СОДЕРЖАНИЕ

[Введение 4](#_Toc105350330)

[1 Анализ предметной области 5](#_Toc105350331)

[2 Постановка задачи 6](#_Toc105350332)

[3 Проектирование программного модуля 7](#_Toc105350333)

[3.1 Классификаторы 7](#_Toc105350334)

[3.2 Инфологическая модель 8](#_Toc105350335)

[3.3 Описание входных и выходных данных 10](#_Toc105350336)

[3.4 Алгоритм решения задачи 10](#_Toc105350337)

[4 Технологии и инструменты разработки 11](#_Toc105350338)

[4.1 Технологии для разработки 11](#_Toc105350339)

[4.2 Инструменты разработки 14](#_Toc105350340)

[5 Программное обеспечение задачи 18](#_Toc105350341)

[5.1 Пользовательский интерфейс программного модуля 18](#_Toc105350342)

[5.2 Описание методов разработки 20](#_Toc105350343)

[Заключение 21](#_Toc105350344)

[Список использованных источников: 22](#_Toc105350345)

[Приложение A 23](#_Toc105350346)

[Приложение Б 24](#_Toc105350347)

# Введение

Одной из важнейших проблем в современных кинотеатрах является задача создания автоматизированной работы кинотеатра. В настоящее время использование информационных систем в кинотеатрах не является редкостью. Спектр их применения широк и варьируется от автоматизации отдельно взятых мест до полной автоматизации деятельности кинотеатра.

Целью данного проекта является разработка С# кода и его оптимизации для улучшения интерактивности и функциональности приложения расписания.

Для достижения цели проекта необходимо решить ряд задач:

* изучить понятие языка программирования С#;
* разобраться в синтаксисе языка;
* изучить области применения и возможности С#;
* изучить популярные библиотеки и Фреймворки С#;
* определить, что такое entity framework и как он связан с C#;
* разработать, оптимизировать и отладить итоговый C# код
* рассмотреть средства тестирования и отладки;
* определить редактор исходного кода для разработки приложения;
* разработать приложение;

# 1 Анализ предметной области

Кинотеатр — общественное здание или его часть с оборудованием для публичной демонстрации [кинофильмов](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%BD%D0%B5%D0%BC%D0%B0%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84). Главное помещение кинотеатра — зрительный зал с экраном большого размера и системой воспроизведения звука. В современных кинотеатрах система звуковоспроизведения состоит из нескольких [громкоговорителей](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%BE%D0%BC%D0%BA%D0%BE%D0%B3%D0%BE%D0%B2%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D0%BB%D1%8C), обеспечивающих [объёмный звук](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D1%91%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B7%D0%B2%D1%83%D0%BA).

Первые стационарные кинотеатры назывались в [России](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D1%8F) «электротеатрами».

В современных кинотеатрах часто имеется несколько зрительных залов, обязательна система [кондиционирования воздуха](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B4%D0%B8%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B2%D0%BE%D0%B7%D0%B4%D1%83%D1%85%D0%B0), а современные звуковоспроизводящие системы состоят из множества раздельных звуковых каналов. При проектировании зданий для кинотеатров одной из главных задач считается достижение наилучших [акустических](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%B0%D1%8F_%D0%B0%D0%BA%D1%83%D1%81%D1%82%D0%B8%D0%BA%D0%B0) характеристик кинозала для качественного звучания. В кинотеатрах обычно также имеются фойе для зрителей, гардероб, буфет, служебные помещения. [Киноустановка](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%BD%D0%BE%D1%83%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0) кинотеатра, использующего [киноплёнку](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%BD%D0%BE%D0%BF%D0%BB%D1%91%D0%BD%D0%BA%D0%B0), состоит не менее, чем из двух [кинопроекторов](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%BD%D0%BE%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%BE%D1%80) для обеспечения непрерывности демонстрации фильма. Каждый кинопроектор называется «[постом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82_%D0%BA%D0%B8%D0%BD%D0%BE%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%86%D0%B8%D0%B8)» и все имеющиеся посты централизованно управляются специальной автоматизированной системой кинопоказа, обеспечивающей незаметность переходов между отдельными [частями фильма](https://ru.wikipedia.org/wiki/%D0%A7%D0%B0%D1%81%D1%82%D1%8C_%D1%84%D0%B8%D0%BB%D1%8C%D0%BC%D0%B0) для зрителя.

Цель курсового проекта заключается в создании настольного приложения «Резервирование билетов» на базе ОС Windows.

В курсовом проекте решаются задачи:

* формирование базы данных;
* резервирование билетов в реальном времени.

# 2 Постановка задачи

Программа проста в управлении и понятна, не требует больших системных ресурсов и сложнейших настроек. Для работы будет достаточно компьютера с минимальной мощностью. Цель проекта - создание приложения «Резервирование билетов» для кинотеатров.

Задачей данного проекта является разработка программного модуля C# приложения «Резервирование билетов», который улучшат функциональность и интерактивности взаимодействия для кинотеатра, добавляет приложения с полным функционалом использования для добавление фильмов и закрепления за них определенным залом.

Элементы написаны на языке программирования C# с использованием Entity Framework — чтобы код был без лишних команд и при этом эффективным, а также работал быстро и без ошибок.

## 3 Проектирование программного модуля

## 3.1 Классификаторы

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 работнике приведенная в таблице 3.1, документе показанных на таблице 3.2 и местонахождении документа на таблице 3.3.

Таблица 3.1 – Атрибуты Зал

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Зал | Идентификационный код  Название  Местоположение  Количество мест |

Таблица 3.2 – Атрибуты Кино

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Кино | Идентификационный код  Название фильма  Описание  Жанр  Длительность  Режиссер  Начало |

Таблица 3.3 – Атрибуты Билет

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

Таблица 3.4 – Атрибуты Сеанс

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Сеанс | Идентификационный код  Идентификационный код Кино  Идентификационный код Зала |

Таблица 3.5 – Атрибуты Тип места

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Тип места | Идентификационный код  Идентификационный код Места  Название |

Таблица 3.6 – Атрибуты Место

|  |  |
| --- | --- |
| Объект | Атрибуты |
| Место | Идентификационный код  Идентификационный код Зала |

## 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). Во многих случаях информационная модель очень сложна и содержит множество объектов

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

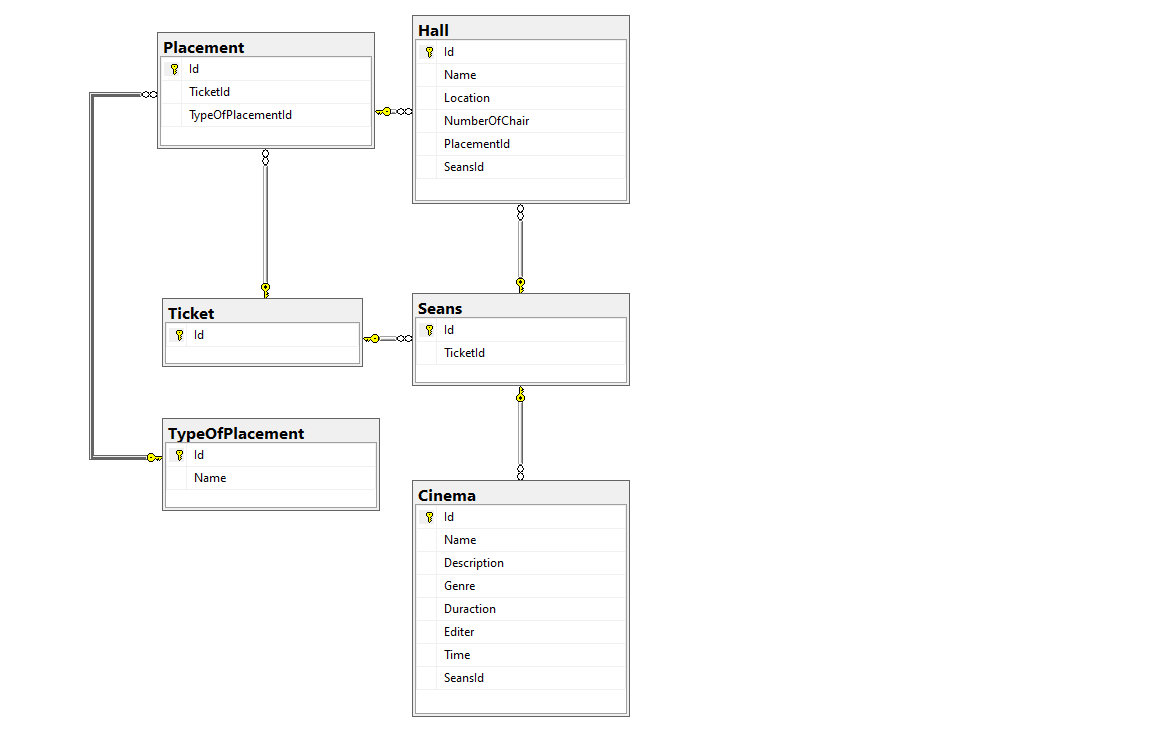


Рисунок 3.1 – Связи таблиц в БД

Структура таблиц Билет, место, зал, сеанс, фильм и тип места базы данных представлена на рисунках 3.2 – 3.7 соответственно.

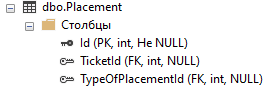


Рисунок 3.2 – Таблица «Место»

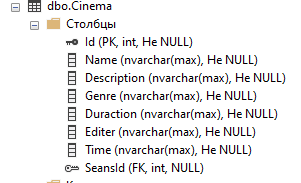


Рисунок 3.3 – Таблица «Кино»



Рисунок 3.4 – Таблица «Билет»

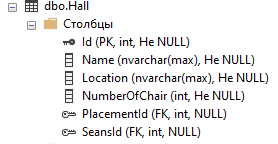


Рисунок 3.5 – Таблица «Зал»

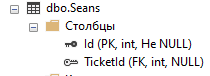


Рисунок 3.6 – Таблица «Сеанс»

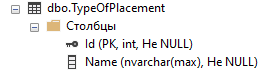


Рисунок 3.7 – Таблица «Тип места»

## 3.3 Описание входных и выходных данных

Входные данные – данные, которые нужны для рассматриваемых заданий. К примеру о таблице, которую необходимо просмотреть. Выходные данные – данные, которые получаются в результате работы программы. К примеру, вывод таблицы по ранее введённым данным.

Входная информация в данном проекте является список фильмов хранящиеся в базе данных. Выходной информацией является зарезервированное место для определенного фильма в конкретном зале.

## 3.4 Алгоритм решения задачи

Последовательное описание блок – схемы приложения приведено ниже:

Шаг 1. Запуск программы.

Шаг 2. Вопрос: Верно ли указан логин и пароль? Если да, то переходим к следующему шагу, иначе остаемся на 2 шаге.

Шаг 3. Вход в приложение под учетной записью

Шаг 4. Вопрос: Выбрано ли действие? Если да, то переходим к 5 шагу, если нет, то переходим к 7 шагу.

Шаг 5. Выполнение

Шаг 6. Возвращаемся к 4 шагу.

Шаг 7. Закрытие приложения.

Блок-схема Приведена в приложении А.

Листинг программы приведен в приложении Б.

## 4 Технологии и инструменты разработки

## 4.1 Технологии для разработки

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 схожий синтаксис с C, C++, Java или JavaScript.

C# — объектно-ориентированный, ориентированный на компоненты язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО [[1](#_Список_использованных_источников:_1)].

WPF – это бесплатная графическая подсистема с открытым исходным кодом, первоначально разработанная Microsoft для рендеринга пользовательских интерфейсов в приложениях на базе Windows. WPF использует язык разметки XAML, для построения визуальной части приложения [[2](#_Список_использованных_источников:_1)].

Entity Framework Core – Объектно-ориентированная технология доступа к данным. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Для облегчения построения используется связка из Windows Communication Foundation и Windows Presentation Foundation, позволяющая строить многоуровневые приложения, реализуя один из шаблонов проектирования MVC, MVP или MVVM [[3](#_Список_использованных_источников:_1)].

MS SQL Server – Система управления реляционными базами данных, 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 с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия. [[4](#_Список_использованных_источников:_1)]

Чтобы написать «Программу» нужно знать синтаксис языка.

Синтаксис C# это набор правил, по которым строится программа. При написании программ используется набор символов Юникода — это надмножество кодировок и поддерживает практически все письменные языки, имеющиеся на планете.

C# чувствителен к регистру символов. Это значит, что ключевые слова, имена переменных и функций и любые другие идентификаторы языка должны всегда содержать одинаковые наборы прописных и строчных букв.

C# игнорирует пробелы, которые могут присутствовать в программе. Кроме того, язык по большей части игнорирует символы перевода строки. Поэтому пробелы и символы перевода строки могут без ограничений использоваться в исходных текстах программ для форматирования и придания им удобного и читаемого внешнего вида.

Как и в других языках программирования, для отделения инструкций друг от друга. Как и в других языках программирования, для отделения инструкций друг от друга в языке С# используется точка с запятой.

Идентификатор — это просто имя. В C# идентификаторы выступают в качестве имен переменных и функций, а также меток некоторых циклов. Идентификаторы в C# должны начинаться с буквы, с символа подчеркивания или знака доллара. Далее могут следовать любые буквы, цифры, символы подчеркивания или знаки доллара. (Цифра не может быть первым символом, так как тогда интерпретатору трудно будет отличать идентификаторы от чисел.).

C# резервирует ряд идентификаторов, которые играют роль ключевых слов самого языка. Эти ключевые слова не могут служить идентификаторами в программах. Для того чтобы понять, как и где можно применить C# следует рассмотреть области, в которых применяется данный язык программирования.

Видеоигры — C# является популярным языком программирования среди создателей видеоигр. Язык используется для разработки игр под Windows, macOS, Android и iOS. Все дело в Unity – платформе для работы с 3D-графикой. С# лучше остальных языков адаптирован под работу с этим движком.

ПО для защиты систем — Безопасность ваших программ и операционных систем обеспечивается благодаря мощным утилитам на базе C#. Колоссальное количество вирусов, на ежедневной основе атакующих компьютеры пользователей, блокируется инструментами, созданными с помощью языка Microsoft. Аналогичная ситуация наблюдается в крупном бизнесе – мировые корпорации защищаются от хакерских атак с помощью ПО, написанного на С#.

Приложения для Windows— Практически вся операционная система Microsoft существует благодаря С#. Привычные нам утилиты и приложения созданы с использованием этого языка и фреймворков, разработанных для него. В эту категорию попадает мессенджер Skype, браузер Internet Explorer, среда для разработки Visual Studio 2012, Microsoft Office (все его составляющие, включая Word, PowerPoint, Excel, Outlook).

В некоторых кругах программистов С# считается чуть ли не лучшим языком для проектирования мобильных приложений. Все благодаря возможности создавать с помощью этого языка нативные программы для любых платформ.

Библиотека С# является многоразовой частью кода, написанной на С#, которая предлагает определенные функции. Это набор функций, объектов и классов, которые вы можете использовать в своем приложении. Библиотека абстрагирует различные слои, так что вам не нужно беспокоиться об их деталях реализации.

Вы можете вызвать библиотечную функцию и передать ей некоторые параметры, и библиотека выполнит ее и вернет вам элемент управления. Это не создает каких-либо структурных ограничений, которые ограничивают использование библиотеки.

Фреймворки С# — это библиотеки программирования C#, в которых есть предварительно написанный код для использования в стандартных функциях и задачах программирования.

По сути, библиотека — это набор функций и способов решения функциональности проекта, для которого пишется код, а фреймворк – это площадка для программы. Они облегчают разработку и объединение разных элементов кода одного большого проекта.

Главным отличием библиотеки от фреймворка является инверсия контроля. Если программист контролирует ситуацию при вызове функции или метода, то фреймворк самостоятельно вызывает код, написанный программистом.

Для работы с базами данных в C# можно использовать Entity Framework. Entity Framework — это решение для работы с базами данных, которое используется в программировании на языках семейства C. Оно позволяет быстро писать код, а также взаимодействовать с СУБД с помощью сущностей (entity), а не таблиц.

Целью «entity framework» является ускорение и упрощение написания кода для взаимодействия с базами данных.

Например, работая с базами данных напрямую, разработчик должен беспокоиться о подключении, подготовке SQL и параметров, отправке запросов и транзакций. На Entity Framework всё это делается автоматически — программист же работает непосредственно с сущностями и только говорит EF, что нужно сохранить изменения.

## 4.2 Инструменты разработки

Для того чтобы разработать и написать код необходим редактор кода.

Редактор исходного кода — программа текстового редактора, предназначенная специально для редактирования исходного кода компьютерных программ. Это может быть автономное приложение или встроенное в интегрированную среду разработки (IDE). Редакторы исходного кода имеют некоторые возможности, упрощающие и ускоряющие написание и изменение кода, такие как:

* подсветка синтаксиса;
* автодополнение;
* отступы;
* проверка правильности расстановки скобок;
* контекстная помощь по коду;

Интегрированная среда разработки (IDE) — это комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО). Среда разработки включает в себя:

* текстовый редактор (Программа, позволяющая вводить, изменять, обрабатывать, хранить и восстанавливать текст);
* компилятор (Программа, переводящая текст, написанный на языке программирования, в набор машинных кодов);
* средства автоматизации сборки (Автоматизация сборки включает в себя скриптовый язык или автоматизацию процесса компиляции исходного кода компьютера в двоичный код);
* отладчик (Компьютерная программа для автоматизации процесса поиска ошибок).

Редакторы исходного кода являются фундаментальным инструментом программирования, поскольку основная работа программистов заключается в написании и редактировании исходного кода.

Visual Studio Code **—** один из самых популярных редакторов кода уже на протяжении нескольких лет.

Универсальный редактор кода от Microsoft, довольно быстрый и функциональный. Среда с открытым исходным кодом поддерживает автоматическое умное дополнение текста программ и подсветку синтаксиса для большинства популярных языков.

Возможностей Visual Studio Code хватит для любительской и профессиональной разработки. Расширить функциональность можно за счёт тысячи плагинов.

Быстрее редактировать код в Visual Studio Code помогут сочетания клавиш. Они обеспечат быструю навигацию по всему проекту и ускорят разработку.

JetBrains Rider — кроссплатформенная интегрированная среда разработки программного обеспечения для платформы .NET, разрабатываемая компанией JetBrains. Поддерживаются языки программирования C#, VB.NET и F#.

Rider — позволяет создавать приложения для Windows, веб-приложения и мобильные приложения, как и Microsoft Visual Studio. Но, в отличие от Visual Studio, Rider является кросс-платформенной средой, ею можно пользоваться на Windows, OS X и Linux (тогда как Visual Studio — только под Windows).

Rider построена на платформе IntelliJ, и включает следующие компоненты: виртуальную файловую систему, модуль пользовательского интерфейса, текстовый редактор, поддержку синтаксиса, компоненты для работы с кодом, функцию контроля версий, отладчик и тестировщик.

[Eclipse](https://www.eclipse.org/) — свободная интегрированная среда разработки модульных кроссплатформенных приложений. Проект Eclipse представляет собой первую столь мощно поддержанную мировым IT-сообществом попытку создания единой открытой интегрированной платформы разработки приложений, обладающей надежностью, функциональностью и уровнем качества коммерческого продукта. Фактически эта платформа предназначена для всего и ни для чего конкретно: она представляет собой основу, имеющую блочную структуру и интегрирующую инструменты разработки ПО различных производителей для создания приложений на любом языке, с использованием любых технологий и для любой программной платформы.

Программный код может содержать синтаксические или логические ошибки. Часто бывает так, что в коде есть ошибки, но нет никаких проявлений. Нет никаких сообщений об ошибках, и нет никаких указаний на то, где эти ошибки искать.

Отладка – это процесс поиска и исправления ошибок в скрипте. Все современные среды разработки поддерживают инструменты для отладки – специальный графический интерфейс, который сильно упрощает отладку. Он также позволяет по шагам отследить, что именно происходит в нашем коде.

Встроенные отладчики можно включить или отключить, отслеживая ход выполнения программы и сообщая пользователю об ошибках.

При помощи отладчика можно устанавливать точки остановки (участки кода, где процесс выполнения будет остановлен) и исследовать содержимое переменных и его изменение в ходе работы скрипта.

В окне отладчика можно задавать точки остановки в коде C#. Точка остановки – это участок кода, где отладчик автоматически приостановит исполнение C#. В каждой такой точке C# будет останавливать выполнение скрипта, предоставляя возможность проверять различные значения C#. Когда проверка значений будет закончена, можно продолжить выполнение скрипта (обычно при помощи кнопки Продолжить).

Что бы установить точку остановки, необходимо слева от номера строки нажать на пустую область, после этого действия на пустой столбце появится красная точка, свидетельствуя о точке остановки для отладки.

## 5 Программное обеспечение задачи

## 5.1 Пользовательский интерфейс программного модуля

В активити входа в приложение, представленном на рисунке 5.1, находится два текстовых поля логин и пароль, кнопка авторизации. Для дальнейшей работы нужно верно указать логин и пароль. Если логин и пароль верны, то откроется основное активити, представленное на рисунке 5.2 с возможностью выбора фильма. При нажатии на кнопку открывается окно с выбором места в конкретном зале которое показано на рисунке 5.3

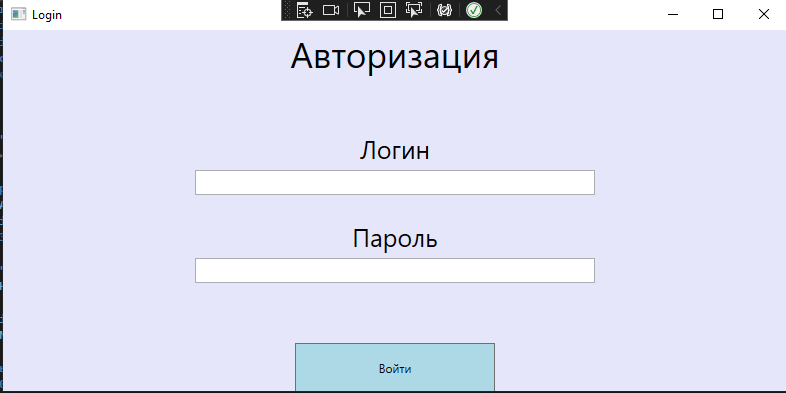


Рисунок 5.1 – Окно входа

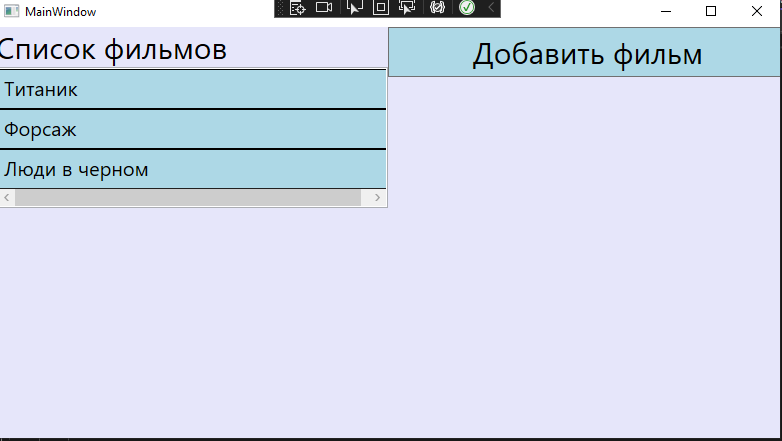


Рисунок 5.2 – Основное окно

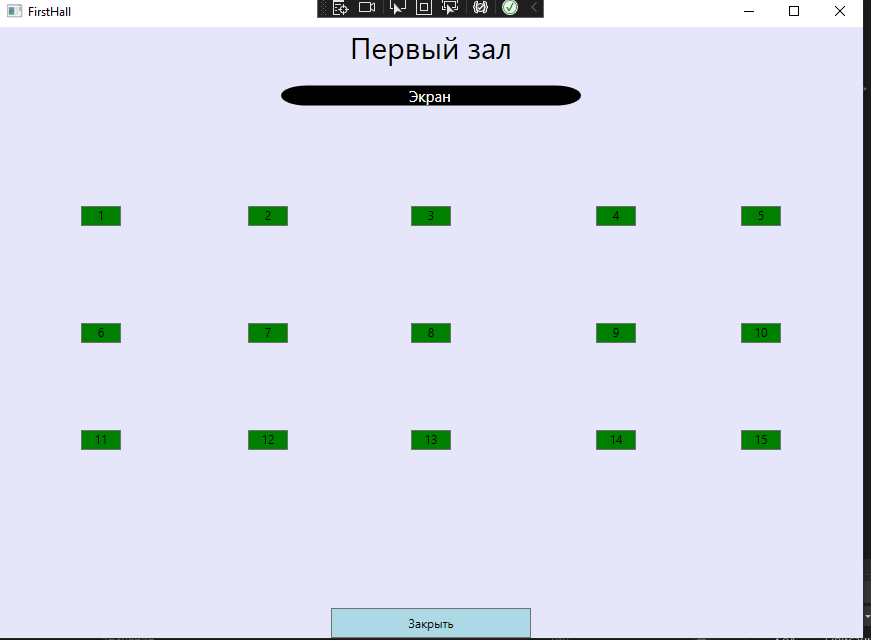


Рисунок 5.3 – Окно выбора места

## 5.2 Описание методов разработки

Программа пишется по паттерну MVVM — (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения [[5](#_Список_использованных_источников:_1)]. В частности, используя команды благодаря интерфейсу ICommand [[6](#_Список_использованных_источников:_1)].

Пример реализации ICommand:

public class CommandBase : ICommand

{

public Predicate<object> Predicate { get; set; }

public Action<object> Func { get; set; }

public event EventHandler? CanExecuteChanged;

public CommandBase(Action<object> command , Predicate<object>? predicate=null)

{

Predicate = predicate;

Func = command;

}

public virtual bool CanExecute(object? parameter)

{

return Predicate==null? true: Predicate(parameter);

}

public virtual void Execute(object? parameter)

{

Func?.Invoke(parameter);

}

protected virtual void OnCanExecuteChanged(object sender, EventArgs e)

{

CanExecuteChanged?.Invoke(sender, e);

}

}

## Заключение

Программа разработана на языке C# в интегрированной середе разработки Visual Studio с использованием СУБД SQL Server. Приложение интуитивно понятно даже неопытному пользователю и не затрудняет его работу с документами с помощью компьютера.

В дальнейшем приложение будет обновляться и будут добавляться новые функции необходимые для работы с резервированием билетов.

В процессе написания кода для данной программы были приобретены практические навыки в работе в СУБД SQL Server и языке программирования C#.

Цель курсового проекта достигнута. Задачи выполнены.

## Список использованных источников

Список Использованной литературы

1. Прайс, М. C# и .Net 5 Разработка и оптимизация, 5-е издание / М.Прайс,- Санкт-Петербург : Питер, 2022. – 832 с.
2. CРихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#, 4-е издание / Д. Рихтер. – Санкт-Петебург: Питер, 2019. – 896 с.
3. Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NETCore / Э.Троелсен, Ф. Джепикс. – Москва : Вильямс, 2018. – 1328 с.

Интернет ресурсы

1. Википедия. Язык программирование СИ – режим доступа: <https://ru.wikipedia.org/wiki/Си_(язык_программирования)>
2. Википедия. Windows Presentation Foundation (WPF) – режим доступа: https://ru.wikipedia.org/wiki/Windows\_Presentation\_Foundation
3. Википедия. ADO.NET Entity Framework – режим доступа: https://ru.wikipedia.org/wiki/ADO.NET\_Entity\_Framework
4. Википедия. Microsoft SQL Server – режим доступа: https://ru.wikipedia.org/wiki/Microsoft\_SQL\_Server
5. Metanit. Паттерн MVVM (Model-View-ViewModel)– Режим доступа: <https://metanit.com/sharp/wpf/22.1.php>
6. Metanit. Команды в Model-View-ViewModel – режим доступа: https://metanit.com/sharp/wpf/22.3.php

## 

## Приложение A

(обязательное)

Блок - схема программы

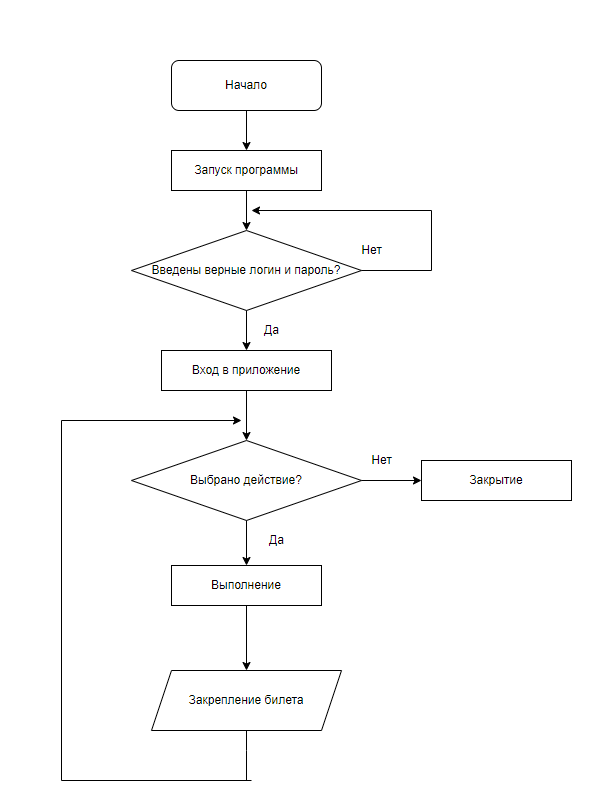


Рисунок А.1–Блок схема программы

## Приложение Б

(обязательное)

Листинг программы

using KontrolWork;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

#nullable disable

namespace KontrolWork.Migrations

{

[DbContext(typeof(DatabaseContext))]

partial class DatabaseContextModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "6.0.4")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder, 1L, 1);

modelBuilder.Entity("KontrolWork.Chair", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<int>("Hall")

.HasColumnType("int");

b.Property<int>("Number")

.HasColumnType("int");

b.HasKey("Id");

b.ToTable("Chair");

});

modelBuilder.Entity("KontrolWork.Models.Cinema", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<string>("CinemaName")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Time")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Cinema");

});

modelBuilder.Entity("KontrolWork.Tickets", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<int>("ChairId")

.HasColumnType("int");

b.Property<int>("CinemaId")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("ChairId");

b.HasIndex("CinemaId");

b.ToTable("Ticket");

});

modelBuilder.Entity("KontrolWork.Tickets", b =>

{

b.HasOne("KontrolWork.Chair", "Chair")

.WithMany()

.HasForeignKey("ChairId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("KontrolWork.Models.Cinema", "Cinema")

.WithMany()

.HasForeignKey("CinemaId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Chair");

b.Navigation("Cinema");

});

#pragma warning restore 612, 618

}

}

}

public partial class film : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Chair",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Number = table.Column<int>(type: "int", nullable: false),

Hall = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Chair", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Cinema",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

CinemaName = table.Column<string>(type: "nvarchar(max)", nullable: false),

Time = table.Column<string>(type: "nvarchar(max)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Cinema", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Ticket",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

CinemaId = table.Column<int>(type: "int", nullable: false),

ChairId = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Ticket", x => x.Id);

table.ForeignKey(

name: "FK\_Ticket\_Chair\_ChairId",

column: x => x.ChairId,

principalTable: "Chair",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Ticket\_Cinema\_CinemaId",

column: x => x.CinemaId,

principalTable: "Cinema",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "IX\_Ticket\_ChairId",

table: "Ticket",

column: "ChairId");

migrationBuilder.CreateIndex(

name: "IX\_Ticket\_CinemaId",

table: "Ticket",

column: "CinemaId");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "Ticket");

migrationBuilder.DropTable(

name: "Chair");

migrationBuilder.DropTable(

name: "Cinema");

}

}

public Cinema(string name, string description, string genre, string duraction, string editer, string time)

{

Name = name;

Description = description;

Genre = genre;

Duraction = duraction;

Editer = editer;

Time = time;

}

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public string Genre { get; set; }

public string Duraction { get; set; }

public string Editer { get; set; }

public string Time { get; set; }

public Hall(string name, string location, int numberOfChair)

{

Name = name;

Location = location;

NumberOfChair = numberOfChair;

}

[Key]

public int Id { get; set; }

public string Name { get; set; }

public string Location { get; set; }

public int NumberOfChair { get; set; }

// public ICollection<Seans> Seans { get; set; }

public ICollection<Placement> placement { get; set; }

}

public class Placement

{

[Key]

public int Id { get; set; }

public ICollection<Ticket> ticket { get; set; }

public class Seans

{

[Key]

public int Id { get; set; }

public ICollection<Ticket> tickets { get; set; }

public Hall halls { get; set; }

public Cinema cinemas { get; set; }

}

public class Ticket

{

[Key]

public int Id { get; set; }

// public ICollection<Seans> Seans { get; set; }

// public ICollection<Placement> Placement { get; set; }

}

public class TypeOfPlacement

{

public TypeOfPlacement(string name)

{

Name = name;

}

[Key]

public int Id { get; set; }

public ICollection<Placement> placement { get; set; }

public string Name { get; set; }

}