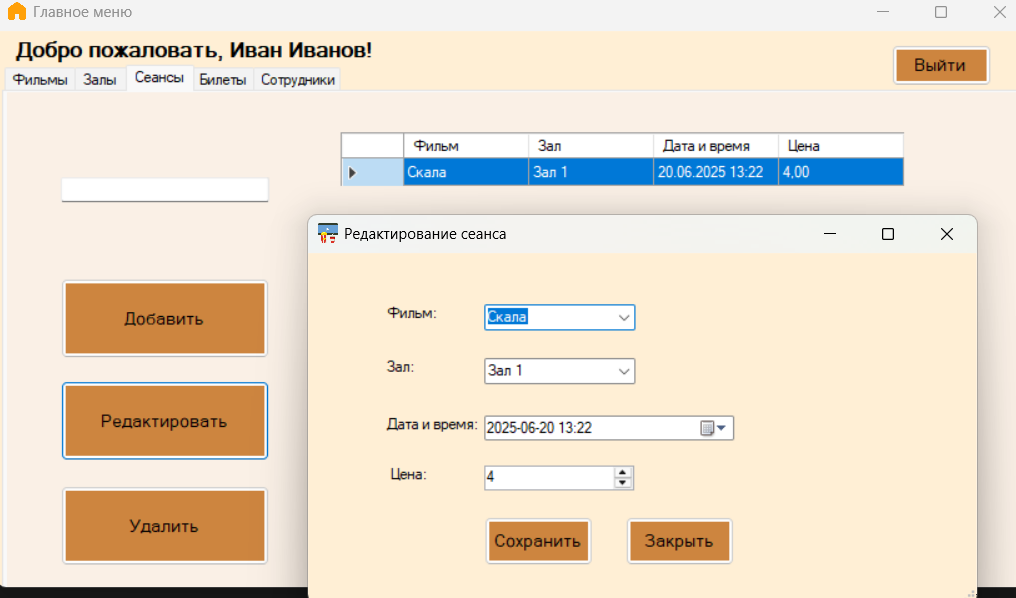


Рисунок Б. 6 - Форма редактирования сеанса

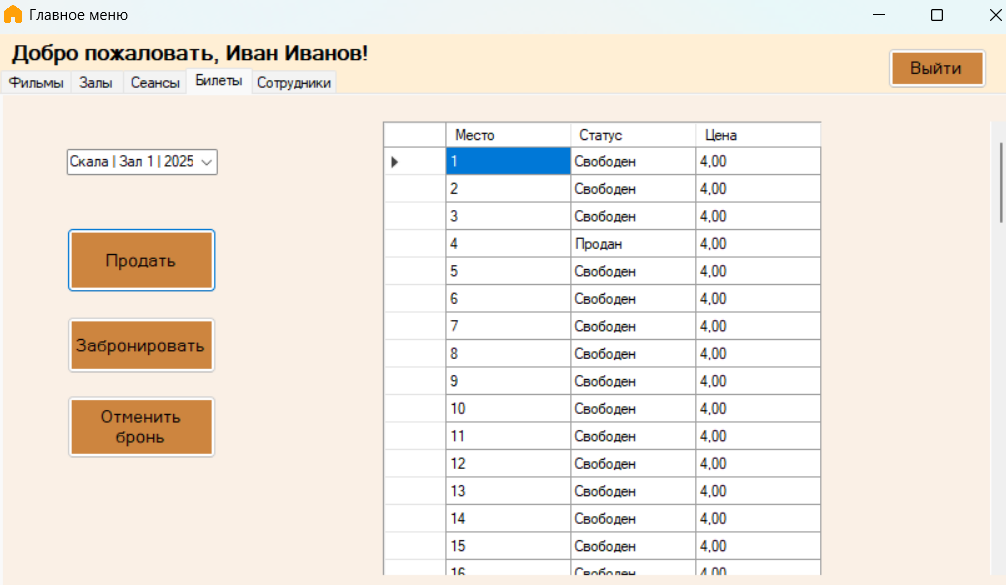


Рисунок Б. 7 - Продажа билета

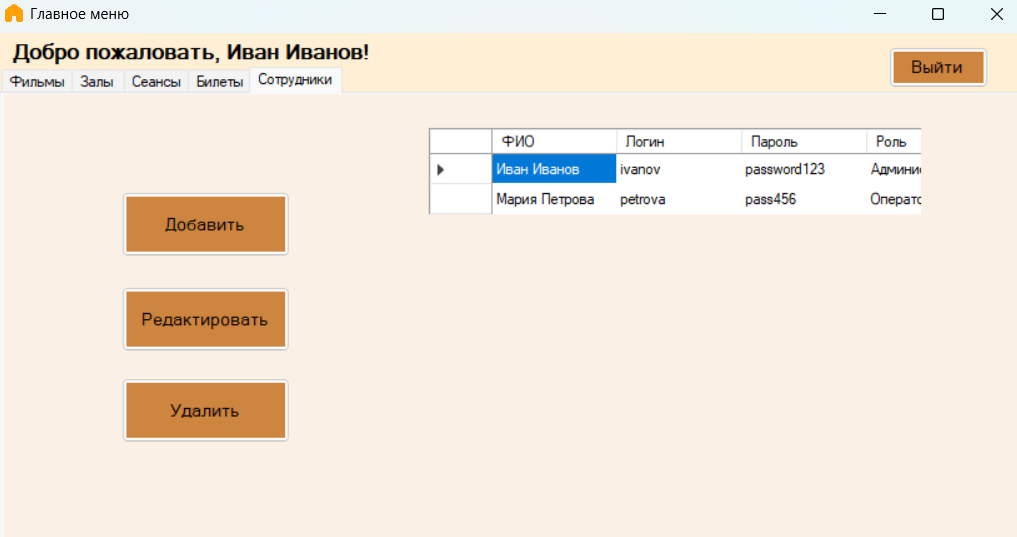


Рисунок Б. 8– Вкладка сотрудники

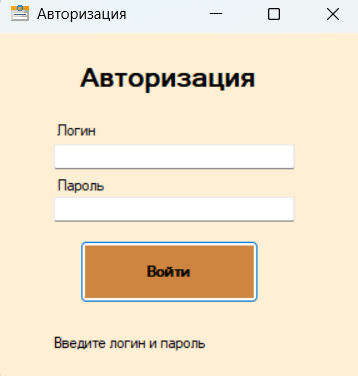


Рисунок Б. 9 - Авторизация с пустыми полями

Таблица Б. 1 – Протокол испытаний

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Название** | **Шаги** | **Тестовые данные** | **Ожидаемые результаты** | **Фактические результаты** | **Статус** |
| 1. Авторизация пользователя | 1. Ввести логин и пароль  2. Нажать «Войти» | Логин: ivanov  Пароль:  password123 | Появляется главное меню с приветствием | Появляется главное меню с приветствием | Пройден |
| 2. Добавление фильма | 1. Ввести логин и пароль.  2. Нажать «Войти».  3. Перейти во вкладку «Фильмы».  4. Нажать добавить.  5. Ввести данные в поля.  6. Нажать «Сохранить». | Логин: ivanov  Пароль:  password123  Название: Ветер  Прод-ость: 89  Описание: Крутой | Фильм добавляется в таблицу. | Фильм добавляется в таблицу. | Пройден |
| 3. Редактирование сеанса | 1. Ввести логин и пароль.  2. Нажать «Войти».  3. Перейти во вкладку «Сеансы».  4.Выбрать самые первый сеанс.  4. Нажать редактировать.  5. Изменить данные в поле «Цена».  6. Нажать «Сохранить». | Цена: 76 | Поле цена изменилась у сеанса в таблице. | Поле цена изменилась у сеанса в таблице. | Пройден |
| 4. Удаление зала | 1. Ввести логин и пароль.  2. Нажать «Войти».  3. Перейти во вкладку «Залы».  4.Выбрать самый первый зал.  5. Нажать удалить. |  | Зал удалился из таблицы. | Зал удалился из таблицы. | Пройден |
| 5. Продажа билета | 1. Ввести логин и пароль.  2. Нажать «Войти».  3. Перейти во вкладку «Билеты».  4.Выбрать самые первый сеанс в выпадающем списке.  4.Выбрать самые первый билет.  4. Нажать продать. |  | У билета в таблице меняется статус на продан. | У билета в таблице меняется статус на продан. | Пройден |
| 6. Добавление сотрудника | 1. Ввести логин и пароль.  2. Нажать «Войти».  3. Перейти во вкладку «сотрудники».  4. Нажать добавить.  5. Ввести данные в поля.  6. Нажать «Сохранить». | ФИО: ФИО  Логин:123  Пароль:123  Роль: Оператор | Сотрудник добавляется в таблицу. | Сотрудник добавляется в таблицу. | Пройден |
| 7. Авторизация пользователя с пустыми полями | 1. Нажать войти. |  | Появляется сообщение «Введите логин и пароль». | Появляется сообщение «Введите логин и пароль». | Пройден |

ПРИЛОЖЕНИЕ В

**В.1 Модуль LoginForm.cs**

using System;

using System.Windows.Forms;

using MySql.Data.MySqlClient;

namespace CinemaApp

{

public partial class LoginForm : Form

{

private string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

public LoginForm()

{

InitializeComponent();

}

private void btnLogin\_Click(object sender, EventArgs e)

{

string username = txtUsername.Text.Trim();

string password = txtPassword.Text.Trim();

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))

{

lblMessage.Text = "Введите логин и пароль";

return;

}

try

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT FullName, Role FROM Employees WHERE Login = @login AND Password = @password";

using (MySqlCommand cmd = new MySqlCommand(query, con))

{

cmd.Parameters.AddWithValue("@login", username);

cmd.Parameters.AddWithValue("@password", password);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

string fullName = reader.GetString("FullName");

string role = reader.GetString("Role");

MessageBox.Show("Успешный вход!", "Добро пожаловать", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Hide();

MainForm mainForm = new MainForm(fullName, role);

mainForm.FormClosed += (s, args) => this.Show();

mainForm.Show();

}

else

{

lblMessage.Text = "Неверный логин или пароль";

}

}

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка подключения к базе данных: " + ex.Message);

}

}

}

}

**В.2 Модуль MainForm.cs**

using CinemaApp.Models;

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

namespace CinemaApp

{

public partial class MainForm : Form

{

private string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

private string \_currentUserRole;

public MainForm(string userName, string userRole)

{

InitializeComponent();

LoadMovies();

LoadHalls();

LoadSessions();

LoadEmployees();

LoadSessionsForTickets();

if (cbSessionsFilter.Items.Count > 0)

{

cbSessionsFilter.SelectedIndex = 0;

}

lblWelcome.Text = $"Добро пожаловать, {userName}!";

\_currentUserRole = userRole;

if (!\_currentUserRole.Equals("Администратор", StringComparison.OrdinalIgnoreCase))

{

tabControlMain.TabPages.Remove(tabPageEmployees);

}

}

private void LoadSessionsForTickets()

{

try

{

var sessions = Session.GetAll(connectionString);

var movies = Movie.GetAll(connectionString).ToDictionary(m => m.MovieID, m => m.Title);

var halls = Hall.GetAll(connectionString).ToDictionary(h => h.HallID, h => h.Name);

var displayList = sessions.Select(s => new

{

s.SessionID,

SessionInfo = $"{movies[s.MovieID]} | {halls[s.HallID]} | {s.StartTime:yyyy-MM-dd HH:mm}"

}).ToList();

cbSessionsFilter.DataSource = displayList;

cbSessionsFilter.ValueMember = "SessionID";

cbSessionsFilter.DisplayMember = "SessionInfo";

if (cbSessionsFilter.Items.Count > 0)

{

cbSessionsFilter.SelectedIndex = 0;

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки сеансов для фильтра: " + ex.Message);

}

}

private void LoadTickets(int sessionId)

{

try

{

List<Ticket> tickets = Ticket.GetAllBySession(sessionId, connectionString);

var result = from t in tickets

join seat in GetSeatsBySession(sessionId, connectionString) on t.SeatID equals seat.SeatID

join session in Session.GetAll(connectionString) on t.SessionID equals session.SessionID

select new

{

seat.SeatID,

seat.SeatNumber,

t.Status,

session.Price

};

dgvTickets.DataSource = result.ToList();

dgvTickets.Columns["SeatID"].Visible = false;

dgvTickets.Columns["SeatNumber"].HeaderText = "Место";

dgvTickets.Columns["Status"].HeaderText = "Статус";

dgvTickets.Columns["Price"].HeaderText = "Цена";

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки билетов: " + ex.Message);

}

}

private List<Seat> GetSeatsBySession(int sessionId, string connStr)

{

using (var con = new MySqlConnection(connStr))

{

con.Open();

Session session = Session.GetById(sessionId, connStr);

if (session == null)

{

return new List<Seat>();

}

int hallId = session.HallID;

return Seat.GetAllByHall(hallId, connStr);

}

}

private void LoadMovies()

{

try

{

List<Movie> movies = Movie.GetAll(connectionString);

dgvMovies.DataSource = movies;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки фильмов: " + ex.Message);

}

}

private void LoadHalls()

{

try

{

List<Hall> halls = Hall.GetAll(connectionString);

dgvHalls.DataSource = halls;

dgvHalls.Columns["HallID"].Visible = false;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки залов: " + ex.Message);

}

}

private void LoadEmployees()

{

try

{

List<Employee> employees = Employee.GetAll(connectionString);

dgvEmployees.DataSource = employees;

dgvEmployees.Columns["EmployeeID"].Visible = false;

dgvEmployees.Columns["FullName"].HeaderText = "ФИО";

dgvEmployees.Columns["Login"].HeaderText = "Логин";

dgvEmployees.Columns["Password"].HeaderText = "Пароль";

dgvEmployees.Columns["Role"].HeaderText = "Роль";

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки сотрудников: " + ex.Message);

}

}

private List<Session> \_allSessions;

private Dictionary<int, string> \_movieTitles;

private Dictionary<int, string> \_hallNames;

private void LoadSessions(string movieFilter = "")

{

try

{

if (\_allSessions == null || \_movieTitles == null || \_hallNames == null)

{

\_allSessions = Session.GetAll(connectionString);

var movies = Movie.GetAll(connectionString);

var halls = Hall.GetAll(connectionString);

\_movieTitles = movies.ToDictionary(m => m.MovieID, m => m.Title);

\_hallNames = halls.ToDictionary(h => h.HallID, h => h.Name);

}

var filtered = string.IsNullOrEmpty(movieFilter)

? \_allSessions

: \_allSessions.Where(s =>

{

string title;

return \_movieTitles.TryGetValue(s.MovieID, out title) &&

title.IndexOf(movieFilter, StringComparison.OrdinalIgnoreCase) >= 0;

}).ToList();

var displayList = filtered.Select(s =>

{

string movieTitle;

string hallName;

\_movieTitles.TryGetValue(s.MovieID, out movieTitle);

\_hallNames.TryGetValue(s.HallID, out hallName);

return new

{

s.SessionID,

Movie = movieTitle ?? "Неизвестен",

Hall = hallName ?? "Неизвестен",

s.StartTime,

s.Price

};

}).ToList();

dgvSessions.DataSource = displayList;

FormatSessionGrid();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки сеансов: " + ex.Message);

}

}

private void FormatSessionGrid()

{

if (dgvSessions.Columns.Count == 0) return;

dgvSessions.Columns["SessionID"].Visible = false;

dgvSessions.Columns["Movie"].HeaderText = "Фильм";

dgvSessions.Columns["Hall"].HeaderText = "Зал";

dgvSessions.Columns["StartTime"].HeaderText = "Дата и время";

dgvSessions.Columns["Price"].HeaderText = "Цена";

}

private void btnDeleteMovie\_Click(object sender, EventArgs e)

{

if (dgvMovies.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите фильм для удаления.");

return;

}

int movieId = Convert.ToInt32(dgvMovies.SelectedRows[0].Cells["MovieID"].Value);

var movie = Movie.GetById(movieId, connectionString);

if (movie == null)

{

MessageBox.Show("Фильм не найден.");

return;

}

if (MessageBox.Show($"Удалить фильм '{movie.Title}'?", "Подтверждение", MessageBoxButtons.YesNo) == DialogResult.Yes)

{

try

{

movie.Delete(connectionString);

LoadMovies();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка удаления фильма: " + ex.Message);

}

}

}

private void btnAddMovie\_Click(object sender, EventArgs e)

{

MovieEditForm form = new MovieEditForm();

if (form.ShowDialog() == DialogResult.OK)

{

LoadMovies();

}

}

private void btnEditMovie\_Click(object sender, EventArgs e)

{

if (dgvMovies.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите фильм для редактирования.");

return;

}

int movieId = Convert.ToInt32(dgvMovies.SelectedRows[0].Cells["MovieID"].Value);

var movie = Movie.GetById(movieId, connectionString);

if (movie == null)

{

MessageBox.Show("Фильм не найден.");

return;

}

MovieEditForm form = new MovieEditForm(movie.MovieID, movie.Title, movie.Duration, movie.Description);

if (form.ShowDialog() == DialogResult.OK)

{

LoadMovies();

}

}

private void btnAddHall\_Click(object sender, EventArgs e)

{

HallEditForm form = new HallEditForm();

if (form.ShowDialog() == DialogResult.OK)

{

LoadHalls();

}

}

private void btnEditHall\_Click(object sender, EventArgs e)

{

if (dgvHalls.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите зал для редактирования.");

return;

}

int hallId = Convert.ToInt32(dgvHalls.SelectedRows[0].Cells["HallID"].Value);

var hall = Hall.GetById(hallId, connectionString);

if (hall == null)

{

MessageBox.Show("Зал не найден.");

return;

}

HallEditForm form = new HallEditForm(hall.HallID, hall.Name, hall.SeatsCount);

if (form.ShowDialog() == DialogResult.OK)

{

LoadHalls();

}

}

private void btnDeleteHall\_Click(object sender, EventArgs e)

{

if (dgvHalls.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите зал для удаления.");

return;

}

int hallId = Convert.ToInt32(dgvHalls.SelectedRows[0].Cells["HallID"].Value);

var hall = Hall.GetById(hallId, connectionString);

if (hall == null)

{

MessageBox.Show("Зал не найден.");

return;

}

if (MessageBox.Show($"Удалить зал '{hall.Name}'?", "Подтверждение", MessageBoxButtons.YesNo) == DialogResult.Yes)

{

try

{

hall.Delete(connectionString);

ClearSessionCache();

LoadHalls();

LoadSessions();

LoadSessionsForTickets();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка удаления зала: " + ex.Message);

}

}

}

private void btnAddSession\_Click(object sender, EventArgs e)

{

SessionEditForm form = new SessionEditForm();

if (form.ShowDialog() == DialogResult.OK)

{

ClearSessionCache();

LoadSessions();

LoadSessionsForTickets();

}

}

private void btnEditSession\_Click(object sender, EventArgs e)

{

if (dgvSessions.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите сеанс для редактирования.");

return;

}

int sessionId = Convert.ToInt32(dgvSessions.SelectedRows[0].Cells["SessionID"].Value);

var session = Session.GetById(sessionId, connectionString);

if (session == null)

{

MessageBox.Show("Сеанс не найден.");

return;

}

string movieTitle = dgvSessions.SelectedRows[0].Cells["Movie"].Value.ToString();

string hallName = dgvSessions.SelectedRows[0].Cells["Hall"].Value.ToString();

SessionEditForm form = new SessionEditForm(

session.SessionID,

movieTitle,

hallName,

session.StartTime,

session.Price

);

if (form.ShowDialog() == DialogResult.OK)

{

ClearSessionCache();

LoadSessions();

LoadSessionsForTickets();

}

}

private void btnDeleteSession\_Click(object sender, EventArgs e)

{

if (dgvSessions.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите сеанс для удаления.");

return;

}

int sessionId = Convert.ToInt32(dgvSessions.SelectedRows[0].Cells["SessionID"].Value);

string movieTitle = dgvSessions.SelectedRows[0].Cells["Movie"].Value.ToString();

if (MessageBox.Show($"Удалить сеанс фильма '{movieTitle}'?", "Подтверждение", MessageBoxButtons.YesNo) == DialogResult.Yes)

{

try

{

var session = Session.GetById(sessionId, connectionString);

if (session == null)

{

MessageBox.Show("Сеанс не найден.");

return;

}

session.Delete(connectionString);

ClearSessionCache();

LoadSessions();

LoadSessionsForTickets();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка удаления сеанса: " + ex.Message);

}

}

}

private void cbSessionsFilter\_SelectedIndexChanged(object sender, EventArgs e)

{

if (cbSessionsFilter.SelectedValue is int sessionId)

{

LoadTickets(sessionId);

}

else if (cbSessionsFilter.SelectedValue is long sessionIdLong)

{

LoadTickets((int)sessionIdLong);

}

}

private void btnSellTicket\_Click(object sender, EventArgs e)

{

int? result = GetSelectedSeatAndSession();

if (result == null) return;

int seatId = result.Value;

int sessionId = (int)cbSessionsFilter.SelectedValue;

ChangeTicketStatus(seatId, sessionId, "Продан");

}

private void btnReserveTicket\_Click(object sender, EventArgs e)

{

int? result = GetSelectedSeatAndSession();

if (result == null) return;

int seatId = result.Value;

int sessionId = (int)cbSessionsFilter.SelectedValue;

ChangeTicketStatus(seatId, sessionId, "Забронирован");

}

private void btnCancelReservation\_Click(object sender, EventArgs e)

{

int? result = GetSelectedSeatAndSession();

if (result == null) return;

int seatId = result.Value;

int sessionId = (int)cbSessionsFilter.SelectedValue;

ChangeTicketStatus(seatId, sessionId, "Свободен");

}

private int? GetSelectedSeatAndSession()

{

if (dgvTickets.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите место.");

return null;

}

try

{

int seatId = Convert.ToInt32(dgvTickets.SelectedRows[0].Cells["SeatID"].Value);

return seatId;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка получения номера места: " + ex.Message);

return null;

}

}

private void ChangeTicketStatus(int seatId, int sessionId, string newStatus)

{

try

{

Ticket ticket = Ticket.GetAllBySession(sessionId, connectionString)

.FirstOrDefault(t => t.SeatID == seatId);

if (ticket == null)

{

ticket = new Ticket(sessionId, seatId, newStatus);

}

else

{

ticket.Status = newStatus;

}

ticket.Save(connectionString);

LoadTickets(sessionId);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка обновления статуса билета: " + ex.Message);

}

}

private void btnAddEmployee\_Click(object sender, EventArgs e)

{

EmployeeEditForm form = new EmployeeEditForm();

if (form.ShowDialog() == DialogResult.OK)

{

LoadEmployees();

}

}

private void btnEditEmployee\_Click(object sender, EventArgs e)

{

if (dgvEmployees.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите сотрудника для редактирования.");

return;

}

int employeeId = Convert.ToInt32(dgvEmployees.SelectedRows[0].Cells["EmployeeID"].Value);

var employee = Employee.GetById(employeeId, connectionString);

if (employee == null)

{

MessageBox.Show("Сотрудник не найден.");

return;

}

EmployeeEditForm form = new EmployeeEditForm(

employee.EmployeeID,

employee.FullName,

employee.Login,

employee.Password,

employee.Role

);

if (form.ShowDialog() == DialogResult.OK)

{

LoadEmployees();

}

}

private void btnDeleteEmployee\_Click(object sender, EventArgs e)

{

if (dgvEmployees.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите сотрудника для удаления.");

return;

}

int employeeId = Convert.ToInt32(dgvEmployees.SelectedRows[0].Cells["EmployeeID"].Value);

var employee = Employee.GetById(employeeId, connectionString);

if (employee == null)

{

MessageBox.Show("Сотрудник не найден.");

return;

}

if (MessageBox.Show($"Удалить сотрудника '{employee.FullName}'?", "Подтверждение", MessageBoxButtons.YesNo) == DialogResult.Yes)

{

try

{

employee.Delete(connectionString);

LoadEmployees();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка удаления сотрудника: " + ex.Message);

}

}

}

private void btnLogout\_Click(object sender, EventArgs e)

{

this.Close();

LoginForm loginForm = new LoginForm();

loginForm.Show();

}

private void txtMovieSearch\_TextChanged(object sender, EventArgs e)

{

LoadSessions(txtMovieSearch.Text.Trim());

}

private void ClearSessionCache()

{

\_allSessions = null;

\_movieTitles = null;

\_hallNames = null;

}

}

}

**В.3 Модуль MovieEditForm.cs**

using CinemaApp.Models;

using MySql.Data.MySqlClient;

using System;

using System.Windows.Forms;

namespace CinemaApp

{

public partial class MovieEditForm : Form

{

private int \_movieId = 0;

private string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

public MovieEditForm()

{

InitializeComponent();

}

public MovieEditForm(int movieId, string title, int duration, string description) : this()

{

\_movieId = movieId;

txtTitle.Text = title;

numDuration.Value = duration;

txtDescription.Text = description;

}

private void btnSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(txtTitle.Text))

{

MessageBox.Show("Введите название фильма.");

return;

}

try

{

// Создаем или обновляем объект Movie

if (\_movieId == 0)

{

// Добавление нового фильма

var movie = new Movie(

txtTitle.Text.Trim(),

(int)numDuration.Value,

txtDescription.Text.Trim()

);

movie.Save(connectionString);

}

else

{

// Редактирование существующего фильма

var movie = Movie.GetById(\_movieId, connectionString);

if (movie == null)

{

MessageBox.Show("Фильм не найден.");

return;

}

movie.Title = txtTitle.Text.Trim();

movie.Duration = (int)numDuration.Value;

movie.Description = txtDescription.Text.Trim();

movie.Save(connectionString);

}

this.DialogResult = DialogResult.OK;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении: " + ex.Message);

}

}

private void btnCancel\_Click(object sender, EventArgs e)

{

this.DialogResult = DialogResult.Cancel;

}

}

}

**В.4 Модуль SessionEditForm.cs**

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

namespace CinemaApp

{

public partial class SessionEditForm : Form

{

private int \_sessionId = 0;

private string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

public SessionEditForm()

{

InitializeComponent();

LoadMovies();

LoadHalls();

}

public SessionEditForm(int sessionId, string movie, string hall, DateTime startTime, decimal price) : this()

{

\_sessionId = sessionId;

cbMovies.SelectedIndex = cbMovies.FindStringExact(movie);

cbHalls.SelectedIndex = cbHalls.FindStringExact(hall);

dtpStartTime.Value = startTime;

numPrice.Value = price;

}

private void LoadMovies()

{

try

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT MovieID, Title FROM Movies ORDER BY Title";

MySqlCommand cmd = new MySqlCommand(query, con);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

DataTable dt = new DataTable();

dt.Load(reader);

cbMovies.DataSource = dt;

cbMovies.ValueMember = "MovieID";

cbMovies.DisplayMember = "Title";

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки фильмов: " + ex.Message);

}

}

private void LoadHalls()

{

try

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT HallID, Name FROM Halls ORDER BY Name";

MySqlCommand cmd = new MySqlCommand(query, con);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

DataTable dt = new DataTable();

dt.Load(reader);

cbHalls.DataSource = dt;

cbHalls.ValueMember = "HallID";

cbHalls.DisplayMember = "Name";

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка загрузки залов: " + ex.Message);

}

}

private void btnSave\_Click(object sender, EventArgs e)

{

if (cbMovies.SelectedIndex < 0)

{

MessageBox.Show("Выберите фильм.");

return;

}

if (cbHalls.SelectedIndex < 0)

{

MessageBox.Show("Выберите зал.");

return;

}

try

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (\_sessionId == 0)

{

// Добавление сеанса

cmd = new MySqlCommand(

"INSERT INTO Sessions (MovieID, HallID, StartTime, Price) VALUES (@movieId, @hallId, @startTime, @price)",

con);

cmd.Parameters.AddWithValue("@movieId", cbMovies.SelectedValue);

cmd.Parameters.AddWithValue("@hallId", cbHalls.SelectedValue);

cmd.Parameters.AddWithValue("@startTime", dtpStartTime.Value);

cmd.Parameters.AddWithValue("@price", numPrice.Value);

cmd.ExecuteNonQuery();

long newSessionId = cmd.LastInsertedId;

int newHallId = Convert.ToInt32(cbHalls.SelectedValue);

// Создаём билеты только если их ещё нет

CreateTicketsForSession((int)newSessionId, newHallId);

}

else

{

// Обновление сеанса

cmd = new MySqlCommand(

"UPDATE Sessions SET MovieID = @movieId, HallID = @hallId, StartTime = @startTime, Price = @price WHERE SessionID = @id",

con);

cmd.Parameters.AddWithValue("@id", \_sessionId);

cmd.Parameters.AddWithValue("@movieId", cbMovies.SelectedValue);

cmd.Parameters.AddWithValue("@hallId", cbHalls.SelectedValue);

cmd.Parameters.AddWithValue("@startTime", dtpStartTime.Value);

cmd.Parameters.AddWithValue("@price", numPrice.Value);

cmd.ExecuteNonQuery();

}

}

this.DialogResult = DialogResult.OK;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении: " + ex.Message);

}

}

private void btnCancel\_Click(object sender, EventArgs e)

{

this.DialogResult = DialogResult.Cancel;

}

private void CreateTicketsForSession(int sessionId, int hallId)

{

try

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

// Получаем все места из нужного зала

string querySeats = "SELECT SeatID FROM Seats WHERE HallID = @hallId";

MySqlCommand cmdSeats = new MySqlCommand(querySeats, con);

cmdSeats.Parameters.AddWithValue("@hallId", hallId);

List<int> seatIds = new List<int>();

using (MySqlDataReader reader = cmdSeats.ExecuteReader())

{

while (reader.Read())

{

seatIds.Add(reader.GetInt32("SeatID"));

}

}

foreach (int seatId in seatIds)

{

// Проверяем, существует ли билет

string checkQuery = "SELECT COUNT(\*) FROM Tickets WHERE SessionID = @session AND SeatID = @seat";

MySqlCommand checkCmd = new MySqlCommand(checkQuery, con);

checkCmd.Parameters.AddWithValue("@session", sessionId);

checkCmd.Parameters.AddWithValue("@seat", seatId);

int count = Convert.ToInt32(checkCmd.ExecuteScalar());

if (count == 0)

{

// Если билета нет — создаём его

string insertQuery = "INSERT INTO Tickets (SessionID, SeatID, Status) VALUES (@session, @seat, 'Свободен')";

MySqlCommand insertCmd = new MySqlCommand(insertQuery, con);

insertCmd.Parameters.AddWithValue("@session", sessionId);

insertCmd.Parameters.AddWithValue("@seat", seatId);

insertCmd.ExecuteNonQuery();

}

}

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при создании билетов: " + ex.Message);

}

}

}

}

**В.5 Модуль HallEditForm.cs**

using CinemaApp.Models;

using MySql.Data.MySqlClient;

using System;

using System.Windows.Forms;

using System.Linq;

namespace CinemaApp

{

public partial class HallEditForm : Form

{

private int \_hallId = 0;

private string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

public HallEditForm()

{

InitializeComponent();

}

public HallEditForm(int hallId, string name, int seatsCount) : this()

{

\_hallId = hallId;

txtName.Text = name;

numSeatsCount.Value = seatsCount;

}

private void btnSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(txtName.Text))

{

MessageBox.Show("Введите название зала.");

return;

}

try

{

// Получаем данные из формы

string name = txtName.Text.Trim();

int newSeatsCount = (int)numSeatsCount.Value;

if (\_hallId == 0)

{

// Добавление нового зала

var hall = new Hall(name, newSeatsCount);

hall.Save(connectionString); // Создаст зал и места

// Места создаются внутри метода Save() или отдельно

for (int i = 1; i <= newSeatsCount; i++)

{

var seat = new Seat(hall.HallID, i);

seat.Save(connectionString);

}

}

else

{

// Редактирование существующего зала

var hall = Hall.GetById(\_hallId, connectionString);

if (hall == null)

{

MessageBox.Show("Зал не найден.");

return;

}

hall.Name = name;

hall.SeatsCount = newSeatsCount;

hall.Save(connectionString);

// Если количество мест увеличилось — добавляем новые места

var existingSeats = Seat.GetAllByHall(\_hallId, connectionString);

int oldSeatsCount = existingSeats.Count;

if (newSeatsCount > oldSeatsCount)

{

for (int i = oldSeatsCount + 1; i <= newSeatsCount; i++)

{

var seat = new Seat(\_hallId, i);

seat.Save(connectionString);

}

}

// Если количество мест уменьшилось — удаляем лишние

else if (newSeatsCount < oldSeatsCount)

{

foreach (var seat in existingSeats.Where(s => s.SeatNumber > newSeatsCount))

{

seat.Delete(connectionString);

}

}

}

this.DialogResult = DialogResult.OK;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении: " + ex.Message);

}

}

private void btnCancel\_Click(object sender, EventArgs e)

{

this.DialogResult = DialogResult.Cancel;

}

}

}

**В.6 Модуль EmployeeEditForm.cs**

using CinemaApp.Models;

using MySql.Data.MySqlClient;

using System;

using System.Windows.Forms;

namespace CinemaApp

{

public partial class EmployeeEditForm : Form

{

private int \_employeeId = 0;

private string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

public EmployeeEditForm()

{

InitializeComponent();

}

public EmployeeEditForm(int employeeId, string fullName, string login, string password, string role) : this()

{

\_employeeId = employeeId;

txtFullName.Text = fullName;

txtLogin.Text = login;

txtPassword.Text = password;

cbRole.SelectedItem = role;

}

private void btnCancel\_Click(object sender, EventArgs e)

{

this.DialogResult = DialogResult.Cancel;

}

private void btnSave\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(txtFullName.Text))

{

MessageBox.Show("Введите ФИО.");

return;

}

if (string.IsNullOrWhiteSpace(txtLogin.Text))

{

MessageBox.Show("Введите логин.");

return;

}

if (cbRole.SelectedIndex < 0)

{

MessageBox.Show("Выберите роль.");

return;

}

try

{

if (\_employeeId == 0)

{

// Добавление нового сотрудника

var newEmployee = new Employee(

txtFullName.Text.Trim(),

txtLogin.Text.Trim(),

txtPassword.Text.Trim(),

cbRole.SelectedItem.ToString()

);

newEmployee.Save(connectionString);

}

else

{

// Редактирование существующего

var existingEmployee = Employee.GetById(\_employeeId, connectionString); if (existingEmployee == null)

{

MessageBox.Show("Сотрудник не найден.");

return;

}

existingEmployee.FullName = txtFullName.Text.Trim();

existingEmployee.Login = txtLogin.Text.Trim();

if (!string.IsNullOrEmpty(txtPassword.Text))

{

existingEmployee.Password = txtPassword.Text.Trim();

}

existingEmployee.Role = cbRole.SelectedItem.ToString();

existingEmployee.Save(connectionString);

}

this.DialogResult = DialogResult.OK;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении: " + ex.Message);

}

}

}

}

**В.7 Класс Employee.cs**

using System;

using System.Collections.Generic;

using MySql.Data.MySqlClient;

namespace CinemaApp.Models

{

public class Employee

{

public int EmployeeID { get; set; }

public string FullName { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public string Role { get; set; }

// Конструктор для нового сотрудника

public Employee(string fullName, string login, string password, string role)

{

FullName = fullName;

Login = login;

Password = password;

Role = role;

}

// Конструктор для существующего сотрудника

public Employee(int employeeId, string fullName, string login, string password, string role)

{

EmployeeID = employeeId;

FullName = fullName;

Login = login;

Password = password;

Role = role;

}

// Сохраняет сотрудника (INSERT или UPDATE)

public void Save(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (EmployeeID == 0) // Новый сотрудник

{

cmd = new MySqlCommand(

"INSERT INTO Employees (FullName, Login, Password, Role) VALUES (@fullName, @login, @password, @role)",

con);

}

else // Обновляем существующего

{

cmd = new MySqlCommand(

"UPDATE Employees SET FullName = @fullName, Login = @login, Role = @role" +

(string.IsNullOrEmpty(Password) ? "" : ", Password = @password") +

" WHERE EmployeeID = @id",

con);

cmd.Parameters.AddWithValue("@id", EmployeeID);

}

cmd.Parameters.AddWithValue("@fullName", FullName);

cmd.Parameters.AddWithValue("@login", Login);

cmd.Parameters.AddWithValue("@role", Role);

if (!string.IsNullOrEmpty(Password))

{

cmd.Parameters.AddWithValue("@password", Password);

}

cmd.ExecuteNonQuery();

// Если это новый сотрудник, получаем его ID

if (EmployeeID == 0)

{

EmployeeID = (int)cmd.LastInsertedId;

}

}

}

// Удаляет сотрудника из БД

public void Delete(string connectionString)

{

if (EmployeeID == 0) throw new InvalidOperationException("Невозможно удалить несохранённого сотрудника.");

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand("DELETE FROM Employees WHERE EmployeeID = @id", con);

cmd.Parameters.AddWithValue("@id", EmployeeID);

cmd.ExecuteNonQuery();

}

}

// Получает всех сотрудников

public static List<Employee> GetAll(string connectionString)

{

List<Employee> employees = new List<Employee>();

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT EmployeeID, FullName, Login, Password, Role FROM Employees ORDER BY FullName";

MySqlCommand cmd = new MySqlCommand(query, con);

using (var reader = cmd.ExecuteReader())

{

while (reader.Read())

{

employees.Add(new Employee(

reader.GetInt32("EmployeeID"),

reader.GetString("FullName"),

reader.GetString("Login"),

reader.IsDBNull(reader.GetOrdinal("Password")) ? null : reader.GetString("Password"),

reader.GetString("Role")

));

}

}

}

return employees;

}

// Получает сотрудника по ID

public static Employee GetById(int id, string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT EmployeeID, FullName, Login, Password, Role FROM Employees WHERE EmployeeID = @id";

MySqlCommand cmd = new MySqlCommand(query, con);

cmd.Parameters.AddWithValue("@id", id);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Employee(

reader.GetInt32("EmployeeID"),

reader.GetString("FullName"),

reader.GetString("Login"),

reader.IsDBNull(reader.GetOrdinal("Password")) ? null : reader.GetString("Password"),

reader.GetString("Role")

);

}

else

{

return null;

}

}

}

}

// Изменяет пароль сотрудника

public void ChangePassword(string newPassword, string connectionString)

{

Password = newPassword;

Save(connectionString);

}

}

}

**В.8 Класс Hall.cs**

using System;

using System.Collections.Generic;

using System.Data;

using MySql.Data.MySqlClient;

namespace CinemaApp.Models

{

public class Hall

{

public int HallID { get; set; }

public string Name { get; set; }

public int SeatsCount { get; set; }

// Конструктор для нового зала

public Hall(string name, int seatsCount)

{

Name = name;

SeatsCount = seatsCount;

}

// Конструктор для существующего зала

public Hall(int hallId, string name, int seatsCount)

{

HallID = hallId;

Name = name;

SeatsCount = seatsCount;

}

// Сохраняет зал (INSERT или UPDATE)

public void Save(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (HallID == 0)

{

cmd = new MySqlCommand(

"INSERT INTO Halls (Name, SeatsCount) VALUES (@name, @seatsCount)",

con);

}

else

{

cmd = new MySqlCommand(

"UPDATE Halls SET Name = @name, SeatsCount = @seatsCount WHERE HallID = @id",

con);

cmd.Parameters.AddWithValue("@id", HallID);

}

cmd.Parameters.AddWithValue("@name", Name);

cmd.Parameters.AddWithValue("@seatsCount", SeatsCount);

cmd.ExecuteNonQuery();

// Если это новый зал, получаем его ID

if (HallID == 0)

{

HallID = (int)cmd.LastInsertedId;

}

}

}

// Удаляет зал из БД

public void Delete(string connectionString)

{

if (HallID == 0) throw new InvalidOperationException("Невозможно удалить несохранённый зал.");

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

// Удаление билетов, связанных с этим залом

MySqlCommand deleteTickets = new MySqlCommand(@"

DELETE t FROM Tickets t

JOIN Sessions s ON t.SessionID = s.SessionID

WHERE s.HallID = @hallId", con);

deleteTickets.Parameters.AddWithValue("@hallId", HallID);

deleteTickets.ExecuteNonQuery();

// Удаление сеансов

MySqlCommand deleteSessions = new MySqlCommand(

"DELETE FROM Sessions WHERE HallID = @hallId", con);

deleteSessions.Parameters.AddWithValue("@hallId", HallID);

deleteSessions.ExecuteNonQuery();

// Удаление мест

MySqlCommand deleteSeats = new MySqlCommand(

"DELETE FROM Seats WHERE HallID = @hallId", con);

deleteSeats.Parameters.AddWithValue("@hallId", HallID);

deleteSeats.ExecuteNonQuery();

// Удаление самого зала

MySqlCommand deleteHall = new MySqlCommand(

"DELETE FROM Halls WHERE HallID = @hallId", con);

deleteHall.Parameters.AddWithValue("@hallId", HallID);

deleteHall.ExecuteNonQuery();

}

}

// Получает все залы из БД

public static List<Hall> GetAll(string connectionString)

{

List<Hall> halls = new List<Hall>();

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand(

"SELECT HallID, Name, SeatsCount FROM Halls", con);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

halls.Add(new Hall(

reader.GetInt32("HallID"),

reader.GetString("Name"),

reader.GetInt32("SeatsCount")

));

}

}

}

return halls;

}

// Получает зал по ID

public static Hall GetById(int id, string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand(

"SELECT HallID, Name, SeatsCount FROM Halls WHERE HallID = @id", con);

cmd.Parameters.AddWithValue("@id", id);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Hall(

reader.GetInt32("HallID"),

reader.GetString("Name"),

reader.GetInt32("SeatsCount")

);

}

else

{

return null;

}

}

}

}

}

}

**В.9 Класс Movie.cs**

using System;

using System.Collections.Generic;

using System.Data;

using MySql.Data.MySqlClient;

namespace CinemaApp.Models

{

public class Movie

{

public int MovieID { get; set; }

public string Title { get; set; }

public int Duration { get; set; }

public string Description { get; set; }

// Конструктор для новых фильмов

public Movie(string title, int duration, string description)

{

Title = title;

Duration = duration;

Description = description;

}

// Конструктор для существующих фильмов

public Movie(int movieId, string title, int duration, string description)

{

MovieID = movieId;

Title = title;

Duration = duration;

Description = description;

}

// Сохраняет фильм в БД (INSERT или UPDATE)

public void Save(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (MovieID == 0) // Новый фильм

{

cmd = new MySqlCommand(

"INSERT INTO Movies (Title, Duration, Description) VALUES (@title, @duration, @description)",

con);

}

else // Обновляем существующий

{

cmd = new MySqlCommand(

"UPDATE Movies SET Title = @title, Duration = @duration, Description = @description WHERE MovieID = @id",

con);

cmd.Parameters.AddWithValue("@id", MovieID);

}

cmd.Parameters.AddWithValue("@title", Title);

cmd.Parameters.AddWithValue("@duration", Duration);

cmd.Parameters.AddWithValue("@description", Description ?? "");

cmd.ExecuteNonQuery();

// Если это новый фильм, получаем его ID

if (MovieID == 0)

{

MovieID = (int)cmd.LastInsertedId;

}

}

}

// Удаляет фильм из БД

public void Delete(string connectionString)

{

if (MovieID == 0) throw new InvalidOperationException("Невозможно удалить несохранённый фильм.");

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand("DELETE FROM Movies WHERE MovieID = @id", con);

cmd.Parameters.AddWithValue("@id", MovieID);

cmd.ExecuteNonQuery();

}

}

// Получает все фильмы из БД

public static List<Movie> GetAll(string connectionString)

{

List<Movie> movies = new List<Movie>();

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand("SELECT MovieID, Title, Duration, Description FROM Movies", con);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

movies.Add(new Movie(

reader.GetInt32("MovieID"),

reader.GetString("Title"),

reader.GetInt32("Duration"),

reader.IsDBNull(reader.GetOrdinal("Description")) ? null : reader.GetString("Description")

));

}

}

}

return movies;

}

// Получает фильм по ID

public static Movie GetById(int id, string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand("SELECT MovieID, Title, Duration, Description FROM Movies WHERE MovieID = @id", con);

cmd.Parameters.AddWithValue("@id", id);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Movie(

reader.GetInt32("MovieID"),

reader.GetString("Title"),

reader.GetInt32("Duration"),

reader.IsDBNull(reader.GetOrdinal("Description")) ? null : reader.GetString("Description")

);

}

else

{

return null;

}

}

}

}

}

}

**В.10 Класс Seat.cs**

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

namespace CinemaApp.Models

{

public class Seat

{

public int SeatID { get; set; }

public int HallID { get; set; }

public int SeatNumber { get; set; }

// Конструктор для нового места

public Seat(int hallId, int seatNumber)

{

HallID = hallId;

SeatNumber = seatNumber;

}

// Конструктор для существующего места

public Seat(int seatId, int hallId, int seatNumber)

{

SeatID = seatId;

HallID = hallId;

SeatNumber = seatNumber;

}

// Сохраняет место (INSERT или UPDATE)

public void Save(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (SeatID == 0) // Новое место

{

cmd = new MySqlCommand(

"INSERT INTO Seats (HallID, SeatNumber) VALUES (@hallId, @seatNumber)",

con);

}

else // Обновляем существующее

{

cmd = new MySqlCommand(

"UPDATE Seats SET HallID = @hallId, SeatNumber = @seatNumber WHERE SeatID = @id",

con);

cmd.Parameters.AddWithValue("@id", SeatID);

}

cmd.Parameters.AddWithValue("@hallId", HallID);

cmd.Parameters.AddWithValue("@seatNumber", SeatNumber);

cmd.ExecuteNonQuery();

if (SeatID == 0)

{

SeatID = (int)cmd.LastInsertedId;

}

}

}

// Удаляет место из БД

public void Delete(string connectionString)

{

if (SeatID == 0) throw new InvalidOperationException("Невозможно удалить несохранённое место.");

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand("DELETE FROM Seats WHERE SeatID = @id", con);

cmd.Parameters.AddWithValue("@id", SeatID);

cmd.ExecuteNonQuery();

}

}

// Получает все места зала

public static List<Seat> GetAllByHall(int hallId, string connectionString)

{

List<Seat> seats = new List<Seat>();

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT SeatID, HallID, SeatNumber FROM Seats WHERE HallID = @hallId ORDER BY SeatNumber";

MySqlCommand cmd = new MySqlCommand(query, con);

cmd.Parameters.AddWithValue("@hallId", hallId);

using (var reader = cmd.ExecuteReader())

{

while (reader.Read())

{

seats.Add(new Seat(

reader.GetInt32("SeatID"),

reader.GetInt32("HallID"),

reader.GetInt32("SeatNumber")

));

}

}

}

return seats;

}

// Получает место по ID

public static Seat GetById(int id, string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = "SELECT SeatID, HallID, SeatNumber FROM Seats WHERE SeatID = @id";

MySqlCommand cmd = new MySqlCommand(query, con);

cmd.Parameters.AddWithValue("@id", id);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Seat(

reader.GetInt32("SeatID"),

reader.GetInt32("HallID"),

reader.GetInt32("SeatNumber")

);

}

else

{

return null;

}

}

}

}

}

}

**В.11 Класс Session.cs**

using System;

using System.Collections.Generic;

using System.Data;

using MySql.Data.MySqlClient;

namespace CinemaApp.Models

{

public class Session

{

public int SessionID { get; set; }

public int MovieID { get; set; }

public int HallID { get; set; }

public DateTime StartTime { get; set; }

public decimal Price { get; set; }

public string MovieTitle => Movie.GetById(MovieID, connectionString)?.Title ?? "Неизвестен";

public string HallName => Hall.GetById(HallID, connectionString)?.Name ?? "Неизвестен";

private static string connectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

// Конструктор для нового сеанса

public Session(int movieId, int hallId, DateTime startTime, decimal price)

{

MovieID = movieId;

HallID = hallId;

StartTime = startTime;

Price = price;

}

// Конструктор для существующего сеанса

public Session(int sessionId, int movieId, int hallId, DateTime startTime, decimal price)

{

SessionID = sessionId;

MovieID = movieId;

HallID = hallId;

StartTime = startTime;

Price = price;

}

// Сохраняет сеанс (INSERT или UPDATE)

public void Save(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (SessionID == 0) // Новый сеанс

{

cmd = new MySqlCommand(

"INSERT INTO Sessions (MovieID, HallID, StartTime, Price) VALUES (@movieId, @hallId, @startTime, @price)",

con);

}

else // Обновляем существующий

{

cmd = new MySqlCommand(

"UPDATE Sessions SET MovieID = @movieId, HallID = @hallId, StartTime = @startTime, Price = @price WHERE SessionID = @id",

con);

cmd.Parameters.AddWithValue("@id", SessionID);

}

cmd.Parameters.AddWithValue("@movieId", MovieID);

cmd.Parameters.AddWithValue("@hallId", HallID);

cmd.Parameters.AddWithValue("@startTime", StartTime);

cmd.Parameters.AddWithValue("@price", Price);

cmd.ExecuteNonQuery();

// Если это новый сеанс, получаем его ID

if (SessionID == 0)

{

SessionID = Convert.ToInt32(cmd.LastInsertedId);

}

}

}

// Удаляет сеанс из БД

public void Delete(string connectionString)

{

if (SessionID == 0) throw new InvalidOperationException("Невозможно удалить несохранённый сеанс.");

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

// Удаление связанных билетов

MySqlCommand deleteTickets = new MySqlCommand(

"DELETE FROM Tickets WHERE SessionID = @sessionId", con);

deleteTickets.Parameters.AddWithValue("@sessionId", SessionID);

deleteTickets.ExecuteNonQuery();

// Удаление самого сеанса

MySqlCommand deleteSession = new MySqlCommand(

"DELETE FROM Sessions WHERE SessionID = @id", con);

deleteSession.Parameters.AddWithValue("@id", SessionID);

deleteSession.ExecuteNonQuery();

}

}

// Получает все сеансы из БД

public static List<Session> GetAll(string connectionString)

{

List<Session> sessions = new List<Session>();

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = @"

SELECT SessionID, MovieID, HallID, StartTime, Price

FROM Sessions";

MySqlCommand cmd = new MySqlCommand(query, con);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

sessions.Add(new Session(

reader.GetInt32("SessionID"),

reader.GetInt32("MovieID"),

reader.GetInt32("HallID"),

reader.GetDateTime("StartTime"),

reader.GetDecimal("Price")

));

}

}

}

return sessions;

}

// Получает сеанс по ID

public static Session GetById(int id, string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = @"

SELECT SessionID, MovieID, HallID, StartTime, Price

FROM Sessions WHERE SessionID = @id";

MySqlCommand cmd = new MySqlCommand(query, con);

cmd.Parameters.AddWithValue("@id", id);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Session(

reader.GetInt32("SessionID"),

reader.GetInt32("MovieID"),

reader.GetInt32("HallID"),

reader.GetDateTime("StartTime"),

reader.GetDecimal("Price")

);

}

else

{

return null;

}

}

}

}

// Создаёт билеты для всех мест в зале для данного сеанса

public void CreateTicketsForSession(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

// Получаем места этого зала

string seatsQuery = "SELECT SeatID FROM Seats WHERE HallID = @hallId";

MySqlCommand seatsCmd = new MySqlCommand(seatsQuery, con);

seatsCmd.Parameters.AddWithValue("@hallId", this.HallID);

using (MySqlDataReader reader = seatsCmd.ExecuteReader())

{

while (reader.Read())

{

int seatId = reader.GetInt32("SeatID");

// Проверяем, есть ли уже билет на это место и сеанс

string checkTicket = "SELECT COUNT(\*) FROM Tickets WHERE SessionID = @session AND SeatID = @seat";

MySqlCommand checkCmd = new MySqlCommand(checkTicket, con);

checkCmd.Parameters.AddWithValue("@session", this.SessionID);

checkCmd.Parameters.AddWithValue("@seat", seatId);

int count = Convert.ToInt32(checkCmd.ExecuteScalar());

if (count == 0)

{

// Добавляем билет

string insertTicket = "INSERT INTO Tickets (SessionID, SeatID, Status) VALUES (@session, @seat, 'Свободен')";

MySqlCommand insertCmd = new MySqlCommand(insertTicket, con);

insertCmd.Parameters.AddWithValue("@session", this.SessionID);

insertCmd.Parameters.AddWithValue("@seat", seatId);

insertCmd.ExecuteNonQuery();

}

}

}

}

}

}

}

**В.12 Класс Ticket.cs**

using System;

using System.Collections.Generic;

using System.Data;

using MySql.Data.MySqlClient;

namespace CinemaApp.Models

{

public class Ticket

{

public int TicketID { get; set; }

public int SessionID { get; set; }

public int SeatID { get; set; }

public string Status { get; set; }

// Конструктор для нового билета

public Ticket(int sessionId, int seatId, string status)

{

SessionID = sessionId;

SeatID = seatId;

Status = status;

}

// Конструктор для существующего билета

public Ticket(int ticketId, int sessionId, int seatId, string status)

{

TicketID = ticketId;

SessionID = sessionId;

SeatID = seatId;

Status = status;

}

// Сохраняет билет (INSERT или UPDATE)

public void Save(string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd;

if (TicketID == 0) // Новый билет

{

cmd = new MySqlCommand(

"INSERT INTO Tickets (SessionID, SeatID, Status) VALUES (@session, @seat, @status)",

con);

}

else // Обновляем существующий

{

cmd = new MySqlCommand(

"UPDATE Tickets SET SessionID = @session, SeatID = @seat, Status = @status WHERE TicketID = @id",

con);

cmd.Parameters.AddWithValue("@id", TicketID);

}

cmd.Parameters.AddWithValue("@session", SessionID);

cmd.Parameters.AddWithValue("@seat", SeatID);

cmd.Parameters.AddWithValue("@status", Status);

cmd.ExecuteNonQuery();

// Если это новый билет, получаем его ID

if (TicketID == 0)

{

TicketID = Convert.ToInt32(cmd.LastInsertedId);

}

}

}

// Метод изменения статуса

public void ChangeStatus(string newStatus, string connectionString)

{

Status = newStatus;

Save(connectionString);

}

// Удаляет билет из БД

public void Delete(string connectionString)

{

if (TicketID == 0) throw new InvalidOperationException("Невозможно удалить несохранённый билет.");

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

MySqlCommand cmd = new MySqlCommand("DELETE FROM Tickets WHERE TicketID = @id", con);

cmd.Parameters.AddWithValue("@id", TicketID);

cmd.ExecuteNonQuery();

}

}

// Получает все билеты для указанного сеанса

public static List<Ticket> GetAllBySession(int sessionId, string connectionString)

{

List<Ticket> tickets = new List<Ticket>();

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = @"

SELECT TicketID, SessionID, SeatID, Status

FROM Tickets WHERE SessionID = @sessionId ORDER BY SeatID";

MySqlCommand cmd = new MySqlCommand(query, con);

cmd.Parameters.AddWithValue("@sessionId", sessionId);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

tickets.Add(new Ticket(

reader.GetInt32("TicketID"),

reader.GetInt32("SessionID"),

reader.GetInt32("SeatID"),

reader.GetString("Status")

));

}

}

}

return tickets;

}

// Получает билет по ID

public static Ticket GetById(int id, string connectionString)

{

using (MySqlConnection con = new MySqlConnection(connectionString))

{

con.Open();

string query = @"

SELECT TicketID, SessionID, SeatID, Status

FROM Tickets WHERE TicketID = @id";

MySqlCommand cmd = new MySqlCommand(query, con);

cmd.Parameters.AddWithValue("@id", id);

using (MySqlDataReader reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new Ticket(

reader.GetInt32("TicketID"),

reader.GetInt32("SessionID"),

reader.GetInt32("SeatID"),

reader.GetString("Status")

);

}

else

{

return null;

}

}

}

}

}

}

ПРИЛОЖЕНИЕ Г

using CinemaApp.Models;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using MySql.Data.MySqlClient;

using System;

namespace CinemaApp.Tests

{

[TestClass]

public class CinemaTests

{

private const string ConnectionString = "server=giai5.h.filess.io;port=61002;database=CinemaDB\_specificam;uid=CinemaDB\_specificam;pwd=5db19cb7940d7e78326db5f1ebd75686978758f8";

[TestMethod]

public void Test\_Movie\_CRUD()

{

var movie = new Movie("Тестовый фильм", 120, "Описание фильма");

movie.Save(ConnectionString);

Assert.IsTrue(movie.MovieID > 0);

var loaded = Movie.GetById(movie.MovieID, ConnectionString);

Assert.IsNotNull(loaded);

Assert.AreEqual("Тестовый фильм", loaded.Title);

loaded.Description = "Обновлённое описание";

loaded.Save(ConnectionString);

var updated = Movie.GetById(movie.MovieID, ConnectionString);

Assert.AreEqual("Обновлённое описание", updated.Description);

updated.Delete(ConnectionString);

var deleted = Movie.GetById(movie.MovieID, ConnectionString);

Assert.IsNull(deleted);

}

[TestMethod]

public void Test\_Hall\_CRUD()

{

var hall = new Hall("Тестовый зал", 50);

hall.Save(ConnectionString);

Assert.IsTrue(hall.HallID > 0);

var loaded = Hall.GetById(hall.HallID, ConnectionString);

Assert.IsNotNull(loaded);

Assert.AreEqual("Тестовый зал", loaded.Name);

loaded.Name = "Новый зал";

loaded.SeatsCount = 60;

loaded.Save(ConnectionString);

var updated = Hall.GetById(hall.HallID, ConnectionString);

Assert.AreEqual("Новый зал", updated.Name);

Assert.AreEqual(60, updated.SeatsCount);

updated.Delete(ConnectionString);

var deleted = Hall.GetById(hall.HallID, ConnectionString);

Assert.IsNull(deleted);

}

[TestMethod]

public void Test\_Session\_CRUD()

{

var movie = new Movie("Фильм сессии", 120, null);

movie.Save(ConnectionString);

var hall = new Hall("Зал сессии", 50);

hall.Save(ConnectionString);

var session = new Session(movie.MovieID, hall.HallID, DateTime.Now.AddHours(1), 250.0m);

session.Save(ConnectionString);

Assert.IsTrue(session.SessionID > 0);

var loaded = Session.GetById(session.SessionID, ConnectionString);

Assert.IsNotNull(loaded);

Assert.AreEqual(movie.MovieID, loaded.MovieID);

loaded.Price = 300.0m;

loaded.Save(ConnectionString);

var updated = Session.GetById(session.SessionID, ConnectionString);

Assert.AreEqual(300.0m, updated.Price);

updated.Delete(ConnectionString);

var deleted = Session.GetById(session.SessionID, ConnectionString);

Assert.IsNull(deleted);

movie.Delete(ConnectionString);

hall.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Ticket\_SaveAndGet()

{

var movie = new Movie("Фильм билета", 120, null);

movie.Save(ConnectionString);

var hall = new Hall("Зал билета", 50);

hall.Save(ConnectionString);

var session = new Session(movie.MovieID, hall.HallID, DateTime.Now.AddHours(1), 250.0m);

session.Save(ConnectionString);

var seat = new Seat(hall.HallID, 1);

seat.Save(ConnectionString);

var ticket = new Ticket(session.SessionID, seat.SeatID, "Свободен");

ticket.Save(ConnectionString);

Assert.IsTrue(ticket.TicketID > 0);

var loaded = Ticket.GetById(ticket.TicketID, ConnectionString);

Assert.IsNotNull(loaded);

Assert.AreEqual("Свободен", loaded.Status);

loaded.ChangeStatus("Продан", ConnectionString);

var updated = Ticket.GetById(ticket.TicketID, ConnectionString);

Assert.AreEqual("Продан", updated.Status);

loaded.Delete(ConnectionString);

var deleted = Ticket.GetById(ticket.TicketID, ConnectionString);

Assert.IsNull(deleted);

session.Delete(ConnectionString);

movie.Delete(ConnectionString);

hall.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Employee\_CRUD()

{

var employee = new Employee("Тестовый сотрудник", "login\_test", "pass123", "Оператор");

employee.Save(ConnectionString);

Assert.IsTrue(employee.EmployeeID > 0);

var loaded = Employee.GetById(employee.EmployeeID, ConnectionString);

Assert.IsNotNull(loaded);

Assert.AreEqual("Тестовый сотрудник", loaded.FullName);

loaded.ChangePassword("new\_pass", ConnectionString);

var updated = Employee.GetById(employee.EmployeeID, ConnectionString);

Assert.AreEqual("new\_pass", updated.Password);

updated.Delete(ConnectionString);

var deleted = Employee.GetById(employee.EmployeeID, ConnectionString);

Assert.IsNull(deleted);

}

[TestMethod]

public void Test\_Session\_GetAll\_Returns\_List()

{

var movie = new Movie("Фильм 1", 120, null);

movie.Save(ConnectionString);

var hall = new Hall("Зал 1", 30);

hall.Save(ConnectionString);

var session1 = new Session(movie.MovieID, hall.HallID, DateTime.Now.AddMinutes(10), 200.0m);

session1.Save(ConnectionString);

var session2 = new Session(movie.MovieID, hall.HallID, DateTime.Now.AddMinutes(20), 250.0m);

session2.Save(ConnectionString);

var sessions = Session.GetAll(ConnectionString);

Assert.IsTrue(sessions.Count >= 2);

session1.Delete(ConnectionString);

session2.Delete(ConnectionString);

movie.Delete(ConnectionString);

hall.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Seat\_GetAllByHall\_Returns\_List()

{

var hall = new Hall("Зал мест", 10);

hall.Save(ConnectionString);

for (int i = 1; i <= hall.SeatsCount; i++)

{

var seat = new Seat(hall.HallID, i);

seat.Save(ConnectionString);

}

var seats = Seat.GetAllByHall(hall.HallID, ConnectionString);

Assert.AreEqual(hall.SeatsCount, seats.Count);

foreach (var seat in seats)

{

seat.Delete(ConnectionString);

}

hall.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Ticket\_Status\_Update\_Working()

{

var movie = new Movie("Фильм билета", 120, null);

movie.Save(ConnectionString);

var hall = new Hall("Зал билета", 50);

hall.Save(ConnectionString);

var session = new Session(movie.MovieID, hall.HallID, DateTime.Now, 250.0m);

session.Save(ConnectionString);

var seat = new Seat(hall.HallID, 1);

seat.Save(ConnectionString);

var ticket = new Ticket(session.SessionID, seat.SeatID, "Свободен");

ticket.Save(ConnectionString);

ticket.ChangeStatus("Продан", ConnectionString);

var updated = Ticket.GetById(ticket.TicketID, ConnectionString);

Assert.AreEqual("Продан", updated.Status);

updated.Delete(ConnectionString);

session.Delete(ConnectionString);

movie.Delete(ConnectionString);

hall.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Employee\_Role\_Change()

{

var emp = new Employee("Тестовый сотрудник", "testuser", "password", "Оператор");

emp.Save(ConnectionString);

var loaded = Employee.GetById(emp.EmployeeID, ConnectionString);

Assert.AreEqual("Оператор", loaded.Role);

loaded.Role = "Администратор";

loaded.Save(ConnectionString);

var updated = Employee.GetById(emp.EmployeeID, ConnectionString);

Assert.AreEqual("Администратор", updated.Role);

updated.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Hall\_SeatsCount\_Updates\_Correctly()

{

var hall = new Hall("Тестовый зал", 20);

hall.Save(ConnectionString);

hall.SeatsCount = 30;

hall.Save(ConnectionString);

var updated = Hall.GetById(hall.HallID, ConnectionString);

Assert.AreEqual(30, updated.SeatsCount);

updated.Delete(ConnectionString);

}

[TestMethod]

public void Test\_Session\_Delete\_Removes\_Tickets()

{

var movie = new Movie("Фильм для удаления сеанса", 120, null);

movie.Save(ConnectionString);

var hall = new Hall("Зал для удаления сеанса", 50);

hall.Save(ConnectionString);

var session = new Session(movie.MovieID, hall.HallID, DateTime.Now, 250.0m);

session.Save(ConnectionString);

var seat = new Seat(hall.HallID, 1);

seat.Save(ConnectionString);

var ticket = new Ticket(session.SessionID, seat.SeatID, "Свободен");

ticket.Save(ConnectionString);

session.Delete(ConnectionString);

var foundTicket = Ticket.GetById(ticket.TicketID, ConnectionString);

Assert.IsNull(foundTicket);

movie.Delete(ConnectionString);

hall.Delete(ConnectionString);

}

}

}