

Министерство образования и науки России

**Федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

**«Казанский национальный исследовательский технологический университет»**

**(ФГБОУ ВО «КНИТУ»)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра «Интеллектуальных систем и управления информационными ресурсами»

Направление специальность: Математическое обеспечение и администрирование информационных систем и технологий.

Тема курсовой работы (проекта): разработка прототипа информационной системы «Заказ билетов в кинотеатре»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КУРСОВОЙ ПРОЕКТ**

Заведующий кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Кирпичников А.П.) Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Мангушева А.Р.)

Студент группы 4371-21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Давыдкина Е.М)

Нормоконтролер\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Мангушева.А.Р)

Казань 2020г.

**Оглавление**

[ВВЕДЕНИЕ 3](#_Toc8893328)

[ЗАДАНИЕ 4](#_Toc8893329)

[ЛИСТ НОРМОКОНТРОЛЕРА 5](#_Toc8893330)

[Глава 1. РАЗРАБОТКА ПРОЕКТА пРОТОТИПА ИНФОРМАЦИОННОЙ СИСТЕМЫ 6](#_Toc8893331)

[1.1. иССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc8893332)

[1.2. Действующие лица и варианты использования системы 7](#_Toc8893332)

[1.3. концептуальное проектирование базы данных 13](#_Toc8893333)

[1.3.1 Перечень сущностей и Атрибутов 14](#_Toc8893334)

[ГЛАВА 2. РАЗРАБОТКА ПРОТОТИПА ИНФОРМАЦИОННОЙ СИСТЕМЫ 20](#_Toc8893334)

2.1 [подключение баз данных к Visual Studio, создание программы и форм 20](#_Toc8893329)

[2.2 КЛАССЫ ДЛЯ СОЗДАНИЯ, УДАЛЕНИЯ, ПОИСКА ДАННЫХ 23](#_Toc8893335)

[ЗАКЛЮЧЕНИЕ 34](#_Toc8893329)

[СПИСОК ЛИТЕРАТУРЫ 35](#_Toc8893337)

[ПРИЛОЖЕНИЕ 36](#_Toc8893329)

# ВВЕДЕНИЕ

В век современных технологий все большее применение получают компьютерные методы учёта продукции, тем самым всё больше предприятий нуждаются в цифровизации своих данных.

Целью данного курсового проекта является создание прототипа информационной системы «Кинотеатр» с помощью полученных знаний при изучении курса «Визуальное программирование». А также, самостоятельное обучение технологиям создания сложных программных продуктов.

Интерфейс программы должен быть удобным для пользователя. Основная функция программы должна заключается в том, чтобы пользователь программы мог легко работать с базой кинотеатра, добавлять и искать необходимую ему информацию о фильмах.

Задачи, поставленные перед началом процесса создания проекта:

• проанализировать предметную область;

• разработать прототип информационной системы с использованием Windows Form;

• создать таблицы в PostgreSQL, хранимые процедуры для добавления, изменения и поиска данных.

• изучить возможность подключения PostgreSQL к Visual Studio и разработать прототип, которым в дальнейшем можно пользоваться.

### З А Д А Н И Е

На курсовую работу (проект) студенту кафедры

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тема курсовой работы(проекта):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные к проекту:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание расчетно-пояснительной записки (включая перечень подлежащих разработке вопросов, включая вопросы стандартизации и контроля качества) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перечень графического материала (схемной документации)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Консультанты по проекту (с указанием относящихся к ним разделов)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

### ЛИСТ НОРМОКОНТРОЛЕРА

1. Лист является обязательным приложением к пояснительной записке дипломного (курсового) проекта.

2. Нормоконтролер имеет право возвращать документацию без рассмотрения в случаях:

- нарушения установленной комплектности,

- отсутствия обязательных подписей,

- нечеткого выполнения текстового и графического материала.

3. Устранение ошибок, указанных нормоконтролером, обязательно

П е р е ч е н ь

замечаний и предложений нормоконтролера по дипломному (курсовому) проекту, студента

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(группа, инициалы, фамилия)

|  |  |  |
| --- | --- | --- |
| Лист (страница) | Условное обозначение (код ошибок) | Содержание замечаний и предложений со ссылкой на нормативный документ, стандарт или типовую документацию |
|  |  |  |

Дата «\_\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_г.

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (инициалы, фамилия)

# Глава 1. рАЗРАБОТКА пРОЕКТА ПРОТОТИПА ИНФОРМАЦИОННОЙ СИСТЕМЫ.

# 1.1 Исследование предметной области

Имеем небольшой многозальный кинотеатр, в котором располагаются места разной комфортности и соответственно разной ценовой категории. Для просмотра фильма необходимо приобрести билет, в котором указан номер билета, название фильма, зал, время, место и цена. При этом кинотеатром ведется учёт занято или свободно место для продажи и возврата билета. Кроме того, клиентам предоставляются услуги по бронированию билетов.

В функционирование предметной области входит следующее:

• Продажа билетов

• Контроль наполняемости зала

• Услуги бронирования билетов и снятия брони

• Возврат билетов (допускается возврат билета и снятие его с брони)

• Предоставление информации о фильмах

Сотрудникам кинотеатра предоставляется возможность модифицировать данные, вносить необходимую информацию о фильмах, просматривать текущие фильмы, которые идут в кинотеатре, а также им доступна возможность делать возврат билета.

Работа программы устроена следующим образом: в базе данных хранится информация о фильмах. С помощью визуального приложения на Windows нам доступны такие функции как бронирование и покупка билетов, просмотр свободных мест в зале, редактирование данных и возврат билетов.

# 1.2. Действующие лица и варианты использования системы.

Действующими лицами разрабатываемой системы являются пользователь (кассир) и администратор. Диаграмму вариантов использования можно увидеть на рисунке 1.

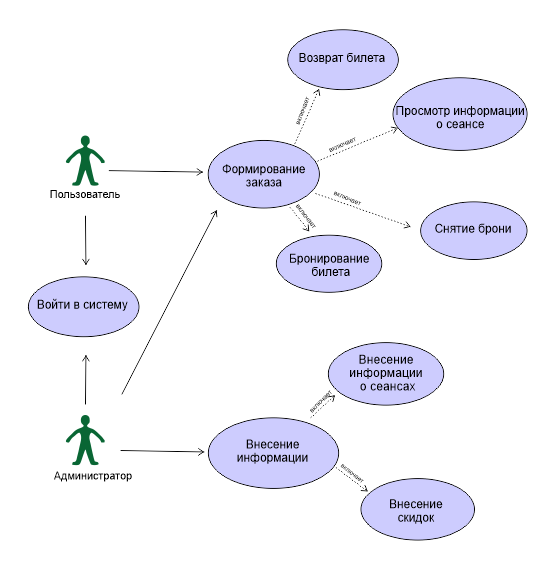


Рисунок 1. Диаграмма вариантов использования

Далее подробно рассмотрим варианты использования системы.

**Вариант использования “Формирование заказа”:**

Данный вариант использования начинает исполняться, когда пользователь продаёт билет клиенту.

Основной поток событий:

Данный вариант использования начинается, когда клиент указал что хочет приобрести билет.

1.Система предоставляет пользователю возможность выбора фильма, времени сеанса и зала

2.Пользователь выбирает сеанс из тех, что предоставляет система, выбрав из списка фильмов.

3.Пользователь выбирает время начала фильма из тех, что предоставляет система, выбрав из списка время сеанса.

4.Пользователь выбирает место в зале из доступных в зрительном зале.

5.Пользователь соглашается с введенными требованиями.

6.Система формирует заказ на основании введенных требований.

7.Система предлагает пользователю проверить правильность введённых данных в билете.

8.Клиент подтверждает, что данные корректны

9.Система выводит сообщение о том, что данные записаны, что подразумевает покупку билета

10.Система записывает покупку билета в базу данных.

Альтернативный поток:

Нет свободных мест. Если во время выполнения основного потока обнаружится, что на данный сеанс нет свободных мест, система выдаёт сообщение о том, что нет свободных мест.

**Вариант использования “Возврат билета”:**

Данный вариант использования начинается исполняться, когда клиент хочет вернуть билет.

Основной поток:

Данный вариант использования начинается, когда клиент сообщает пользователю-кассиру, что хочет вернуть билет.

1. Пользователь выбирает возможность «Вернуть билет»

2. Пользователь вводит номер билета или сканирует штрих-код

3. Если данный билет действительно существует, то пользователь может продолжить процедуру возврата билета, иначе запускается альтернативный поток.

4. Система запрашивает подтверждение на возврат билета.

5. Пользователь соглашается с запросом системы.

6. Система отмечает в базе данных те места, что были в билете как свободные.

Альтернативный поток:

Введен неверный номер билета. Система выдает сообщение, что билет не найден.

**Вариант использования “Бронирование билета”:**

Данный вариант использования начинает исполняться, когда клиент хочет забронировать билет.

Основной поток событий:

Данный вариант использования начинается, когда клиент указал что хочет забронировать билет.

1.Система предоставляет пользователю таблицу с выпадающими списками для заполнения.

2.Пользователь выбирает сеанс из тех, что предоставляет система, выбрав из списка расписания сеансов.

3.Пользователь выбирает время начала сеанса из тех, что предоставляет система.

4.Пользователь выбирает место из доступных в зрительном зале.

5.Система формирует заказ на основании введенных требований.

6.Система предлагает пользователю проверить правильность введённых данных в билете.

7.Клиент подтверждает, что данные корректны

9.Пользователь указывает что это «бронирование»

10.Система записывает бронь в базу данных

Альтернативный поток:

Нет свободных мест. Если во время выполнения основного потока обнаружится, что на данный сеанс нет свободных мест, система выдаёт сообщение, что нет свободных мест.

**Вариант использования “Снятия с брони”**

Данный вариант использования начинает исполняться, когда клиент хочет снять бронь.

Основной поток:

Данный вариант использования начинается, когда клиент сообщает пользователю-кассиру, что хочет снять с брони билет.

1. Пользователь выбирает возможность «Снять бронь»

2. Пользователь вводит номер билета или сканирует штрих-код

3. Если данный билет действительно существует, то пользователь может продолжить процедуру снятия с брони билета, иначе запускается альтернативный поток.

4. Система запрашивает подтверждение на отмену бронирования.

5. Пользователь соглашается с запросом системы.

6. Система отмечает в базе данных те места что были в билете как свободные.

Альтернативные потоки:

Введен неверный номер билета. Система выдает сообщение, что билет не найден.

**Вариант использования “Просмотр информации о сеансах”**

Данный вариант использования начинает исполняться, когда пользователь желает просмотреть информацию о сеансах в кинотеатре.

*Основной поток событий:*

Данный вариант использования начинается, когда клиент хочет узнать какие фильмы можно посмотреть в кинотеатре.

1.Система предоставляет пользователю возможность выбора фильма, времени сеанса и зала

2.Пользователь выбирает фильм из тех, что предоставляет система, выбрав из списка фильмов.

3.Пользователь выбирает время начала фильма из тех, что предоставляет система, выбрав из списка время сеанса.

4.Система в графическом виде предоставляет информацию о свободных местах в зале на выбранный пользователем сеанс.

Альтернативные потоки:

Нет.

**Вариант использования “Внесение информации”**

Данный вариант использования начинает исполняться, когда администратор хочет внести информацию о доступных местах в зале, скидках и расписании фильмов.

Основной поток:

1.Данный вариант исполняется, когда администратор выбирает возможность внесение информации.

2.Администратор заполняет или изменяет необходимые поля для внесения информации о фильме.

3. Система предлагает сохранить изменения в базе данных.

4. Администратор соглашается с запросом системы.

5.Система записывает в базу данных изменения, введенные администратором.

Альтернативные потоки:

Нет.

**Вариант использования “Внесение информации о сеансах”**

Данный вариант использования начинает исполняться, когда администратор хочет внести информацию о расписании фильмов, времени и зала.

Основной поток:

1.Данный вариант исполняется, когда администратор открывает вкладку внесение информации.

2.Администратор заполняет или изменяет необходимые поля внесения необходимой информации о фильме.

3. Система предлагает сохранить изменения в базе данных.

4. Администратор соглашается с запросом системы.

5.Система записывает в базу данных изменения, введенные администратором.

Альтернативные потоки:

Нет.

**Вариант использования “Внесение информации о скидках”**

Данный вариант использования начинает исполняться, когда администратор хочет добавить информацию о фильме со скидками.

Основной поток:

1. Данный вариант начинает исполняется, когда администратор открывает вкладку внесение информации.

3.Администратор выбирает и вводит необходимую информацию для внесения данных о скидках.

4. Система предлагает сохранить изменения в базе данных.

5. Администратор соглашается с запросом системы.

6.Система записывает в базу данных изменения, введенные администратором.

Альтернативные потоки:

Нет.

**Вариант использования войти в систему:**

Данный вариант использования описывает вход пользователя в систему кинотеатра.

Основной поток событий

Данный вариант использования начинает выполняться, когда пользователь хочет войти в систему кинотеатра.

1. Система запрашивает имя пользователя и пароль.
2. Пользователь вводит имя и пароль.
3. Система проверяет имя и пароль, после чего открывается доступ в систему и отправляет пользователя на вкладку формирование заказа.
4. Для выхода из системы пользователь может выбрать возможность выход в углу основного окна программы.
5. Для последующего входа в систему у пользователя появляется возможность войти в систему.

Альтернативные потоки

Неправильное имя/пароль.Если во время выполнения Основного потокаобнаружится, что пользователь ввел неправильное имя и/или пароль, система выводит сообщение об ошибке. Пользователь может вернуться к началу Основного потокаили отказаться от входа в систему, при этом выполнение варианта использования завершается.

# 1.3. Концептуальное проектирование базы данных

Концептуальное проектирование — создание БД, включающая в себя определение таблиц, но с высоким уровнем абстракции от модели бд и физической реализации.

Первым делом мы определяем какую информацию необходимо хранить в базе данных. Входные данные данного проекта:

• Информация о фильмах;

• Информация о залах.

• Информация о купленных билетах.

## 1.3.1 Перечень сущностей И АТРИБУТОВ.

Сущность — это некоторая абстракция реального мира, которая может существовать независимо. Сущность может иметь экземпляры, отличающиеся друг от друга значениями атрибутов и допускающие однозначную идентификацию. [2] В нашей системе кинотеатра будет храниться информация о фильмах и расписании сеансов.

Выделим базовые сущности данной предметной области:

1.сущность Movie(фильмы) содержит информацию о фильмах;

2. сущность Showtime (расписание сеансов) содержит информацию о сеансе;

3. сущность Ticket (билет) содержит сведения о приобретенном билете.

4. сущность Hall (зал) содержит сведения о залах в кинотеатре.

Атрибутом является свойство объекта сущности.[4]

Определим список атрибутов от каждой сущности:

1. Для сущности Movie атрибутами являются:

- movieID

-year\_movie

-title

-duration

-director

-genre

-Address

-cast\_movie

2. Для сущности Showtime атрибутами являются:

- showtime\_id

- date

- time

- price

- movieid

3. Для сущности Ticket атрибутами являются:

- ticket\_id

- hall\_id

- price

- seat

- shouwtime\_id

- role

4. Для сущности Roles атрибутами являются:

- role

- login

- password

5. Для сущности Hall атрибутами являются:

- hall\_id

- hall\_name

- seats

6. Для сущности Hall\_Showtime атрибутами являются:

- hSh\_id

- showtime\_id

- hall\_id

- seats\_map

Результат диаграммы (ER-модель) можно увидеть на рисунке 1.

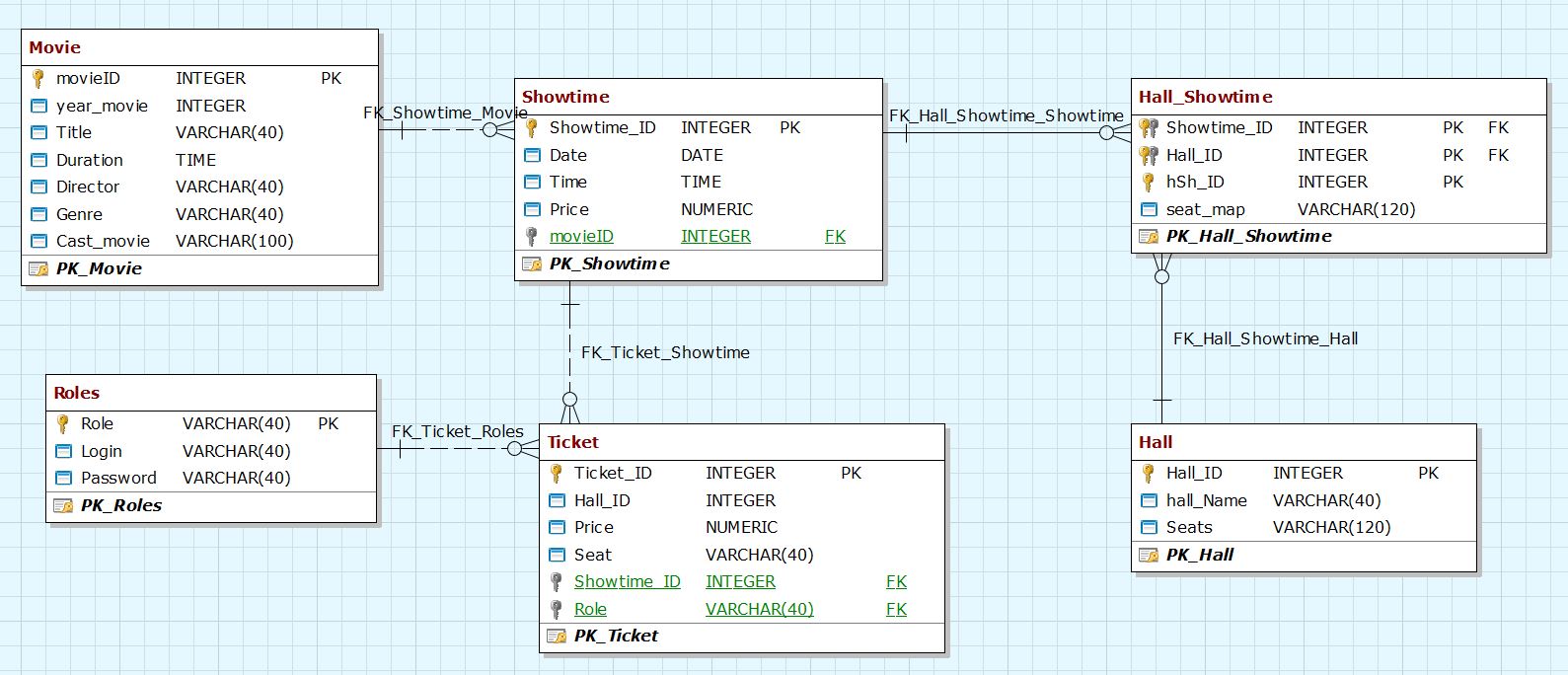


Рисунок 1 – ER-модель

Чтобы пользователю иметь представление как будет выглядеть система, создаём и согласовываем макет интерфейса программы. Данный макет можно увидеть на рисунках 2,3,4,5 и 6. Где, рисунок 2 — это вкладка для смены пользователя, рисунок 3 — это вкладка формирования заказа, рисунок 4 — это вкладка просмотра информации о сеансе и рисунок 5 — это вкладка для внесения информации, рисунок 6 — это вкладка для возврата билета.

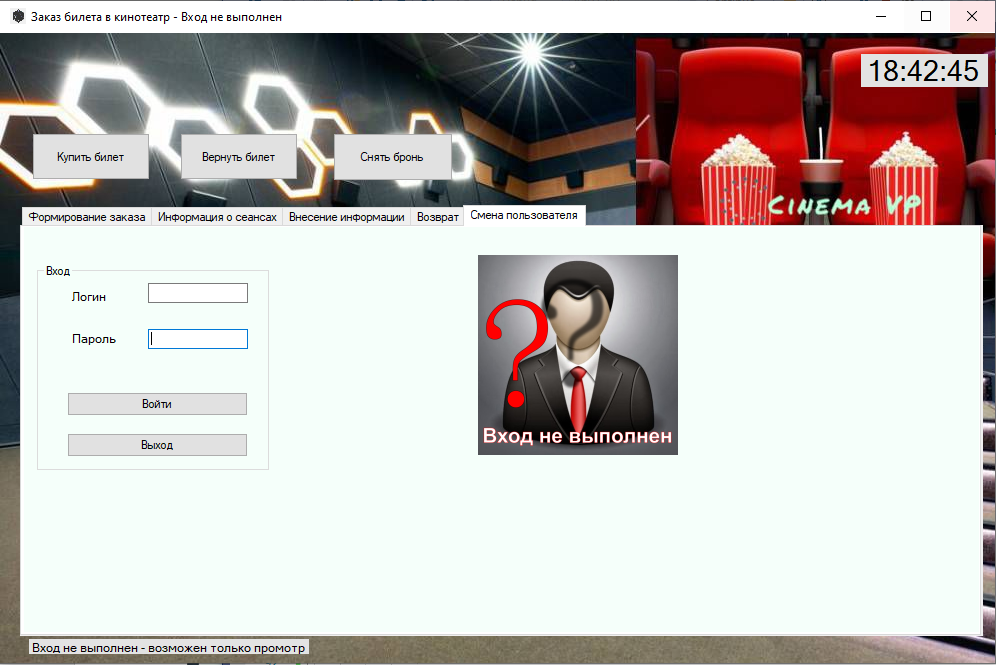


Рисунок 2 – Смена пользователя

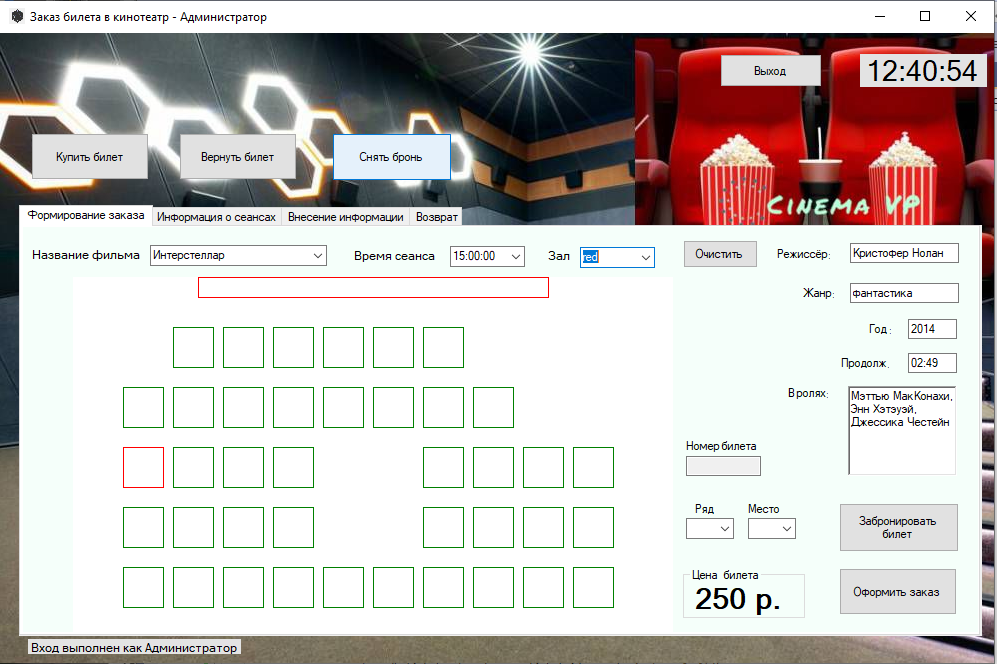


Рисунок 3 – Формирование заказа

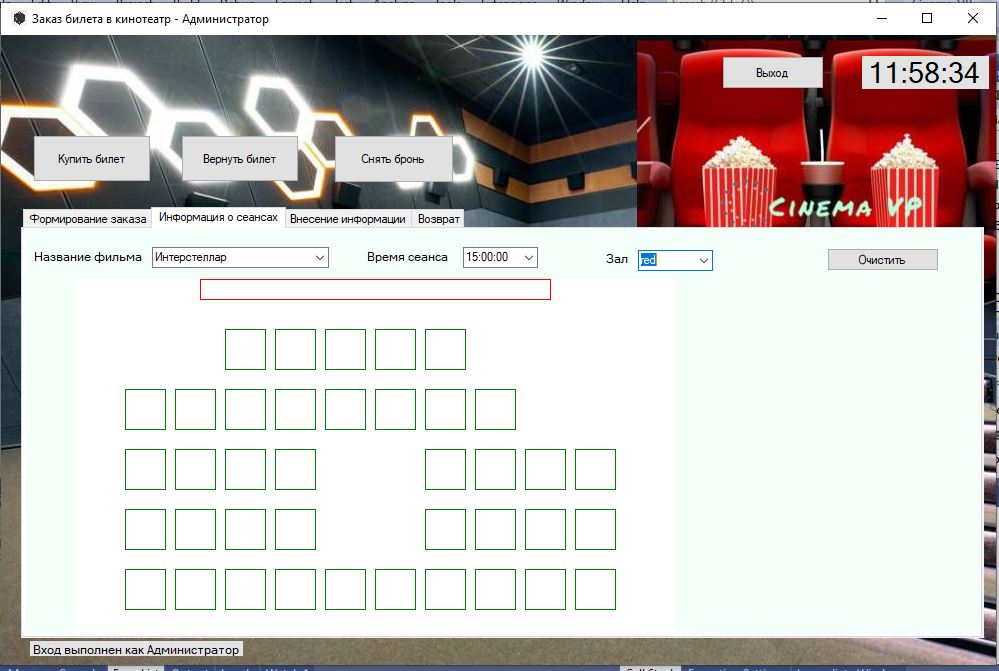


Рисунок 4- Информация о сеансе

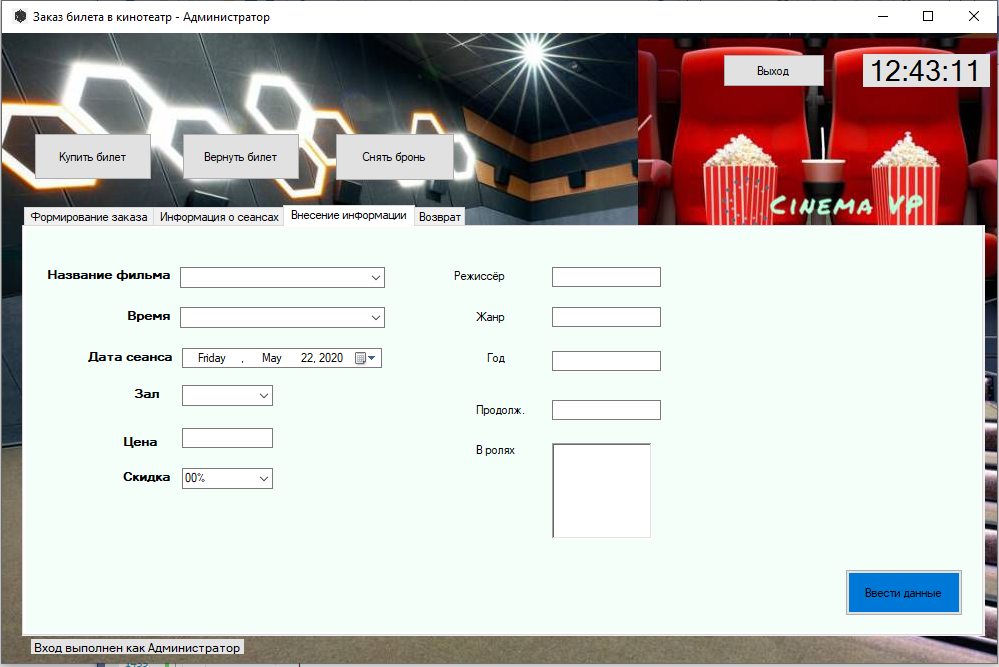


Рисунок 5- Внесение информации

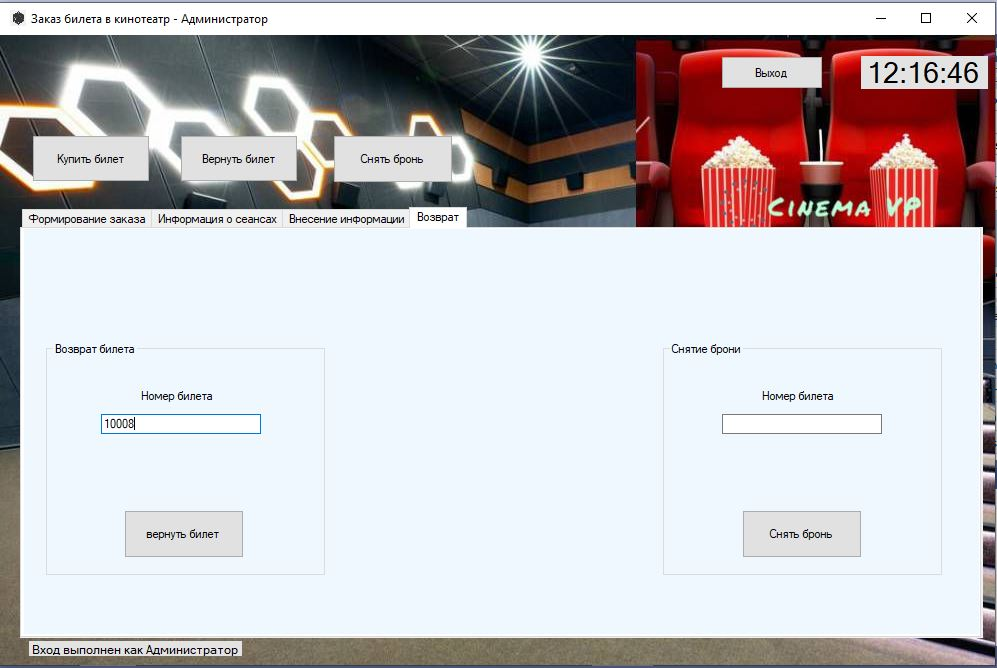


Рисунок 6 - возврат

**ГЛОССАРИЙ**

1. **PostgreSQL** — это свободная и открытая система управления реляционными базами данных, подчеркивающая расширяемость и соответствие SQL.
2. **Диаграмма вариантов использования** — представление взаимодействия пользователя с системой, которое показывает связь между пользователем и различными вариантами использования, в которых участвует пользователь.
3. **Интерфейс программы** — это общая граница, через которую два или более отдельных компонента компьютерной системы обмениваются информацией.
4. **ER-модель**— модель отношений между сущностями, описывает взаимосвязанные вещи, представляющие интерес в конкретной области знаний. Базовая модель ER состоит из типов сущностей и определяет отношения, которые могут существовать между сущностями
5. **Цифровизация** — процесс перехода предприятий или экономических отраслей на новые модели ведения бизнеса, основанных на информационных технологиях.

# Глава 2. рАЗРАБОТКА ПРОТОТИПА ИНФОРМАЦИОННОЙ СИСТЕМЫ.

# 2.1. подключение баз данных к Visual Studio, создание программы и форм

После того как схема базы данных была определена, переходим к созданию визуального приложения в среде Visual Studio. Для этого подключаем postgres к Visual Studio, используя entity framework. Entity Framework — это набор библиотек API для работ с бд, которое находит применение в программировании на языке C#. [3] Он позволяет взаимодействовать с СУБД с помощью сущностей, а не таблиц. Для того, чтобы использовать entity framework в нашем проект добавляем такой компонент как entity framework и NpgSql в менеджере пакетов NuGet. [1] Проверяем соединение и подключаем базу данных к проекту. Подключенную базу данных можно увидеть на рисунке 2.

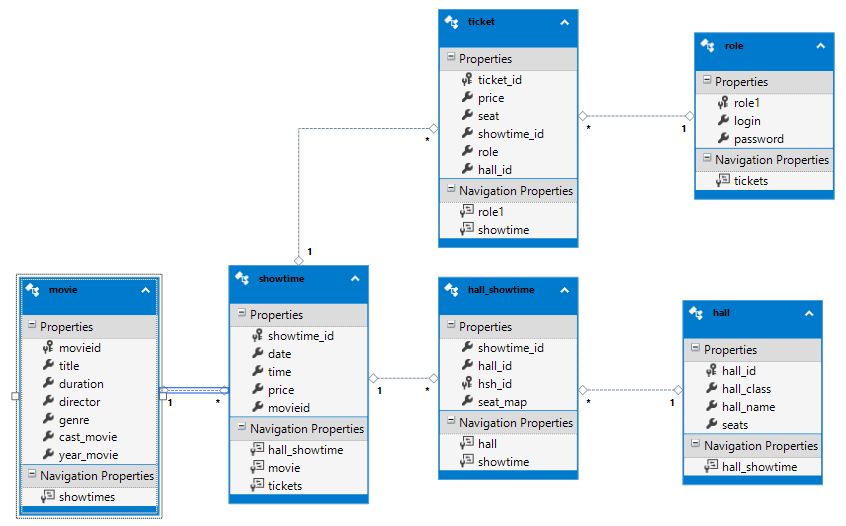


Рисунок 3- соединение базы данных с Visual Studio

Для удобства использования приложение будет написано с оконным интерфейсом Windows Form. Forms представляет из себя диалоговое окно, являющейся пользовательским интерфейсом приложения. В Windows Forms формой является площадь окна, на которую выводится информация для пользователя. Чтобы построить приложение Windows Form, нужно поместить элементы управления на форму и написать код для обработки действий пользователя. [6] Windows Forms включает в себя множество элементов управления, которые можно использовать при создании приложения: текстовые поля, раскрывающиеся списки, кнопки, переключатели и даже веб-страницы. В данном приложении на основную форму помещены кнопки быстрого доступа, а весь основной функционал, такой как покупка билета, возврат билета, просмотр информации о сеансах реализован на вкладках, что позволяет нам быстро переключаться с одной задачи на другую.

В приложении “заказ билета в кинотеатр” созданы формы, которые представлены на рисунках 2,3,4 и 5.

Первая и самая главная форма — это вкладка для входа пользователя в систему, рисунок 2. Она позволяет нам сберечь систему от посторонних лиц и защитить от несанкционированных действий. Однако, перед тем как войти в систему, необходимо ввести логин и пароль, чтобы система открыла доступ определённых функций для определенного сотрудника. При входе в систему, система выводит картинки, которые показывают нам в какой роли мы вошли, что очень удобно для пользования и дальнейшей работы с программой.

После того, как был произведён вход в систему, пользователь переходит во вкладку с формированием заказа, рисунок 3, чтобы продать билет, но при этом ему доступна и другие вкладки для работы, а именно оформить заказ, посмотреть информацию о сеансе, вернуть билет и внести данные о фильме или же он может перейти через кнопки к необходимой ему вкладке.

На вкладке формирование заказа, которая представлена на рисунке 4, пользователям доступны две кнопки “забронировать билет” и “оформить заказ”, при помощи который можно забронировать и купить билет на определённое место и увидеть есть ли свободные места на определенный фильм, однако перед этим нужно сначала выбрать тот фильм, на который хотят приобрести билет, время сеанса, зал и место, после этого можно оформить заказ, который выводит купленный билет для пользователя. Если же пользователи после того, как закончили свою работу, хотят выйти из системы, они могут выбрать возможность выход в углу основного окна программы и им больше не будет доступна продажа билетов, внесение информации и их возврат.

На второй вкладке, представленной на рисунке 4, сотрудникам кинотеатра доступен просмотр текущих фильмов в определенное время и определённом зале. Рядом с ними находится кнопка “очистить”, куда нужно обратиться, чтобы очистить текущий фильм и посмотреть доступные места на другие фильмы. Тем самым, мы можем посмотреть свободные места на сеанс в удобном графическом виде.

На третьей вкладке, представленной на рисунке 5, доступно внесение информации о фильме и предоставляется скидка на билет для того или иного фильма, но лишь для администратора, так как потом могут возникнуть проблемы при продаже билетов.

На рисунке 6 можно увидеть, как вернуть купленный билет. Для этого пользователю необходимо ввести номер билета, который он может взять на вкладке оформления заказа и нажать на кнопку вернуть билет. В программе уже есть все некоторые данные о фильмах, поэтому нам доступны такие функции как возврат и снятие брони, а как их внести, удалить и произвести еще какие-либо действия над программой показано в подпункте 2.2.

# 2.2. КЛАССЫ ДЛЯ СОЗДАНИЯ, УДАЛЕНИЯ И ПОИСКА ДАННЫХ

Чтобы пользователь системы мог изменять, добавлять, удалять и находить данные в системе, необходимо создать методы и классы в Visual Studio. [5]

К примеру, на рисунке 7 представлена метод для поиска фильмов, а также класс на рисунке 8, который необходим для работы данного метода.



Рисунок 7 – метод для поиска фильмов по его названию

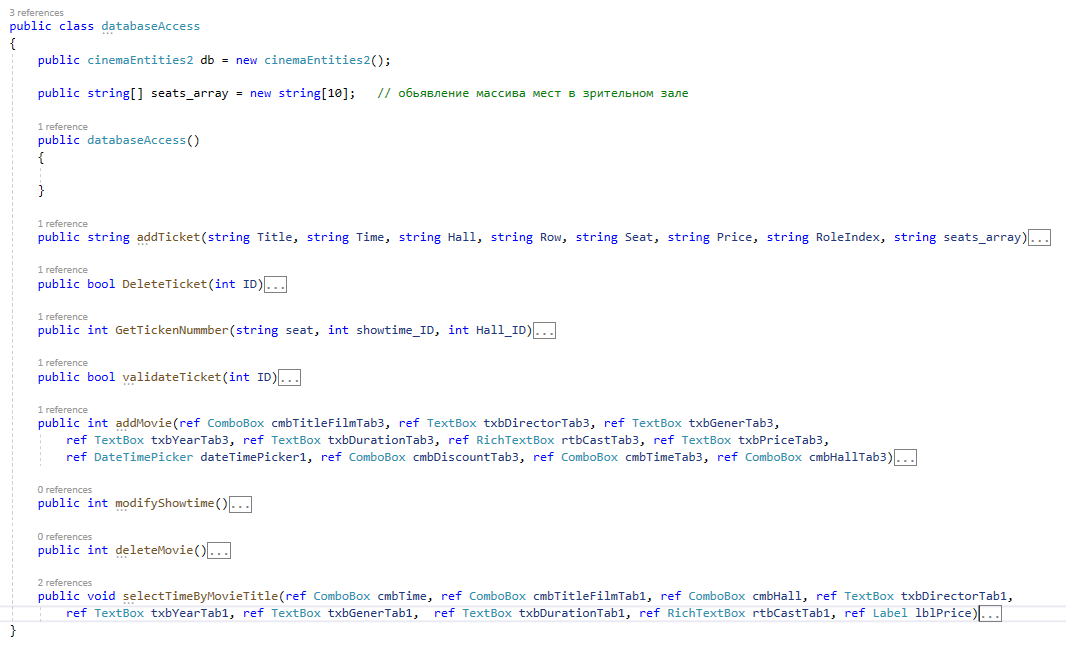


Рисунок 8 – класс для работы с базой данных.

На рисунке 9 показана работа класса:

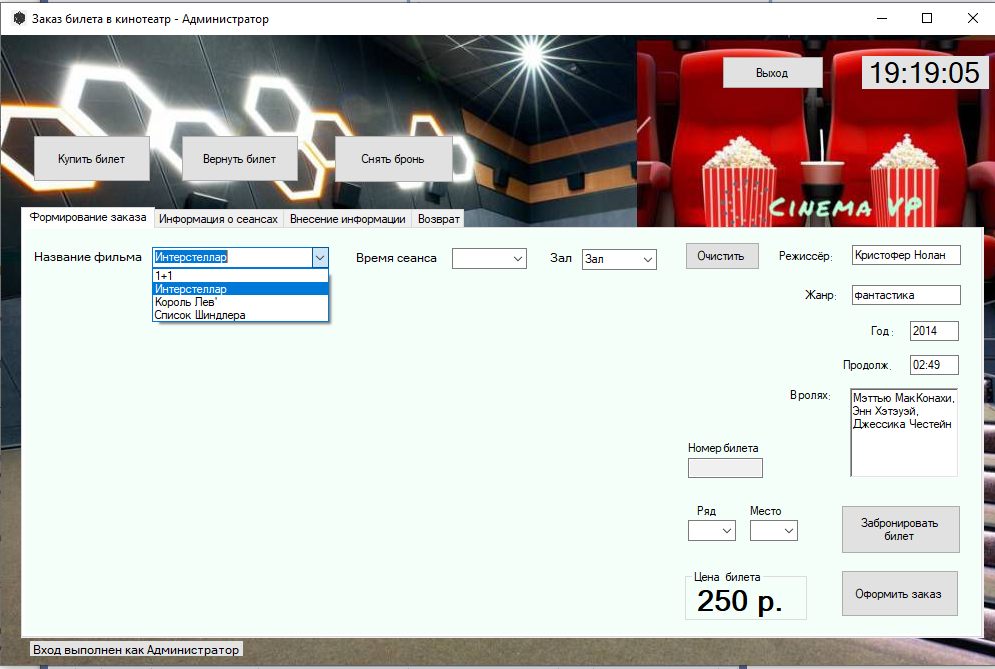


Рисунок 9- выбор фильма для сортировки по его названию

На данной форме есть элемент tabcontrol, которая занимает большую часть формы. Ее основные свойства:

BackColor: MintcCream – цвет фона

Font: Microsoft Sans Serif; 8 pt – шрифт

Size: 963; 431 – размер панели

BorderStyle: none; - отображает сведения о стиле границы элементы управления

Name: tabControl1; - название панели

Modifiers: Public; - модификаторы

На этой панели есть 5 вкладок, на которых располагаются такие элементы управления как кнопки button, combobox, textbox, richtextbox и label. Рассмотрим кратко каждую из элементов. Основные свойства кнопки AllContact:

Button – Кнопка на которую пользователь нажимает мышкой

Combobox – поле для ввода текста с возможностью раскрытия списка подсказок

Textbox – поле для ввода текста пользователем

Richtextbox – поле для ввода- вывода большого количества текста

Label – тестовая надпись на форме

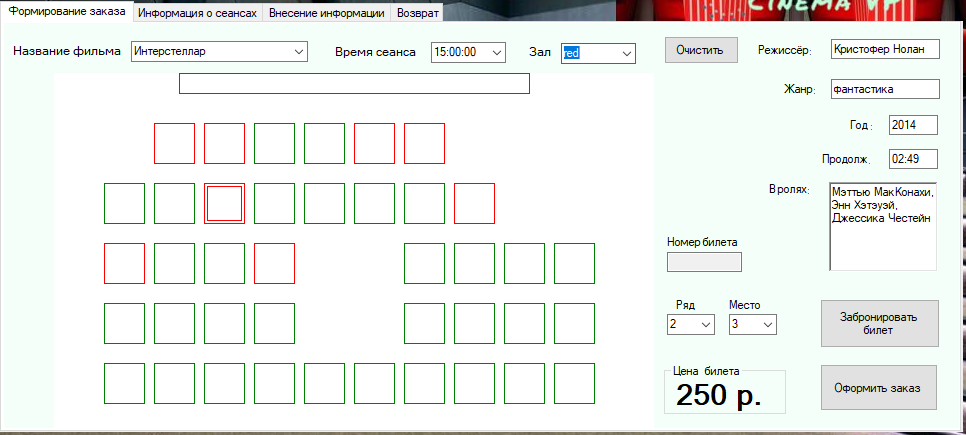


Рисунок 10- Результат работы программы

Алгоритм работы класса databaseAccess: создаем экземпляр базы данных db, который устанавливает связь с базой данных. Далее вызываем метод класса selectTimeByMovieTitle, который можно увидеть на рисунке 7. После этого следует обращение к базе данных с запросами о фильме, доступных сеансах и залах. Чтобы передать информацию, которая хранится в базе данных, нужно чтобы параметры совпадали. Такой алгоритм будет действовать во всех методах.

Далее идёт заполнение comboBox информацией, полученной из базы данных, принимая во внимание, что количество значений будет больше 1, используя цикл while и foreach.

Второй метод класса databaseAccess - GetTickenNummber. Этот метод нужен для определения наличия билета в базе данных с определённым номером. Создаём переменную result для хранения номера билета и присваиваем ей значение 0, что для нас будет индикатором, что билет не найден. Формируем и отправляем запрос в базу данных с параметрами seat, hall и showtime\_id, которые передаются при вызове метода. Если из базы данных вернулся номер билета, то записываем её в переменную result и возвращаем из метода, иначе возвращается 0. На рисунке 11 показан данный метод.

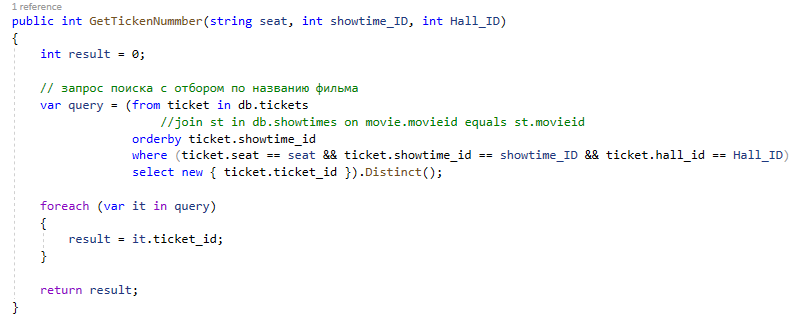


Рисунок 11 – метод для поиска билета

Третий метод класса databaseAccess - DeleteTicket. Как следует из названия, данный метод предназначен для удаления билета из базы данных. Объявляем переменные для хранения всех id базы данных, отправляем запросы в базы данных и получаем необходимые промежуточные параметры для формирования запроса на удаление билета из базы данных. Подготавливаем для удаления из списка занятых мест определенное место, приобретенное покупателем. Далее отправляем запрос на удаление билета в базу данных. Данный метод можно увидеть на рисунке 12. Результат работы этого метода можно увидеть на рисунке 13.

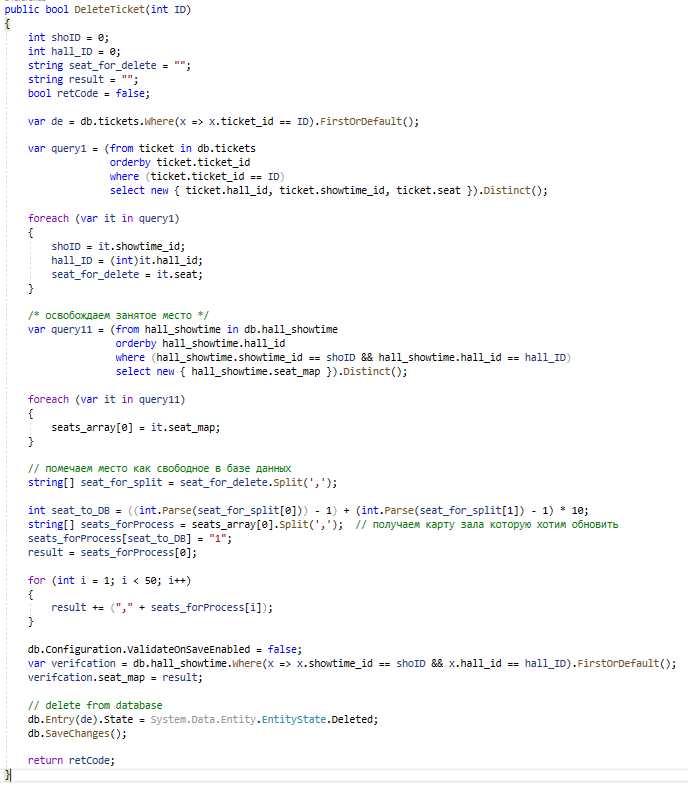


Рисунок 12 – метод DeleteTicket

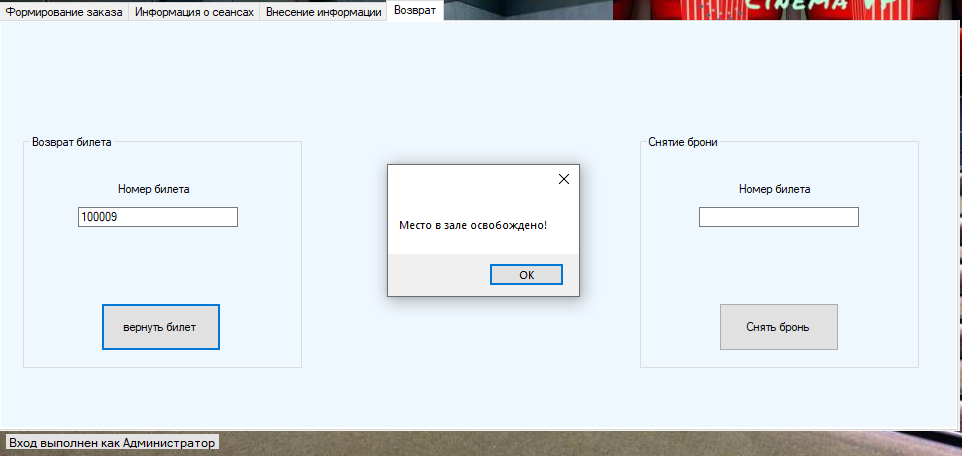


Рисунок 13 –результат работы метода DeleteTicket

Для оформления заказа применяется событие btnPlaceOrder\_Click, которое срабатывает по кнопке “оформить заказ”. Всплывает окно уточнения «Проверьте правильность данных» (Рисунок 14).

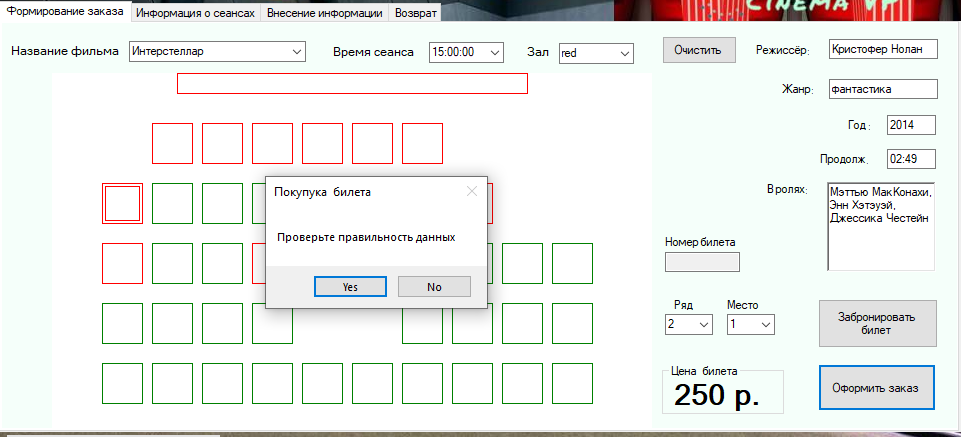


Рисунок 14- окно уточнения

Нажав на кнопку «Yes», можно увидеть успешное добавление в базу данных, которое показано на рисунке 15.

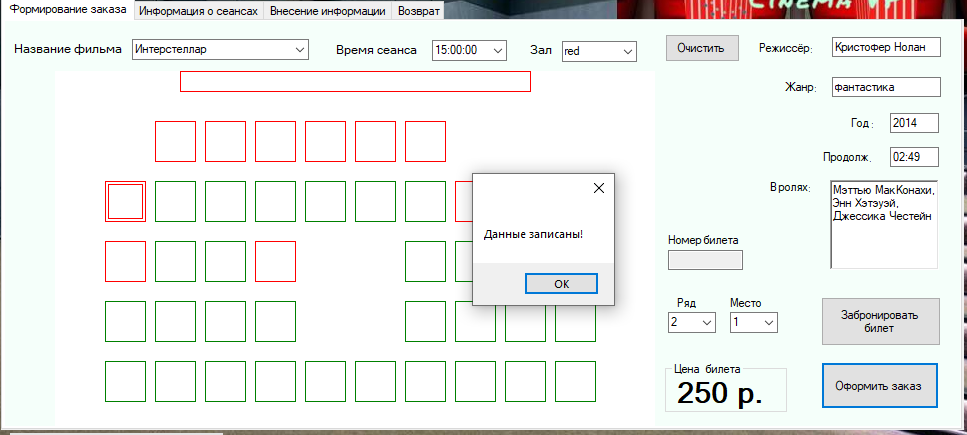


Рисунок 15 – добавление

Далее открывается форма с билетом, на котором указан фильм, зал, место, время, цена и номер билета. Сформированный билет можно увидеть на рисунке 16.

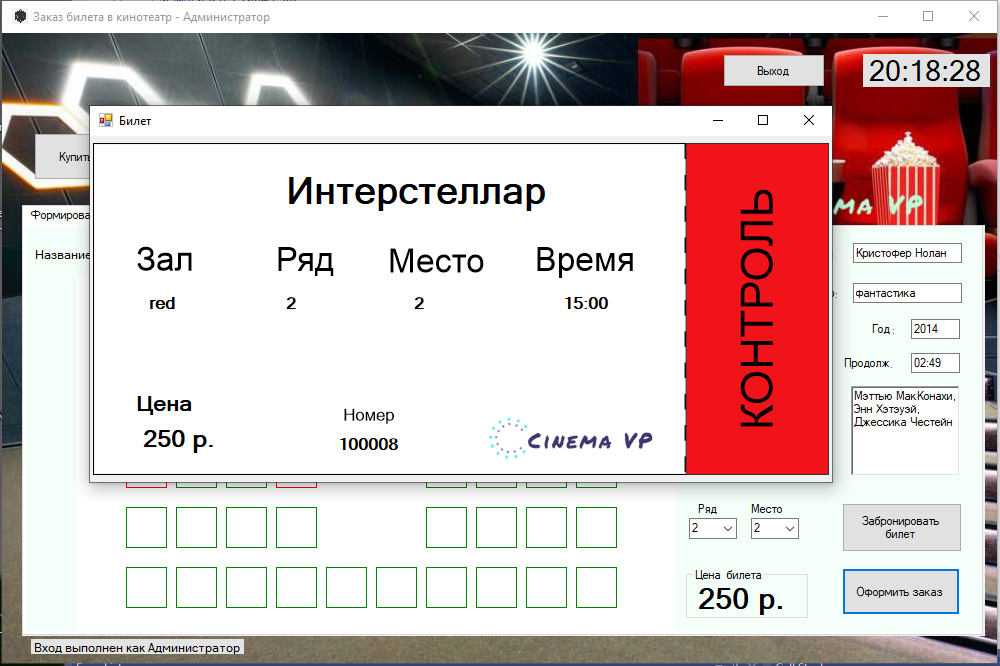


Рисунок 16 – сформированный билет

Для передачи информации в открывающуюся форму с билетом, мы используем класс tickFRM, который содержит такие поля, которые необходимы для отображения картинки билета. Данный класс можно увидеть на рисунке 17.

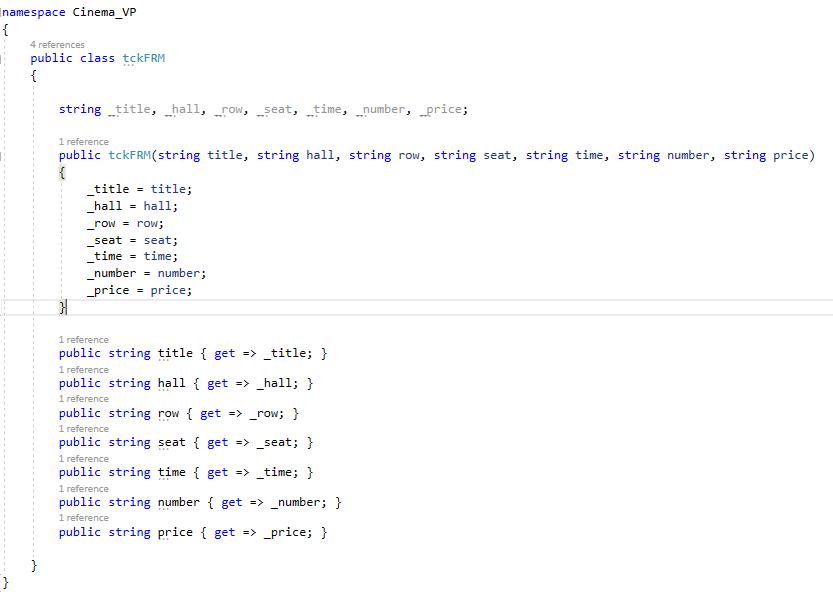


Рисунок 17 – класс tickFRM

Для того, чтобы записать в базу данных новые фильмы и сеансы существует метод addMovie, который вызывается событием btnAddDataTab3\_Click. В метод передаются comboBox с необходимой для записи в бд информацией. Первым делом запрашиваем количество записей в таблице фильмов. Далее проверяем нет ли вводимого фильма в бд. Если такого фильма нет, мы добавляем, а иначе обновляем информацию о сеансах. Записываем данные в таблицу сеансов и залов, в которых идут фильмы. Далее добавляем пустой зал к сеансу и сохраняем изменение в базе данных. Данный метод можно увидеть на рисунке 18,19 и 20.

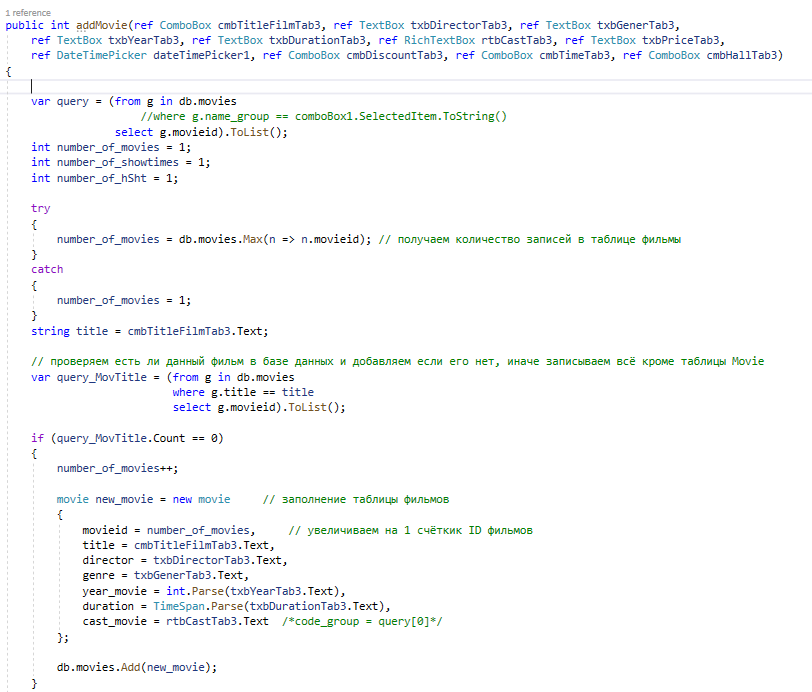


Рисунок 18 – метод addMovie



Рисунок 19 – метод addMovie

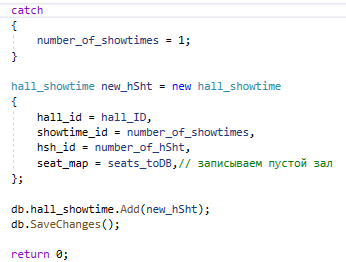


Рисунок 20 – метод addMovie

Так как логин и пароль у нас также находятся в базе данных, то для идентификации пользователя, мы используем метод loginFunc класса databaseAccess. Данный метод принимает строку логина и пароля из textbox формы входа, запрашивает в базе данных существует ли такая комбинация и, если такая комбинация находится в бд, он возвращает нам роль, с которой выполнен вход, в случае неудачи возвращается строка failed. Эти данные используются в последствии для указания роли заголовки окна. Данный метод можно увидеть на рисунке 21.

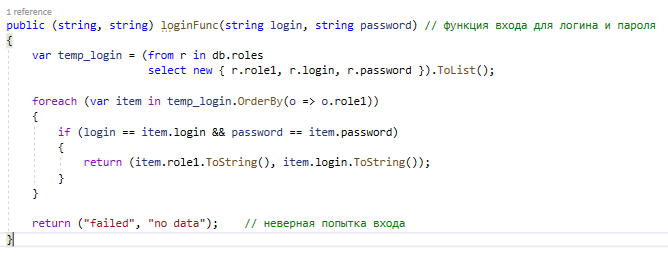


Рисунок 21 – метод идентификации пользователя

ЗАКЛЮЧЕНИЕ

Таким образом, в результате выполнения курсового проектирования, можно сказать о том, что было разработано визуальное приложение «Кинотеатр», а также была подключена и протестирована база данных, созданная в системе PostgreSQL. Поставленные цели и задачи были успешно выполнены.

Система “Кинотеатр” полностью готова к использованию сотрудникам кинотеатра, а также позволяет оперативно вносить и получать необходимую информацию о фильмах, доступ посторонним пользователям закрыт, что является необходимым для оперативной работы сотрудников кинотеатра.

Полученные знания в ходе работы над курсовым проектом явились хорошей практикой для курса визуального программирования на языке Visual Studio и PostgreSQL.

В заключении хочется сказать о том, что данное приложение позволит кинотеатру автоматизировать процесс поиска взаимодействия с клиентами, продавать билеты быстрее, избегая длинных очередей.

СПИСОК ЛИТЕРАТУРЫ

1. "Герберт Шилдт - C# 4.0: полное руководство"– ООО "И.Д. Вильямс", 2011 г. – 294 с.
2. К. Дж. Дейт. SQL и реляционная теория. Как грамотно писать код на SQL: Символ-плюс, 2010 г.– 127 с.
3. С.Макконнелл "Совершенный код. Практическая разработка по разработке программного обеспечения ", 2017 г.– 893 с.
4. Туманов, В.Е. "Основы проектирования реляционных баз данных" Бином 2012 г. - 420 c.
5. Разработка Windows- приложений на основе Visual C#: Ч. А. Кариев – Москва, 2017 г. – 768 с.
6. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов: Алекс Макки – Москва, Вильямс, 2010 г. – 416 с.

ПРИЛОЖЕНИЕ

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace Cinema\_VP

{

public partial class Form1 : Form

{

int couner = 0; //счётчик морганий label

private Bitmap MyImage; // картинка пользователя

private string roleIndex = "null"; // Роль= Администратор, Пользователь, Управляюший

public cinemaEntities2 db = new cinemaEntities2();

public List<role> roleSheet; //список ролей для вхождения в систему

public string[] seats\_array = new string[10]; // обьявление массив мест в зрительном зале

public int seats\_array\_index = 0;

int hall\_ID = 0;

databaseAccess dbAccess = new databaseAccess();

public Form1()

{

InitializeComponent();

ShowMyImage(Properties.Resources.User\_Guest1, 200, 200); // картинка гостя

}

public void btnPlaceOrder\_Click(object sender, EventArgs e)

{

DialogResult dialogResult = MessageBox.Show("Проверьте правильность данных", "Покупука билета", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

seats\_array[0] = dbAccess.addTicket(cmbTitleFilmTab1.Text, cmbTimeTab1.Text, cmbHallTab1.Text, cmbRowTab1.Text, cmbSeatTab1.Text, lblPrice.Text, roleIndex, seats\_array[0]);

clear\_seatMap(tabPage1);

draw\_seatMap(seats\_array[0], tabPage1);

}

else if (dialogResult == DialogResult.No)

{

// do something else

}

}

public void clear\_seatMap(TabPage tabPage)

{

System.Drawing.SolidBrush myBrush = new System.Drawing.SolidBrush(System.Drawing.Color.White); //рисуем белый прямоугольник по всему полю нашей карты

System.Drawing.Graphics formGraphics;

formGraphics = tabPage.CreateGraphics();

formGraphics.FillRectangle(myBrush, new Rectangle(50, 50, 600, 600)); //начало координат(50,50) с размерами 600\*600 пикселей

myBrush.Dispose();

formGraphics.Dispose(); //удаляем обьект рисования

}

/\* принцип кодирования мест: 0-нет места 1-место не занято, 2-место занято, 3- диванчик, 4- диванчик занятый \*/

public void draw\_seatMap(string Map, TabPage Tab\_Page)

{

int i = 0;

try

{

string[] seat = Map.Split(',');

Graphics g = Tab\_Page.CreateGraphics();

g.DrawRectangle(Pens.Red, 175, 50, 350, 20); // рисуем экран зрительного зала

for (int y = 100; y < 400; y += 60) // рисуем места

{

for (int x = 100; x < 600; x += 50)

{

if (seat[i] == "1")

{

g.DrawRectangle(Pens.Green, x, y, 40, 40);

}

if (seat[i] == "2")

{

g.DrawRectangle(Pens.Red, x, y, 40, 40);

}

if (seat[i] == "5") // это место выбрано для оформления билета

{

g.DrawRectangle(Pens.Red, x, y, 40, 40);

g.DrawRectangle(Pens.Red, x + 3, y + 3, 34, 34);

}

if (seat[i] == "3")

{

g.DrawRectangle(Pens.Green, x, y, 90, 40);

x += 50;

i++;

}

if (seat[i] == "4")

{

g.DrawRectangle(Pens.Red, x, y, 90, 40);

x += 50;

i++;

}

if (seat[i] == "6") // выбранное место диванчика для оформления билета

{

g.DrawRectangle(Pens.Red, x, y, 90, 40);

g.DrawRectangle(Pens.Red, x + 3, y + 3, 84, 34);

x += 50;

i++;

}

i++;

}

}

}

catch

{

MessageBox.Show("Сначала выберите фильм и время");

}

}

// бронирование билета

private void btnBookTicketTab1\_Click(object sender, EventArgs e)

{

DialogResult dialogResult = MessageBox.Show("Проверьте правильность данных", "Покупука билета", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

seats\_array[0] = dbAccess.addTicket(cmbTitleFilmTab1.Text, cmbTimeTab1.Text, cmbHallTab1.Text, cmbRowTab1.Text, cmbSeatTab1.Text, "000 p.", roleIndex, seats\_array[0]);

clear\_seatMap(tabPage1);

draw\_seatMap(seats\_array[0], tabPage1);

}

else if (dialogResult == DialogResult.No)

{

// do something else

}

}

private void Form1\_Load(object sender, EventArgs e)

{

// не меньше начального размера

this.MinimumSize = new System.Drawing.Size(this.Width, this.Height);

// не больше размера экрана для формы

this.MaximumSize = new System.Drawing.Size(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);

load\_data();

}

/\* функция добавления распределения мест в зале \*/

void Add\_to\_seats(string seats)

{

seats\_array[seats\_array\_index] = seats;

seats\_array\_index++;

}

/\* заполнение всех compbobox и других полей \*/

void load\_data()

{

var hall\_forList = (from s in db.halls // Подготовка свободных мест в зале

orderby s.hall\_name

select new { s.hall\_name, s.hall\_id, s.seats }).Distinct();

cmbHallTab1.Items.Clear();

cmbHallTab2.Items.Clear();

cmbHallTab3.Items.Clear();

tbxTicketID.Clear();

seats\_array\_index = 0;

Array.Clear(seats\_array, 0, seats\_array.Length);

foreach (var it in hall\_forList)

{

cmbHallTab1.Items.Add(it.hall\_name);

cmbHallTab2.Items.Add(it.hall\_name);

cmbHallTab3.Items.Add(it.hall\_name);

Add\_to\_seats(it.seats);

}

cmbHallTab2.DrawMode = DrawMode.OwnerDrawFixed;

var movie\_forList = (from s in db.movies // фильмы для списка

orderby s.title

select s.title).Distinct();

foreach (string it in movie\_forList)

{

cmbTitleFilmTab1.Items.Add(it);

cmbTitleFilmTab2.Items.Add(it);

cmbTitleFilmTab3.Items.Add(it);

}

var time\_forList = (from s in db.showtimes // время сеансов для списка

orderby s.time

select s.time).Distinct();

foreach (var it in time\_forList)

{

cmbTimeTab1.Items.Add(it.ToString());

cmbTimeTab2.Items.Add(it.ToString());

}

}

private void timer1\_Tick(object sender, EventArgs e)

{

label1.Text = DateTime.Now.ToString("HH:mm:ss");

}

private void btnBuyTicket\_Click(object sender, EventArgs e) // кнопка купить билет

{

if (roleIndex == "Администратор" || roleIndex == "Пользователь")

{

tabControl1.SelectedTab = tabPage1;

}

else

{

loginRequired();

}

}

private void btnRetTicket\_Click(object sender, EventArgs e) // кнопка вернуть билет

{

if (roleIndex == "Администратор" || roleIndex == "Пользователь")

{

tabControl1.SelectedTab = tabPage5;

}

else

{

loginRequired();

}

}

private void btnBookRet\_Click(object sender, EventArgs e) // кнопка бронировать билет

{

if (roleIndex == "Администратор" || roleIndex == "Пользователь")

{

tabControl1.SelectedTab = tabPage5;

}

else

{

loginRequired();

}

}

void loginRequired() // функция для моргания при не выолненном входе

{

tabControl1.SelectedTab = tabPage4;

Timer\_errUser\_event.Enabled = true;

}

private void Timer\_errUser\_event\_Tick(object sender, EventArgs e) //моргание label при необходимости войти в систему

{

if (this.label14.ForeColor == SystemColors.ControlText)

{

label14.ForeColor = System.Drawing.Color.Red;

tabPage4.ForeColor = System.Drawing.Color.Red;

}

else

{

label14.ForeColor = SystemColors.ControlText;

tabPage4.ForeColor = SystemColors.ControlText;

}

if (couner++ == 5) //кол-во морганий label краным цветом

{

Timer\_errUser\_event.Enabled = false;

couner = 0;

}

}

private void btnExit\_Click(object sender, EventArgs e) // кнопка выхода

{

}

void loginFormLock(bool action) // блокировка формы входа

{

if (action == true)

{

textBox\_Login.Enabled = false;

textBox\_Pass.Enabled = false;

btnLoginEnter.Enabled = false;

}

else

{

textBox\_Login.Enabled = true;

textBox\_Pass.Enabled = true;

btnLoginEnter.Enabled = true;

textBox\_Login.Clear();

textBox\_Pass.Clear();

}

}

public (string, string) loginFunc() // функция входа для логина и пароля

{

var temp\_login = (from r in db.roles

select new { r.role1, r.login, r.password }).ToList();

foreach (var item in temp\_login.OrderBy(o => o.role1))

{

if (textBox\_Login.Text == item.login && textBox\_Pass.Text == item.password)

{

return (item.role1.ToString(), item.login.ToString());

}

}

return ("failed", "no data"); // неверная попытка входа

}

public void btnLoginEnter\_Click(object sender, EventArgs e) // кнопка входа

{

var values = dbAccess.loginFunc(textBox\_Login.Text, textBox\_Pass.Text);

bool result = false;

login lgn = new login();

result = lgn.checkUserInput(textBox\_Login.Text, textBox\_Pass.Text);

if (values.Item2 == "admin")

{

tabPage1.Enabled = true;

tabPage3.Enabled = true;

tabPage5.Enabled = true;

label14.Text = "Вход выполнен как " + values.Item1;

this.Text = "Заказ билета в кинотеатр - " + values.Item1;

tabControl1.SelectedTab = tabPage1;

ShowMyImage(Properties.Resources.User\_Admin, 200, 200);

loginFormLock(true);

roleIndex = "Администратор";

this.tabControl1.Controls.Remove(this.tabPage4);

btnExitMainFrm.Visible = true;

}

else if (values.Item2 == "user")

{

tabPage1.Enabled = true;

tabPage5.Enabled = true;

label14.Text = "Вход выполнен как " + values.Item1 + " Внесение данных невозможно";

this.Text = "Заказ билета в кинотеатр - " + values.Item1;

tabControl1.SelectedTab = tabPage1;

ShowMyImage(Properties.Resources.User\_User, 200, 200);

loginFormLock(true);

roleIndex = "Пользователь";

this.tabControl1.Controls.Remove(this.tabPage4);

btnExitMainFrm.Visible = true;

}

else if (values.Item2 == "user1")

{

tabPage3.Enabled = true;

label14.Text = "Вход выполнен как " + values.Item1 + " Доступно только внесение данных";

this.Text = "Заказ билета в кинотеатр - " + values.Item1;

tabControl1.SelectedTab = tabPage3;

ShowMyImage(Properties.Resources.User\_User1, 200, 200);

loginFormLock(true);

roleIndex = "Управляющий";

this.tabControl1.Controls.Remove(this.tabPage4);

btnExitMainFrm.Visible = true;

}

else

{

label\_wrongLogin.Visible = true;

textBox\_Login.BackColor = System.Drawing.Color.Red;

textBox\_Pass.BackColor = System.Drawing.Color.Red;

}

}

public class login

{

public login()

{

}

public bool checkUserInput (string login, string pass)

{

if (login == "admin")

{

return true;

}

return false;

}

}

private void textBox\_Login\_TextChanged(object sender, EventArgs e) //очищает поле логина

{

if (textBox\_Login.BackColor != SystemColors.Window)

{

clear\_userInput();

}

}

private void textBox\_Pass\_TextChanged(object sender, EventArgs e) //очищает поле пароля

{

if (textBox\_Pass.BackColor != SystemColors.Window)

{

textBox\_Pass.Clear();

label\_wrongLogin.Visible = false;

textBox\_Pass.BackColor = SystemColors.Window;

textBox\_Login.BackColor = SystemColors.Window;

}

}

public void clear\_userInput() //очищает всё

{

textBox\_Login.Clear();

textBox\_Pass.Clear();

label\_wrongLogin.Visible = false;

textBox\_Pass.BackColor = SystemColors.Window;

textBox\_Login.BackColor = SystemColors.Window;

}

// функция для смены картинки

public void ShowMyImage(Image fileToDisplay, int xSize, int ySize)

{

// выбор картинки при входк в систему для отображения роли

if (MyImage != null)

{

MyImage.Dispose();

}

// расширение картинки на размер pictureBox

pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;

MyImage = new Bitmap(fileToDisplay);

pictureBox1.ClientSize = new Size(xSize, ySize);

pictureBox1.Image = (Image)MyImage;

}

/\* выбор фильма на 2 вкладке\*/

private void cmbTitleFilmTab2\_SelectedIndexChanged(object sender, EventArgs e)

{

cmbTimeTab2.Items.Clear();

cmbHallTab2.Items.Clear();

dbAccess.selectTimeByMovieTitle(ref cmbTimeTab2, ref cmbTitleFilmTab2, ref cmbHallTab2, ref txbDirectorTab1, ref txbYearTab1, ref txbGenerTab1, ref txbDurationTab1, ref rtbCastTab1, ref lblPrice );

}

// событие выбора времени

private void cmbTimeTab2\_SelectedIndexChanged(object sender, EventArgs e)

{

//int[] sh\_ID = new int[10];

//int i = 0;

//int temp = 0;

//// запрос поиска с отбором по названию фильма

//var query = (from movie in db.movies

// join st in db.showtimes on movie.movieid equals st.movieid

// orderby movie.title

// where (st.time.ToString() == cmbTimeTab2.Text.ToString())

// select new { movie.title, movie.duration, movie.director, movie.genre, movie.cast\_movie, st.time, st.showtime\_id }).Distinct();

//cmbTitleFilmTab2.Items.Clear();

//cmbHallTab2.Items.Clear();

//tbxTicketID.Clear();

//// заполнение combobox о фильме

//foreach (var item in query.OrderBy(o => o.title))

//{

// i++;

// sh\_ID[i] = item.showtime\_id;

// cmbTitleFilmTab2.Items.Add(item.title.ToString());

//}

//while (i > 0)

//{

// temp = sh\_ID[i];

// var query1 = (from g in db.hall\_showtime

// where g.showtime\_id == temp

// select g.hall\_id).Distinct();

// var query2 = (from g in db.halls

// where g.hall\_id == query1.FirstOrDefault()

// select g.hall\_name).Distinct();

// foreach (string item in query2)

// {

// cmbHallTab2.Items.Add(item);

// }

// i--;

//}

}

// Выбор фильма на 1 вкладке

private void cmbTitleFilmTab1\_SelectedIndexChanged(object sender, EventArgs e)

{

cmbTimeTab1.Items.Clear();

cmbHallTab1.Items.Clear();

tbxTicketID.Clear();

dbAccess.selectTimeByMovieTitle(ref cmbTimeTab1, ref cmbTitleFilmTab1, ref cmbHallTab1, ref txbDirectorTab1, ref txbYearTab1, ref txbGenerTab1, ref txbDurationTab1, ref rtbCastTab1, ref lblPrice);

}

/\* выбор и сортировка по времени \*/

private void cmbTimeTab1\_SelectedIndexChanged(object sender, EventArgs e)

{

tbxTicketID.Clear();

}

/\* кнопка очистить на 2 вкладке \*/

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

{

cmbTitleFilmTab2.Items.Clear();

cmbHallTab2.Items.Clear();

cmbTimeTab2.Items.Clear();

load\_data();

}

/\*кнопка очистить на 1 вкладке \*/

private void btnClearTab1\_Click(object sender, EventArgs e)

{

cmbTitleFilmTab1.Items.Clear();

cmbHallTab1.Items.Clear();

cmbTimeTab1.Items.Clear();

load\_data();

}

/\* активация кнопки Enter при вводе пароля \*/

private void textBox\_Pass\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == Convert.ToChar(Keys.Enter))

{

btnLoginEnter\_Click(null, null);

}

}

/\* активация кнопки Enter при вводе логина \*/

private void textBox\_Login\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == Convert.ToChar(Keys.Enter))

{

btnLoginEnter\_Click(null, null);

}

}

/\* кнопка внесения данных на 3 вкладке \*/

private void btnAddDataTab3\_Click(object sender, EventArgs e)

{

if (txbDirectorTab1.TextLength == 0 || txbGenerTab1.TextLength == 0)

{

MessageBox.Show("Поле оставлено пустым, данные не могут быть добавлены");

}

else

{

dbAccess.addMovie(ref cmbTitleFilmTab3, ref txbDirectorTab3, ref txbGenerTab3, ref txbYearTab3, ref txbDurationTab3, ref rtbCastTab3,

ref txbPriceTab3, ref dateTimePicker1, ref cmbDiscountTab3, ref cmbTimeTab3, ref cmbHallTab3);

MessageBox.Show("Данные записаны!");

}

load\_data(); // обновляем данные в форме

}

// выбор фильма на 3 вкладке

private void cmbTitleFilmTab3\_SelectedIndexChanged(object sender, EventArgs e)

{

int[] sh\_ID = new int[10];

int i = 0;

int temp = 0;

// запрос поиска с отбором по названию фильма

var query = (from movie in db.movies

join st in db.showtimes on movie.movieid equals st.movieid

orderby movie.title

where (movie.title == cmbTitleFilmTab3.Text.ToString())

select new { movie.title, movie.duration, movie.director, movie.genre, movie.cast\_movie, st.time, st.date, st.showtime\_id, st.price, movie.year\_movie }).Distinct();

cmbTimeTab3.Items.Clear();

cmbHallTab3.Items.Clear();

// заполнение combobox о фильме

foreach (var item in query.OrderBy(o => o.title))

{

i++;

sh\_ID[i] = item.showtime\_id;

cmbTimeTab3.Items.Add(item.time.ToString());

txbDirectorTab3.Text = item.director;

txbGenerTab3.Text = item.genre;

txbYearTab3.Text = item.year\_movie.ToString();

txbDurationTab3.Text = item.duration.ToString().Remove(5, 3);

rtbCastTab3.Text = item.cast\_movie;

txbPriceTab3.Text = item.price.ToString();

dateTimePicker1.Value = (DateTime)item.date;

}

while (i > 0)

{

temp = sh\_ID[i];

var query1 = (from g in db.hall\_showtime

where g.showtime\_id == temp

select g.hall\_id).Distinct();

var query2 = (from g in db.halls

where g.hall\_id == query1.FirstOrDefault()

select g.hall\_name).Distinct();

foreach (string item in query2)

{

cmbHallTab3.Items.Add(item);

}

i--;

}

}

/\* обработка события DrawItem для combobox для возможности просмотра карты при движении мыши без щелчка \*/

private void cmbHallTab2\_DrawItem(object sender, DrawItemEventArgs e)

{

int index;

// Рисуем прямоугольник выделения на combobox для каждого элемента

e.DrawBackground();

// отрисовываем чёрным шрифтом текст шрифт на основании

e.Graphics.DrawString(cmbHallTab2.Items[e.Index].ToString(), e.Font, System.Drawing.Brushes.Black, new RectangleF(e.Bounds.X, e.Bounds.Y, e.Bounds.Width, e.Bounds.Height));

// переносим фокус на выбранный элемент

e.DrawFocusRectangle();

if ((e.State & DrawItemState.Selected) != 0)

{

index = cmbHallTab2.SelectedIndex;

clear\_seatMap(tabPage2);

draw\_seatMap(seats\_array[index], tabPage2);

}

}

// выбор зала на 1 вкладке

private void cmbHallTab1\_SelectedIndexChanged(object sender, EventArgs e)

{

// берём название фильма и время чтобы получить movie\_ID чтобы потом получить showtime\_ID используя время

int movID = 0;

int shoID = 0;

//запрос movie ID по его названию

var movieID = (from movie in db.movies

where (movie.title == cmbTitleFilmTab1.Text)

select new { movie.movieid }).Distinct();

foreach (var it in movieID)

{

movID = it.movieid;

}

// запрос showtimeID по фильму

var query = (from showtime in db.showtimes

orderby showtime.movieid

where (showtime.time.ToString() == cmbTimeTab1.Text.ToString() && showtime.movieid == movID)

select new {showtime.showtime\_id}).Distinct();

foreach (var it in query)

{

shoID = it.showtime\_id;

}

// получаем hall\_id по названию зала

var query\_hall = (from g in db.halls

where g.hall\_name == cmbHallTab1.Text

select g.hall\_id).Distinct();

hall\_ID = query\_hall.First();

var query11 = (from hall\_showtime in db.hall\_showtime

orderby hall\_showtime.hall\_id

where (hall\_showtime.showtime\_id == shoID && hall\_showtime.hall\_id == hall\_ID)

select new { hall\_showtime.seat\_map }).Distinct();

seats\_array\_index = 0;

Array.Clear(seats\_array, 0, seats\_array.Length);

foreach (var it in query11)

{

seats\_array[0] = it.seat\_map;

}

clear\_seatMap(tabPage1);

draw\_seatMap(seats\_array[0], tabPage1);

tbxTicketID.Clear();

}

/\* обработка события DrawItem для combobox для возможности просмотра карты при движении мыши без щелчка для 2 вкладке \*/

private void cmbHallTab1\_DrawItem(object sender, DrawItemEventArgs e)

{

int index;

e.DrawBackground();

e.Graphics.DrawString(cmbHallTab1.Items[e.Index].ToString(), e.Font, System.Drawing.Brushes.Black, new RectangleF(e.Bounds.X, e.Bounds.Y, e.Bounds.Width, e.Bounds.Height));

e.DrawFocusRectangle();

if ((e.State & DrawItemState.Selected) != 0)

{

index = cmbHallTab1.SelectedIndex;

clear\_seatMap(tabPage1);

draw\_seatMap(seats\_array[index], tabPage1);

}

}

public void selectTimeByMovieTitle(ComboBox cmbTime, ComboBox cmbHall, string Title)

{

int[] sh\_ID = new int[10];

int i = 0;

int temp = 0;

// запрос поиска с отбором по названию фильма

var query = (from movie in db.movies

join st in db.showtimes on movie.movieid equals st.movieid

orderby movie.title

where (movie.title == Title)

select new { movie.title, movie.duration, movie.director, movie.genre, movie.cast\_movie, st.time, st.showtime\_id, st.price, movie.year\_movie }).Distinct();

// заполнение combobox о фильме

foreach (var item in query.OrderBy(o => o.title))

{

i++;

sh\_ID[i] = item.showtime\_id;

cmbTime.Items.Add(item.time.ToString());

txbDirectorTab1.Text = item.director;

txbGenerTab1.Text = item.genre;

txbYearTab1.Text = item.year\_movie.ToString();

txbDurationTab1.Text = item.duration.ToString().Remove(5, 3);

rtbCastTab1.Text = item.cast\_movie;

lblPrice.Text = item.price.ToString() + " p.";

}

while (i > 0)

{

temp = sh\_ID[i];

var query1 = (from g in db.hall\_showtime

where g.showtime\_id == temp

select g.hall\_id).Distinct();

var query2 = (from g in db.halls

where g.hall\_id == query1.FirstOrDefault()

orderby g.hall\_name

select new { g.hall\_name }).Distinct();

foreach (var item in query2)

{

// если в combobox уже внесён этот зал то мы его не добавляем

if (!cmbHall.Items.Contains(item.hall\_name))

{

cmbHall.Items.Add(item.hall\_name);

}

}

i--;

}

}

// кнопка возврата билета

private void btnReturnTab5\_Click(object sender, EventArgs e)

{

int ID = int.Parse(tbxTicketForReturn.Text);

// проверяем существует ли билет

if (dbAccess.validateTicket(ID) == true)

{

DialogResult dialogResult = MessageBox.Show("Вы уверены что хотите удалить билет", "Удаление билета", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

dbAccess.DeleteTicket(ID);

MessageBox.Show("Место в зале освобождено!");

}

else if (dialogResult == DialogResult.No)

{

}

}

else

{

MessageBox.Show("Билет не найден!");

}

}

/\* кнопка отмены бронирования \*/

private void btnUnBookTab5\_Click(object sender, EventArgs e)

{

int ID = int.Parse(tbxTicketForUnBook.Text);

// проверяем существует ли билет

if (dbAccess.validateTicket(ID) == true )

{

if (dbAccess.isTicketBooked(ID) == true)

{

DialogResult dialogResult = MessageBox.Show("Вы уверены что хотите снять бронь", "Снятие брони", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

dbAccess.DeleteTicket(ID);

MessageBox.Show("Место в зале освобождено!");

}

else if (dialogResult == DialogResult.No)

{

}

}

else

{

MessageBox.Show("Этот билет нужно возвращать как купленный");

}

}

else

{

MessageBox.Show("Билет не найден!");

}

}

// обработка события щелчка мыши по карте распределения мест в зале, выбор места

private void tabPage1\_MouseDown(object sender, MouseEventArgs e)

{

int x, y, index, seat\_to\_change;

string result = "";

x = e.X;

y = e.Y;

if ((x > 100 && x < 600) && (y > 100 && y < 380))

{

x = ((x - 100) / 52);

y = ((y - 100) / 60);

cmbSeatTab1.Text = (x + 1).ToString();

cmbRowTab1.Text = (y + 1).ToString();

seat\_to\_change = x + (y \* 10);

index = cmbHallTab1.SelectedIndex;

clear\_seatMap(tabPage1);

string temp = seats\_array[0];

string[] numbers = temp.Split(',');

if (numbers[seat\_to\_change] == "1") // выбираем только то место которое существует

{

numbers[seat\_to\_change] = "5";

result = numbers[0];

}

if (numbers[seat\_to\_change] == "2") // если место занято запрашиваем и выводим Ticket\_ID

{

string seat\_to\_funnc = (x + 1).ToString() + "," + (y + 1).ToString();

tbxTicketID.Text = dbAccess.GetTicketNummber(seat\_to\_funnc, hall\_ID, cmbTitleFilmTab1.Text, cmbTimeTab1.Text).ToString();

}

else

{

tbxTicketID.Clear();

}

for (int i = 1; i < 50; i++)

{

result += ("," + numbers[i]);

}

draw\_seatMap(result, tabPage1);

}

}

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

{

tabPage1.Enabled = false;

tabPage3.Enabled = false;

tabPage5.Enabled = false;

btnLoginEnter.Enabled = true;

label14.Text = "Вход не выполнен - возможен только промотр";

ShowMyImage(Properties.Resources.User\_Guest1, 200, 200);

clear\_userInput();

this.Text = "Заказ билета в кинотеатр - Вход не выполнен";

loginFormLock(false);

roleIndex = "null";

this.tabControl1.Controls.Add(this.tabPage4);

tabControl1.SelectedTab = tabPage4;

btnExitMainFrm.Visible = false;

}

private void cmbHallTab2\_SelectedIndexChanged(object sender, EventArgs e)

{

// берём название фильма и время чтобы получить movie\_ID чтобы потом получить showtime\_ID используя время

int movID = 0;

int shoID = 0;

//запрос movie ID по его названию

var movieID = (from movie in db.movies

where (movie.title == cmbTitleFilmTab2.Text)

select new { movie.movieid }).Distinct();

foreach (var it in movieID)

{

movID = it.movieid;

}

// запрос showtimeID по фильму

var query = (from showtime in db.showtimes

orderby showtime.movieid

where (showtime.time.ToString() == cmbTimeTab2.Text.ToString() && showtime.movieid == movID)

select new { showtime.showtime\_id }).Distinct();

foreach (var it in query)

{

shoID = it.showtime\_id;

}

// получаем hall\_id по названию зала

var query\_hall = (from g in db.halls

where g.hall\_name == cmbHallTab2.Text

select g.hall\_id).Distinct();

hall\_ID = query\_hall.First();

var query11 = (from hall\_showtime in db.hall\_showtime

orderby hall\_showtime.hall\_id

where (hall\_showtime.showtime\_id == shoID && hall\_showtime.hall\_id == hall\_ID)

select new { hall\_showtime.seat\_map }).Distinct();

seats\_array\_index = 0;

Array.Clear(seats\_array, 0, seats\_array.Length);

foreach (var it in query11)

{

seats\_array[0] = it.seat\_map;

}

clear\_seatMap(tabPage2);

draw\_seatMap(seats\_array[0], tabPage2);

tbxTicketID.Clear();

}

private void tabControl1\_Selected(object sender, TabControlEventArgs e)

{

cmbTitleFilmTab1.Items.Clear();

cmbHallTab1.Items.Clear();

cmbTimeTab1.Items.Clear();

cmbTitleFilmTab2.Items.Clear();

cmbHallTab2.Items.Clear();

cmbTimeTab2.Items.Clear();

cmbTitleFilmTab3.Items.Clear();

load\_data();

}

}

public class databaseAccess

{

public cinemaEntities2 db = new cinemaEntities2();

public string[] seats\_array = new string[10]; // обьявление массива мест в зрительном зале

public databaseAccess()

{

}

public string addTicket(string Title, string Time, string Hall, string Row, string Seat, string Price, string RoleIndex, string seats\_array)

{

int movID = 0;

int shoID = 0;

string result = "";

//запрос movie ID по его названию

var movieID = (from movie in db.movies

where (movie.title == Title)

select new { movie.movieid }).Distinct();

foreach (var it in movieID)

{

movID = it.movieid;

}

// запрос showtimeID по фильму

var query = (from showtime in db.showtimes

orderby showtime.movieid

where (showtime.time.ToString() == Time.ToString() && showtime.movieid == movID)

select new { showtime.showtime\_id }).Distinct();

foreach (var it in query)

{

shoID = it.showtime\_id;

}

int number\_of\_tickets = db.tickets.Max(n => n.ticket\_id) + 1; // ищем макс ID билета из существ., чтобы записать следующий билет

//получаем hall\_ID из названия зала

var query\_hall = (from g in db.halls

where g.hall\_name == Hall

select g.hall\_id).Distinct();

int hall\_ID = query\_hall.First();

string[] seat\_for\_split = new string[2];

seat\_for\_split[0] = Row;

seat\_for\_split[1] = Seat;

int seat\_to\_DB = ((int.Parse(Seat)) - 1) + (int.Parse(Row) - 1) \* 10;

string[] seats\_forProcess = seats\_array.Split(','); // получаем карту зала которую хотим обновить

seats\_forProcess[seat\_to\_DB] = "2";

result = seats\_forProcess[0];

for (int i = 1; i < 50; i++)

{

result += ("," + seats\_forProcess[i]);

}

// запись билета в бд

ticket new\_ticket = new ticket

{

ticket\_id = number\_of\_tickets,

price = int.Parse(Price.Remove(3, 3)),

seat = (Seat + "," + Row),

showtime\_id = shoID,

hall\_id = hall\_ID,

role = RoleIndex,

};

db.tickets.Add(new\_ticket);

// запись занятого места

db.Configuration.ValidateOnSaveEnabled = false;

var verifcation = db.hall\_showtime.Where(x => x.showtime\_id == shoID && x.hall\_id == hall\_ID).FirstOrDefault();

verifcation.seat\_map = result;

db.SaveChanges();

seats\_array = result;

MessageBox.Show("Данные записаны!");

tckFRM tckFRM = new tckFRM(Title, Hall, Row, Seat, Time, number\_of\_tickets.ToString(), Price);

TicketDisplay frm\_ttd = new TicketDisplay(tckFRM);

frm\_ttd.Show();

return seats\_array; // возвращаем 0 если процедура не выолнена успешно, 1 если успешно

}

public bool DeleteTicket(int ID)

{

int shoID = 0;

int hall\_ID = 0;

string seat\_for\_delete = "";

string result = "";

bool retCode = false;

var de = db.tickets.Where(x => x.ticket\_id == ID).FirstOrDefault();

var query1 = (from ticket in db.tickets

orderby ticket.ticket\_id

where (ticket.ticket\_id == ID)

select new { ticket.hall\_id, ticket.showtime\_id, ticket.seat }).Distinct();

foreach (var it in query1)

{

shoID = it.showtime\_id;

hall\_ID = (int)it.hall\_id;

seat\_for\_delete = it.seat;

}

/\* освобождаем занятое место \*/

var query11 = (from hall\_showtime in db.hall\_showtime

orderby hall\_showtime.hall\_id

where (hall\_showtime.showtime\_id == shoID && hall\_showtime.hall\_id == hall\_ID)

select new { hall\_showtime.seat\_map }).Distinct();

foreach (var it in query11)

{

seats\_array[0] = it.seat\_map;

}

// помечаем место как свободное в базе данных

string[] seat\_for\_split = seat\_for\_delete.Split(',');

int seat\_to\_DB = ((int.Parse(seat\_for\_split[0])) - 1) + (int.Parse(seat\_for\_split[1]) - 1) \* 10;

string[] seats\_forProcess = seats\_array[0].Split(','); // получаем карту зала которую хотим обновить

seats\_forProcess[seat\_to\_DB] = "1";

result = seats\_forProcess[0];

for (int i = 1; i < 50; i++)

{

result += ("," + seats\_forProcess[i]);

}

db.Configuration.ValidateOnSaveEnabled = false;

var verifcation = db.hall\_showtime.Where(x => x.showtime\_id == shoID && x.hall\_id == hall\_ID).FirstOrDefault();

verifcation.seat\_map = result;

// удаляем билет с базы данных

db.Entry(de).State = System.Data.Entity.EntityState.Deleted;

db.SaveChanges();

return retCode;

}

public int GetTicketNummber(string seat, int Hall\_ID, string Movietitle, string MovieTime)

{

int result = 0;

int movID = 0;

int shoID = 0;

// подготавливаем Showtime ID и hall\_ID по имени зала

var movieID = (from movie in db.movies

where (movie.title == Movietitle)

select new { movie.movieid }).Distinct();

foreach (var it in movieID)

{

movID = it.movieid;

}

// запрос showtimeID по фильму

var query = (from showtime in db.showtimes

orderby showtime.movieid

where (showtime.time.ToString() == MovieTime && showtime.movieid == movID)

select new { showtime.showtime\_id }).Distinct();

foreach (var it in query)

{

shoID = it.showtime\_id;

}

// запрос поиска с отбором по названию фильма

var query1 = (from ticket in db.tickets

orderby ticket.showtime\_id

where (ticket.seat == seat && ticket.showtime\_id == shoID && ticket.hall\_id == Hall\_ID)

select new { ticket.ticket\_id }).Distinct();

foreach (var it in query1)

{

result = it.ticket\_id;

}

return result;

}

public bool validateTicket(int ID)

{

bool retCode = false;

var de = db.tickets.Where(x => x.ticket\_id == ID).FirstOrDefault();

if (de != null)

{

retCode = true;

}

return retCode;

}

public bool isTicketBooked (int ID)

{

bool result = false;

var query1 = (from ticket in db.tickets

where ticket.ticket\_id == ID

select ticket.price);

if (query1.First() == 0)

{

result = true;

}

return result;

}

public int addMovie(ref ComboBox cmbTitleFilmTab3, ref TextBox txbDirectorTab3, ref TextBox txbGenerTab3,

ref TextBox txbYearTab3, ref TextBox txbDurationTab3, ref RichTextBox rtbCastTab3, ref TextBox txbPriceTab3,

ref DateTimePicker dateTimePicker1, ref ComboBox cmbDiscountTab3, ref ComboBox cmbTimeTab3, ref ComboBox cmbHallTab3)

{

var query = (from g in db.movies

select g.movieid).ToList();

int number\_of\_movies = 1;

int number\_of\_showtimes = 1;

int number\_of\_hSht = 1;

try

{

number\_of\_movies = db.movies.Max(n => n.movieid); // получаем количество записей в таблице фильмы

}

catch

{

number\_of\_movies = 1;

}

string title = cmbTitleFilmTab3.Text;

// проверяем есть ли данный фильм в базе данных и добавляем если его нет, иначе записываем всё кроме таблицы Movie

var query\_MovTitle = (from g in db.movies

where g.title == title

select g.movieid).ToList();

if (query\_MovTitle.Count == 0)

{

number\_of\_movies++;

movie new\_movie = new movie // заполнение таблицы фильмов

{

movieid = number\_of\_movies, // увеличиваем на 1 счёткик ID фильмов

title = cmbTitleFilmTab3.Text,

director = txbDirectorTab3.Text,

genre = txbGenerTab3.Text,

year\_movie = int.Parse(txbYearTab3.Text),

duration = TimeSpan.Parse(txbDurationTab3.Text),

cast\_movie = rtbCastTab3.Text

};

db.movies.Add(new\_movie);

}

else

{

number\_of\_movies = query\_MovTitle.First();

}

try

{

number\_of\_showtimes = db.showtimes.Max(n => n.showtime\_id) + 1;

}

catch

{

number\_of\_showtimes = 1;

}

showtime new\_showtime = new showtime

{

showtime\_id = number\_of\_showtimes,

movieid = number\_of\_movies,

price = (int)(float.Parse(txbPriceTab3.Text) \* ((100 - float.Parse(cmbDiscountTab3.Text.Remove(2, 1))) / 100)),

date = dateTimePicker1.Value,

time = TimeSpan.Parse(cmbTimeTab3.Text)

};

string hall = cmbHallTab3.Text;

var query\_hall = (from g in db.halls

where g.hall\_name == hall

select g.hall\_id).Distinct();

int hall\_ID = query\_hall.First();

var query\_seats = (from g in db.halls

where g.hall\_id == hall\_ID

select g.seats).Distinct();

string seats\_toDB = query\_seats.First();

try

{

number\_of\_hSht = db.hall\_showtime.Max(n => n.hsh\_id) + 1; //Запись с max ID билета,чтобы записать новый с ID на 1 больше

}

catch

{

number\_of\_hSht = 1;

}

hall\_showtime new\_hSht = new hall\_showtime

{

hall\_id = hall\_ID,

showtime\_id = number\_of\_showtimes,

hsh\_id = number\_of\_hSht,

seat\_map = seats\_toDB, // записываем пустой зал из таблицы залов

};

// проверяем не является ли запрос добавлением скидки. Новое время и старое должны совпадать

var time = TimeSpan.Parse(cmbTimeTab3.Text);

var query\_sho = (from g in db.showtimes

where g.showtime\_id == number\_of\_hSht - 1

select g.time).Distinct();

string time12 = query\_sho.First().ToString();

if (time12 == cmbTimeTab3.Text)

{

db.showtimes.Add(new\_showtime);

db.hall\_showtime.Add(new\_hSht);

}

else

{

//--------------------------- обновляем только цену при добовлении скидки------------------

int priceAfterDiscount = (int)(float.Parse(txbPriceTab3.Text) \* ((100 - float.Parse(cmbDiscountTab3.Text.Remove(2, 1))) / 100));

int shoID = 0;

string time\_1 = cmbTimeTab3.Text.ToString();

// запрос showtimeID по фильму

var querySho = (from showtime in db.showtimes

orderby showtime.movieid

where (showtime.time.ToString() == time\_1 && showtime.movieid == number\_of\_movies)

select new { showtime.showtime\_id }).Distinct();

foreach (var it in querySho)

{

shoID = it.showtime\_id;

}

db.Configuration.ValidateOnSaveEnabled = false;

MessageBox.Show(number\_of\_showtimes.ToString() + " " + number\_of\_movies.ToString());

var verifcation = db.showtimes.Where(x => x.showtime\_id == shoID && x.movieid == number\_of\_movies).FirstOrDefault();

verifcation.price = priceAfterDiscount;

MessageBox.Show("Скидка записана");

//

}

db.SaveChanges();

return 0;

}

public (string, string) loginFunc(string login, string password) // функция входа для логина и пароля

{

var temp\_login = (from r in db.roles

select new { r.role1, r.login, r.password }).ToList();

foreach (var item in temp\_login.OrderBy(o => o.role1))

{

if (login == item.login && password == item.password)

{

return (item.role1.ToString(), item.login.ToString());

}

}

return ("failed", "no data"); // неверная попытка входа

}

public void selectTimeByMovieTitle(ref ComboBox cmbTime, ref ComboBox cmbTitleFilmTab1, ref ComboBox cmbHall, ref TextBox txbDirectorTab1,

ref TextBox txbYearTab1, ref TextBox txbGenerTab1, ref TextBox txbDurationTab1, ref RichTextBox rtbCastTab1, ref Label lblPrice)

{

int[] sh\_ID = new int[10];

int i = 0;

int temp = 0;

string title = cmbTitleFilmTab1.Text;

// запрос поиска с отбором по названию фильма

var query = (from movie in db.movies

join st in db.showtimes on movie.movieid equals st.movieid

orderby movie.title

where (movie.title == title)

select new { movie.title, movie.duration, movie.director, movie.genre, movie.cast\_movie, st.time, st.showtime\_id, st.price, movie.year\_movie }).Distinct();

// заполнение combobox о фильме

foreach (var item in query.OrderBy(o => o.title))

{

i++;

sh\_ID[i] = item.showtime\_id;

cmbTime.Items.Add(item.time.ToString());

txbDirectorTab1.Text = item.director;

txbGenerTab1.Text = item.genre;

txbYearTab1.Text = item.year\_movie.ToString();

txbDurationTab1.Text = item.duration.ToString().Remove(5, 3);

rtbCastTab1.Text = item.cast\_movie;

lblPrice.Text = item.price.ToString() + " p.";

}

while (i > 0)

{

temp = sh\_ID[i];

var query1 = (from g in db.hall\_showtime

where g.showtime\_id == temp

select g.hall\_id).Distinct();

var query2 = (from g in db.halls

where g.hall\_id == query1.FirstOrDefault()

orderby g.hall\_name

select new { g.hall\_name }).Distinct();

foreach (var item in query2)

{

// если в combobox уже внесён этот зал то мы его не добавляем

if (!cmbHall.Items.Contains(item.hall\_name))

{

cmbHall.Items.Add(item.hall\_name);

}

}

i--;

}

}

}

}