**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

на тему: «**СИСТЕМА ПРОДАЖИ БИЛЕТОВ В ТЕАТРЕ**»

Исполнитель: студент гр. ИТП-21

Бондарев Е.Ю.

Руководитель: доцент

Курочка К.С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc134908731)

[1 ТЕХНИЧЕСКИЕ ПОДХОДЫ К РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ ДЛЯ СИСТЕМЫ ПРОДАЖИ БИЛЕТОВ 7](#_Toc134908732)

[1.1 Обзор аналогов 7](#_Toc134908733)

[1.2 Объектно-ориентированное программирование 8](#_Toc134908734)

[1.3 Принципы *SOLID* 9](#_Toc134908735)

[1.4 Модели хранения данных 10](#_Toc134908736)

[1.5 *Windows Forms* технология 10](#_Toc134908737)

[1.6 Язык разметки документов *XML* 11](#_Toc134908738)

[1.7 Технология *LINQ* 12](#_Toc134908739)

[2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ПОСТАВЛЕНОЙ ЗАДАЧИ И ОПИСАНИЕ ПРОГРАММНОГО КОМПЛЕКСА 13](#_Toc134908740)

[2.1 Архитектура программного комплекса 13](#_Toc134908741)

[2.2 Структура данных 13](#_Toc134908742)

[2.3 Компоненты приложения 15](#_Toc134908743)

[2.4 Структура классов разработанного программного комплекса 16](#_Toc134908744)

[2.4 Графический интерфейс пользователя 22](#_Toc134908745)

[3 ВЕРИФИКАЦИЯ ПРИЛОЖЕНИЯ 26](#_Toc134908746)

[3.1 Тестирование пользовательского интерфейса 26](#_Toc134908747)

[3.2 Модульное тестирование приложния 28](#_Toc134908748)

[ЗАКЛЮЧЕНИЕ 29](#_Toc134908749)

[Список используемых источников 30](#_Toc134908750)

[ПРИЛОЖЕНИЕ А 31](#_Toc134908751)

[Графические представления интерфейса программы 31](#_Toc134908752)

[ПРИЛОЖЕНИЕ Б 33](#_Toc134908753)

[Листинг программы 33](#_Toc134908754)

[ПРИЛОЖЕНИЕ В 68](#_Toc134908755)

[Руководство системного программиста 68](#_Toc134908756)

[ПРИЛОЖЕНИЕ Г 69](#_Toc134908757)

[Руководство программиста 69](#_Toc134908758)

[ПРИЛОЖЕНИЕ Д 70](#_Toc134908759)

[Руководство пользователя 70](#_Toc134908760)

[ПРИЛОЖЕНИЕ Е 71](#_Toc134908761)

[Иерархическая схема классов приложения 71](#_Toc134908762)

# ВВЕДЕНИЕ

Система продажи билетов имеет важное значение для многих организаций, включая кинотеатры, театры, музеи, концертные залы и другие. Она облегчает процесс покупки билетов для клиентов и обеспечивает безопасный и удобный способ продажи билетов для организаторов мероприятий.

Эта система также может быть использована для контроля доступа на мероприятия, что обеспечивает безопасность и предотвращает мошенничество. Кроме того, система продажи билетов может предоставлять информацию о доступности мест, ценах на билеты, расписании и других деталях мероприятия. Это помогает клиентам выбрать подходящее время и место для посещения мероприятия и обеспечивает гладкое и эффективное проведение самого мероприятия.

Существует множество различных систем продажи билетов, включая онлайн-платформы, автоматизированные кассы и системы, управляемые операторами. Каждая система имеет свои преимущества и недостатки, и организации выбирают систему, которая наиболее соответствует их потребностям и бюджету.

Одним из главных преимуществ систем продажи билетов является их доступность для клиентов. Клиенты могут легко купить билеты, не выходя из дома, используя онлайн-платформы или мобильные приложения. Это особенно удобно для тех, кто не имеет времени или возможности посетить кассу в физическом месте продажи билетов.

Благодаря системам продажи билетов организации могут лучше контролировать количество проданных билетов и заранее знать количество зрителей на мероприятии.

Более того, система продажи билетов может помочь в сборе данных и аналитике. Организаторы мероприятий могут использовать данные о продажах билетов для анализа поведения клиентов и популярности определенных мероприятий. Это может помочь им в принятии решений относительно маркетинговых стратегий и улучшения опыта посетителей в будущем.

В конечном итоге, система продажи билетов является важным элементом в организации мероприятий, которая обеспечивает удобство и безопасность для клиентов и организаторов. С помощью системы продажи билетов клиенты могут легко выбирать и оплачивать желаемые места на мероприятии, а организаторы могут контролировать доступ к мероприятию и отслеживать продажи. Система продажи билетов также управляет финансами организации, позволяя ей получать доход от проданных билетов и распределять его среди всех заинтересованных сторон. Кроме того, система продажи билетов помогает планировать будущие мероприятия, анализируя данные о посещаемости, предпочтениях и отзывах клиентов. Она может повысить эффективность организации мероприятий и улучшить опыт посетителей.

# 1 ТЕХНИЧЕСКИЕ ПОДХОДЫ К РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ ДЛЯ СИСТЕМЫ ПРОДАЖИ БИЛЕТОВ

## 1.1 Обзор аналогов

В настоящее время на рынке существует огромное количество различных систем продажи билетов, предоставляемых компаниями-поставщиками. Каждая из этих систем обладает своими уникальными особенностями и функциональностью, которые могут быть востребованы для различных типов мероприятий и организаторов.

Некоторые системы продажи билетов могут предоставлять возможность онлайн-бронирования мест, что особенно удобно для крупных мероприятий, где необходимо забронировать много мест для групп посетителей. Другие системы могут предоставлять возможность создания персонализированных билетов с уникальными *QR*-кодами для более безопасного контроля доступа на мероприятие.

Для театров, как и для любых других мероприятий, существует множество систем продажи билетов, которые предоставляют различные инструменты для управления продажами и улучшения опыта посетителей. Одной из таких систем является *Ticketmaster*. Это крупнейшая система продажи билетов в мире, которая позволяет театрам создавать схему зала и бронировать места, а также отправлять электронные билеты на почту или мобильный телефон посетителя. Кроме того, *Ticketmaster* предлагает инструменты для управления продажами, маркетинга и аналитики, что может быть полезно для организаторов театральных мероприятий.

Другой системой продажи билетов, которая может быть полезна для театров, является *Eventbrite*. Эта система позволяет театрам создавать персонализированные страницы мероприятий с подробной информацией, фотографиями и видео. Театры могут использовать *Eventbrite* для продажи билетов на свои спектакли и события, а также для сбора пожертвований и регистрации гостей.

Еще одной системой продажи билетов, которая может быть полезна для театров, является *Brown Paper Tickets*. Эта система специализируется на мероприятиях с небольшим бюджетом, включая театральные спектакли. *Brown Paper Tickets* предоставляет возможность продажи билетов онлайн и в реальном времени, а также отправляет билеты на почту или мобильный телефон посетителя.

Наконец, система *Spektrix* также может быть полезна для театров. Она предлагает инструменты для управления продажами, маркетинга и аналитики, а также позволяет театрам создавать персонализированные билеты с уникальными *QR*-кодами, которые могут использоваться для контроля доступа на мероприятие.

В целом, выбор системы продажи билетов для театров зависит от конкретных потребностей и желаемых функций. Каждая из упомянутых выше платформ предлагает уникальные возможности для управления продажами и улучшения опыта посетителей.

## 1.2 Объектно-ориентированное программирование

ООП (объектно-ориентированное программирование) *–* это подход к программированию, в котором программа разбивается на объекты, каждый из которых имеет свои свойства и методы. В ООП данные и функции, работающие с этими данными, объединяются в объекты, которые обладают определенным поведением.

Основные принципы объектно-ориентированного программирования включают в себя наследование, инкапсуляцию и полиморфизм. Наследование позволяет создавать новые классы на основе уже существующих, а инкапсуляция *–* скрыть детали реализации объекта от пользователя и предоставить только интерфейс для работы с ним. Полиморфизм позволяет использовать один и тот же интерфейс для работы с разными объектами, что делает код более гибким и удобным для использования.

Объектно-ориентированное программирование позволяет писать более читаемый и понятный код, что упрощает его сопровождение и развитие. В ООП используются классы, объекты, наследование и другие конструкции, которые позволяют разбивать код на логические блоки и обеспечивают лучшую организацию и структурирование кода.

ООП широко используется в современном программировании и применяется в различных областях, от создания веб-сайтов до разработки игр и мобильных приложений. Некоторые языки программирования, такие как *Java*, *Python* и *C*++, предназначены для ООП и предоставляют мощные инструменты для создания объектно-ориентированных программ.

Объектно-ориентированное программирование также позволяет использовать наследование для повторного использования кода и уменьшения дублирования. Кроме того, объектно-ориентированное программирование позволяет разделять ответственность между различными объектами, что упрощает разработку сложных систем.

Важным принципом ООП являются абстракция, полезная для описания общих понятий и выделения ключевых характеристик, и композиция, позволяющая объединять объекты в более крупные структуры.

Важным аспектом ООП является использование классов для создания объектов. Класс определяет свойства и методы объекта, а каждый созданный объект имеет свои собственные значения свойств, но использует одни и те же методы класса.

Другим важным принципом ООП является инкапсуляция, которая обеспечивает скрытие внутренней реализации объекта и предоставляет интерфейс для работы с ним. Это позволяет изменять внутреннюю реализацию объекта без влияния на другие части программы, использующие этот объект. Объектно-ориентированное программирование также обеспечивает возможность полиморфизма, который позволяет объектам разных классов иметь одинаковые методы и использоваться в качестве аргументов функций без необходимости знать их конкретный тип.

Еще одним важным аспектом ООП является использование наследования интерфейсов, который позволяет создавать новые классы на основе уже существующих интерфейсов. Это позволяет увеличивать гибкость и расширяемость программы.

Наконец, ООП поддерживает принцип композиции объектов, который позволяет создавать сложные объекты из более простых, используя их в качестве компонентов. Это позволяет создавать гибкие и масштабируемые системы.

Общие преимущества ООП включают повторное использование кода, уменьшение дублирования, упрощение разработки и поддержки кода, улучшение гибкости и расширяемости программы, а также увеличение понятности и читаемости кода.

## 1.3 Принципы *SOLID*

*SOLID* – это аббревиатура, которая означает пять принципов объектно-ориентированного программирования, разработанных Робертом Мартином (*Robert C. Martin*). Эти принципы помогают разработчикам создавать гибкие, расширяемые и легко поддерживаемые приложения.

Вот краткое описание каждого из принципов *SOLID*:

1. Принцип единственной ответственности (*Single Responsibility Principle, SRP*) - класс должен иметь только одну причину для изменения. Каждый класс должен быть ответственен только за одну часть функциональности программы. Это позволяет легко поддерживать и тестировать код.
2. Принцип открытости/закрытости (*Open/Closed Principle, OCP*) - классы должны быть открыты для расширения, но закрыты для изменения. Это означает, что вы должны использовать наследование или интерфейсы для добавления новой функциональности, а не изменять существующий код.
3. Принцип подстановки Барбары Лисков (*Liskov Substitution Principle, LSP*) - объекты должны быть заменяемы друг за друга без изменения поведения программы. Это означает, что вы должны использовать наследование, только если это абсолютно необходимо, и если классы-наследники действительно могут заменять своих родителей.
4. Принцип разделения интерфейса (*Interface Segregation Principle, ISP*) - клиенты не должны зависеть от интерфейсов, которые они не используют. Лучше создавать несколько узких интерфейсов, чем один большой. Это уменьшает связанность между классами и делает код более гибким.
5. Принцип инверсии зависимостей (*Dependency Inversion Principle, DIP*) - модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба типа модулей должны зависеть от абстракций. Это означает, что вы должны программировать на уровне абстракций, а не конкретных реализаций.
6. Применение принципов *SOLID* помогает создавать более чистый, гибкий и легко поддерживаемый код, который легче расширять и модифицировать в будущем.

## 1.4 Модели хранения данных

Существует ряд различных методов сохранения информации, каждый из которых специализирован для решения определенных задач и имеет свои сильные и слабые стороны.

Например, реляционные базы данных широко используются для хранения структурированных данных, таких как финансовые отчеты, данные клиентов и т.д. Однако они могут быть неэффективными для работы с неструктурированными данными, такими как изображения или видео.

Для хранения неструктурированных данных могут использоваться *NoSQL* базы данных, такие как *MongoDB* или *Cassandra*. Они хорошо подходят для хранения больших объемов данных, не имеющих строгой структуры, таких как данные социальных медиа, информация об устройствах Интернета вещей (*IoT*) и т.д.

Еще один подход - использование облачных хранилищ, таких как *Amazon S3* или *Microsoft Azure*. Эти хранилища обеспечивают высокую отказоустойчивость и масштабируемость, что делает их идеальными для хранения крупных объемов данных.

Независимо от выбранной модели хранения данных, важно выбирать ее в соответствии с требованиями вашей конкретной задачи, а также учитывать ограничения бюджета и доступные ресурсы.

Кроме того, при выборе модели хранения данных необходимо учитывать требования к скорости чтения и записи данных, а также к безопасности и конфиденциальности. Например, если данные содержат конфиденциальную информацию, то необходимо выбирать модель хранения данных, которая обеспечивает надежную защиту от несанкционированного доступа.

Также стоит учитывать возможность интеграции выбранной модели хранения данных с другими системами и приложениями, с которыми ваша компания работает.

В целом, выбор модели хранения данных - это компромисс между различными факторами, такими как требования к хранению данных, доступные ресурсы и бюджет, безопасность, скорость доступа и масштабируемость. Важно выбрать модель, которая наилучшим образом соответствует вашим требованиям и поможет эффективно решать поставленные задачи.

## 1.5 *Windows Forms* технология

*Windows Forms* – это платформа пользовательского интерфейса для создания классических приложений *Windows*. Она обеспечивает один из самых эффетивных способов создания классических приложений с помощью визуального конструктора в *Visual Studio*. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений

В *Windows Forms* можно разрабатывать графически сложные приложения, которые просто развертывать, обновлять, и с которыми удобно работать как в автономном режиме, так и в сети. Приложения *Windows Forms* могут получать доступ к локальному оборудованию и файловой системе компьютера, на котором работает приложение.

Приложение *Windows Forms* представляет собой событийно-ориентированное приложение, поддерживаемое *Microsoft .NET Framework*. В отличие от пакетных программ, большая часть времени тратится на ожидание от пользователя каких-либо действий, как, например, ввод текста в текстовое поле или клика мышкой по кнопке. При выполнении пользователем какого-либо действия с формой или одним из ее элементов управления создается событие. Приложение реагирует на эти события, как задано в коде, и обрабатывает события при их возникновении.

*Windows Forms* также поддерживает многопоточность, что позволяет создавать приложения, которые могут выполнять несколько задач одновременно. Она также предоставляет широкий спектр визуальных элементов управления, таких как кнопки, поля ввода, списки и многое другое, которые могут быть легко настроены и управляемы в коде.

*Windows Forms* была представлена в .*NET Framework* 1.0 и быстро стала одной из наиболее популярных платформ для разработки классических приложений *Windows*. С развитием технологий, таких как *WPF* и *UWP, Windows Forms* по-прежнему остается популярным выбором для создания классических приложений и остается поддерживаемой в последних версиях .*NET Framework.*

## 1.6 Язык разметки документов *XML*

*XML (Extensible Markup Language) –* это язык разметки документов, используемый для хранения и обмена данными. XML был разработан с целью предоставить стандартный формат для обмена информацией между различными системами и приложениями.

*XML* используется во многих областях, включая веб-приложения, базы данных, настройки программного обеспечения, научные исследования и многое другое. В отличие от *HTML, XML* не описывает, как должна выглядеть информация на экране, а скорее определяет ее структуру и семантику.

Файлы *XML* состоят из элементов, которые определяют структуру данных. Каждый элемент содержит открывающий и закрывающий теги, между которыми находится содержимое элемента. Некоторые элементы могут содержать атрибуты, которые задают дополнительные характеристики элемента.

*XML* является расширяемым языком, что означает, что его можно легко адаптировать под конкретные потребности проекта. Например, разработчики могут определить свои собственные элементы, атрибуты и правила, которые будут использоваться в файле *XML*. Это делает *XML* очень гибким и удобным для использования в различных областях.

XML также поддерживает возможность использования схемы (XSD), которая определяет структуру и типы данных, которые могут содержаться в XML-документе. С помощью схемы можно проверять корректность структуры и содержимого XML-документа, что делает его более надежным и удобным для обмена данными.

## 1.7 Технология *LINQ*

*LINQ* (*Language Integrated Query*) - это технология, которая позволяет интегрировать запросы в язык программирования *C#* и другие языки .*NET. С* помощью *LINQ* разработчики могут использовать единый язык запросов для доступа к различным источникам данных, таким как базы данных, *XML*-документы, коллекции объектов и т.д. Основным преимуществом *LINQ* является то, что он предоставляет удобный и интуитивно понятный синтаксис запросов, что позволяет разработчикам писать более читаемый и понятный код. *LINQ* также предоставляет много возможностей для фильтрации, сортировки, группировки и агрегации данных, что позволяет упростить код и сделать его более эффективным.

LINQ состоит из двух частей: *LINQ to Objects* и *LINQ to SQL*. *LINQ to Objects* позволяет выполнять запросы к коллекциям объектов в памяти, таким как массивы, списки и словари. *LINQ to SQL* позволяет выполнять запросы к базам данных с помощью языка *SQL*.

*LINQ* также поддерживает другие типы источников данных, такие как *XML*-документы и сетевые потоки. Благодаря этому *LINQ* предоставляет универсальный подход к работе с данными, что упрощает процесс разработки приложений.

В целом, *LINQ* - это мощный инструмент для работы с данными, который позволяет разработчикам писать более читаемый и эффективный код, а также повышает производительность и универсальность приложений.

# 2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ПОСТАВЛЕНОЙ ЗАДАЧИ И ОПИСАНИЕ ПРОГРАММНОГО КОМПЛЕКСА

## 2.1 Архитектура программного комплекса

В рамках данного курсового проекта разрабатывается универсальное и надежное приложение, которое позволяет эффективно и удобно продавать билеты на спектакли в театре и вести их учет. Для создания приложения используется мощный и гибкий язык программирования C#, а в качестве источника данных используются XML-файлы, что обеспечивает быстрый доступ и удобство в работе с информацией. Для доступа к коллекциям используется эффективная и удобная технология LINQ, что позволяет быстро находить необходимые данные. Для создания графического интерфейса применяется инструмент Windows form. В есть возможность многопользовательского доступа к системе, что обеспечивает максимальную целостность и безопасность данных. Это позволяет использовать приложение на практике в любом месте и в любых условиях.

## 2.2 Структура данных

Для успешного выполнения курсовой работы необходимо разработать хранилище данных в виде базы данных основанных на XML-файлов. База данных обеспечивает надежность и целостность данных, что является важным аспектом при создании программного комплекса. В данном курсовом проекте будет использоваться XML-файлы в качестве источника данных:

– выделение сущностей и их атрибутов, которые будут храниться в базе данных, и формирование по ним таблиц;

– определение уникальных идентификаторов (первичных ключей) объектов, которые хранятся в строках таблицы;

– определение отношений между таблицами с помощью внешних ключей;

– нормализация данных в базе данных.

Основные задачи проектирования базы данных:

– хранение в XML-файлах всей необходимой информации;

– возможность получения данных по всем необходимым запросам;

– избыточность и дублирование данных;

– целостность базы данных.

База данных состоит из четырех таблиц:

– *SpectacleModel;*

–*TicketsModel;*

–*UsersModel;*

Модель *SpectacleModel* содержит данные о спектаклях, которые включают в себя название спектакля, информацию об авторе и жанре, дату проведения спектакля, уникальный идентификатор каждого спектакля, словарь категорий и цен на каждую категорию, а также количество свободных мест на спектакле. Для связывания таблицы *SpectacleModel* с таблицей *Genres* используется внешний ключ. Этот ключ позволяет установить связь между таблицами на основе значений в полях. В данном случае, таблица *Genres* содержит информацию о жанрах спектаклей, которая связана с таблицей *SpectacleModel* через внешний ключ, который ссылается на поле *Genre*. Данные в таблице *SpectacleModel* могут быть использованы для мониторинга продаж билетов на спектакли, планирования расписания спектаклей и анализа популярности жанров и авторов.

Таблица *TicketModel* содержит информацию о билетах на спектакли. Эта таблица включает в себя шесть полей. Поле *Id* хранит уникальный идентификатор билета, а поле *ticketId* также хранит *id* билета, которое используется для связи с другими таблицами. Поле *Owner* хранит уникальное имя владельца билета из таблицы *UsersModel.* Это поле используется для связи с таблицей пользователей, которая содержит информацию о зарегистрированных пользователях системы. Поля *Date, Title, Category* и *Price* связаны с таблицей *SpectacleModel* по внешнему ключу *Date*. Это означает, что каждый билет связан с конкретным спектаклем из таблицы *SpectacleModel*, используя дату проведения спектакля. Поле *Date* в таблице *TicketModel* связывается с полем *Date* в таблице *SpectacleModel*, которое также является уникальным идентификатором спектакля. Поля *Title, Category* и *Price* в таблице *TicketModel* содержат соответствующие данные о спектакле, на который был куплен билет.

В таблице *UsersModel* содержатся данные о зарегистрированных пользователях системы. Таблица состоит из трех полей: *Login, Password* и *Role*. Поле *Login* в таблице *UsersModel* используется в качестве первичного ключа, поскольку каждый пользователь имеет уникальное имя. Поле *Password* хранит пароль пользователя, который используется для аутентификации при входе в систему. Поле *Role* является перечислением состоящим из трех ролей: *admin,* *registered* и *guest*. Роль определяет уровень доступа пользователя в системе. Пользователи с ролью *admin* имеют полный доступ ко всем функциям системы, пользователи с ролью *registered* имеют доступ только к определенным функциям, а пользователи с ролью *guest* имеют ограниченный доступ к системе.

Таблица *Log* создана для ведения статистических отчетов и содержит информацию о покупках и возвратах билетов на спектакли. Таблица состоит из семи полей, каждое из которых содержит информацию о соответствующей операции. Поле *Method* хранит информацию о типе операции - покупке или возврате билета пользователем. Поле *Owner* содержит уникальное имя владельца билета из таблицы *UsersModel* и используется для связи с таблицей пользователей, которая содержит информацию о зарегистрированных пользователях системы. Поля *SpectacleName, SpectacleDate, Category* и *Price* содержат информацию о соответствующем спектакле, которая была получена из таблицы *TicketModel*. В поле *Timestamp* хранится дата и время проведения соответствующей операции, дата и время выводятся в формате обратного преобразования даты и времени в соответствии со стандартом *ISO* 8601, например, 2015-07-17T17:04:43.4092892+03:00.

Для повышения удобочитаемости отчетности, было принято решение о конвертации *XML*-файла с информацией о покупках и возвратах билетов на спектакли в формат *Excel*. При конвертации сохраняются все столбцы из таблицы *Log*. Кроме того, для лучшей визуализации данных, было установлено окрашивание строк в зеленый цвет для информации о покупках билетов и в красный цвет для информации о возвратах билетов.

* 1. **Компоненты приложения**

Разработка приложения на основе компонентов - это очень эффективный способ создания программного обеспечения. Каждый компонент включает в себя определенную функциональность, что позволяет разрабатывать их независимо от других, ускоряя процесс разработки и облегчая поддержку кода в будущем.

Как правило, приложение состоит из нескольких блоков: пользовательского интерфейса, логики приложения и классов, которые отвечают за взаимодействие с данными. Каждый из этих блоков может быть разработан отдельно, что делает изменение их отдельно друг от друга более простым и безопасным. Один из преимуществ использования компонентов заключается в том, что если необходимо изменить способ хранения данных, например, перейти от *XML*-файлов к базе данных, необходимо изменить только несколько классов, ответственных за работу с данными. В таком случае другие компоненты приложения не затрагиваются, что делает процесс изменения более простым и безопасным.

Классы, работающие с данными, обычно не зависят от источника данных, что означает, что они не знают, откуда получены данные, и им важно только получить нужную информацию, вызвав соответствующий метод класса, отвечающего за работу с данными. Такой подход делает алгоритмы приложения более универсальными, поскольку они могут использоваться в различных системах без серьезных изменений.

Для корректной работы приложения важно правильно реализовать классы, которые будут работать с разными типами объектов. Классы должны содержать информацию, необходимую для работы с объектами, например, класс «билет» должен содержать информацию о купленном билете, дате покупки и сведения о пользователе, совершившем покупку. Классы также должны иметь соответствующий функционал для работы с объектами.

Для удобства работы с объектами приложения необходимо создать следующие объекты: «билет», «спектакль», «пользователь», «модель пользователя», «типы пользователей» (администратор, гость, авторизованный пользователь), «учет продажи билетов» и «фабрику пользователей». Каждый объект имеет свои особенности и функционал, например, объект «учет продажи билетов» содержит информацию о проданных билетах на спектакль, а объект «фабрика пользователей» создает пользователей определенного типа.

Важно учесть, что различные типы пользователей могут иметь разный доступ к функционалу приложения, например, только у администратора может быть доступ к функционалу управления пользователями.

Кроме того, каждый класс должен иметь определенный набор методов для работы с объектами. Например, у класса «билет» должны быть методы для создания, редактирования и удаления билетов, а у класса «пользователь» – методы для аутентификации, регистрации и изменения учетных данных и просмотра информации о спектаклях.

Для удобства работы с объектами можно использовать различные структуры данных, такие как массивы, списки, словари и т.д. Например, можно использовать перечисление для хранения информации ролях пользователей и категориях спектаклей.

Кроме того, важно учитывать особенности взаимодействия между объектами. Например, при продаже билетов необходимо учитывать информацию о спектакле и количестве доступных мест, а также проверять права доступа пользователя при редактировании информации о других пользователях.

В целом, разработка приложения, работающего с различными типами объектов, требует внимательного проектирования архитектуры и правильной реализации классов и методов для работы с объектами.

## 2.4 Структура классов разработанного программного комплекса

Для облегчения группировки и структурирования данных, было решено создать несколько папок:

– папка *img* содержит все картинки содержащиеся в курсовом проекте*;*

– папка *Logs* содержит классы для работы и формирования статистических отчетов;

– папка *Models* содержит сущности описанные выше;

– папка *Repositories* содержит три класса содержащих основные *CRUD* операции для работы со всеми базами данных в проекте;

– папка *Services* содержит классы для взаимодействия графического интерфейса с классами находящимися в папке *Repositories*;

– папка *Users* содержит классы пользователей с определенными ролями;

– папка *XMLData* содержит *XML*-файлы содержащие данные о жанрах, спектаклях, билетах и пользователях, также в папке хранится информация о статистических отчетах.

Для работы с формированием статических отчетов были созданы специальные классы *TicketServicesXmlLoggingDecorator* и *XmlToExcel*. Класс *TicketServicesXmlLoggingDecorator* является декоратором класса *TicketServices* и дополняет методы *AddTicket* и *DeleteTicket,* которые добавляют и удаляют билеты в базе данных. Заполнение данными происходит в методе *LogOperation*, в результате которого в XML-файл добавляется элемент, содержащий поля, описывающие вид операции, данные о пользователе, информацию о спектакле и время проведения операции.

Для конвертации *XML*-файла в формат *Excel* был создан класс *XmlToExcel.* Класс конвертации содержит метод разделения данных в соответствии с видом исходного *XML*-файла и добавляет окраску полей в зависимости от вида операции.

Класс *SpectacleManager* содержит методы добавления, удаления, обновления и вывода всех пользователей.

Метод *GetAll* возвращает коллекцию объектов типа *SpectacleModel*, которая содержит информацию о представлениях, полученную из XML-документа. Метод использует *LINQ to XML* для выбора всех элементов *spectacle* класса *SpectacleModel* из корневого элемента *XML*-документа и преобразует каждый элемент *spectacle* в объект *SpectacleModel.* Для каждого элемента *spectacle* метод выбирает значения свойств объекта *SpectacleModel* из соответствующих элементов *XML*-документа. Кроме того, метод преобразует строковые значения элементов *XML* в соответствующие типы данных, такие как *DateTime* и *Int32*, используя методы *Parse*. Также, метод использует метод *ToDictionary* для преобразования элементов *category* в словарь, где ключом является значение атрибута *name*, а значением - десятичное число, полученное из значения элемента *category*.

Метод *Add* добавляет новый объект *SpectacleModel* в *XML*-документ. Сначала метод выполняет проверку входных данных с помощью метода *DataValidate*, который определен ниже в коде. Если данные корректны, то метод продолжает свое выполнение. Далее, метод проверяет, есть ли уже спектакль на указанную дату в *XML*-документе. Если есть, то метод генерирует исключение с сообщением об ошибке. Затем, метод получает идентификатор жанра по названию с помощью метода *GetGenreIdByName*. Метод создает новый элемент *spectacle* с помощью класса *XElement* и заполняет его свойствами объекта *SpectacleModel*, переданного в качестве параметра. Значения для свойств *title*, *author*, *genre* и *freePlase* извлекаются непосредственно из объекта *SpectacleModel*. Значение свойства *date* преобразуется в строку в формате *yyyy-MM-dd* и добавляется в новый элемент *spectacle*. Для свойства *categories* из объекта *SpectacleModel* метод создает новый элемент *category* и добавляет его к новому элементу *spectacle*. Каждый элемент *category* содержит значение из словаря *Categories* в свойстве *Value* и ключ из словаря *Categories* в атрибуте *name*. Наконец, метод добавляет новый элемент *spectacle* к корневому элементу *XML*-документа и сохраняет изменения в файл. Если элемент *spectacle* с таким же названием уже существует, то метод перезапишет его значениями из переданного объекта *SpectacleModel*.

Метод *Update* обновляет значения свойств объекта *SpectacleModel* в *XML*-документе. Сначала метод выполняет проверку входных данных с помощью метода *DataValidate*, который определен где-то в коде. Если данные корректны, то метод продолжает свое выполнение. Затем метод вызывает метод *GetElement(item)*, который получает элемент *XML*-документа, соответствующий переданному объекту *SpectacleModel*. Далее, метод обновляет значения свойств объекта *SpectacleModel* в *XML*-документе с помощью метода *SetElementValue.* Для свойств *author*, *genre* и *freePlase* метод устанавливает новые значения, переданные в объекте *SpectacleModel*. Значение свойства *date* преобразуется в строку в формате *yyyy-MM-dd* и устанавливается в качестве нового значения свойства *date* элемента *XML*-документа. Затем метод проходит по каждому элементу *category* элемента *XML*-документа, связанному с переданным объектом *SpectacleModel*. Для каждого элемента *category* метод проверяет, существует ли элемент с атрибутом *name*, равным ключу текущего элемента словаря *Categories* объекта *SpectacleModel*. Если элемент существует, то метод устанавливает новое значение свойства *Value* элемента *category.* Если элемент не существует, то метод генерирует исключение с сообщением об ошибке.

Метод *Delete* удаляет элемент *XML*-документа, соответствующий объекту *SpectacleModel*, переданному в качестве аргумента метода. Сначала метод вызывает метод *GetElement(item),* который получает элемент *XML*-документа, соответствующий переданному объекту *SpectacleModel*. Затем метод вызывает метод *Remove()* для полученного элемента, который удаляет его из *XML*-документа. Наконец, метод сохраняет изменения в *XML*-документе с помощью метода *Save*.

Метод *GetElement* ищет элемент *XML*-документа, соответствующий объекту *SpectacleModel*, переданному в качестве аргумента метода. Сначала метод получает дату из объекта *SpectacleModel* и приводит ее к типу *DateTime*, чтобы использовать ее для поиска элемента в *XML*-документе. Затем метод вызывает метод *FirstOrDefault* для выборки первого элемента из корневого элемента *XML*-документа, который содержит элементы *spectacle* с датой, соответствующей переданной дате. Если элемент найден, то метод возвращает его. Если элемент не найден, то метод выбрасывает исключение *ArgumentException*. Метод проверяет корректность переданных данных объекта *SpectacleModel.* Он создает временный *XML*-элемент, используя переданные данные, и добавляет к нему элементы *category* из словаря *Categories*, содержащего категории и их стоимости. Затем метод создает новый *XML*-документ на основе временного элемента, и использует метод *Validate* для проверки его соответствия схемам *XML*-документа, заданным в поле *schemas*. Если при проверке данных была обнаружена ошибка, метод выбрасывает исключение *ArgumentException* с сообщением об ошибке. Если ошибок не было обнаружено, метод возвращает *true*.

Также в классе находятся четыре метода для работы с жанрами спектаклей. Метод *GetGenreIdByName* получает на вход название жанра (представленное в виде строки) и ищет элемент в документе, который соответствует данному названию. Если элемент найден и его атрибут *id* может быть преобразован в целочисленное значение, то метод возвращает это значение в качестве идентификатора жанра. Если жанр не найден в базе данных, метод выбрасывает соответствующее исключение с сообщением.

Cтатический метод с именем *GetGenreNameById* принимает на вход целочисленный параметр *id*. Метод ищет в документе жанры (*genre*), ищет элемент, у которого атрибут *id* совпадает со значением, переданным в параметре *id*, и возвращает текстовое значение (*Value*) найденного элемента. Если жанр с заданным *id* не найден, метод выбрасывает исключение *ArgumentException* с сообщением об ошибке.

Метод *AddGenre* добавляет новый жанр в *XML*-документ, содержащий информацию о жанрах спектаклей. Сначала метод определяет максимальный идентификатор жанра среди уже существующих записей в документе. Затем создается новый элемент *genre* с указанным именем и новым идентификатором, который увеличивается на один от максимального значения. Этот элемент добавляется в корневой элемент документа. После этого происходит сохранение документа в файл.

Метод *GetAllGenres* возвращает список всех жанров, которые содержатся в файле с жанрами. В методе используется *LINQ to XML* для выборки элементов с тегом *genre* из корневого элемента *XML*-документа *genreDoc*. Затем из каждого элемента выбирается значение *Value* и добавляется в список. В итоге метод возвращает список всех жанров, представленных в *XML*-файле.

Статический класс *TicketManager* создан для взаимодействия с *XML*-файлом содержащем данные о билетах. Метод GetAll возвращает все билеты, представленные в *XML*-документе в виде списка объектов *TicketModel*. Для каждого элемента *ticket* метод создает новый объект *TicketModel*, заполняя его свойства значениями из соответствующих элементов *ticket* и *spectacle* в *XML*-документе. При заполнении свойства *Title* используется метод *ShowSpectacle* класса *SpectacleServices*, который возвращает объект *SpectacleModel* для спектакля, соответствующего дате, указанной в элементе *ticket*. Свойство *Category* заполняется на основе значения элемента *category* в *XML*-документе, которое преобразуется в значение перечисления *Categorias*. Полученные объекты T*icketModel* добавляются в результирующий список и возвращаются из метода.

Метод *Add* добавляет новый билет в *XML*-файл с информацией о покупателе, дате, категории, цене и присваивает уникальный идентификатор для нового билета. Если данные не проходят валидацию, выбрасывается исключение *ArgumentException*. После добавления билета *XML*-файл сохраняется.

Метод *Delete* удаляет из файла билет с указанным идентификатором *ticketId*. Если билет с таким идентификатором найден в файле, то он удаляется из дерева элементов XML и изменения сохраняются в файле. Если билет не найден, выбрасывается исключение с сообщением о том, что билет с указанным идентификатором не существует.

*DataValidate* осуществляет валидацию объекта типа *TicketModel* путем создания временного экземпляра типа *XElement*, заполненного данными из объекта *TicketModel*, и проверки его на соответствие заданной схеме *XSD* при помощи метода *Validate* класса *XDocument*. Если валидация прошла успешно, метод возвращает значение *true*, иначе – *false*.

Класс *UserManager* предназначен для обработки и управления данными о пользователях в соответствующем *XML*-файле. В классе находится семь методов для работы с базой данных.

Метод *GetAll* возвращает список всех пользователей из *XML*-документа. Он использует *LINQ to XML* для выборки элементов *user* из корневого элемента *XML*-документа, а затем создает и инициализирует объекты *UserModel* из элементов *user*. Каждый объект *UserModel* получает свои значения свойств из элементов *login*, *password* и *role*. Значение свойства *Role* является строкой, которую нужно преобразовать в соответствующее значение перечисления *Role*.

Метод *Add* добавляет нового пользователя в файл *XML* базы данных. Если данные о пользователе не проходят проверку на валидность или если пользователь с таким логином уже существует, выбрасывается исключение. Затем создается новый элемент *XML* с информацией о пользователе, а затем добавляется в корневой элемент документа. Документ сохраняется в файле. Важно отметить, что сохранение файла происходит через промежуточный *MemoryStream*, чтобы предотвратить запись поврежденного файла, если что-то пойдет не так во время сохранения.

Метод *Update* обновляет информацию о пользователе в *XML*-файле. Если переданный в метод объект *UserModel* содержит неверные данные, то выбрасывается исключение *ArgumentException.* Если пользователь с таким логином не существует, то также выбрасывается исключение *ArgumentException*. В противном случае метод ищет в *XML*-файле пользователя с переданным логином, обновляет его пароль и роль и сохраняет изменения в *XML*-файле.

Метод *Delete* удаляет пользователя из *XML*-файла по его логину. Сначала метод проверяет, что такой пользователь существует в базе, если пользователь не найден, то выбрасывается исключение. Если пользователь найден, то соответствующий элемент удаляется и изменения сохраняются в файле.

Метод *UserValid* проверяет, существует ли в *XML*-файле запись о пользователе с логином, указанным в параметре *user*. Если существует, метод возвращает *false*, иначе *true*.

Метод *DataValidate* производит валидацию данных пользователя *UserModel*, используя *XML*-схему. Сначала он создает новый экземпляр *XElement*, который содержит переданный в метод объект *UserModel*. Затем он создает новый экземпляр *XDocument*, содержащий только этот элемент, и вызывает метод *Validate*, передавая ему этот документ и обработчик ошибок. Если в результате валидации не было обнаружено ошибок, метод возвращает *true*, иначе – false.

Классы *SpectacleServices, TicketServices* и *UserServices* – это важная часть приложения, которая обеспечивает корректную работу методов графического интерфейса и классов, которые работают с соответствующими базами данных. Каждый из этих классов содержит методы для добавления, удаления и изменения соответствующих данных в базе данных. Без этих классов приложение было бы не в состоянии осуществлять взаимодействие с базой данных и предоставлять пользователю все необходимые функции. Например, *SpectacleServices* содержит методы для добавления новых спектаклей, удаления существующих спектаклей и обновления информации о существующих спектаклях. Аналогично, *TicketServices* и *UserServices* предоставляют методы для работы с билетами и пользователями соответственно.

Классы *Administrator, Guest* и *Registered* наследуются от класса *User* и реализуют соответствующие их правам методы. Базовый класс *User* содержит основные методы отображения спектаклей, так как по условию курсовой работы пользователи из любой категории могут просматривать информацию о спектаклях в том или ином виде. Для этого в классе были добавлены методы *ShowAllSpectacles* и *ShowSpectacle* из класса *SpectacleServices*.

Класс *Registered* расширяет функциональность базового класса *User*, предоставляя методы для работы с данными пользователя, такие как *UpdateUserData* для обновления личных данных, а также методы из класса *TicketServices* для добавления, удаления и обновления данных о билетах. Кроме того, класс Registered предоставляет метод *GetThisSpectacle* для просмотра более подробной информации о спектаклях. Эти методы доступны только для зарегистрированных пользователей с соответствующими правами доступа.

Класс *Administrator*, также наследуясь от класса *User*, имеет доступ к административной панели, которая позволяет управлять спектаклями, пользователями и билетами. В частности, у класса *Administrator* есть методы для добавления, редактирования и удаления спектаклей, пользователей и билетов, а также методы для генерации статистических отчетов.

Класс *Guest*, наследуясь от класса *User*, имеет возможность просматривать информацию о спектаклях, но не имеет доступа к функционалу регистрации, авторизации и работы с билетами.

Класс *UserFactory* представляет фабрику пользователей, которая создает объекты пользователей (*User*) в зависимости от их роли (*Role*). Конструктор класса *UserFactory* принимает три параметра типа *UserServices, SpectacleServices* и *TicketServices*, которые представляют сервисы для работы с пользователями, спектаклями и билетами соответственно. Метод *CreateUser* принимает объект *UserModel* и на основе его поля Role создает новый объект пользователя, соответствующий данной роли. Если роль пользователя – *Role.admin*, то создается объект типа *Administrator*, если *Role.registered*, то создается объект типа *Registered*, в противном случае создается объект типа *Guest*. Для созданных объектов пользователей задается логин, а также передаются сервисы *SpectacleServices, UserServices* и *TicketServices*, которые были переданы в конструктор класса.

Преимущества использования фабрики пользователей заключаются в том, что она позволяет создавать объекты пользователей с помощью одного метода, основываясь на определенном параметре (роли пользователя). Это делает код более гибким и удобным для дальнейшей разработки и поддержки, так как если в будущем необходимо добавить новую роль пользователя, достаточно будет расширить метод *CreateUser*, а не переписывать весь код. Кроме того, фабрика пользователей позволяет изолировать создание объектов пользователей от остального кода приложения, что способствует соблюдению принципа единственной ответственности и повышает его модульность и расширяемость. Также использование фабрики пользователей облегчает тестирование кода, так как при необходимости можно создавать объекты пользователей с разными ролями для проверки различных сценариев работы приложения.

## 2.4 Графический интерфейс пользователя

В программе находится три формы, которые описывают каждого из пользователей:

*– LoginForm;*

*– MainForm;*

*– RegistrForm;*

*– TicketBuyForm;*

*–UserTicketsForm.*

При первом запуске программы нас встречает окно гостя, так как у гостя ограниченные права доступа, то не зарегистрированный пользователь может только просмотреть представления которые будут проводится в этом театре. Гость может видеть:

– дату представления спектакля;

– время спектакля;

– количество мест в зале;

– количество проданных мест;

– название представления;

– автора произведения;

– тип представления.

Так же гость может воспользоваться поиском, чтобы найти интересующее его представление (рисунок 2.11).

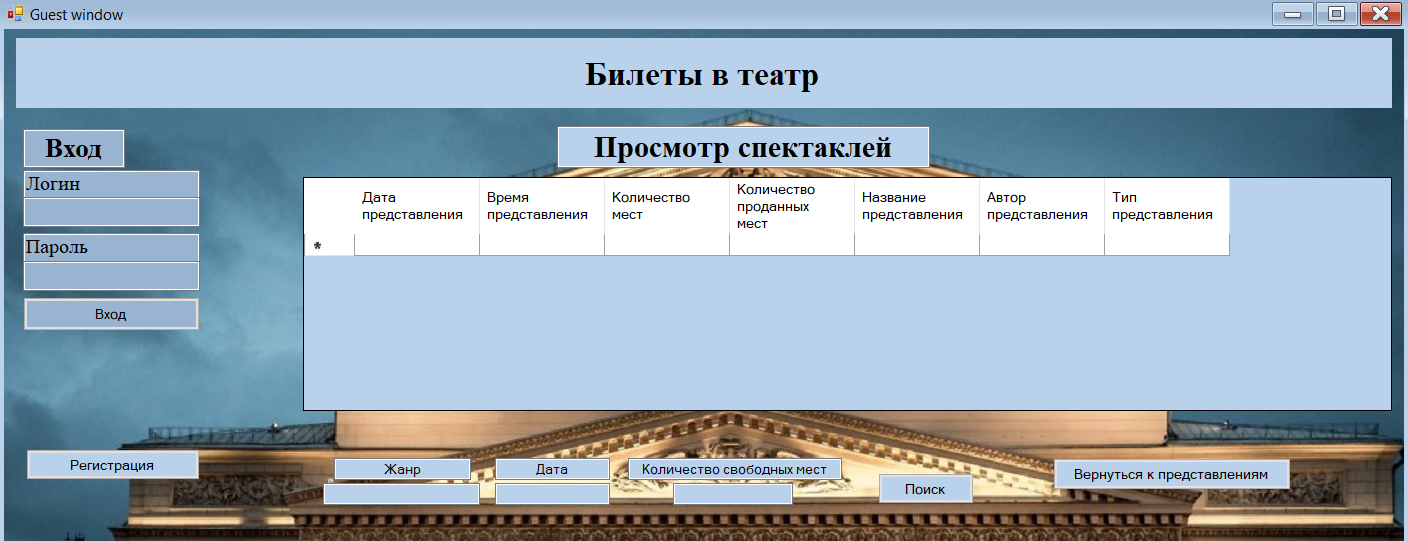


Рисунок 2.11 – Форма гостя

Если пользователь зарегистрирован в данном приложении, он может зайти в свой личный кабинет, а если не зарегистрирован, то может зарегистрироваться. Чтобы войти в свой профиль для этого он должен ввести свой логин и пароль. После перехода в личный кабинет зарегистрированного пользователя мы видим форму *UserForm* (рисунок 2.12)*.*

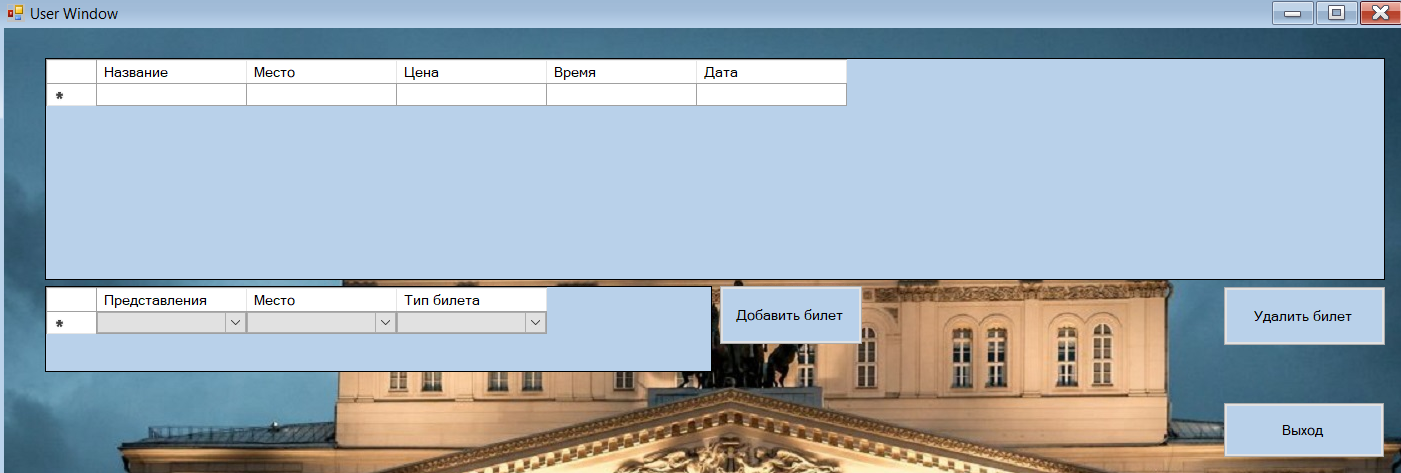


Рисунок 2.12 – Форма зарегистрированного пользователя

В личном кабинете пользователь может видеть свои заказанные билеты, а именно он видит:

– название представления;

– свое посадочное место;

– цену билета;

– время представления;

– дату представления.

Цена билета зависит от выбранного пользователем типа билета. Типы билетов бывают трех видов:

– балкон;

– партер;

– бельэтаж.

Партер – нижний этаж зрительного зала в театре с местами для публики в пространстве от сцены или от оркестра до противоположной стены или до амфитеатра.

Бельэтаж – места в зрительном зале, как правило, расположенные полукругом или по кривой линии, сзади и выше партера и амфитеатра. Иногда рассматриваются как балкон первого яруса театра.

Балкон – это места для зрителей, расположенные выше партера, в различных ярусах зрительного зала.

Пользователь не может заказать билет на место, если на это место уже купил билет другой пользователь или он сам. Так же для того чтобы купить билет пользователь должен ввести следующие данные:

– выбрать представление из доступных на которое он хочет купить билет;

– выбрать свободное место;

– выбрать тип билета из трех доступных.

Если пользователь не хочет идти на этот спектакль или у него не получается, то он может удалить свой билет. На этом возможности пользователя заканчиваются и если пользователю больше ничего не надо, он может выйти из аккаунта.

Остается только последняя форма, *AdminWindow.* Чтобы администратору зайти на свой профиль, ему так же как и пользователю нужно знать свой логин и пароль. После правильного ввода своих данных у администратора откроется его окно (рисунок 2.13).

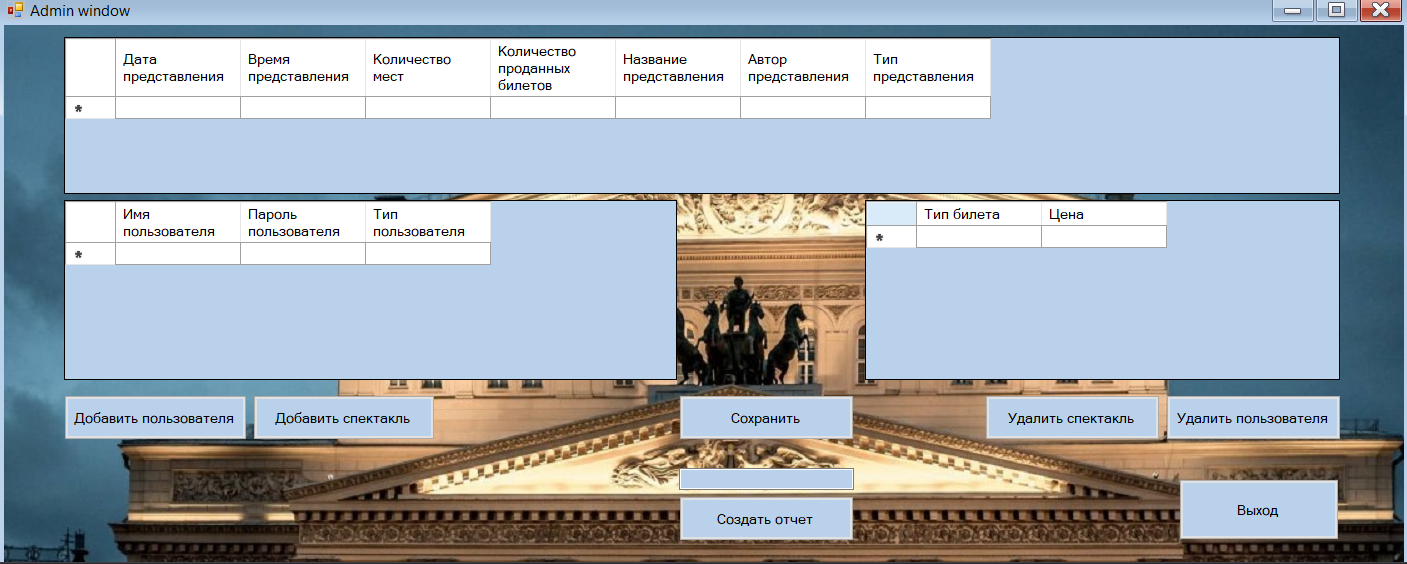


Рисунок 2.13 – Форма администратора

У администратора самый большой функционал возможностей. Администратор видит все представления которые проводятся и будут проводится в этом театре, а именно:

– дату представления;

– время представления;

– количество мест;

– количество проданных билетов;

– название представления;

– автора представления;

– тип представления.

Так же администратор видит всех пользователей, а в таблице всех пользователей находятся следующие поля:

– имя пользователя;

– пароль пользователя;

– тип пользователя.

Рядом с таблице всех пользователей находится таблица в которой находятся поля:

– тип билета;

– цена.

На все таблицы находящиеся в форме, администратор может непосредственно влиять, например:

– администратор может добавить нового пользователя и указать кем он будет являться, обычном пользователем или таким же администратором;

– администратор может удалить пользователя;

– так же является возможным изменить тип билета и цену.

– администратор может добавить новое представление и указать все необходимые данные;

– администратор может удалить спектакль;

– создавать отчеты за определенный день;

На этом функционал администратора заканчивается.

# 3 ВЕРИФИКАЦИЯ ПРИЛОЖЕНИЯ

## 3.1 Тестирование пользовательского интерфейса

Так же как любое приложение нуждается в тестировании, надо проверить и данную программу на работоспособность. Это можно сделать многими способами, одними из таких способов является:

– проверка при помощи модульных тестов;

– запуск программы в режиме отладки во время использования.

При запуске программы пользователя встречает окно гостя на рисунке А.1 (Приложение А).

Так как пользователь может выполнить поиск по интересующим его представлениям ему достаточно ввести данные в строки поиска, такие как:

– жанр;

– дату представления;

– количество свободных мест на спектакль.

После введенных данных пользователю нужно нажать на кнопку «Поиск». После ему будет показано представление по заданному запросу. Чтобы вернуться на главную ко всем представлениям, нужно нажать на кнопку «Вернуться к представлениям». Так как функционал пользователя ограничен, то ему нужно выйти из приложения или если он заинтересовался каким либо спектаклем и захотел приобрести билет на какое-то представление, то нажать на кнопку «Регистрация», после чего откроется окно регистрации на рисунке А.3 (Приложение А).

Когда пользователь ввел свои данные, ему нужно нажать кнопку «Зарегистрироваться», после чего пользователь станет зарегистрированным и сможет использовать функционал не только гостя, а так же и зарегистрированного пользователя.

Для того чтобы зайти в свой профиль ему нужно ввести свои данные, такие как логин и пароль, который он использовал при регистрации (рисунок 3.1).

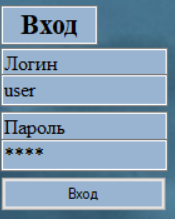


Рисунок 3.1 – Авторизация пользователя

На данном примере показано, что в качестве логина и пароля используется *user,* пароль показан в качестве символа \*, это сделано для конфиденциальности данных, чтобы другой пользователь не видел пароль и не смог зайти в профиль другого пользователя. Чтобы зайти в профиль достаточно нажать на кнопку «Вход».

После входа на аккаунт, у зарегистрированного пользователя повышается возможность действий, тут пользователь может видеть заказанные ранее билеты, добавить новые билеты на интересующие его представления или отменить заказ на рисунке А.2 (Приложение А).

Для того чтобы заказать билет пользователю нужно выбрать название представления на которое он бы хотел сходить, место, тип билета и нажать на кнопку добавить, после чего добавленный им билет отобразиться вместе с остальными билетами (рисунок 3.2).



Рисунок 3.2 – Добавление билета

Если же у пользователя пропал интерес к этому спектаклю или по каким-то причинам у него не получается прийти, он должен отменить билет, это можно сделать просто выбрав представление и нажать на кнопку «Удалить билет», после чего место которое было забронировано освободится, это позволит другим пользователям сделать заказ на это место. На этом возможность пользователя заканчиваются. Для того чтобы выйти со своего аккаунта пользователю необходимо нажать на кнопку «Выход».

Есть еще один тип пользователей который может пользоваться данной программой – администратор. У администратора самый большой функционал возможностей. Для того чтобы ему авторизироваться нужно знать свой логин и пароль (рисунок 3.3).

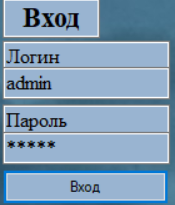


Рисунок 3.3 – Авторизация администратора

После входа на профиль администратора открывается окно администратора на рисунке А.4 (Приложение А).

После входа на аккаунт видим все возможности, администратор видит всех зарегистрированных пользователей, все спектакли и типы билетов, на все это администратор может влиять. Например, может добавлять новых пользователей и указать его тип, администратор или обычный пользователь, так же если пользователь вел себя не прилично, его можно удалить. Администратор может изменять тип билета и его цену. Так же может добавить новое представление, для этого нужно ввести все данные по представлению. Чтобы сохранить все внесенные изменения нужно нажать на кнопку «Сохранить». В возможности администратора входит еще создание отчетов, чтобы это сделать ему требуется ввести дату когда было проведено представление и нажать на кнопку «Создать отчет». На этом возможность администратора заканчиваются, для того чтобы выйти на главную форму нужно нажать на кнопку «Выход».

## 3.2 Модульное тестирование приложния

Данное приложение уже было протестировано с помощью отладки, теперь протестируем с помощью модульных тестов, это делается для того чтобы можно было протестировать каждый отдельный модуль программы (рисунок 3.4).

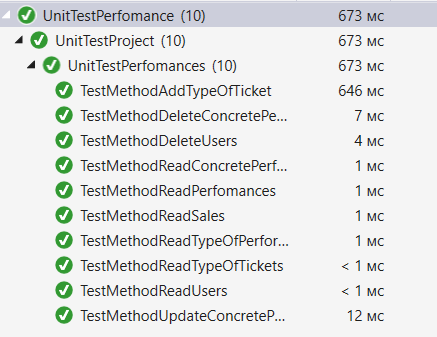


Рисунок 3.4 – Модульные тесты

На данном скриншоте видим как имеется один класс *UnitTestPerfomances* в котором тестируются десять методов. Например, в методе *TestMethodReadUsers* тестируется считывание данных из таблицы *users*. В методе *TestMethodDeleteUsers* тестируется удаление данных из таблицы *users*. В методе *TestMethodUpdateConcretePerformance* тестируется обновление данных в таблице *ConcretePerformance*. В методе *TestMethodAddTypeOfTicket* тестируется добавление данных в таблицу *TypeOfTicket.* В методе *TestMethodReadTypeOfTickets* тестируется считывание данных из таблицы *TypeOfTicket.* В методе *TestMethodReadTypeOfPerformance* тестируется считывание данных из таблицы *TypeOfPerformance.* В методе *TestMethodReadSales* тестируется считывание данных из таблицы *Sales.* Все десять методов помечены галочкой, это значит что тесты успешно прошли проверку и программа работает правильно.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта было разработано приложение для реализации продаж и учета проданных билетов на спектакли в театре. Приложение было разработано на языке *C#* с помощью СУБД *MySQL.* Для доступа к данным было использовано *ADO.NET.*

Результатом разработки курсового проекта, является приложение с графическим интерфейсом *WFA*. Приложение имеет многопользовательский доступ к системе, что решает проблему с целостностью данных, а также с безопасностью системы.

Приложение имеет простой интерфейс, который доступен любому пользователю. Это приложение пригодится тем людям, которые хотят сходить в театр на какое-нибудь представление, а так же театрам чтобы пользователи могли просматривать представления и заказывать билеты, приложение многофункционально и простое в использовании. В данном приложении пользователь может просматривать какие есть в наличии представления, чтобы облегчить поиск представлений была разработана кнопка поиска. Так же пользователь может добавить билет и удалить его. Администратор же может добавлять и удалять пользователей, изменять тип, цену представления, так же добавлять новые представления в театр. По результатам курсового проекта было видно, что все задачи выполнены и реализованы в полной мере.

# Список используемых источников

1. Практическое руководство к курсовому проектированию по курсу «Ин­форматика» для студентов технических специальностей дневной и заочной форм обучения – Гомель: ГГТУ им. П.О. Сухого, 2019. – 32 с.

2. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# – ООО Издательство «Питер», 2013. – 896 с.

3. Шилдт Герберт. C# 4.0: полное руководство: учебное пособие – ООО «И.Д. Вильямс», 2011. – 41 с.

4. Мухортов В.В., Рылов В.Ю. Объектно-ориентированное программирование, анализ и дизайн: Методическое пособие. – Новосибирск, издательство «ИМ СО РАН», 2002. – 108 с.

5. Крис Фиайли. Быстрый старт– Издательство «Москва», 2017. – 12 с.

6. MSDN [Электронный ресурс].: Объект Database (DAO) – Режим доступа: https://docs.microsoft.com/ru-ru/office/client-developer/access/desktop-database-reference/database-object-dao – Дата доступа: 12.03.2020

7. Metanit [Электронный ресурс].: Введение в ADO.NET – Режим доступа: https://metanit.com/sharp/adonet/1.1.php – Дата доступа: 12.03.2020

8. MSDN [Электронный ресурс].: Рекомендации по модульному тестированию для .NET Core и .NET Standard – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-best-practices – Дата доступа: 13.03.2020

9. Шварц Б., Зайцев П., Ткаченко В. Ш33 MySQL по максимуму. 3-е изд. - СПб.: Питер, 2018. - 864 с.: ил. - (Серия «Бестселлеры O'Reilly»).

10. Professor Web [Электронный ресурс].: Появление LINQ – Режим доступа: https://professorweb.ru/my/LINQ/base/level1/1\_1.php – Дата доступа: 15.03.2020

# ПРИЛОЖЕНИЕ А

(обязательное)

# Графические представления интерфейса программы

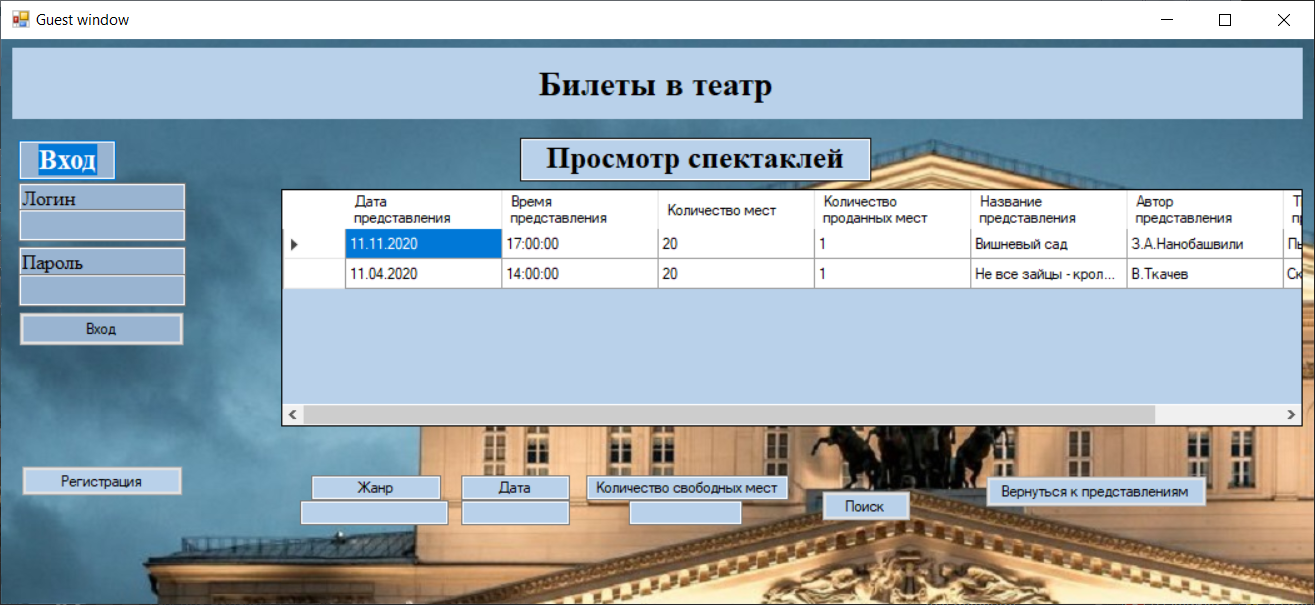


Рисунок А1 – Форма гостя

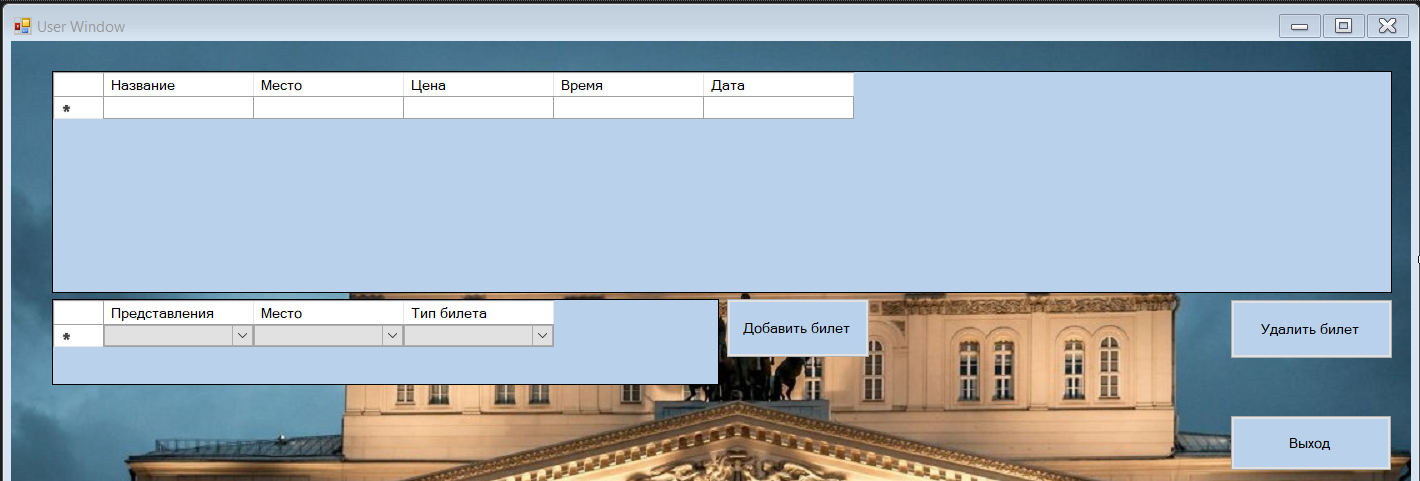


Рисунок А2 – Форма зарегистрированного пользователя

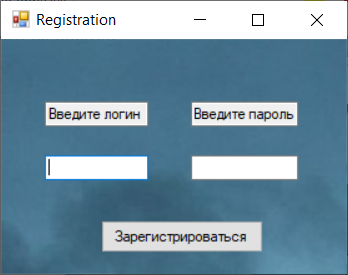


Рисунок А3 – Форма регистрации

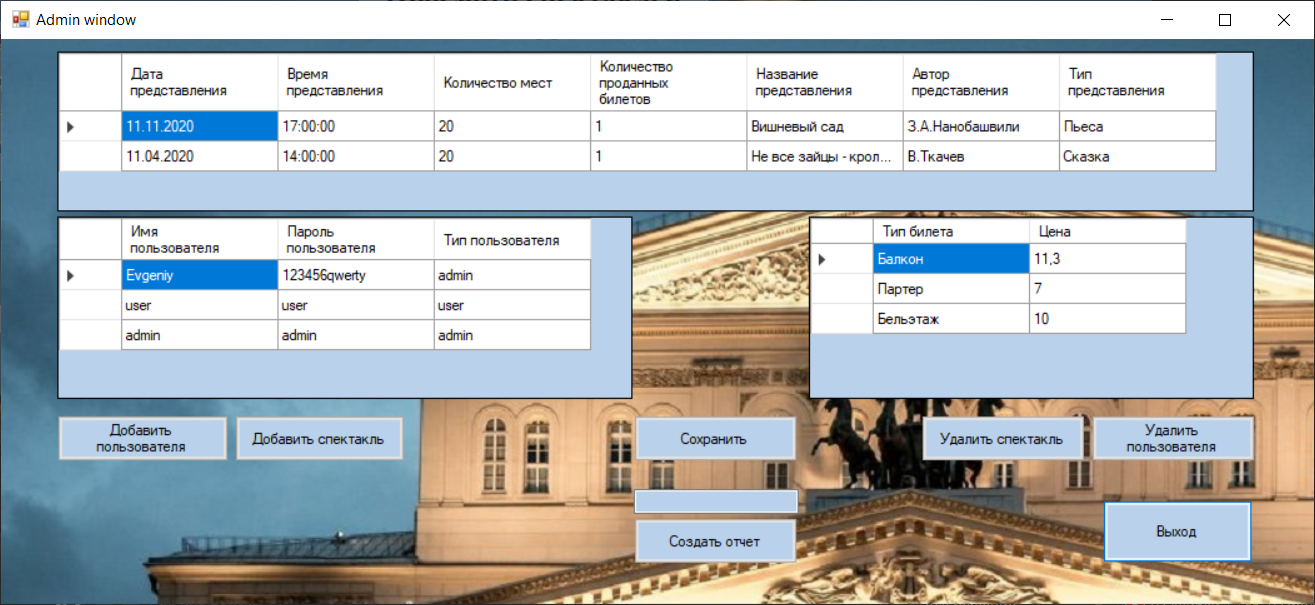


Рисунок А4 – Форма администратора

# ПРИЛОЖЕНИЕ Б

(обязательное)

# Листинг программы

ConcretePerformanceTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class ConcretePerformance

{

public int concretePerformanceId { get; set; }

public int performanceId { get; set; }

public DateTime dateOfConcretePerformance { get; set; }

public TimeSpan timeOfPerformance { get; set; }

public int count { get; set; }

public int countSold { get; set; }

public string title { get; set; }

public string author { get; set; }

public string typeName { get; set; }

}

}

PerfomancesTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class Perfomances

{

public int perfomancesId { get; set; }

public string title { get; set; }

public string author { get; set; }

public int typeOfPerformanceId { get; set; }

}

}

SalesTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class Sales

{

public int SalesId { get; set; }

public int ticketId { get; set; }

public int userId { get; set; }

public string Title { get; set; }

public int place { get; set; }

public float Cost { get; set; }

public TimeSpan TimeSpan { get; set; }

public string DateTime { get; set; }

}

}

TicketsTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class Tickets

{

public int ticketsId { get; set; }

public int typeOfTicketId { get; set; }

public int performanceId { get; set; }

public int place { get; set; }

public string Title { get; set; }

public float Cost { get; set; }

public TimeSpan TimeSpan { get; set; }

public string DateTime { get; set; }

}

}

TypeOfPerformanceTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class TypeOfPerformance

{

public int typeOfPerformanceId { get; set; }

public string typeName { get; set; }

}

}

TypeOfTicketsTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class TypeOfTickets

{

public int typeOfTicketsId { get; set; }

public string typeOfTickets { get; set; }

public float cost { get; set; }

}

}

UserTable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Table

{

public partial class User

{

public int userId { get; set; }

public string userName { get; set; }

public string userPassword { get; set; }

public string is\_admin { get; set; }

}

}

MyConnection

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public class Connection

{

protected string connectionString = ConfigurationManager.ConnectionStrings["mainsql"].ConnectionString;

}

}

ConcreteperformanceSQL

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public class ConcreteperformanceSQL: Connection

{

/// <summary>

/// Метод считывание данных таблицы "concreteperformance" из БД;

/// </summary>

/// <returns></returns>

public List<ConcretePerformance> ConcretePerformanceRead()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "use mainsql;" +

"select concreteperformance.concretePerformanceId, perfomances.perfomancesId, concreteperformance.dateOfConcretePerformance, " +

"concreteperformance.timeOfPerformance, concreteperformance.count, concreteperformance.countSold, " +

"perfomances.title, perfomances.author, typeofperformance.typeName from concreteperformance " +

"join perfomances on concreteperformance.performanceId = perfomances.perfomancesId " +

"join typeofperformance on perfomances.typeOfPerformanceId = typeofperformance.typeOfPerformanceId;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<ConcretePerformance> concretePerformances = new List<ConcretePerformance>();

if (reader.HasRows)

{

while (reader.Read())

{

concretePerformances.Add(new ConcretePerformance

{

concretePerformanceId = reader.GetInt32(0),

performanceId = reader.GetInt32(1),

dateOfConcretePerformance = reader.GetDateTime(2),

timeOfPerformance = reader.GetTimeSpan(3),

count = reader.GetInt32(4),

countSold = reader.GetInt32(5),

title = reader.GetString(6),

author = reader.GetString(7),

typeName = reader.GetString(8)

});

}

}

return concretePerformances;

}

}

/// <summary>

/// Метод удаления данных таблицы "concreteperformance" из БД

/// </summary>

/// <param name="id"></param>

public void ConcreteperformanceDelete(int id)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"delete from concreteperformance where concreteperformance.concretePerformanceId = {id};";

MySqlCommand command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

}

}

/// <summary>

/// Метод обновления информации в БД (Concreteperformance);

/// </summary>

/// <param name="concretePerformances"></param>

public void ConcreteperformanceUpdate(List<ConcretePerformance> concretePerformances)

{

int dbCount = ConcretePerformanceRead().Count;

// Если есть новые спектакли;

if (concretePerformances.Count > ConcretePerformanceRead().Count)

{

ConcreteperformanceAdd(concretePerformances, dbCount);

}

else

{

// Обновляю записи;

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql;

MySqlCommand command;

foreach (ConcretePerformance concretePerformance in concretePerformances)

{

if (string.IsNullOrEmpty(concretePerformance.dateOfConcretePerformance.ToString()) ||

string.IsNullOrEmpty(concretePerformance.timeOfPerformance.ToString()) ||

string.IsNullOrEmpty(concretePerformance.count.ToString()) ||

string.IsNullOrEmpty(concretePerformance.countSold.ToString()))

throw new Exception();

else

{

sql = $"update concreteperformance set dateOfConcretePerformance = '{concretePerformance.dateOfConcretePerformance.ToString("dd.MM.yyyy")}'," +

$" timeOfPerformance = '{concretePerformance.timeOfPerformance.ToString()}', " +

$"count = '{concretePerformance.count.ToString()}', countSold = '{concretePerformance.countSold.ToString()}' " +

$"where concreteperformance.concretePerformanceId = {concretePerformance.concretePerformanceId};";

command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

}

}

}

}

}

/// <summary>

/// Метод добавления информации в таблицу "concreteperformance"

/// </summary>

/// <param name="concretePerformances"></param>

/// <param name="dbCount"></param>

public void ConcreteperformanceAdd(List<ConcretePerformance> concretePerformances, int dbCount)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into concreteperformance (performanceId, dateOfConcretePerformance, timeOfPerformance, count, countSold) values ";

for (int i = dbCount; i < concretePerformances.Count; i++)

{

if (string.IsNullOrEmpty(concretePerformances[i].dateOfConcretePerformance.ToString()) || string.IsNullOrEmpty(concretePerformances[i].timeOfPerformance.ToString()) || string.IsNullOrEmpty(concretePerformances[i].count.ToString()) || string.IsNullOrEmpty(concretePerformances[i].countSold.ToString()))

throw new Exception();

else

{

sql += $"({concretePerformances[i].performanceId}, '{concretePerformances[i].dateOfConcretePerformance.ToString("d")}', '{concretePerformances[i].timeOfPerformance}', '{concretePerformances[i].count}', '{concretePerformances[i].countSold}')";

if (i + 1 == concretePerformances.Count)

sql += ";";

else

sql += ", ";

}

}

MySqlCommand command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

}

}

/// <summary>

/// Метод считывания данных по типу билета, дате, месту из БД

/// </summary>

/// <param name="typeName"></param>

/// <param name="date"></param>

/// <param name="place"></param>

/// <returns></returns>

public List<ConcretePerformance> GetByDateAndName(string typeName, DateTime date, int place)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = "use mainsql;" +

"select concreteperformance.concretePerformanceId, perfomances.perfomancesId, concreteperformance.dateOfConcretePerformance, " +

"concreteperformance.timeOfPerformance, concreteperformance.count, concreteperformance.countSold, " +

"perfomances.title, perfomances.author, typeofperformance.typeName from concreteperformance " +

"join perfomances on concreteperformance.concretePerformanceId = perfomances.perfomancesId " +

"join typeofperformance on perfomances.perfomancesId = typeofperformance.typeOfPerformanceId " +

"where concreteperformance.dateOfConcretePerformance = @date and typeofperformance.typeName = @typeName and concreteperformance.count - concreteperformance.countSold = @place;";

MySqlCommand command = new MySqlCommand(sql, connection);

MySqlParameter dateof = new MySqlParameter("@date", date);

command.Parameters.Add(dateof);

MySqlParameter idof = new MySqlParameter("@typeName", typeName);

command.Parameters.Add(idof);

MySqlParameter plof = new MySqlParameter("@place", place);

command.Parameters.Add(plof);

List<ConcretePerformance> concretePerformances = new List<ConcretePerformance>();

MySqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

concretePerformances.Add(new ConcretePerformance

{

concretePerformanceId = reader.GetInt32(0),

performanceId = reader.GetInt32(1),

dateOfConcretePerformance = reader.GetDateTime(2),

timeOfPerformance = reader.GetTimeSpan(3),

count = reader.GetInt32(4),

countSold = reader.GetInt32(5),

title = reader.GetString(6),

author = reader.GetString(7),

typeName = reader.GetString(8)

});

}

}

return concretePerformances;

}

}

}

}

PerfomanceSQL

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public partial class PerfomanceSQL: Connection

{

/// <summary>

/// Метод считывание данных таблицы "perfomance" из БД;

/// </summary>

/// <returns></returns>

public List<Perfomances> PerfomanceRead()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select \* from perfomances ";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<Perfomances> perfomances = new List<Perfomances>();

if (reader.HasRows)

{

while (reader.Read())

{

perfomances.Add(new Perfomances

{

perfomancesId = reader.GetInt32(0),

title = reader.GetString(1),

author = reader.GetString(2),

typeOfPerformanceId = reader.GetInt32(3)

});

}

}

return perfomances;

}

}

/// <summary>

/// Метод добавления информации в таблицу "perfomances"

/// </summary>

/// <param name="title"></param>

/// <param name="typeOfPerformanceId"></param>

/// <param name="author"></param>

public void PerfomancesAdd(string title, int typeOfPerformanceId, string author)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into perfomances (typeOfPerformanceId, title, author) values (@typeOfPerformanceid, @Title, @Author);";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@typeOfPerformanceid", typeOfPerformanceId));

command.Parameters.Add(new MySqlParameter("@Title", title));

command.Parameters.Add(new MySqlParameter("@Author", author));

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

}

}

}

}

SalesSQL

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public partial class SalesSQL: Connection

{

/// <summary>

/// Метод считывание данных таблицы "sales" из БД;

/// </summary>

/// <returns></returns>

public List<Sales> SalesRead()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select \* from sales " +

"join users on sales.userId = users.userId " +

";";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<Sales> sales = new List<Sales>();

if (reader.HasRows)

{

while (reader.Read())

{

sales.Add(new Sales

{

SalesId = reader.GetInt32(0),

ticketId = reader.GetInt32(1),

userId = reader.GetInt32(2)

});

}

}

return sales;

}

}

/// <summary>

/// Метод удаления данных таблицы "sales" из БД

/// </summary>

/// <param name="id"></param>

public void SalesDelete(int id, TicketSQL ticketSQL, string title)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "delete from sales where SalesId = @id;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlParameter dateid = new MySqlParameter("@id", id);

command.Parameters.Add(dateid);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

sql = $"update concreteperformance set countSold = countSold - 1 where concreteperformance.concretePerformanceId = @id;";

command = new MySqlCommand(sql, my);

id = ticketSQL.GetPerfromanceId(title);

command.Parameters.Add(new MySqlParameter("@id", id));

number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

}

}

/// <summary>

/// Метод считывания конкретного пользователя по продажам билетов

/// </summary>

/// <param name="userid"></param>

/// <returns></returns>

public List<Sales> GetConcreteUserSales(int userid)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select tickets.ticketsId, perfomances.title, typeoftickets.cost, concreteperformance.timeOfPerformance, concreteperformance.dateOfConcretePerformance, tickets.place, sales.salesid from tickets " +

"join typeoftickets on tickets.typeOfTicketId = typeoftickets.typeOfTicketsId " +

"join concreteperformance on tickets.performanceId = concreteperformance.concretePerformanceId " +

"join perfomances on concreteperformance.PerformanceId = perfomances.perfomancesId " +

"join sales on sales.ticketId = tickets.ticketsId " +

"where sales.userId = @id;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlParameter dateid = new MySqlParameter("@id", userid);

command.Parameters.Add(dateid);

MySqlDataReader reader = command.ExecuteReader();

List<Sales> sales = new List<Sales>();

if (reader.HasRows)

{

while (reader.Read())

{

sales.Add(new Sales

{

ticketId = reader.GetInt32(0),

Title = reader.GetString(1),

Cost = reader.GetFloat(2),

TimeSpan = reader.GetTimeSpan(3),

DateTime = reader.GetDateTime(4).ToString("d"),

place = reader.GetInt32(5),

SalesId = reader.GetInt32(6)

});

}

}

return sales;

}

}

/// <summary>

/// Метод добавления информации в таблицу "sales"

/// </summary>

/// <param name="title"></param>

/// <param name="place"></param>

/// <param name="userid"></param>

/// <param name="ticketSQL"></param>

public void SalesAdd(string title, int place, int userid, TicketSQL ticketSQL)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into sales (ticketId, userId) values (@titcketid, @userid);";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@titcketid", GetTicketId(title, place)));

command.Parameters.Add(new MySqlParameter("@userid", userid));

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

sql = $"update concreteperformance set countSold = countSold + 1 where concreteperformance.concretePerformanceId = @id;";

command = new MySqlCommand(sql, connection);

int id = ticketSQL.GetPerfromanceId(title);

command.Parameters.Add(new MySqlParameter("@id", id));

number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

}

}

/// <summary>

/// Метод считывания билета по id

/// </summary>

/// <param name="title"></param>

/// <param name="place"></param>

/// <returns></returns>

public int GetTicketId(string title, int place)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"select tickets.ticketsId from tickets " +

"join concreteperformance on tickets.performanceId = concreteperformance.concretePerformanceId " +

"join perfomances on concreteperformance.performanceId = perfomances.perfomancesId " +

"where perfomances.title = @title and tickets.place = @place;";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@title", title));

command.Parameters.Add(new MySqlParameter("@place", place));

MySqlDataReader reader = command.ExecuteReader();

int id = -1;

if (reader.HasRows)

{

while (reader.Read())

{

id = reader.GetInt32(0);

}

}

return id;

}

}

/// <summary>

/// Метод удаления данных таблицы "sales" из БД

/// </summary>

/// <param name="id"></param>

public void SalesDel(int id)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "delete \* from sales where SalesId = @id;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlParameter dateid = new MySqlParameter("@id", id);

command.Parameters.Add(dateid);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

}

}

}

}

TicketSQL

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public partial class TicketSQL: Connection

{

/// <summary>

/// Метод считывание данных таблицы "tickets" из БД;

/// </summary>

/// <returns></returns>

public List<Tickets> ReadAllTickets()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select tickets.ticketsId, typeoftickets.typeOfTicketsId, typeoftickets.cost, concreteperformance.timeOfPerformance, concreteperformance.dateOfConcretePerformance from tickets " +

"join typeoftickets on tickets.typeOfTicketId = typeoftickets.typeOfTicketsId " +

"join concreteperformance on tickets.performanceId = concreteperformance.concretePerformanceId;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

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

if (reader.HasRows)

{

while (reader.Read())

{

tickets.Add(new Tickets

{

ticketsId = reader.GetInt32(0),

typeOfTicketId = reader.GetInt32(1),

performanceId = reader.GetInt32(2),

place = reader.GetInt32(3)

});

}

}

return tickets;

}

}

/// <summary>

/// Метод считывания конкретного пользователя по билету

/// </summary>

/// <param name="username"></param>

/// <returns></returns>

public List<Tickets> GetConcreteUserTickets(string username)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select tickets.ticketsId, perfomances.title, typeoftickets.cost, concreteperformance.timeOfPerformance, concreteperformance.dateOfConcretePerformance, tickets.place from tickets " +

"join typeoftickets on tickets.typeOfTicketId = typeoftickets.typeOfTicketsId " +

"join concreteperformance on tickets.performanceId = concreteperformance.concretePerformanceId " +

"join perfomances on concreteperformance.PerformanceId = perfomances.perfomancesId";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

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

if (reader.HasRows)

{

while (reader.Read())

{

tickets.Add(new Tickets

{

ticketsId = reader.GetInt32(0),

Title = reader.GetString(1),

Cost = reader.GetFloat(2),

TimeSpan = reader.GetTimeSpan(3),

DateTime = reader.GetDateTime(4).ToString("d"),

place = reader.GetInt32(5)

});

}

}

return tickets;

}

}

/// <summary>

/// Метод добавления информации в таблицу "tickets"

/// </summary>

/// <param name="title"></param>

/// <param name="place"></param>

/// <param name="typeOfTicket"></param>

public void AddTicket(string title, int place, string typeOfTicket)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into tickets (performanceId, place, typeOfTicketId) values (@title, @place, @typeOfTicket)";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@title", GetPerfromanceId(title)));

command.Parameters.Add(new MySqlParameter("@place", place));

command.Parameters.Add(new MySqlParameter("@typeOfTicket", GetTypeOfTicketId(typeOfTicket)));

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

}

}

/// <summary>

/// Метод считывания представления по id

/// </summary>

/// <param name="title"></param>

/// <returns></returns>

public int GetPerfromanceId(string title)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"select concreteperformance.concretePerformanceId from concreteperformance " +

"join perfomances on concreteperformance.performanceId = perfomances.perfomancesId " +

"where perfomances.title = @title;";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@title", title));

MySqlDataReader reader = command.ExecuteReader();

int id = -1;

if (reader.HasRows)

{

while (reader.Read())

{

id = reader.GetInt32(0);

}

}

return id;

}

}

/// <summary>

/// Метод считывания тип билета по id

/// </summary>

/// <param name="typeOfTicket"></param>

/// <returns></returns>

public int GetTypeOfTicketId(string typeOfTicket)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"select typeOfTicketsId from typeoftickets " +

"where typeOfTickets = @typeOfTickets;";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@typeOfTickets", typeOfTicket));

MySqlDataReader reader = command.ExecuteReader();

int id = -1;

if (reader.HasRows)

{

while (reader.Read())

{

id = reader.GetInt32(0);

}

}

return id;

}

}

/// <summary>

/// Метод считывание данных таблицы "tickets" из БД;

/// </summary>

/// <param name="title"></param>

/// <returns></returns>

public List<Tickets> ReadTicekts(string title)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"select \* from tickets where tickets.performanceId = @performanceId";

MySqlCommand command = new MySqlCommand(sql, connection);

command.Parameters.Add(new MySqlParameter("@performanceId", GetPerfromanceId(title)));

MySqlDataReader reader = command.ExecuteReader();

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

if (reader.HasRows)

{

while (reader.Read())

{

tickets.Add(new Tickets

{

ticketsId = reader.GetInt32(0),

typeOfTicketId = reader.GetInt32(1),

performanceId = reader.GetInt32(2),

place = reader.GetInt32(3)

});

}

}

return tickets;

}

}

/// <summary>

/// Метод удаления данных таблицы "tickets" из БД

/// </summary>

/// <param name="id"></param>

public void TicketDelete(int id)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "delete from tickets where ticketsId = @id;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlParameter dateid = new MySqlParameter("@id", id);

command.Parameters.Add(dateid);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

}

}

}

}

TypeofperformanceSQL

using System;

using System.Collections.Generic;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public partial class TypeofperformanceSQL: Connection

{

/// <summary>

/// Метод считывания таблицы "typeofperformance" из БД;

/// </summary>

/// <returns></returns>

public List<TypeOfPerformance> TypeOfPerformanceRead()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select \* from typeofperformance;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<TypeOfPerformance> typeOfPerformances = new List<TypeOfPerformance>();

if (reader.HasRows)

{

while (reader.Read())

{

typeOfPerformances.Add(new TypeOfPerformance

{

typeOfPerformanceId = reader.GetInt32(0),

typeName = reader.GetString(1),

});

}

}

return typeOfPerformances;

}

}

/// <summary>

/// Вставляет новую запись в таблицу typeofperformance(добавление)

/// </summary>

/// <param name="typeName"></param>

public void Insert(string typeName)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "insert into typeofperformance(typeName) values (@typeName);";

MySqlCommand command = new MySqlCommand(sql, my);

command.Parameters.Add(new MySqlParameter("@typeName", typeName));

int num = command.ExecuteNonQuery();

}

}

}

}

TypeofticketSQL

using System;

using System.Collections.Generic;

using System.Globalization;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public partial class TypeofticketSQL: Connection

{

/// <summary>

/// Метод считывания таблицы "typeOfTickets" из БД;

/// </summary>

/// <returns></returns>

public List<TypeOfTickets> TypeOfTicketRead()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select \* from typeOfTickets;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<TypeOfTickets> typeOfTickets = new List<TypeOfTickets>();

if (reader.HasRows)

{

while (reader.Read())

{

typeOfTickets.Add(new TypeOfTickets

{

typeOfTicketsId = reader.GetInt32(0),

typeOfTickets = reader.GetString(1),

cost = reader.GetFloat(2),

});

}

}

return typeOfTickets;

}

}

/// <summary>

/// Метод обновления информации в БД (Typeofticket);

/// </summary>

/// <param name="typeoftickets"></param>

public void TypeOfTicketUpdate(List<TypeOfTickets> typeOfTickets)

{

int dbCount = TypeOfTicketRead().Count;

// Если есть новые пользователи;

if (typeOfTickets.Count > TypeOfTicketRead().Count)

{

TypeOfTicketAdd(typeOfTickets, dbCount);

}

else

{

// Обновляю записи;

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql;

MySqlCommand command;

foreach (TypeOfTickets typeOfTicket in typeOfTickets)

{

if (string.IsNullOrEmpty(typeOfTicket.typeOfTickets) || string.IsNullOrEmpty(typeOfTicket.cost.ToString()))

throw new Exception();

else

{

sql = $"update typeoftickets set typeOfTickets = '{typeOfTicket.typeOfTickets}', cost = {typeOfTicket.cost.ToString("G", CultureInfo.InvariantCulture) } where typeoftickets.typeOfTicketsId = {typeOfTicket.typeOfTicketsId};";

command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

}

}

}

}

}

/// <summary>

/// Метод добавления информации в таблицу "typeoftickets"

/// </summary>

/// <param name="typeOfTicket"></param>

/// <param name="dbCount"></param>

public void TypeOfTicketAdd(List<TypeOfTickets> typeOfTicket, int dbCount)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into typeoftickets (typeOfTickets, cost) values ";

for (int i = dbCount; i < typeOfTicket.Count; i++)

{

if (string.IsNullOrEmpty(typeOfTicket[i].typeOfTickets) || string.IsNullOrEmpty(typeOfTicket[i].cost.ToString()))

throw new Exception();

else

{

sql += $"('{typeOfTicket[i].typeOfTickets}', '{typeOfTicket[i].cost}')";

if (i + 1 == typeOfTicket.Count)

sql += ";";

else

sql += ", ";

}

}

MySqlCommand command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception();

}

}

}

}

}

UserSQL

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Diagnostics.Eventing.Reader;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public partial class UserSQL : Connection

{

/// <summary>

/// Метод получения зарегистрированного пользователя;

/// </summary>

/// <param name="login"></param>

/// <param name="password"></param>

/// <returns></returns>

public User IsWho(string login, string password)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = "select \* from users where users.userName = @login and users.userPassword = @password;";

MySqlCommand command = new MySqlCommand(sql, connection);

MySqlParameter loginname = new MySqlParameter("@login", login);

command.Parameters.Add(loginname);

MySqlParameter passwordname = new MySqlParameter("@password", password);

command.Parameters.Add(passwordname);

MySqlDataReader reader = command.ExecuteReader();

User user = null;

if (reader.HasRows)

{

while (reader.Read())

{

user = new User

{

userId = reader.GetInt32(0),

userName = reader.GetString(1),

userPassword = reader.GetString(2),

is\_admin = reader.GetString(3),

};

}

}

return user;

}

}

/// <summary>

/// Метод считывания таблицы "users" из БД;

/// </summary>

/// <returns></returns>

public List<User> UserRead()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select \* from users;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<User> users = new List<User>();

if (reader.HasRows)

{

while (reader.Read())

{

users.Add(new User

{

userId = reader.GetInt32(0),

userName = reader.GetString(1),

userPassword = reader.GetString(2),

is\_admin = reader.GetString(3),

});

}

}

return users;

}

}

/// <summary>

/// Метод удаления данных таблицы "users" из БД

/// </summary>

/// <param name="id"></param>

public void DeleteUser(int id)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"delete from users where users.userId = {id};";

MySqlCommand command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception("Перед тем как удалить пользователя, добавьте его в базу данных, нажав кнопку 'Сохранить'");

}

}

/// <summary>

/// Метод обновления информации в БД (users);

/// </summary>

/// <param name="users"></param>

public void UpdateUser(List<User> users)

{

int dbCount = UserRead().Count;

// Если есть новые пользователи;

if (users.Count > UserRead().Count)

{

AddUser(users, dbCount);

}

else

{

// Обновляю записи;

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql;

MySqlCommand command;

foreach (User user in users)

{

if (string.IsNullOrEmpty(user.userName) || string.IsNullOrEmpty(user.userPassword) || string.IsNullOrEmpty(user.is\_admin))

throw new Exception();

else

{

sql = $"update users set userName = '{user.userName}', userPassword = '{user.userPassword}', is\_admin = '{user.is\_admin}' where users.userId = {user.userId}";

command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

throw new Exception();

}

}

}

}

}

/// <summary>

/// Метод добавления информации в таблицу "users"

/// </summary>

/// <param name="users"></param>

/// <param name="dbCount"></param>

public void AddUser(List<User> users, int dbCount)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into users (userName, userPassword, is\_admin) values ";

for (int i = dbCount; i < users.Count; i++)

{

if (string.IsNullOrEmpty(users[i].userName) || string.IsNullOrEmpty(users[i].userPassword) || string.IsNullOrEmpty(users[i].is\_admin))

throw new Exception("Перед тем как удалить пользователя, добавьте его в базу данных, нажав кнопку 'Сохранить' " +

"или чтобы добавить пользователя нужно ввести его данные");

else

{

sql += $"('{users[i].userName}', '{users[i].userPassword}', '{users[i].is\_admin}')";

if (i + 1 == users.Count)

sql += ";";

else

sql += ", ";

}

}

MySqlCommand command = new MySqlCommand(sql, connection);

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception("такой пользователь уже существует");

}

}

}

/// <summary>

/// Метод добавления нового пользователя в таблицу "users"

/// </summary>

/// <param name="login"></param>

/// <param name="pass"></param>

public void AddNewUser(string login, string pass)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string sql = $"insert into users (userName, userPassword, is\_admin) values (@username, @password, 'user')";

MySqlCommand command = new MySqlCommand(sql, connection);

MySqlParameter username = new MySqlParameter("@username", login);

command.Parameters.Add(username);

MySqlParameter password = new MySqlParameter("@password", pass);

command.Parameters.Add(password);

int number = command.ExecuteNonQuery();

if (number == 0)

{

throw new Exception("Такой пользователь уже существует");

}

}

}

}

}

GetReport

using Main\_Project.Reports;

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.SQL

{

public class GetReport : Connection

{

/// <summary>

/// Метод получения информации о проданных билетов по кол-ву проданных мест и названию спектакля

/// </summary>

/// <returns></returns>

public List<TicketReport> GetReportSales()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select concreteperformance.countSold, perfomances.title from concreteperformance " +

"join perfomances on concreteperformance.concretePerformanceId = perfomances.perfomancesId;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<TicketReport> ticketReports = new List<TicketReport>();

if (reader.HasRows)

{

while (reader.Read())

{

ticketReports.Add(new TicketReport { countSold = reader.GetInt32(0), title = reader.GetString(1) });

}

}

else

throw new Exception();

return ticketReports;

}

}

/// <summary>

/// Метод получения информации о проданных билетов по типу билетов и цене

/// </summary>

/// <returns></returns>

public List<TypeOfTicketsReport> GetTypeOfTicketsReport()

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select typeoftickets.typeOfTickets, typeoftickets.cost from typeoftickets;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlDataReader reader = command.ExecuteReader();

List<TypeOfTicketsReport> typeOfTicketsReports = new List<TypeOfTicketsReport>();

if (reader.HasRows)

{

while (reader.Read())

{

typeOfTicketsReports.Add(new TypeOfTicketsReport { typeOfTickets = reader.GetString(0), cost = reader.GetFloat(1) });

}

}

else

throw new Exception();

return typeOfTicketsReports;

}

}

/// <summary>

/// Метод получения информации о проданных билетов по дате, типу спектакля и по кол-ву мест

/// </summary>

/// <param name="date"></param>

/// <returns></returns>

public List<PlaceAndDateReports> GetPlaceAndDateReports(DateTime date)

{

using (MySqlConnection my = new MySqlConnection(connectionString))

{

my.Open();

string sql = "select count(sales.SalesId), concreteperformance.dateOfConcretePerformance, typeoftickets.typeOfTickets from sales " +

"join tickets on tickets.ticketsId = sales.ticketId " +

"join typeoftickets on typeoftickets.typeofticketsId = tickets.typeOfTicketId " +

"join concreteperformance on concreteperformance.concretePerformanceId = tickets.performanceId " +

"where typeoftickets.typeOfTickets = 'Балкон' and concreteperformance.dateOfConcretePerformance = @date;";

MySqlCommand command = new MySqlCommand(sql, my);

MySqlParameter dateof = new MySqlParameter("@date", date);

command.Parameters.Add(dateof);

MySqlDataReader reader = command.ExecuteReader();

List<PlaceAndDateReports> placeAndDateReports = new List<PlaceAndDateReports>();

if (reader.HasRows)

{

while (reader.Read())

{

if (reader.GetInt32(0) == 0)

{

placeAndDateReports.Add(new PlaceAndDateReports

{

dateOfConcretePerformance = date,

typeOfTickets = "Балкон",

count = 0

});

}

else

{

placeAndDateReports.Add(new PlaceAndDateReports

{

dateOfConcretePerformance = reader.GetDateTime(1),

typeOfTickets = reader.GetString(2),

count = reader.GetInt32(0)

});

}

}

}

else

throw new Exception();

my.Close();

my.Open();

sql = "select count(sales.SalesId), concreteperformance.dateOfConcretePerformance, typeoftickets.typeOfTickets from sales " +

"join tickets on tickets.ticketsId = sales.ticketId " +

"join typeoftickets on typeoftickets.typeofticketsId = tickets.typeOfTicketId " +

"join concreteperformance on concreteperformance.concretePerformanceId = tickets.performanceId " +

"where typeoftickets.typeOfTickets = 'Партер' and concreteperformance.dateOfConcretePerformance = @date;";

command = new MySqlCommand(sql, my);

dateof = new MySqlParameter("@date", date);

command.Parameters.Add(dateof);

reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

if (reader.GetInt32(0) == 0)

{

placeAndDateReports.Add(new PlaceAndDateReports

{

dateOfConcretePerformance = date,

typeOfTickets = "Партер",

count = 0

});

}

else

{

placeAndDateReports.Add(new PlaceAndDateReports

{

dateOfConcretePerformance = reader.GetDateTime(1),

typeOfTickets = reader.GetString(2),

count = reader.GetInt32(0)

});

}

}

}

my.Close();

my.Open();

sql = "select count(sales.SalesId), concreteperformance.dateOfConcretePerformance, typeoftickets.typeOfTickets from sales " +

"join tickets on tickets.ticketsId = sales.ticketId " +

"join typeoftickets on typeoftickets.typeofticketsId = tickets.typeOfTicketId " +

"join concreteperformance on concreteperformance.concretePerformanceId = tickets.performanceId " +

"where typeoftickets.typeOfTickets = 'Бельэтаж' and concreteperformance.dateOfConcretePerformance = @date;";

command = new MySqlCommand(sql, my);

dateof = new MySqlParameter("@date", date);

command.Parameters.Add(dateof);

reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

if (reader.GetInt32(0) == 0)

{

placeAndDateReports.Add(new PlaceAndDateReports

{

dateOfConcretePerformance = date,

typeOfTickets = "Бельэтаж",

count = 0

});

}

else

{

placeAndDateReports.Add(new PlaceAndDateReports

{

dateOfConcretePerformance = reader.GetDateTime(1),

typeOfTickets = reader.GetString(2),

count = reader.GetInt32(0)

});

}

}

}

return placeAndDateReports;

}

}

}

}

PlaceAndDateReports

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Reports

{

public class PlaceAndDateReports: Connection

{

public DateTime dateOfConcretePerformance { get; set; }

public string typeOfTickets { get; set; }

public int count { get; set; }

}

}

ReportXML

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

using Main\_Project.SQL;

namespace Main\_Project.Reports

{

public class ReportXML : Connection

{

public void SaveReportsXML(List<TicketReport> ticketReports)

{

XmlSerializer formatter = new XmlSerializer(typeof(List<TicketReport>));

using (FileStream fs = new FileStream(@"SalesReports.xml", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, ticketReports);

}

}

public void SaveTypeOfTicketsReport(List<TypeOfTicketsReport> typeOfTicketsReports)

{

XmlSerializer formatter = new XmlSerializer(typeof(List<TypeOfTicketsReport>));

using (FileStream fs = new FileStream(@"TypeOfTicketsReport.xml", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, typeOfTicketsReports);

}

}

public void SavePlaceAndDateReports(List<PlaceAndDateReports> placeAndDateReports)

{

XmlSerializer formatter = new XmlSerializer(typeof(List<PlaceAndDateReports>));

using (FileStream fs = new FileStream(@"PlaceAndDateReports.xml", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, placeAndDateReports);

}

}

}

}

TicketReport

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using Main\_Project.Table;

using MySql.Data.MySqlClient;

namespace Main\_Project

{

public class TicketReport: Connection

{

public int countSold { get; set; }

public string title { get; set; }

}

}

TypeOfTicketsReport

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Main\_Project.Reports

{

public class TypeOfTicketsReport: Connection

{

public string typeOfTickets { get; set; }

public float cost { get; set; }

}

}

UnitTestPerfomances

using System;

using System.Collections.Generic;

using Main\_Project.Table;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Main\_Project;

using System.Linq;

namespace UnitTestProject

{

[TestClass]

public class UnitTestPerfomances

{

[TestMethod]

public void TestMethodReadUsers()

{

UserSQL user = new UserSQL();

List<User> users = new List<User>();

users = user.UserRead();

CollectionAssert.AllItemsAreNotNull(users);

}

[TestMethod]

public void TestMethodReadTypeOfTickets()

{

TypeofticketSQL typeOfTickets = new TypeofticketSQL();

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

tickets = typeOfTickets.TypeOfTicketRead();

CollectionAssert.AllItemsAreNotNull(tickets);

}

[TestMethod]

public void TestMethodReadTypeOfPerformance()

{

TypeofperformanceSQL typeofperformance = new TypeofperformanceSQL();

List<TypeOfPerformance> typeofperformanceSQLs = new List<TypeOfPerformance>();

typeofperformanceSQLs = typeofperformance.TypeOfPerformanceRead();

CollectionAssert.AllItemsAreNotNull(typeofperformanceSQLs);

}

[TestMethod]

public void TestMethodReadSales()

{

SalesSQL salesSQL = new SalesSQL();

List<Sales> sales = new List<Sales>();

sales = salesSQL.SalesRead();

CollectionAssert.AllItemsAreNotNull(sales);

}

[TestMethod]

public void TestMethodReadPerfomances()

{

PerfomanceSQL perfomanceSQL = new PerfomanceSQL();

List<Perfomances> perfomances = new List<Perfomances>();

perfomances = perfomanceSQL.PerfomanceRead();

CollectionAssert.AllItemsAreNotNull(perfomances);

}

[TestMethod]

public void TestMethodReadConcretePerformance()

{

ConcreteperformanceSQL concreteperformanceSQL = new ConcreteperformanceSQL();

List<ConcretePerformance> concretePerformances = new List<ConcretePerformance>();

concretePerformances = concreteperformanceSQL.ConcretePerformanceRead();

CollectionAssert.AllItemsAreNotNull(concretePerformances);

}

[TestMethod]

public void TestMethodUpdateConcretePerformance()

{

ConcreteperformanceSQL concreteperformanceSQL = new ConcreteperformanceSQL();

List<ConcretePerformance> concretePerformances = new List<ConcretePerformance>();

concretePerformances = concreteperformanceSQL.ConcretePerformanceRead();

int predicated = 123;

concretePerformances.Last().count = 123;

concreteperformanceSQL.ConcreteperformanceUpdate(concretePerformances);

concretePerformances = concreteperformanceSQL.ConcretePerformanceRead();

Assert.AreEqual(predicated, concretePerformances.Last().count);

}

[TestMethod]

public void TestMethodDeleteConcretePerformance()

{

ConcreteperformanceSQL concreteperformanceSQL = new ConcreteperformanceSQL();

List<ConcretePerformance> concretePerformances = new List<ConcretePerformance>();

int id = concreteperformanceSQL.ConcretePerformanceRead().Count;

concreteperformanceSQL.ConcreteperformanceDelete(id);

CollectionAssert.AllItemsAreNotNull(concretePerformances);

}

[TestMethod]

public void TestMethodDeleteUsers()

{

UserSQL user = new UserSQL();

List<User> users = new List<User>();

int id = user.UserRead().Count;

user.DeleteUser(id);

CollectionAssert.AllItemsAreNotNull(users);

}

[TestMethod]

public void TestMethodAddTypeOfTicket()

{

TypeofticketSQL typeofticketSQL = new TypeofticketSQL();

List<TypeOfTickets> typeOfTickets = typeofticketSQL.TypeOfTicketRead();

typeOfTickets.Add(new TypeOfTickets

{

typeOfTickets = "Asdg",

cost = 12,

typeOfTicketsId = 1

});

int dbCount = typeofticketSQL.TypeOfTicketRead().Count;

typeofticketSQL.TypeOfTicketAdd(typeOfTickets, dbCount);

typeOfTickets = typeofticketSQL.TypeOfTicketRead();

Assert.IsTrue(typeOfTickets.Count > dbCount);

}

}

}

GuestWindow

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using Main\_Project.Table;

using Main\_Project;

namespace MainForm

{

public partial class GuestForm : Form

{

public Connection connection;

List<ConcretePerformance> perfomances;

List<ConcretePerformance> newConcrete;

public ConcreteperformanceSQL concreteperformanceSQL = new ConcreteperformanceSQL();

private UserSQL userSQL = new UserSQL();

Registration registration = new Registration();

public GuestForm()

{

InitializeComponent();

connection = new Connection();

AllPerfomance();

}

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

{

// Считываю введенные пользователем данные;

string login = winlog.Text;

string password = passlog.Text;

// Возвращаю пользователя, если он не зарегистрирован, возвращаю null,

User user = userSQL.IsWho(login, password);

// Если пользователь существует;

if (!(user == null))

{

// Проверяется его статус и вызывается окно для соответствующего сотрудника;

switch (user.is\_admin)

{

case "admin":

passlog.Text = "";

AdminForm adminWindow = new AdminForm(user, this);

adminWindow.Show();

break;

case "user":

passlog.Text = "";

UserForm userForm = new UserForm(user, this);

userForm.Show();

break;

}

}

else

MessageBox.Show("Вы вввели неверный логин или пароль.");

}

private void AllPerfomance()

{

// Считывание данных таблицы "perfomances" из БД и сохранение в список;

perfomances = concreteperformanceSQL.ConcretePerformanceRead();

// Вывод списка на экран;

dataGridView1.DataSource = perfomances;

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

string typeName = (textBox10.Text);

DateTime date = Convert.ToDateTime(textBox9.Text);

int place = Convert.ToInt32(textBox6.Text);

newConcrete = concreteperformanceSQL.GetByDateAndName(typeName, date, place);

dataGridView1.DataSource = newConcrete;

}

catch

{

MessageBox.Show("Введите жанр, дату представления и количество свободных мест");

}

}

private void button2\_Click\_1(object sender, EventArgs e)

{

dataGridView1.DataSource = perfomances;

}

private void button4\_Click(object sender, EventArgs e)

{

registration.Show();

}

}

}

Registration

using Main\_Project;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MainForm

{

public partial class Registration : Form

{

UserSQL userSQL = new UserSQL();

public Registration()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

string login = textBox6.Text;

string password = textBox2.Text;

try

{

if (login == "" || password == "")

{

throw new Exception();

}

else

{

userSQL.AddNewUser(login, password);

this.Close();

}

}

catch

{

MessageBox.Show("Введите логин и пароль, если выдает ошибку, значит такой пользователь уже зарегистрирован");

}

}

}

}

UserWindow

using Main\_Project;

using Main\_Project.Table;

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace MainForm

{

public partial class UserForm : Form

{

List<Sales> sales;

List<ConcretePerformance> concretePerformances;

List<TypeOfTickets> typeOfTickets;

List<string> freeTickets;

GuestForm guestForm;

User user;

private SalesSQL salesSQL = new SalesSQL();

private TicketSQL ticketSQL = new TicketSQL();

private ConcreteperformanceSQL concreteperformanceSQL = new ConcreteperformanceSQL();

private TypeofticketSQL typeofticketSQL = new TypeofticketSQL();

public UserForm(User user, GuestForm guestForm)

{

concretePerformances = concreteperformanceSQL.ConcretePerformanceRead();

typeOfTickets = typeofticketSQL.TypeOfTicketRead();

freeTickets = new List<string>();

sales = salesSQL.GetConcreteUserSales(user.userId);

this.user = user;

InitializeComponent();

AllSales(user);

this.guestForm = guestForm;

}

private void AllSales(User user)

{

// Считывание данных таблицы "sales" из БД и сохранение в список;

sales = salesSQL.GetConcreteUserSales(user.userId);

dataGridView2.AutoGenerateColumns = false;

dataGridView2.DataSource = sales;

titles.ValueMember = "title";

titles.DisplayMember = "title";

titles.DataSource = concretePerformances;

places.ValueMember = "place";

places.DisplayMember = "place";

TypeOfTicket.ValueMember = "typeOfTickets";

TypeOfTicket.DisplayMember = "typeOfTickets";

TypeOfTicket.DataSource = typeOfTickets;

dataGridView1.Rows.Add(1);

dataGridView1.AllowUserToAddRows = false;

}

private void dataGridView1\_CellValueChanged(object sender, DataGridViewCellEventArgs e)

{

UpdateFreeTickets();

}

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

{

string title = dataGridView1.Rows[0].Cells[0].Value.ToString();

int place = int.Parse(dataGridView1.Rows[0].Cells[1].Value.ToString());

string typeOfTicket = dataGridView1.Rows[0].Cells[2].Value.ToString();

ticketSQL.AddTicket(title, place, typeOfTicket);

salesSQL.SalesAdd(title, place, user.userId, ticketSQL);

sales = salesSQL.GetConcreteUserSales(user.userId);

dataGridView2.DataSource = null;

dataGridView2.DataSource = sales;

dataGridView1.Rows.RemoveAt(0);

dataGridView1.Rows.Add(1);

guestForm.dataGridView1.DataSource = guestForm.concreteperformanceSQL.ConcretePerformanceRead();

}

private void dataGridView1\_DataError(object sender, DataGridViewDataErrorEventArgs e)

{

}

private void UpdateFreeTickets()

{

if (dataGridView1.Rows[0].Cells[0].Value != null)

{

freeTickets.Clear();

foreach (ConcretePerformance concrete in concretePerformances)

{

if (dataGridView1.Rows[0].Cells[0].Value.ToString() == concrete.title)

{

List<Tickets> tickets = ticketSQL.ReadTicekts(concrete.title);

for (int i = 1; i < concrete.count + 1; i++)

{

int count = 0;

foreach (Tickets ticket in tickets)

{

if (i != ticket.place)

count++;

}

if (count == tickets.Count)

freeTickets.Add(i.ToString());

}

places.DataSource = null;

places.DataSource = freeTickets;

break;

}

}

}

}

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

{

ticketSQL.TicketDelete(sales[dataGridView2.CurrentRow.Index].ticketId);

salesSQL.SalesDelete(sales[dataGridView2.CurrentRow.Index].SalesId, ticketSQL, sales[dataGridView2.CurrentRow.Index].Title);

sales.RemoveAt(dataGridView2.CurrentRow.Index);

dataGridView2.DataSource = null;

dataGridView2.DataSource = sales;

guestForm.dataGridView1.DataSource = guestForm.concreteperformanceSQL.ConcretePerformanceRead();

}

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

{

this.Close();

}

}

}

AdminWindow

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using Main\_Project.Table;

using Main\_Project;

using Main\_Project.Reports;

using Main\_Project.SQL;

using System.Linq;

namespace MainForm

{

public partial class AdminForm : Form

{

GuestForm main;

List<User> users;

List<ConcretePerformance> concreteperformances;

List<TypeOfTickets> typeofTicket;

private UserSQL userSQL = new UserSQL();

private ConcreteperformanceSQL concreteperformanceSQL = new ConcreteperformanceSQL();

private TypeofticketSQL typeOfTicketSQL = new TypeofticketSQL();

private PerfomanceSQL perfomanceSQL = new PerfomanceSQL();

private TypeofperformanceSQL typeofperformanceSQL = new TypeofperformanceSQL();

private List<TicketReport> ticketReport = new List<TicketReport>();

private List<TypeOfTicketsReport> typeOfTicketsReports = new List<TypeOfTicketsReport>();

private List<PlaceAndDateReports> placeAndDateReports = new List<PlaceAndDateReports>();

public AdminForm(User user, GuestForm form)

{

main = form;

InitializeComponent();

AllUsers();

}

private void AllUsers()

{

// Считывание данных таблицы "users" из БД и сохранение в список;

users = userSQL.UserRead();

concreteperformances = concreteperformanceSQL.ConcretePerformanceRead();

typeofTicket = typeOfTicketSQL.TypeOfTicketRead();

// Вывод списка на экран;

dataGridUsers.DataSource = users;

dataGridUsers.AllowUserToAddRows = true;

dataGridConcretePerformance.DataSource = concreteperformances;

dataGridTypeOfTicket.DataSource = typeofTicket;

}

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

{

users.Add(new User());

dataGridUsers.DataSource = null;

dataGridUsers.DataSource = users;

}

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

{

userSQL.DeleteUser(users[dataGridUsers.CurrentRow.Index].userId);

users.RemoveAt(dataGridUsers.CurrentRow.Index);

dataGridUsers.DataSource = null;

dataGridUsers.DataSource = users;

}

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

{

concreteperformances.Add(new ConcretePerformance());

dataGridConcretePerformance.DataSource = null;

dataGridConcretePerformance.DataSource = concreteperformances;

main.dataGridView1.DataSource = main.concreteperformanceSQL.ConcretePerformanceRead();

}

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

{

concreteperformanceSQL.ConcreteperformanceDelete(concreteperformances[dataGridConcretePerformance.CurrentRow.Index].concretePerformanceId);

concreteperformances.RemoveAt(dataGridConcretePerformance.CurrentRow.Index);

dataGridConcretePerformance.DataSource = null;

dataGridConcretePerformance.DataSource = concreteperformances;

main.dataGridView1.DataSource = main.concreteperformanceSQL.ConcretePerformanceRead();

}

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

{

userSQL.UpdateUser(users);

string title = concreteperformances.Last().title;

if(!typeofperformanceSQL.TypeOfPerformanceRead().Any(t => t.typeName == concreteperformances.Last().typeName))

{

typeofperformanceSQL.Insert(concreteperformances.Last().typeName);

}

int id = typeofperformanceSQL.TypeOfPerformanceRead().Where(c => c.typeName == concreteperformances.Last().typeName).First().typeOfPerformanceId;

string author = concreteperformances.Last().author;

perfomanceSQL.PerfomancesAdd(title, id, author);

id = perfomanceSQL.PerfomanceRead().Last().perfomancesId;

concreteperformances.Last().performanceId = id;

concreteperformanceSQL.ConcreteperformanceUpdate(concreteperformances);

typeOfTicketSQL.TypeOfTicketUpdate(typeofTicket);

main.dataGridView1.DataSource = main.concreteperformanceSQL.ConcretePerformanceRead();

MessageBox.Show("Данные успешно сохранены.");

}

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

{

this.Close();

}

private void button1\_Click\_1(object sender, EventArgs e)

{

GetReport getReport = new GetReport();

ticketReport = getReport.GetReportSales();

ReportXML reportXML = new ReportXML();

reportXML.SaveReportsXML(ticketReport);

typeOfTicketsReports = getReport.GetTypeOfTicketsReport();

reportXML.SaveTypeOfTicketsReport(typeOfTicketsReports);

DateTime date = DateTime.Parse(textBox1.Text);

placeAndDateReports = getReport.GetPlaceAndDateReports(date);

reportXML.SavePlaceAndDateReports(placeAndDateReports);

MessageBox.Show("Данные успешно сохранены.");

}

}

}

# ПРИЛОЖЕНИЕ В

(обязательное)

# Руководство системного программиста

Программа предназначена для реализации продаж и учета проданных билетов на спектакли в театре. В программе можно производить поиск по спектаклям, зарегистрировать нового пользователя. Зарегистрированный пользователь может добавить и удалить билет. Администратор может добавить и удалить пользователя, изменять тип билета и его цену, добавлять новый спектакль и удалить его, так же формировать отчеты. Приложение было разработана на языке *C#* с помощью СУБД *MySQL.* Для доступа к данным было использовано *ADO.NET.*

Программный комплекс состоит из трех составных частей а именно:

– библиотека классов;

– модульное тестирование;

– *WF* приложение.

Для корректной работы приложения необходима следующая конфигурация ЭВМ:

– *Windows* 7 с *.NET Framework* 4.7.2;

– объем оперативной памяти не менее 2гб;

– объем видеокарты должен быть не менее 128 Мбайт;

– объем свободного места на жестком диске должен быть не менее 20мб;

– разрешение экрана 800 x 600.

Настройка приложения. Для запуска приложения требуется открыть файл с расширением *.exe.* Длявзаимодействия с приложением нужна клавиатура и мышь.

Проверка приложения. Для проверки работоспособности программы требуется запустить *WF* приложение, а так же имеются модульные тесты. Чтобы их запустить нужно перейти во вкладку «Тест» и нажать на кнопку «Запуск всех тестов».

Сообщения системному программисту. При тестировании программы после ввода некорректных данных в поля логина и пароля, выдаст ошибку, для исправления нужно ввести логин и пароль пользователя, зарегистрированного в данной программе. Если администратор захочет создать новый отчет и не выберет дату за которую должен будет сформироваться отчет, произойдет ошибка, чтобы это исправить администратору нужно ввести конкретную дату представления.

# ПРИЛОЖЕНИЕ Г

(обязательное)

# Руководство программиста

Программа предназначена для театров. В программе реализуется продажи и учет проданных билетов на спектакли в театре. В программе можно производить поиск по спектаклям, зарегистрировать нового пользователя. Зарегистрированный пользователь может добавить и удалить билет. Администратор может добавить и удалить пользователя, изменять тип билета и его цену, добавлять новый спектакль и удалить его, так же формировать отчеты. Приложение было разработана на языке *C#* с помощью СУБД *MySQL.* Для доступа к данным было использовано *ADO.NET.* Для написания запросов было использовано *LINQ to object.*

Минимальные требования для запуска программы:

– *Windows* 7 с *.NET Framework* 4.7.2;

– объем оперативной памяти не менее 2гб;

– объем видеокарты должен быть не менее 128 Мбайт;

– объем свободного места на жестком диске должен быть не менее 20мб;

– разрешение экрана 800 x 600.

Программу можно запустить несколькими способами:

1. С помощью файла с раcширением .*exe*;
2. С помощью *Visual Studio.*

В качестве входных данных служит считывание информации из базы данных *MySQL*.

В качестве выходных данных служат отчеты, которые может формировать администратор.

Сообщения. При тестировании программы после ввода некорректных данных в поля логина и пароля, выдаст ошибку, для исправления нужно ввести логин и пароль пользователя, зарегистрированного в данной программе. Чтобы провести поиск по спектаклям нужно ввести данные во все три поля, если же это не сделать будет выдана ошибка.

# ПРИЛОЖЕНИЕ Д

(обязательное)

# Руководство пользователя

Уданной программы не большая область применения, программа предназначена в основном для театров. В программе реализуется поиск по билетам, просмотр представлений, регистрация нового пользователя, заказ и отмена билетов. Администратор же может изменять типы билетов, цену, удалять пользователей и представления, добавлять пользователей и представление. Так же формировать отчеты за определенную дату. Чтобы использовать данную программу, пользователю нужно уметь пользоваться компьютером и мышью. Чтобы пользоваться данной программой достаточно прочитать Руководство пользователя.

Программа упрощает заказ билетов, формирование отчетов.

Условия при котором обеспечивается работа программы:

– *Windows* 7 с *.NET Framework* 4.7.2;

– объем оперативной памяти не менее 2гб;

– объем видеокарты должен быть не менее 128 Мбайт;

– объем свободного места на жестком диске должен быть не менее 20мб;

– разрешение экрана 800 x 600;

– установленная база данных *MySQL*.

Программы необходимые для работоспособности, сперва стоит загрузить *.NET Framework* 4.7.2, а потом установить базу данных *MySQL*.

Для того чтобы пользователю проверить работоспособность программы достаточно будет запустить файл с расширением .*exe.* В самой же программе нужно будет правильно ввести данные и протестировать кнопки.

При отказе программы пользователю нужно ее перезапустить это можно сделать по нажатию крестика в правом верхнем углу или нажать сочетание клавиш *Alt* + *F4.*

Чтобы пользоваться данной программой достаточно иметь не большие познания в эксплуатации ЭВМ, прочитать руководство пользователя, конфигурация компьютера должна соответствовать минимальным системным требованиям.

# ПРИЛОЖЕНИЕ Е

(обязательное)

# Иерархическая схема классов приложения