СОДЕРЖАНИЕ

[Введение 4](#_Toc105405616)

[1 Анализ предметной области 5](#_Toc105405617)

[2 Постановка задачи 7](#_Toc105405618)

[3 Проектирование программного модуля 8](#_Toc105405619)

[3.1 Классификаторы 8](#_Toc105405620)

[3.2 Инфологическая модель 9](#_Toc105405621)

[3.3 Описание входных и выходных данных 10](#_Toc105405622)

[3.4 Алгоритм решения задачи 11](#_Toc105405623)

[4 Технологии и инструменты разработки 12](#_Toc105405624)

[4.1 Технологии для разрабоки 12](#_Toc105405625)

[4.2 Инструменты разработки 1](#_Toc105405626)4

[5 Разработка программного модуля 1](#_Toc105405627)6

[5.1 Пользовательский интерфейс программного модуля 1](#_Toc105405628)6

[5.2 Описание методов разработки](#_Toc105405629) 18

[Заключение](#_Toc105405630) 20

[Приложение А 22](#_Toc105405631)

[Приложение Б 23](#_Toc105405632)

# Введение

В данной курсовой работе рассматривается проблема отсутствия свободных мест на нужный сеанс во время прихода в кинотеатр. Следствием чего является впустую потраченное время и испорченное настроение. Целью данной курсовой работы является решение данной проблемы путем реализации программного продукта на одном из языков программирования. Задача проектирования состоит в том, чтобы максимально просто добиться результата поставленной задачи. Было предложено реализовать программный продукт, который позволял бы пользователям покупать билеты онлайн – “Система покупки билетов онлайн”.

Для решения данной проблемы были выдвинуты следующие цели:

* пользователи (посетители кинотеатра) могли приобретать билеты онлайн, не выходя из дому;
* пользователи могли просматривать расписание сеансов на неделю вперёд, в том числе занятость зала на интересующий их сеанс;
* администратор (работник кинотеатра) мог бы добавлять или удалять фильмы, сеансы; изменять количество мест в зале, цены билетам; а также продавать билеты на сеансы.

# 1 Анализ предметной области

Деятельность кинотеатра направлена на удовлетворение развлекательной и культурно-просветительной потребности населения. На основе миссии определяются цели деятельности. Следовательно, целью деятельности данного кинотеатра является максимально большее привлечение кинозрителей к просмотру максимально большего количества фильмов разнообразных жанров и тематик, для развлечения кинозрителей и оказания ни них культурно-научно-просветительского воздействия.

Рассмотрим достоинства и недостатки существующей организационной структуры компании. Управление компанией носит линейный характер, основанный на единоначалии. К основным недостаткам такого метода управления можно отнести следующие:

1. Высокие требования к руководителю, который должен иметь багаж всесторонних знаний, необходимых для управления компанией. Остановимся на этой проблеме управления компанией более подробно. Управление компанией на уровне генерального директора предполагает решение вопросов по позиционированию на рынке реализуемого продукта или услуги, уменьшение затрат на закупку и реализацию, подбор нужных кадров, т.е. объем знаний и умений, в котором сложно быть компетентным одному человеку.
2. Линейная структура лишена необходимых промежуточных звеньев по планированию и подготовке решений, что приводит к отсутствию видения ближайшей перспективы развития компании и негативно отражается на ее будущей конкурентоспособности.
3. Средние уровни управления компанией перегружены лишней организационной информацией, что мешает им решать на должном уровне конкретные необходимые для развития компании задачи.
4. Излишняя перегруженность средних уровней ослабляет связи между ними, что влияет на качество принимаемых на этих уровнях управленческих решений.
5. Так как общее управление компанией организовано по линейному принципу, при отсутствии одного из ключевых руководителей нарушается работа как одного из звеньев компании, так и всей компании целиком. Это способствует дополнительным рискам в работе компании и делает ее менее конкурентоспособной. Ещё одним недостатком в управлении компанией является программная автономия ее отдельных подразделений, занимающихся реализацией билетов.

Для устранения описанных недостатков управления компанией предлагается сделать ее более интегрированной и постепенно перевести механизм управления с линейного на функциональный, используя для этого средства дополнительной автоматизации процесса продажи билетов.

# 2 Постановка задачи

Администрирование сети кинотеатров заключается в добавлении и редактировании и удалении информации о кинотеатрах, фильмах, которые демонстрируются в кинотеатрах. Отдельными процессами являются создание сеанса и заказ билетов

Для каждого кинотеатра существует возможность создания залов с заданным числом рядов и мест в каждом зале. Каждый зал может быть использован для показа любого фильма.

Для предоставления возможности заказа билетов сотруднику необходимо создать сеанс, назначив кинотеатр, зал и время показа, а также фильм, который будет демонстрироваться. После этого сотрудники и зарегистрированные интернет-пользователи получают возможность заказа билетов.

Необходимо предусмотреть невозможность заказа билетов на уже занятые места, а также сохранение информации об интернет-заказчиках для последующей реализации забронированных билетов.

Также необходимо обеспечить невозможность заказа билетов через сеть интернет незарегистрированными пользователям.

В результате анализа постановки задачи, будет спроектирована база данных. Спроектированная база данных будет отражает сущности предметной области, а также будет содержать вспомогательные таблицы для работы приложения.

# 3 Проектирование программного модуля

## 3.1 Классификаторы

Классификатор представляет собой набор таблиц (справочников).

Объект – некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением в данной предметной области.

Атрибут – элемент, описывающий любую характеристику объекта, важную для данной предметной области. Объектом в данной базе данных будет: информация о фильмах приведенная в таблице 3.1, залах показанных на таблице 3.2, саенсах на таблице 3.3 и билетах на таблице 3.4.

Таблица 3.1 – Атрибуты фильма

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Фильм | Идентификационный код  Название фильма |

Таблица 3.2 – Атрибуты зала

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Зал | Идентификационный код  Выбор зала  Ряд  Место |

Таблица 3.3 – Атрибуты сеанса

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Сеанс | Идентификационный код  Идентификационный код фильма  Цена  Время |

Таблица 3.4 – Атрибуты билета

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Билет | Идентификационный код  Идентификационный код зала  Идентификационный код сеанса |

## 3.2 Инфологическая модель

Инфологическая модель – это набор сущностей и связей между ними. Она содержит информацию о сущностях системы и способах их взаимодействия, включает идентификацию объектов, важных для предметной области (сущностей), свойств этих объектов (атрибутов) и их отношений с другими объектами (связей).

Для выполнения проекта была выбрана СУБД SQL Server. Схема БД представлена на рисунке 3.1.

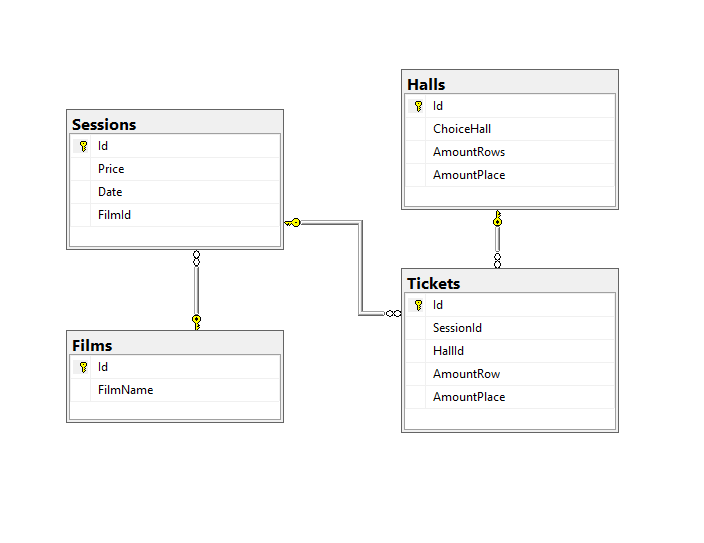


Рисунок 3.1 – Связи таблиц в БД

Структура таблиц фильмы, залы, сеансы и билеты базы данных представлена на рисунках 3.2 – 3.5 соответственно.

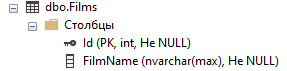


Рисунок 3.2 – Таблица «Фильмы»

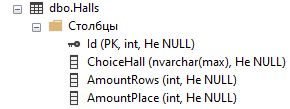


Рисунок 3.2 – Таблица «Залы»

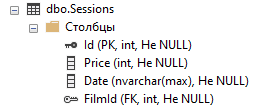


Рисунок 3.2 – Таблица «Сеансы»

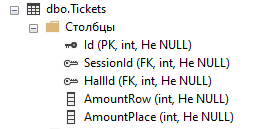


Рисунок 3.2 – Таблица «Билеты»

## 3.3 Описание входных и выходных данных

Входной является информация, которую пользователь вносит в файл базы данных, заполняя необходимые поля выбранной таблицы, а также вводя информацию в базу данных с помощью запросов SQL.

Выходная информация - результат выполнения запросов, фильтрации данных, вывод необходимой информации в отчет, печать информации. Информация, которая несет вывод либо обобщает указанные данные в общем виде или по определенному критерию.

## 3.4 Алгоритм решения задачи

Для проведения каких-либо действий с объектами необходимо выбрать действие: удалить, редактировать или добавить.

Последовательное описание блок – схемы приложения приведено ниже:

Шаг 1. Запуск программы.

Шаг 2. Вопрос: Есть ли у вас аккаунт? Если нет, то переходим к шагу 3. Если есть то переходим к шагу 4.

Шаг 3. Регистрируемся. Возвращаемя к авторизации.

Шаг 4. Вход в приложение под своей учетной записью.

Шаг 5. Вопрос: Выбрано ли действие? Если да, то переходим к шагу 6, если нет, то переходим к 8 шагу.

Шаг 6. Выполнение

Шаг 7. Возвращаемся к 4 шагу.

Шаг 8. Закрытие приложения.

Блок-схема приведена в приложении А.

Листинг программы приведен в приложении Б.

# 4 Технологии и инструменты разработки

## 4.1 Технологии для разрабоки

Для разработки данного приложения был выбран язык C# - современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. Здесь представлен обзор основных компонентов языка C# 8 и более ранних версий.

Windows Presentation Foundation (WPF) — это платформа пользовательского интерфейса для создания клиентских приложений для настольных систем. Платформа разработки WPF поддерживает широкий набор компонентов для разработки приложений, включая модель приложения, ресурсы, элементы управления, графику, макет, привязки данных, документы и безопасность.

WPF является частью платформы .NET, и если ранее вы создавали приложения в .NET с помощью ASP.NET или Windows Forms, то общий процесс программирования вам уже знаком. WPF использует расширяемый язык разметки для приложений (XAML), чтобы предоставить декларативную модель для программирования приложений.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL является общепринятым интерфейсом к базам данных. «Все промышленные базы — Oracle, Microsoft SQL Server, PostgreSQL, MySQL — работают на SQL.

SQL Server характеризуется такими особенностями как:

* Производительность. SQL Server работает очень быстро.
* Надежность и безопасность. SQL Server предоставляет шифрование данных.
* Простота. С данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

## 4.2 Инструменты разработки

В качестве среды для разработки приложения была выбрана программа Microsoft Visual Studio, линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight. Существует множество сред разработки, но данная была выбрана в ввиду ее удобного графического интерфейса и средств отладки. Microsoft Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных.

Главные возможности среды разработки Microsoft Visual Studio:

* включены все «интеллектуальные» возможности по редактированию кода;
* есть возможность визуального просмотра будущего приложения;
* сборка проекта работает быстро;
* удобный конструктор интерфейсов;
* удобное и интуитивно понятное логирование проекта;
* указанные цвета и рисунки, использованные в layout'e отображаются на границе в виде небольших превью, которые легко помогают понять какой конкретно ресурс используется;
* среда разработки является технологиями компании Microsoft;
* при выборе ресурса, его содержимое отображается во всплывающих окнах;
* возможность создания одним кликом новых окон и страниц;
* мониторинг используемой памяти;
* возможность подключать сторонние дополнения(плагины);
* возможность добавления новых инструментов.

Рекомендуемые системные требования к компьютеру на которой будет установлена Visual Studio следующие:

* Операционная система – Microsoft® Windows® 7/8/10 (32- или 64-bit);
* 2 ГБ ОЗУ. рекомендуется 8 ГБ ОЗУ (минимум 2,5 ГБ при выполнении на виртуальной машине) ;
* Место на жестком диске: до 210 ГБ (минимум 800 МБ) свободного места в зависимости от установленных компонентов; обычно для установки требуется от 20 до 50 ГБ свободного места;
* Видеоадаптер с минимальным разрешением 720p (1280 на 720 пикселей); для оптимальной работы Visual Studio рекомендуется разрешение WXGA (1366 на 768 пикселей) или более высокое.

# 5 Разработка программного модуля

## 5.1 Пользовательский интерфейс программного модуля

В окне входа в приложение, представленном на рисунке 5.1, находятся два текстовых поля логин и пароль, и кнопка входа в приложение. Для дальнейшей работы нужно верно указать логин и пароль. Если логин и пароль верны, то откроется основное окно, представленное на рисунке 5.2 с возможностью выбора действий.

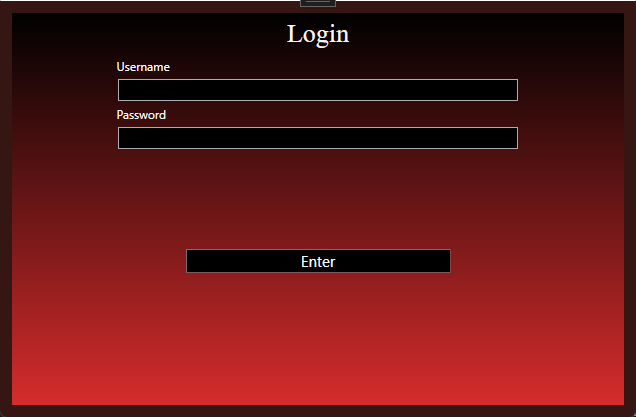


Рисунок 5.1 – Окно входа

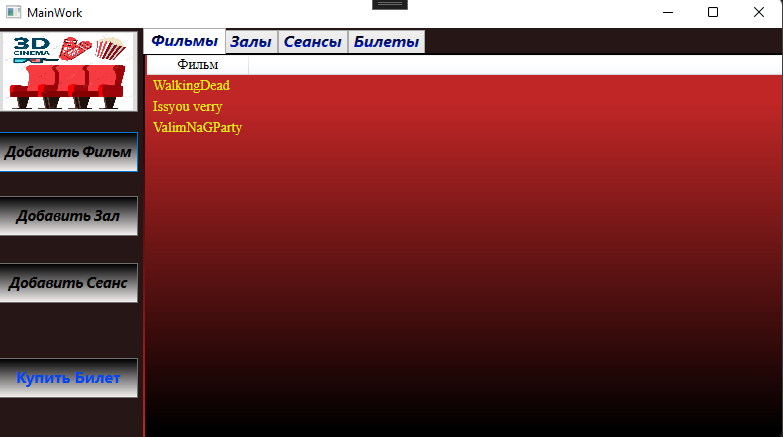


Рисунок 5.2 – Основное окно

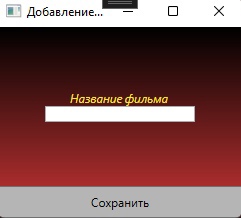


Рисунок 5.3 – Добавление фильма

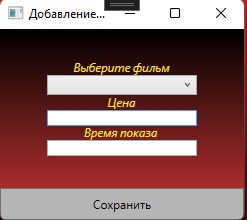


Рисунок 5.4 – Добавление сеанса

Для успешной работы с программой необходимо:

1. ОС Windows.
2. Соеденение с БД.
3. Учетная запись.

## 5.2 Описание методов разработки (MVVM, Command, EF Core)

**MVVM (Model-View-ViewModel)** - это шаблон, который появился для обхода ограничений паттернов MVC и MVP, и объединяющий некоторые из их сильных сторон. Эта модель впервые появилась в составе фреймворка Small Talk в 80-х, и была позднее улучшена с учетом обновленной модели презентаций (MVP).

Шаблон MVVM имеет три основных компонента: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса XAML, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

Команды применяются в различных целях. Первой задачей является отделение семантики и объекта, вызывающего команду, от логики, которая выполняет команду. Это позволяет нескольким и разнородным источникам вызывать одну логику команды, а также настраивать логику команды для различных целевых объектов. Например, операции редактирования **Копировать**, **Вырезать** и **Вставить**, которые встречаются во многих приложениях, могут вызываться с помощью разных действий пользователя, если они реализованы с помощью команд. Приложение может поддерживать вырезание выделенных объектов или текста путем нажатия определенной кнопки, выбора пункта меню или сочетания клавиш, например CTRL+X. С помощью команд можно привязать разные типы действий пользователя к одной логике.

Команды являются привязанными объектами, что позволяет разделить логику и пользовательский интерфейс друг от друга.

Если рассматривать команды более подробно, то они представляют из себя следующее:

* Команды представляют собой объекты, реализующие интерфейс ICommand
* Обычно команды связанны с какой либо функцией
* Элементы пользовательского интерфейса привязываются к командам кода интерфейс активируется пользователем, то выполняется команда вызывается соответствующая функция
* Команды знают, включены ли они или нет
* Функции могут отключать команды – автоматическое отключение всех пользовательских элементов ассоциированных с ней.

Entity Framework Core - представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер.

# Заключение

В процессе работы над программным продуктом автором были выявлены и решены проблемы доступа из основного интерфейса к дополнительным меню и формам программы путем использования оригинального дизайна интерфейса и его эргономичного решения.

Также была разработана и реализована необходимая система доступа и перехода к основным и дополнительным блокам программного продукта, обеспечивающего точное описание необходимых в работе подразделения ИТ серверных и сетевых механизмов доступа каждого рабочего места сотрудника.

Автором в процессе работы над программным продуктом был произведен необходимый научный и технический анализ для проектирования и реализации программного продукта «Резервирования билетов в кинотеатре».

Цель курсового проекта достигнута. Задачи выполнены.

Список использованных источников

Список использованной литературы

1. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер. – Санкт-Петербург : Издательство Питер, 2019. – 896 с.
2. Фримен, Э. Head First. Паттерны проектирования. / Э. Фримен, Э. Робсон, К. Сьерра. – Санкт-Петербург : Питер, 2021. – 640 с.

Интернет ресурсы

1. Документация по Entity Framework. [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/ef/
2. Википедия. Язык программирования C# [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/C\_Sharp
3. Шаблон Model-View-ViewModel. [Электронный ресурс] – https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/enterprise-application-patterns/mvvm
4. SQL Server 2019 [Электронный ресурс] – https://www.microsoft.com/ru-ru/sql-server/sql-server-2019
5. Windows Presentation Foundation (WPF) [Электронный ресурс] – https://docs.microsoft.com/ru-ru/visualstudio/designers/getting-started-with-wpf?view=vs-2022

# Приложение А

(обязательное)

Блок – схема программы

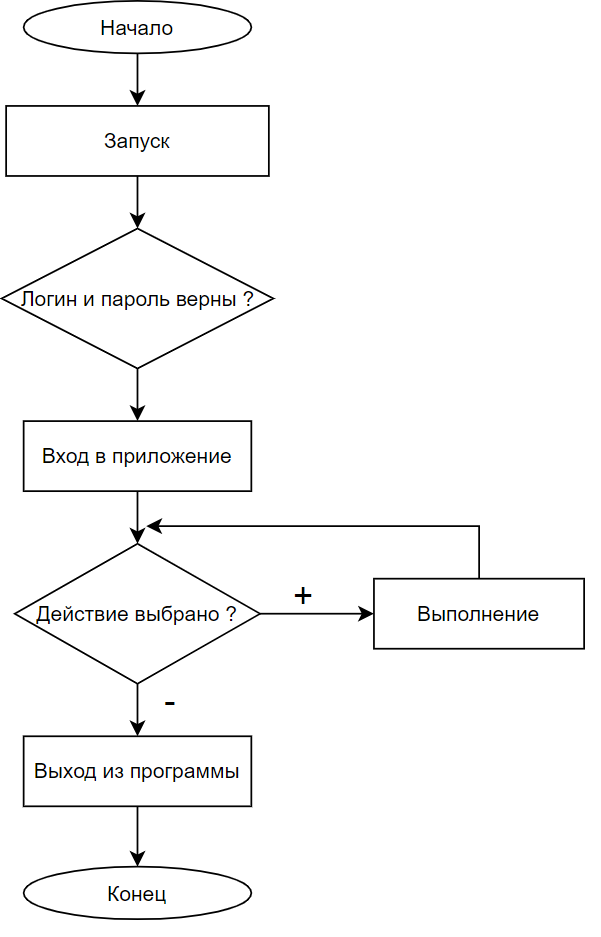


Рисунок А.1–Блок схема программы

# Приложение Б

(обязательное)

Листинг программы

using System.Collections.Generic;

using System.Linq;

using WpfAppCinema.Resources.MVVM.Models.Data;

using WpfAppCinema.Resources.MVVM.Command;

namespace WpfAppCinema.Resources.MVVM.Models

{

public class DataWorker

{

#region[Вся таблица]

public static List<Film> GetAllFilms()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Films.ToList();

return result;

}

}

public static List<Hall> GetAllHalls()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Halls.ToList();

return result;

}

}

public static List<Session> GetAllSessions()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Sessions.ToList();

return result;

}

}

public static List<Ticket> GetAllTickets()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Tickets.ToList();

return result;

}

}

#endregion

#region[Список по Id]

public static List<Session> GetAllSessionByFilmId(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

List<Session> students = (from Session in GetAllSessions() where Session.FilmId == id select Session).ToList();

return students;

}

}

public static List<Hall> GetAllHallsById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

List<Hall> cp = (from Hall in GetAllHalls() where Hall.Id == id select Hall).ToList();

return cp;

}

}

public static List<Session> GetAllSessionsById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

List<Session> cp = (from Session in GetAllSessions() where Session.Id == id select Session).ToList();

return cp;

}

}

public static List<Ticket> GetAllTicketsById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

List<Ticket> cp = (from Ticket in GetAllTickets() where Ticket.Id == id select Ticket).ToList();

return cp;

}

}

#endregion

#region[По Id]

public static Film GetFilmById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

Film pos = db.Films.FirstOrDefault(p => p.Id == id);

return pos;

}

}

public static Hall GetHallById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

Hall pos = db.Halls.FirstOrDefault(p => p.Id == id);

return pos;

}

}

public static Session GetSessionById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

Session pos = db.Sessions.FirstOrDefault(p => p.Id == id);

return pos;

}

}

public static Ticket GetTicketById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

Ticket pos = db.Tickets.FirstOrDefault(p => p.Id == id);

return pos;

}

}

#endregion

#region[Film]

public static string CreateFilm(string filmName)

{

string result = "Уже существует";

using (ApplicationContext db = new ApplicationContext())

{

//проверяем сущесвует ли фильм

bool checkIsExist = db.Films.Any(el => el.FilmName == filmName);

if (!checkIsExist)

{

Film newFilms = new Film

{

FilmName = filmName

};

db.Films.Add(newFilms);

db.SaveChanges();

result = "Фильм добавлен!";

}

return result;

}

}

public static string EditFilm(Film oldFilm, string filmName)

{

string result = "Такого Резервирование не существует";

using (ApplicationContext db = new ApplicationContext())

{

Film film = db.Films.FirstOrDefault(d => d.Id == oldFilm.Id);

film.FilmName = filmName;

db.SaveChanges();

result = "Сделано! filmName " + film.FilmName + " изменено";

}

return result;

}

public static string DeleteFilm(Film films)

{

string result = "Такой Film не существует";

using (ApplicationContext db = new ApplicationContext())

{

db.Films.Remove(films);

db.SaveChanges();

result = "Сделано! " + films.FilmName + " Удалена";

}

return result;

}

#endregion

#region[Hall]

public static string CreateHall(string choiceHall, int amountRows, int amountPlace)

{

string result = "Уже существует";

using (ApplicationContext db = new ApplicationContext())

{

//проверяем сущесвует ли

bool checkIsExist = db.Halls.Any(el => el.ChoiceHall == choiceHall &&

el.AmountRows == amountRows && el.AmountPlace == amountPlace);

if (!checkIsExist)

{

Hall newHalls = new Hall

{

ChoiceHall = choiceHall,

AmountRows = amountRows,

AmountPlace = amountPlace

};

db.Halls.Add(newHalls);

db.SaveChanges();

result = " добавлен!";

}

return result;

}

}

public static string EditHall(Hall oldHall, string choiceHall, int amountRows, int amountPlace)

{

string result = "Такого Резервирование не существует";

using (ApplicationContext db = new ApplicationContext())

{

Hall halls = db.Halls.FirstOrDefault(d => d.Id == oldHall.Id);

halls.ChoiceHall = choiceHall;

halls.AmountRows = amountRows;

halls.AmountPlace = amountPlace;

db.SaveChanges();

result = "Сделано! choiceHall " + halls.ChoiceHall + " изменено";

}

return result;

}

public static string DeleteHall(Hall halls)

{

string result = "Такой Film не существует";

using (ApplicationContext db = new ApplicationContext())

{

db.Halls.Remove(halls);

db.SaveChanges();

result = "Сделано! " + halls.ChoiceHall + " Удалена";

}

return result;

}

#endregion

#region[Session]

public static string CreateSession(Film film, int price, string date)

{

string result = "Уже существует";

using (ApplicationContext db = new ApplicationContext())

{

//проверяем сущесвует ли

bool checkIsExist = db.Sessions.Any(el => el.Price == price &&

el.Date == date && el.FilmId == film.Id);

if (!checkIsExist)

{

Session newSession = new Session

{

FilmId = film.Id,

Price = price,

Date = date

};

db.Sessions.Add(newSession);

db.SaveChanges();

result = " добавлен!";

}

return result;

}

}

public static string EditSession(Session oldSession, Film filmName, int price, string date)

{

string result = "Такого Резервирование не существует";

using (ApplicationContext db = new ApplicationContext())

{

Session session = db.Sessions.FirstOrDefault(d => d.Id == oldSession.Id);

session.FilmId = filmName.Id;

session.Price = price;

session.Date = date;

db.SaveChanges();

result = "Сделано! choiceHall " + session.Price + " изменено";

}

return result;

}

public static string DeleteSession(Session sessions)

{

string result = "Такой Film не существует";

using (ApplicationContext db = new ApplicationContext())

{

db.Sessions.Remove(sessions);

db.SaveChanges();

result = "Сделано! " + sessions.Price + " Удалена";

}

return result;

}

#endregion

#region[Ticket]

public static string CreateTicket(Session session, Hall hall,

int amountRow, int amountPlace)

{

string result = "Уже существует";

using (ApplicationContext db = new ApplicationContext())

{

//проверяем сущесвует ли

bool checkIsExist = db.Tickets.Any(el => el.AmountRow == amountRow &&

el.HallId == hall.Id && el.AmountPlace == amountPlace && el.SessionId == session.Id);

if (!checkIsExist)

{

Ticket newtickets = new Ticket

{

SessionId = session.Id,

AmountPlace = amountPlace,

AmountRow = amountRow,

HallId = hall.Id,

};

db.Tickets.Add(newtickets);

db.SaveChanges();

result = " добавлен!";

}

return result;

}

}

public static string EditTicket(Ticket oldTicket, Film film, Session session, Hall hall, int row, int place)

{

string result = "Такого Резервирование не существует";

using (ApplicationContext db = new ApplicationContext())

{

Ticket ticket = db.Tickets.FirstOrDefault(d => d.Id == oldTicket.Id);

ticket.Session.FilmId = film.Id;

ticket.Session.Date = session.Date;

ticket.Session.FilmId = hall.Id;

ticket.AmountRow = row;

ticket.AmountPlace = place;

db.SaveChanges();

result = "Сделано! choiceHall " + ticket.AmountRow + " изменено";

}

return result;

}

public static string DeleteTicket(Ticket tickets)

{

string result = "Такой Film не существует";

using (ApplicationContext db = new ApplicationContext())

{

db.Tickets.Remove(tickets);

db.SaveChanges();

result = "Сделано! " + tickets.AmountPlace + tickets.AmountRow + " Удалена";

}

return result;

}

public static int GetIndexFilm(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

var index = db.Films.ToList().FindLastIndex(p => p.Id == id);

return index;

}

}

public static List<AmountRow> GetAmountRows(Hall hall)

{

using (ApplicationContext db = new ApplicationContext())

{

int amount = db.Halls.FirstOrDefault(p => p.Id == hall.Id).AmountRows;

List<AmountRow> result = new List<AmountRow>();

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

{

AmountRow amountRow = new AmountRow

{

Row = i

};

result.Add(amountRow);

}

return result;

}

}

public static List<AmountPlace> GetAmountPlace(Hall hall)

{

using (ApplicationContext db = new ApplicationContext())

{

int amount = db.Halls.FirstOrDefault(p => p.Id == hall.Id).AmountPlace;

List<AmountPlace> result = new List<AmountPlace>();

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

{

AmountPlace amountPlace = new AmountPlace

{

Place = i

};

result.Add(amountPlace);

}

return result;

}

}

}

namespace WpfAppCinema.Resources.MVVM.Models

{

public class Film

{

public int Id { get; set; }

public string FilmName { get; set; }

}

}

namespace WpfAppCinema.Resources.MVVM.Models

{

public class Hall

{

public int Id { get; set; }

public string ChoiceHall { get; set; }

public int AmountRows { get; set; }

public int AmountPlace { get; set; }

}

}

using System;

using System.Windows.Input;

namespace WpfAppCinema.Resources.MVVM.Models

{

public class RelayCommand : ICommand

{

private Action<object> \_execute;

private Func<object, bool> \_canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)

{

\_execute = execute;

\_canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return \_canExecute == null || \_canExecute(parameter);

}

public void Execute(object parameter)

{

\_execute(parameter);

}

}

}

using System.ComponentModel.DataAnnotations.Schema;

namespace WpfAppCinema.Resources.MVVM.Models

{

public class Session

{

public int Id { get; set; }

public int Price { get; set; }

public string Date { get; set; }

public int FilmId { get; set; }

public virtual Film Film { get; set; }

[NotMapped]

public Film SessionFilm

{

get

{

return DataWorker.GetFilmById((int)FilmId);

}

}

}

}

using System.ComponentModel.DataAnnotations.Schema;

namespace WpfAppCinema.Resources.MVVM.Models

{

public class Ticket

{

public int Id { get; set; }

public int SessionId { get; set; }

public Session Session { get; set; }

public int HallId { get; set; }

public Hall Hall { get; set; }

public int AmountRow { get; set; }

public int AmountPlace { get; set; }

[NotMapped]

public Session TicketSession

{

get

{

return DataWorker.GetSessionById((int)SessionId);

}

}

[NotMapped]

public Hall TicketHall

{

get

{

return DataWorker.GetHallById((int)HallId);

}

}

}

}

using Microsoft.EntityFrameworkCore;

namespace WpfAppCinema.Resources.MVVM.Models.Data

{

public class ApplicationContext : DbContext

{

public DbSet<Film> Films { get; set; }

public DbSet<Hall> Halls { get; set; }

public DbSet<Session> Sessions { get; set; }

public DbSet<Ticket> Tickets { get; set; }

public ApplicationContext() => Database.EnsureCreated();

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

//optionsBuilder.UseSqlServer(@"Data Source=localhost\SQLEXPRESS;Initial Catalog=WPFCinema19;Integrated Security=True");

optionsBuilder.UseSqlServer(@"Data Source=MSI-AEGIS-TI3;Initial Catalog=WPFCinema19;Integrated Security=True");

}

}

}

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class AddFilmVM : DataManageVM

{

public static string FilmName { get; set; }

public AddFilmVM() { }

public RelayCommand AddNewFilm

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

if (FilmName == null || FilmName.Replace(" ", "").Length == 0)

{

SetRedBlockControll(window, "TextBoxFilmName");

}

else

{

var result = DataWorker.CreateFilm(FilmName);

ShowMessageToUser(result);

FilmName = null;

window.Close();

}

});

}

}

}

}

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class AddHallVM : DataManageVM

{

public static string ChoiceHall { get; set; }

public static int AmountRows { get; set; }

public static int AmountPlace { get; set; }

public AddHallVM() { }

public RelayCommand AddNewHall

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

if (ChoiceHall == null || ChoiceHall.Replace(" ", "").Length == 0 ||

AmountRows <= 0 ||

AmountPlace <= 0)

{

if (ChoiceHall == null || ChoiceHall.Replace(" ", "").Length ==0

SetRedBlockControll(window, "TextBoxChoiceHall");

if (AmountRows <= 0)

SetRedBlockControll(window, "TextBoxAmountRows");

if (AmountPlace <= 0)

SetRedBlockControll(window, "TextBoxAmountPlace");

}

else

{

var result = DataWorker.CreateHall(ChoiceHall, AmountRows,

ShowMessageToUser(result);

AmountRows = 0;

AmountPlace = 0;

ChoiceHall = null;

window.Close();

}

});

}

}

}

}

using System.Collections.Generic;

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class AddSessionVM : DataManageVM

{

public static int Price { get; set; }

public static string Date { get; set; }

public static Film SelectedFilm { get; set; }

public AddSessionVM() { }

private List<Film> \_allFilms = DataWorker.GetAllFilms();

public List<Film> AllFilms

{

get { return \_allFilms; }

set

{

\_allFilms = value;

NotifyPropertyChanged("AllFilms");

}

}

public RelayCommand AddNewSession

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

if (SelectedFilm == null ||

Date == null || Date.Replace(" ", "").Length == 0 ||

Price < 100)

{

if (SelectedFilm == null)

ShowMessageToUser("Не выбран фильм");

if (Date == null || Date.Replace(" ", "").Length == 0)

SetRedBlockControll(window, "TextBoxDate");

if (Price < 100)

SetRedBlockControll(window, "TextBoxPrice");

}

else

{

var result = DataWorker.CreateSession(SelectedFilm, Price, Date);

ShowMessageToUser(result);

Price = 0;

Date = null;

SelectedFilm = null;

window.Close();

}

});

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WpfAppCinema.Resources.MVVM.Command;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class AddTicket2 : DataManageVM

{

public static AmountRow SelectedAmountRow { get; set; }

public static Hall SelectedHall { get; set; }

public static Session SelectedSession { get; set; }

private List<AmountRow> \_allAmountRows;

public List<AmountRow> AllAmountRows

{

get { return \_allAmountRows; }

set

{

\_allAmountRows = value;

NotifyPropertyChanged("AllAmountRows");

}

}

private List<AmountPlace> \_allAmountPlace;

public List<AmountPlace> AllAmountPlace

{

get { return \_allAmountPlace; }

set

{

\_allAmountPlace = value;

NotifyPropertyChanged("AllAmountPlace");

}

}

public AddTicket2(Hall hall,Session session)

{

SelectedHall = hall;

SelectedSession = session;

AllAmountRows = DataWorker.GetAmountRows(SelectedHall);

AllAmountPlace = DataWorker.GetAmountPlace(SelectedHall);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

using WpfAppCinema.Resources.MVVM.Command;

using WpfAppCinema.Resources.MVVM.Views;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class AddTicketVM : DataManageVM

{

public static Hall SelectedHall { get; set; }

public static Session SelectedSession { get; set; }

//public static AmountRow SelectedAmountRow { get; set; }

public AddTicketVM()

{

//AllAmountRows = DataWorker.GetAmountRows(SelectedHall);

}

private List<Hall> \_allHalls = DataWorker.GetAllHalls();

public List<Hall> AllHalls

{

get { return \_allHalls; }

set

{

\_allHalls = value;

NotifyPropertyChanged("AllHalls");

}

}

private List<Session> \_allSession = DataWorker.GetAllSessions();

public List<Session> AllSession

{

get { return \_allSession; }

set

{

\_allSession = value;

NotifyPropertyChanged("AllSessions");

}

}

private List<AmountRow> \_allAmountRows;

public List<AmountRow> AllAmountRows

{

get { return \_allAmountRows; }

set

{

\_allAmountRows = value;

NotifyPropertyChanged("AllAmountRows");

}

}

public RelayCommand NextTicket

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

if (SelectedHall == null ||

SelectedSession == null)

{

if (SelectedHall == null)

ShowMessageToUser("Вы не выбрали зал");

if (SelectedSession == null)

ShowMessageToUser("Вы не выбрали сеанс");

}

else

{

AddTicketViews2 addTicketViews2 = new AddTicketViews2(SelectedHall,SelectedSession,window);

SetCenterPositionAndOpen(addTicketViews2);

}

});

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using WpfAppCinema.Resources.MVVM.Models;

using WpfAppCinema.Resources.MVVM.Views;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class DataMainVM : DataManageVM

{

public TabItem SelectedTabItem { get; set; }

public static Film SelectedFilm { get; set; }

public static Hall SelectedHall { get; set; }

public static Session SelectedSession { get; set; }

public static Ticket SelectedTicket { get; set; }

public int IndexTabControl { get; set; }

public DataMainVM()

{

IndexTabControl = 0;

}

#region [Full table]

private List<Film> \_allFilms = DataWorker.GetAllFilms();

public List<Film> AllFilms

{

get { return \_allFilms; }

set

{

\_allFilms = value;

NotifyPropertyChanged("AllFilms");

}

}

private List<Hall> \_allHalls = DataWorker.GetAllHalls();

public List<Hall> AllHalls

{

get { return \_allHalls; }

set

{

\_allHalls = value;

NotifyPropertyChanged("AllHalls");

}

}

private List<Session> \_allSession = DataWorker.GetAllSessions();

public List<Session> AllSession

{

get { return \_allSession; }

set

{

\_allSession = value;

NotifyPropertyChanged("AllSessions");

}

}

private List<Ticket> \_allTicket = DataWorker.GetAllTickets();

public List<Ticket> AllTickets

{

get { return \_allTicket; }

set

{

\_allTicket = value;

NotifyPropertyChanged("AllTickets");

}

}

#endregion

public RelayCommand OpenAddNewFilmWnd

{

get

{

return null ?? new RelayCommand(obj =>

{

AddFilmVIews addFilmVIews = new AddFilmVIews();

SetCenterPositionAndOpen(addFilmVIews);

UpdateFilmsList();

});

}

}

public RelayCommand OpenAddNewHallWnd

{

get

{

return null ?? new RelayCommand(obj =>

{

AddHallViews addHallViews = new AddHallViews();

SetCenterPositionAndOpen(addHallViews);

UpdateHallList();

});

}

}

public RelayCommand OpenAddNewSessionWnd

{

get

{

return null ?? new RelayCommand(obj =>

{

AddSessionViews addSessionViews = new AddSessionViews();

SetCenterPositionAndOpen(addSessionViews);

UpdateSessionList();

});

}

}

public RelayCommand OpenAddNewTicketWnd

{

get

{

return null ?? new RelayCommand(obj =>

{

AddTicketViews addTicketViews = new AddTicketViews();

SetCenterPositionAndOpen(addTicketViews);

UpdateTicketList();

});

}

}

public RelayCommand OpenEditItemWnd

{

get

{

return null ?? new RelayCommand(obj =>

{

if (SelectedTabItem.Name == "FilmTab" && SelectedFilm != null)

{

EditFilmViews editFilmViews = new EditFilmViews(SelectedFilm);

SetCenterPositionAndOpen(editFilmViews);

SelectedFilm = null;

UpdateFilmsList();

}

if (SelectedTabItem.Name == "HallTab" && SelectedHall != null)

{

EditHallViews editHallViews = new EditHallViews(SelectedHall);

SetCenterPositionAndOpen(editHallViews);

SelectedHall = null;

UpdateHallList();

}

if (SelectedTabItem.Name == "SessionTab" && SelectedSession != null)

{

EditSessionViews editSessionViews = new EditSessionViews(SelectedSession);

SetCenterPositionAndOpen(editSessionViews);

SelectedSession = null;

UpdateSessionList();

}

if (SelectedTabItem.Name == "TicketTab" && SelectedTicket != null)

{

EditTicketViews editTicketViews = new EditTicketViews(SelectedTicket);

SetCenterPositionAndOpen(editTicketViews);

SelectedTicket = null;

UpdateTicketList();

}

});

}

}

public RelayCommand DeleteItem

{

get

{

return null ?? new RelayCommand(obj =>

{

string result = "Ничего не выбрано";

if (SelectedTabItem.Name == "FilmTab" && SelectedFilm != null)

{

result = DataWorker.DeleteFilm(SelectedFilm);

UpdateFilmsList();

}

if (SelectedTabItem.Name == "HallTab" && SelectedHall != null)

{

result = DataWorker.DeleteHall(SelectedHall);

UpdateHallList();

}

if (SelectedTabItem.Name == "SessionTab" && SelectedSession != null)

{

result = DataWorker.DeleteSession(SelectedSession);

UpdateSessionList();

}

if (SelectedTabItem.Name == "TicketTab" && SelectedTicket != null)

{

result = DataWorker.DeleteTicket(SelectedTicket);

UpdateTicketList();

}

ShowMessageToUser(result);

});

}

}

protected void UpdateFilmsList()

{

AllFilms = DataWorker.GetAllFilms();

MainWork.AllFilmsInfoListView.ItemsSource = null;

MainWork.AllFilmsInfoListView.Items.Clear();

MainWork.AllFilmsInfoListView.ItemsSource = AllFilms;

MainWork.AllFilmsInfoListView.Items.Refresh();

}

protected void UpdateHallList()

{

AllHalls = DataWorker.GetAllHalls();

MainWork.AllHallsInfoListView.ItemsSource = null;

MainWork.AllHallsInfoListView.Items.Clear();

MainWork.AllHallsInfoListView.ItemsSource = AllHalls;

MainWork.AllHallsInfoListView.Items.Refresh();

}

protected void UpdateSessionList()

{

AllSession = DataWorker.GetAllSessions();

MainWork.AllSessionsInfoListView.ItemsSource = null;

MainWork.AllSessionsInfoListView.Items.Clear();

MainWork.AllSessionsInfoListView.ItemsSource = AllSession;

MainWork.AllSessionsInfoListView.Items.Refresh();

}

protected void UpdateTicketList()

{

AllTickets = DataWorker.GetAllTickets();

MainWork.AllTiketsInfoListView.ItemsSource = null;

MainWork.AllTiketsInfoListView.Items.Clear();

MainWork.AllTiketsInfoListView.ItemsSource = AllTickets;

MainWork.AllTiketsInfoListView.Items.Refresh();

}

}

}

using System;

using System.ComponentModel;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using WpfAppCinema.Resources.MVVM.Views;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class DataManageVM : INotifyPropertyChanged

{

public static void ShowMessageToUser(string message)

{

MessageWindow newMessageWindow = new MessageWindow(message);

newMessageWindow.WindowStartupLocation = WindowStartupLocation.CenterScreen;

newMessageWindow.ShowDialog();

}

protected void SetCenterPositionAndOpen(Window window)

{

window.Owner = Application.Current.MainWindow;

window.WindowStartupLocation = WindowStartupLocation.CenterOwner;

window.ShowDialog();

}

protected void SetRedBlockControll(Window wnd, string blockName)

{

Control block = wnd.FindName(blockName) as Control;

block.BorderBrush = Brushes.Red;

}

public event PropertyChangedEventHandler PropertyChanged;

public void NotifyPropertyChanged(String propertyName)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

}

}

}

using System.Collections.Generic;

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class EditFilmVM : DataManageVM

{

private string \_filmName;

public static Film SelectedFilm { get; set; }

public EditFilmVM(Film selectedFilm)

{

SelectedFilm = selectedFilm;

FilmName = SelectedFilm.FilmName;

}

public string FilmName

{

get => \_filmName;

set

{

\_filmName = value;

NotifyPropertyChanged(nameof(FilmName));

}

}

private List<Film> \_allFilms = DataWorker.GetAllFilms();

public List<Film> AllFilms

{

get { return \_allFilms; }

set

{

\_allFilms = value;

NotifyPropertyChanged("AllFilms");

}

}

public RelayCommand EditFilm

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

string result = "Не выбран фильм";

if (SelectedFilm != null)

{

if (FilmName == null || FilmName.Replace(" ", "").Length == 0)

{

SetRedBlockControll(window, "TextBoxFilmName");

}

else

{

result = DataWorker.EditFilm(SelectedFilm, FilmName);

ShowMessageToUser(result);

FilmName = null;

SelectedFilm = null;

window.Close();

}

}

else ShowMessageToUser(result);

});

}

}

}

}

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class EditHallVM : DataManageVM

{

private string \_choiceHall;

private int \_amountRows { get; set; }

private int \_amountPlace { get; set; }

public static Hall SelectedHall { get; set; }

public EditHallVM(Hall selectedHall)

{

SelectedHall = selectedHall;

ChoiceHall = SelectedHall.ChoiceHall;

AmountRows = SelectedHall.AmountRows;

AmountPlace= SelectedHall.AmountPlace;

}

public string ChoiceHall

{

get => \_choiceHall;

set

{

\_choiceHall = value;

NotifyPropertyChanged(nameof(ChoiceHall));

}

}

public int AmountRows

{

get => \_amountRows;

set

{

\_amountRows = value;

NotifyPropertyChanged(nameof(AmountRows));

}

}

public int AmountPlace

{

get => \_amountPlace;

set

{

\_amountPlace = value;

NotifyPropertyChanged(nameof(AmountPlace));

}

}

public RelayCommand EditHall

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

string result = "Не выбран зал";

if (SelectedHall != null)

{

if (ChoiceHall == null || ChoiceHall.Replace(" ", "").Length == 0 ||

AmountRows <= 0 ||

AmountPlace <= 0)

{

if (ChoiceHall == null || ChoiceHall.Replace(" ", "").Length == 0)

SetRedBlockControll(window, "TextBoxChoiceHall");

if (AmountRows <= 0)

SetRedBlockControll(window, "TextBoxAmountRows");

if (AmountPlace <= 0)

SetRedBlockControll(window, "TextBoxAmountPlace");

}

{

result = DataWorker.EditHall(SelectedHall, ChoiceHall, AmountRows, AmountPlace);

ShowMessageToUser(result);

ChoiceHall = null;

SelectedHall = null;

AmountRows = 0;

AmountPlace = 0;

window.Close();

}

}

else ShowMessageToUser(result);

});

}

}

}

}

using System.Collections.Generic;

using System.Windows;

using WpfAppCinema.Resources.MVVM.Models;

namespace WpfAppCinema.Resources.MVVM.ViewModels

{

public class EditSessionVM : DataManageVM

{

public static Session SelectedSession { get; set; }

private int \_price;

private string \_date;

private int \_indexFilm;

public static Film SelectedFilm { get; set; }

public EditSessionVM(Session selectedSession)

{

SelectedSession = selectedSession;

Price = SelectedSession.Price;

Date = SelectedSession.Date;

IndexFilm = DataWorker.GetIndexFilm(SelectedSession.FilmId);

}

public int Price

{

get => \_price;

set

{

\_price = value;

NotifyPropertyChanged(nameof(Price));

}

}

public string Date

{

get => \_date;

set

{

\_date = value;

NotifyPropertyChanged(nameof(Date));

}

}

public int IndexFilm

{

get => \_indexFilm;

set

{

\_indexFilm = value;

NotifyPropertyChanged(nameof(IndexFilm));

}

}

private List<Film> \_allFilms = DataWorker.GetAllFilms();

public List<Film> AllFilms

{

get { return \_allFilms; }

set

{

\_allFilms = value;

NotifyPropertyChanged("AllFilms");

}

}

public RelayCommand EditSession

{

get

{

return null ?? new RelayCommand(obj =>

{

Window window = obj as Window;

string result = "Не выбран сеанс";

if (SelectedSession != null)

{

if (SelectedFilm == null ||

Date == null || Date.Replace(" ", "").Length == 0 ||

Price < 100)

{

if (SelectedFilm == null)

ShowMessageToUser("Не выбран фильм");

if (Date == null || Date.Replace(" ", "").Length == 0)

SetRedBlockControll(window, "TextBoxDate");

if (Price < 100)

SetRedBlockControll(window, "TextBoxPrice");

}

else

{

result = DataWorker.EditSession(SelectedSession, SelectedFilm, Price, Date);

ShowMessageToUser(result);

Price = 0;

Date = null;

SelectedFilm = null;

SelectedFilm = null;

window.Close();

}

}

else ShowMessageToUser(result);

});

}

}

}

}