**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

Егоров Е. А.

Руководитель: доцент

Курочка К.С.

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

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

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

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

Подписи членов комиссии

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

Гомель 2020

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc42738242)

[1 ОПИСАНИЯ СРЕДСТВ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ 5](#_Toc42738243)

[СИСТЕМ 5](#_Toc42738244)

[1.1 Используемые средства реализации информационной системы 5](#_Toc42738245)

[1.2 Основы объектно-ориентированного представления 6](#_Toc42738246)

[1.3 Язык программирования *SQL* 8](#_Toc42738247)

[1.4 Слой доступа к данным *DAO* 9](#_Toc42738248)

[1.5 Технология для доступа к данным *ADO* 11](#_Toc42738249)

[1.6 База данных *MySQL* 13](#_Toc42738250)

[1.7 Появление *LINQ* 14](#_Toc42738251)

[2 АГОРИТМИЧЕСКИЙ АНАЛИЗ ПОСТАВЛЕНОЙ ЗАДАЧИ И ОПИСАНИЕ ПРОГРАММНОГО КОМПЛЕКСА 16](#_Toc42738252)

[2.1 Архитектура программного комплекса 16](#_Toc42738253)

[2.2 Структура базы данных 16](#_Toc42738254)

[2.3 Структура классов разработанного программного комплекса 21](#_Toc42738255)

[2.4 Графический интерфейс пользователя 23](#_Toc42738256)

[3 ВЕРИФИКАЦИЯ ПРИЛОЖЕНИЯ 27](#_Toc42738257)

[3.1 Тестирование пользовательского интерфейса 27](#_Toc42738258)

[3.2 Модульное тестирование приложния 29](#_Toc42738259)

[ЗАКЛЮЧЕНИЕ 31](#_Toc42738260)

[Список используемых источников 32](#_Toc42738261)

[ПРИЛОЖЕНИЕ А](#_Toc42738262) [Графические представления интерфейса программы 33](#_Toc42738263)

[ПРИЛОЖЕНИЕ Б](#_Toc42738264) [Листинг программы 35](#_Toc42738265)

[ПРИЛОЖЕНИЕ В](#_Toc42738266) [Руководство системного программиста 83](#_Toc42738267)

[ПРИЛОЖЕНИЕ Г](#_Toc42738268) [Руководство программиста 84](#_Toc42738269)

[ПРИЛОЖЕНИЕ Д](#_Toc42738270) [Руководство пользователя 85](#_Toc42738271)

[ПРИЛОЖЕНИЕ Е](#_Toc42738272) [Иерархическая схема классов приложения 86](#_Toc42738273)

# ВВЕДЕНИЕ

Театр – один из видов искусства, культурный и социальный институт, который выполняет ряд важных функций, но, в первую очередь, учит быть человеком, учит чувствовать и сопереживать. Может возникнуть вопрос, нужен ли такой инструментарий в наш информационный, полный динамизма и технологичности век. Еще Максим Горький сказал: «Человек – это звучит гордо». И как раз-таки театральное направление в искусстве как нельзя лучше помогает развиваться человеку как личности, поэтому и актуально со времен Древней Римской империи и до наших дней.

Получение любой информации, распоряжение способами проведения досуга происходит на основе личного выбора и предпочтений. Современный человек, естественно, склоняется к наименее энергозатратным способам получения этой самой услуги или информации. Оплата коммунальных услуг, банковские операции, продажа и покупка разных товаров – все это в наше время можно сделать через определенные компьютерные приложения. Удобство данных операций, несомненно: высокая скорость выполнения, возможность реализации из любой точки мира, выбор формы оплаты и времени, высокая вариативность и т.п. Удивительно, но такой, казалось бы, ретроградный способ проведения досуга, как посещение театра, благодаря информационным технологиями в наше время может выйти на совсем другой уровень.

Создание компьютерных приложений по оптимизации работы театральных касс на данном этапе развития общества крайне необходимо. Это поможет решить следующие задачи:

– структурировать и упорядочить большой объем информации;

– облегчить круглосуточное донесение самой свежей информации о предстоящих мероприятиях до огромного количества людей максимально быстрым и удобным способом;

– своевременно информировать о каких-либо изменениях в программе мероприятий;

– максимально удобно приобретать билеты на выбранные мероприятия и, в случае изменения личных планов, осуществлять возврат приобретенных билетов;

– анализировать спрос и, соответственно, корректировать предложения.

Высоко оптимизированный и хорошо продуманный интернет-магазин или удобное смарт-приложение как маркетинговые каналы являются одними из самых эффективных способов продвижения не только продукции какого-либо производства, но и такого вида искусства, как театр.

# 1 ОПИСАНИЯ СРЕДСТВ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ

# СИСТЕМ

## 1.1 Используемые средства реализации информационной системы

Любая информационная система предназначена для своевременного обеспечения данными о системе. Она имеет системы сбора, накопления, поиска и передачи информации.

Существуют различные методы реализации информационных систем классифицируемые по степени автоматизации: ручные, автоматические и автоматизированные.

Автоматизированные информационные системы предполагают участия в процессе обработки информации человека и технических средств, причём главная роль по обработке информации отводится компьютеру. Именно этот класс систем соответствует современному представлению о информационных системах.

Для реализации информационной системы будет создан автоматизированный программный комплекс на платформе *.NET Framework* написанный на языке программирования *C#* при помощи объектно-ориентированного подхода в среде разработки *Visual Studio 2019*, с использованием шаблонов проектирования и слоя доступа к данным, который будет взаимодействовать с СУБД (система управления базами данных) *MySQL*. Для реализации пользовательского интерфейса будет использована технология *WPF* (*Windows Presentation Foundation*).

*.NET Framework* – программная платформа, выпущенная компанией *Microsoft* в 2002 году. Основой платформы является общеязыковая среда исполнения *Common Language Runtime (CLR)*, которая подходит для разных языков программирования. Функциональные возможности *CLR* доступны в любых языках программирования, использующих эту среду. Платформа *.NET* *Framework* поддерживает множество языков программирования, таких как *C#*, *Visual Basic*, *Iron Python*, *C*, *C++* и пр. Благодаря тому, что программа сначала переводится компилятором в единый для *.NET* байт-код *Common Intermediate Language* (*CIL*). Затем код либо исполняется виртуальной машиной *Common Language Runtime (CLR)*, либо транслируется утилитой *NGen.exe* в исполняемый код для конкретного целевого процессора. Благодаря *CLR* встроенный в неё *JIT* (*Just–in–time*)компилятор преобразует промежуточный байт-код в машинные коды нужного процессора [2, с. 126].

## 1.2 Основы объектно-ориентированного представления

Основным понятием *C#* является объектно-ориентированное программирование (ООП). Методика ООП неотделима от *С#*, и поэтому все программы на *C#* являются объектно-ориентированными хотя бы в самой малой степени. В связи с этим очень важно и полезно усвоить основополагающие принципы ООП, прежде чем приступать к написанию самой простой программы на *С#*. ООП представляет собой эффективный подход к программированию. Методики программирования претерпели существенные изменения с момента изобретения компьютера, постепенно приспосабливаясь, главным образом, к повышению сложности программ. Когда, например, появились первые электроннвычислительные машины (ЭВМ), программирование заключалось в ручном переключении на разные двоичные машинные команды с переднего пульта управления ЭВМ. Такой подход был вполне оправданным, поскольку программы состояли всего из нескольких сотен команд. Дальнейшее усложнение программ привело к разработке языка ассемблера, который давал программистам возможность работать с более сложными программами, используя символическое представление отдельных машинных команд. Постоянное усложнение программ вызвало потребность в разработке и внедрении в практику программирования таких языков высокого уровня, как, например, *FORTRAN* и *COBOL*, которые предоставляли программистам больше средств для того, чтобы как-то справиться с постоянно растущей сложностью программ. Но как только возможности этих первых языков программирования были полностью исчерпаны, появились разработки языков структурного программирования, в том числе и *C* [3, с. 41]*.*

Программа реализованная с помощью объектно-ориентированного программирования состоит из объектов. Для того чтобы лучше разобраться в этом, дадим определение понятию объект.

Объект ­– это модель или абстракция реальной сущности в программной системе.

Так же при написании программы надо соблюдать основные принципы объектно-ориентированного программирования, это такие принципы как:

– абстракция;

– инкапсуляция;

– наследование;

– полиморфизм.

Абстракция – в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

Инкапсуляция – свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента (что у него внутри?), а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Наследование – позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию – например, реализация класса может быть изменена в процессе наследования [4].

Так же одной из главных составляющих программы является ее тестирование на работоспособность, проверить программу можно при помощи модульных тестов или как их еще называют юнит–тесты.

Существует множество преимуществ написания модульных тестов: они помогают с регрессией, предоставляют документацию и способствуют хорошей структуре кода. Но трудночитаемые и ненадежные модульные тесты могут негативно отразиться на базе кода.

Модульные тесты, занимают миллисекунды, выполняются простым нажатием кнопки и не обязательно требуют знаний о всей системе в целом. Успешность прохождения теста зависит от средства выполнения теста, а не от пользователя.

С модульным тестированием можно повторно запускать весь набор тестов после каждой сборки или даже после изменения строки кода. Это дает вам уверенность, что ваш новый код не нарушил существующие функциональные возможности.

Характеристики хорошего модульного теста:

– быстрый. В хорошо разработанных проектах могут быть тысячи модульных тестов. Модульные тесты должны выполняться очень быстро. За миллисекунды;

– изолированный. Модульные тесты являются автономными, могут выполняться изолированно и не имеют зависимостей от внешних факторов, таких как файловая система или база данных;

– повторяемый. Запуски модульного теста должны иметь согласованные результаты, то есть всегда возвращать одинаковый результат, если вы не вносите никаких изменений между запусками;

– самопроверяющий. Тест должен автоматически определять, пройден он или нет, без участия пользователя;

– уместный. Время на написание модульного теста не должно значительно превышать время написания тестируемого кода. Если вам кажется, что тестирование кода занимает слишком много времени по сравнению с написанием кода, продумайте структуру, более подходящую для тестирования.

## 1.3 Язык программирования *SQL*

*SQL* – один из формальных языков, то есть средство, с помощью которого вы передаете компьютеру инструкции, называемые программой. Программное обеспечение вашей базы данных выполняет эту программу, написанную на языке *SQL*. Это значит, что СУБД выполняет те запросы, которые вы ей передали, и отображает результаты их работы, в том числе какое-нибудь сообщение об ошибке. Надо сказать, что языки программирования, называемые также формальными языками, отличаются от языков общения, называемых неформальными или естественными языками, главным образом тем, что создаются под конкретную цель, полностью лишены двусмысленности, имеют весьма ограниченные словарный запас и гибкость. Таким образом, если вы не получили результата от работы своей программы, на который рассчитывали при ее написании, это произошло потому, что ваша программа содержит какую-либо ошибку (логическую или синтаксическую – в последнем случае, скорее всего, будет выведено соответствующее сообщение, описывающее ошибку), а не потому, что компьютер неправильно понял ваши инструкции, формализованные в виде программы (эта информация проясняет, почему отладка программ считается одной из основных задач программирования) [5, с.12].

С помощью *SQL* вы можете, в частности, превратить любой вопрос типа «А где живут наши клиенты?» в такую команду, которую программное обеспечение вашей базы данных сможет понять и выполнить (для приведенного вопроса это может быть команда *SELECT city*, *state FROM customers*). Если вы уже умеете извлекать информацию аналогичного типа с помощью графического инструментария построения запросов, то, скорее всего, заметили, что он становится весьма ограничивающим и громоздким по мере того, как сложность ваших запросов возрастает. Вот здесь и нужен *SQL*, хотя решением указанной проблемы его возможности не ограничиваются. Например, вы можете применять *SQL* для того, чтобы добавлять, модифицировать и удалять данные и объекты базы данных. И именно потому, что язык SQL такой мощный, его поддерживают наиболее популярные СУБД, в частности *Microsoft Access*, *Oracle* и *MySQL*, хотя уровень этой поддержки существенно зависит от того, о какой именно СУБД идет речь.

Изначально *SQL* был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

– создание в базе данных новой таблицы;

– добавление в таблицу новых записей;

– изменение записей;

– удаление записей;

– выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);

– изменение структур таблиц.

Со временем *SQL* усложнился – обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) – и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях *SQL* остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

## 1.4 Слой доступа к данным *DAO*

При проектировании информационной системы выявляются некоторые слои, которые отвечают за взаимодействие различных модулей системы. Соединение с базой данных является одной из важнейшей составляющей приложения. Всегда выделяется часть кода, модуль, отвечающий за передачу запросов в базе данных и обработку полученных от неё ответов. В общем случае, определение *Data Access Object* описывает его как прослойку между БД и системой. *DAO* абстрагирует сущности системы и делает их отображение на БД, определяет общие методы использования соединения, его получение, закрытие и (или) возвращение в *Connection Pool*.

Вершиной иерархии *DAO* является абстрактный класс или интерфейс с описанием общих методов, которые будут использоваться при взаимодействии с базой данных. Как правило, это методы поиска, удаление по ключу, обновление и т.д.

Объект *Database*, его методы и свойства используются для операций с открытой базы данных. В базе данных любого типа вы можете:

– использовать метод *Execute*, чтобы выполнить запрос на изменение;

– задать свойство *Connect*, чтобы установить соединение с источником данных *ODBC*;

– задать свойство *QueryTimeout*, чтобы ограничить продолжительность ожидания выполнения запроса в источнике данных *ODBC*;

– использовать свойство *RecordsAffected*, чтобы определить количество измененных записей в результате запроса на изменение;

– использовать метод *OpenRecordset*, чтобы выполнить запрос на выборку и создать объект *Recordset*;

– использовать свойство *Version*, чтобы определить версию ядра СУБД, создавшую базу данных.

С помощью базы данных ядра СУБД *Microsoft* *Access* можно также использовать другие методы, свойства и коллекции для работы с объектом *Database*, а также создавать, изменять или получать сведения о его таблицах, запросах и отношениях. Например, вы можете:

– использовать методы *CreateTableDef* и *CreateRelation*, чтобы создавать таблицы и отношения соответственно;

– использовать метод *CreateProperty*, чтобы определить новые свойства *Database*;

– использовать метод *CreateQueryDef*, чтобы создать постоянное или временное определение запроса;

– использовать методы *MakeReplica*, *Synchronize* и *PopulatePartial*, чтобы создать и синхронизировать полные или частичные копии базы данных;

– задать свойство *CollatingOrder*, чтобы установить порядок сортировки по алфавиту для символьных полей на разных языках.

Метод *CreateDatabase* используется для создания постоянного объекта *Database*, автоматически добавляемого в коллекцию *Databases*, что обеспечивает его сохранение на диске.

Не нужно указывать объект *DBEngine* при использовании метода *OpenDatabase*.

Открытие базы данных со связанными таблицами не создает автоматически ссылки на указанные внешние файлы. Необходимо либо создать ссылку на объект *TableDef* или объекты *Field* таблицы, либо открыть объект *Recordset*. Если не удается создать ссылки на эти таблицы, возникает перехватываемая ошибка. Может также требоваться разрешение на доступ к базе данных или другой пользователь мог открыть базу данных в монопольном режиме. В этих случаях возникают перехватываемые ошибки.

Когда выполнена процедура, объявляющая объект *Database*, локальные объекты *Database* закрываются вместе с любыми открытыми объектами *Recordset*. Любые ожидающие обновления пропадают, и выполняется откат всех ожидающих транзакций, но перехватываемая ошибка не возникает. Следует явным образом завершить все ожидающие транзакции или изменения и закрыть объекты *Recordset* и *Database* перед процедурами выхода, локально объявляющими эти переменные объектов.

Если вы используете один из методов транзакции (*BeginTrans*, *CommitTrans* или *Rollback*) для объекта *Workspace*, эти транзакции применяются ко всем базам данных, открытым в объекте *Workspace*, из которого был открыт объект *Database*. Если нужно использовать независимые транзакции, необходимо сначала открыть дополнительный объект *Workspace*, а затем открыть другой объект *Database* в этом объекте *Workspace* [6].

## 1.5 Технология для доступа к данным *ADO*

Сегодня большое значение имеет работа с данными. Для хранения данных используются различные системы управления базами данных: *MS SQL* *Server*, *Oracle*, *MySQL* и так далее. И большинство крупных приложений так или иначе используют для хранения данных эти системы управления базами данных. Однако чтобы осуществлять связь между базой данных и приложением на *C#* необходим посредник. И именно таким посредником является технология ADO.NET [7].

*ADO.NET* предоставляет собой технологию работы с данными, которая основана на платформе .*NET* *Framework*. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Основу интерфейса взаимодействия с базами данных в *ADO.NET* представляет ограниченный круг объектов: *Connection*, *Command*, *DataReader*, *DataSet* и *DataAdapter*. С помощью объекта *Connection* происходит установка подключения к источнику данных. Объект *Command* позволяет выполнять операции с данными из БД. Объект *DataReader* считывает полученные в результате запроса данные. Объект *DataSet* предназначен для хранения данных из БД и позволяет работать с ними независимо от БД. И объект *DataAdapter* является посредником между *DataSet* и источником данных. Главным образом, через эти объекты и будет идти работа с базой данных.

Однако чтобы использовать один и тот же набор объектов для разных источников данных, необходим соответствующий провайдер данных. Собственно через провайдер данных в *ADO.NET* и осуществляется взаимодействие с базой данных. Причем для каждого источника данных в *ADO.NET* может быть свой провайдер, который собственно и определяет конкретную реализацию вышеуказанных классов.

По умолчанию в *ADO.NET* имеются следующие встроенные провайдеры:

– провайдер для *MS SQL* *Server;*

– провайдер для *OLE DB* (Предоставляет доступ к некоторым старым версиям *MS SQL Server*, а также к БД *Access*, *DB2*, *MySQL* и *Oracle*);

– провайдер для *ODBC* (Провайдер для тех источников данных, для которых нет своих провайдеров);

– провайдер для *Oracle;*

– провайдер *EntityClient*. Провайдер данных для технологии *ORM Entity Framework;*

– провайдер для сервера *SQL Server Compact* 4.0.

Кроме этих провайдеров, которые являются встроенными, существует также множество других, предназначенных для различных баз данных, например, для MySQL.

Основные пространства имен, которые используются в *ADO.NET*:

*– system.Data*: определяет классы, интерфейсы, делегаты, которые реализуют архитектуру *ADO.NET;*

*– system.Data.Common*: содержит классы, общие для всех провайдеров *ADO.NET;*

*– system.Data.Design*: определяет классы, которые используются для создания своих собственных наборов данных;

*– system.Data.Odbc*: определяет функциональность провайдера данных для *ODBC;*

*– system.Data.OleDb*: определяет функциональность провайдера данных для *OLE DB;*

*– system.Data.Sql*: хранит классы, которые поддерживают специфичную для *SQL Server* функциональность;

*– system.Data.OracleClient*: определяет функциональность провайдера для баз данных *Oracle;*

*– system.Data.SqlClient*: определяет функциональность провайдера для баз данных *MS SQL Server;*

*– system.Data.SqlServerCe*: определяет функциональность провайдера для *SQL Server Compact* 4.0;

*– system.Data.SqlTypes*: содержит классы для типов данных *MS SQL Servera;*

*– microsoft.SqlServer.Server*: хранит компоненты для взаимодействия *SQL Server* и среды *CLR.*

Схематично архитектуру *ADO.NET* (рисунок 1.1).



Рисунок 1.1 – Схема архитектуры *ADO.NET*

## 1.6 База данных *MySQL*

База данных сайта *MySQL* – это система, предназначенная для хранения и обработки информации. Комплекс таблиц, взаимосвязанных между собой, для доступа к которым применяется система управления базами данных (СУБД) *MySQL*. По сути, *MySQL* – это специальная программа с открытым кодом, которая используется на сервере *SQL*. Данная программа не способна обрабатывать большое количество информации, однако она идеальна для небольших и крупных веб-ресурсов.

Архитектура *MySQL* очень отличается от архитектур иных серверов баз данных, что делает эту СУБД полезной для одних целей, но одновременно неудачным выбором для других. *MySQL* не идеальна, но достаточно гибка для того, чтобы хорошо работать в очень требовательных средах, например в веб-приложениях.

В то же время *MySQL* позволяет применять встраиваемые приложения, хранилища данных, индексирование содержимого, программное обеспечение для доставки, высоконадежные системы с резервированием, обработку транзакций в реальном времени (*OLTP*) и многое другое.

Для того чтобы максимально эффективно использовать *MySQL*, нужно разобраться в ее устройстве. Гибкость системы проявляется во многом. Например, вы можете настроить ее для работы на различном оборудовании и поддержки разных типов данных.

Однако самой необычной и важной особенностью *MySQL* является такая архитектура подсистемы хранения, в которой обработка запросов и другие серверные задачи отделены от хранения и извлечения данных. Подобное разделение задач позволяет выбирать способ хранения данных, а также настраивать производительность, ключевые характеристики и др.

Чтобы хорошо понимать работу сервера, нужно иметь представление о взаимодействии его компонентов. Представлен логический вид архитектуры *MySQL* (рисунок 1.2).

**

Рисунок 1.2 – Логический вид архитектуры сервера *MySQL*

На верхнем уровне располагаются службы, не являющиеся уникальными компонентами *MySQL*. Они необходимы большинству сетевых клиент-серверных инструментов или серверов: для обслуживания соединений, аутентификации, обеспечения безопасности и т. п.

Второй уровень намного интереснее. Здесь находится большая часть «мозгов» *MySQL*: код для обработки, анализа, оптимизации и кэширования запросов, а также все встроенные функции (например, функции даты/времени, математические и функции шифрования). Здесь также расположены все инструменты, используемые в подсистемах хранения, например хранимые процедуры, триггеры и представления.

Третий уровень содержит подсистемы хранения данных. Они отвечают за хранение всех данных в *MySQL* и их извлечение. Подобно различным файловым системам, доступным для *GNU* / *Linux*, каждая подсистема хранения данных имеет как сильные, так и слабые стороны. Сервер взаимодействует с ними через *API* подсистемы хранения данных. Этот интерфейс скрывает различия между такими подсистемами и делает их практически прозрачными на уровне запросов. *API* содержит пару десятков низкоуровневых функций, выполняющих операции типа «начать транзакцию» или «извлечь строку с таким первичным ключом». Подсистемы хранения не анализируют запросы *SQL* и не взаимодействуют друг с другом, они просто отвечают на исходящие от сервера запросы [9, c. 29].

## 1.7 Появление *LINQ*

По мере становления платформы .*NET Framework* и поддерживаемых ею языков *C#* и *VB*, стало ясно, что одной из наиболее проблемных областей для разработчиков остается доступ к данным из разных источников. В частности, доступ к базе данных и манипуляции *XML* часто в лучшем случае запутаны, а в худшем – проблематичны.

Проблемы, связанные с базами данных, многочисленны. Первая сложность в том, что нельзя программно взаимодействовать с базой данных на уровне естественного языка. Это приводит к синтаксическим ошибкам, которые не проявляются вплоть до момента запуска. Неправильные ссылки на поля базы данных тоже не обнаруживаются. Это может пагубно отразиться на программе, особенно если произойдет во время выполнения кода обработки ошибок. Нет ничего хуже, чем сбой механизма обработки ошибок из-за синтаксически неверного кода, который никогда не тестировался. Иногда это неизбежно из-за непредсказуемого поведения ошибки.

Вторая проблема связана с неудобством, которое вызвано различными типами данных используемыми определенным доменом данных, например, разница между типами базы данных или типами *XML* и типами данных в языке, на котором написана программа. В частности, серьезные сложности могут вызывать типы времени и даты.

Разбор итерация и манипулирование *XML* – разметкой могут быть достаточно утомительными. Часто фрагмент *XML* – это все, что нужно, но из-за требований интерфейса *W3C* *DOM* *XML* *API* объект *XmlDocument* должен быть обязательно создан, чтобы выполнять различные операции над фрагментом *XML*.

Вместо того чтобы просто добавить больше классов и методов для постепенного восполнения этих недостатков, в *Microsoft* решили пойти на один шаг дальше в абстрагировании основ запросов данных из этих конкретных доменов данных. В результате появился *LINQ* – технология *Microsoft*, предназначенная для поддержки запросов к данным всех типов на уровне языка. Эти типы включают массивы и коллекции в памяти, базы данных, документы *XML* и многое другое.

*LINQ* включает в себя около пятидесяти стандартых операций запросов, разделяемых на две большие группы – отложенные операции (выполняются не во время инициализации, а только при их вызове) и не отложенные операции (выполняются сразу)[10]. Ниже наглядно показана "градация" операций *LINQ* (рисунок 1.2):

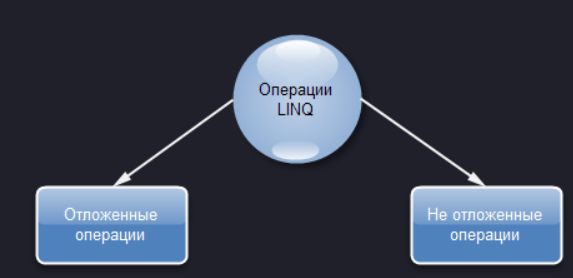


Рисунок 1.2 – Градация операций *LINQ*

# 2 АГОРИТМИЧЕСКИЙ АНАЛИЗ ПОСТАВЛЕНОЙ ЗАДАЧИ И ОПИСАНИЕ ПРОГРАММНОГО КОМПЛЕКСА

## 2.1 Архитектура программного комплекса

В данном курсовом проекте разрабатывается приложение для реализации продаж и учета проданных билетов на спектакли в театре. Приложение разрабатывается на языке *C#* с помощью *СУБД MySQL,* длядоступа к коллекциям используется *LINQ to object*.

Для разработки курсового проекта, используется графический интерфейс *WFA*. В приложении разрабатывается многопользовательский доступ к системе, что решает проблему с целостностью данных, а также с безопасностью системы.

Для доступа к базе данных используется технология *ADO.NET*. Из-за существования большого количества различных СУБД, данная технология поддерживает различные провайдеры, которые позволяют использовать одни и те же объекты для разных источников данных. В данном курсовом проекте в качестве источника данных используется СУБД *MySQL*. Для доступа к данной СУБД используется соответствующий провайдер *MySql.Data.MySqlClient*.

## 2.2 Структура базы данных

Для нашего курсового проекта, необходимо спроектировать хранилище данных, а именно, базы данных. Это делается для того, чтобы увеличить надежность своих данных и студент понимал как должна работать и строиться его программа.

В данном курсовом проекте использовалась СУБД *MySQL.* Необходимо обеспечить:

– выделение сущностей и их атрибутов, которые будут храниться в базе данных, и формирование по ним таблиц. Атомизация сложных атрибутов на более простые;

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

– определение отношений между таблицами с помощью внешних ключей;

– нормализация базы данных.

Основные задачи проектирования базы данных:

– хранение в БД всей необходимой информации;

– возможность получения данных по всем необходимым запросам;

– избыточность и дублирование данных;

– целостность базы данных.

Диаграмма базы данных (рисунок 2.1)

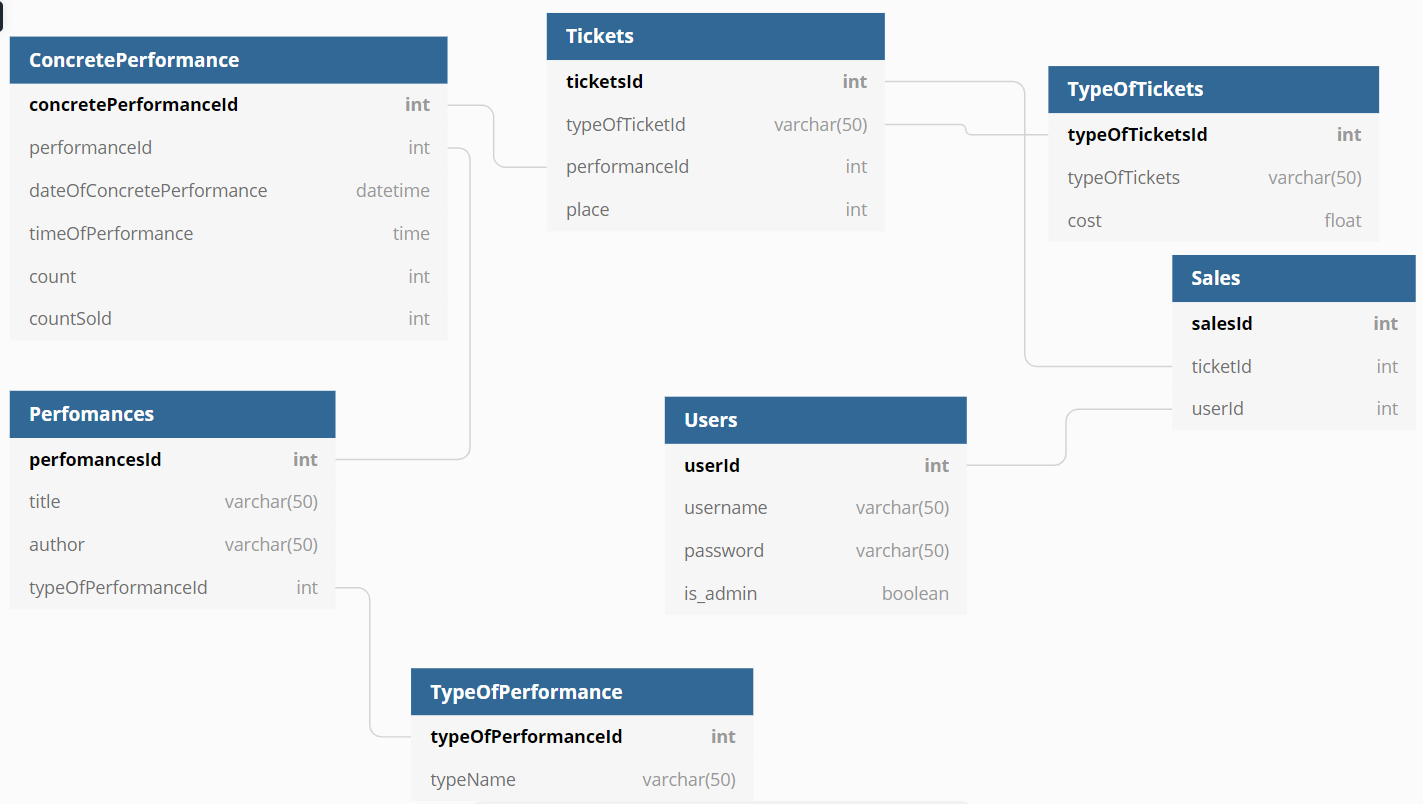


Рисунок 2.1 – Диаграмма базы данных

База данных состоит из семи таблиц:

– *Users;*

– *Sales;*

–*Tickets;*

–*TypeOfTickets;*

–*ConcretePerformance;*

–*Perfomances;*

–*TypeOfPerformance.*

Таблица *Users* содержит данные о пользователях, есть три типа пользователя:

– гость;

– зарегистрированный пользователь;

– администратор.

Таблица *Users* содержит четыре поля (рисунок 2.2)

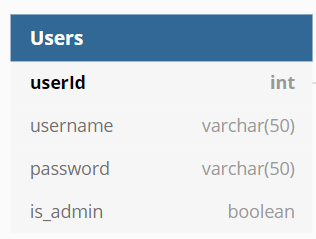


Рисунок 2.2 – Таблица *Users*

Поле *userId* хранит *id* всех пользователей, *username* – хранит данные об имени пользователя, *password* – хранит пароль пользователя, *is\_admin* проверяет кем является данный пользователь. Таблица *Users* соединяется с таблицей *Sales* по внешнему ключу *userId.*

Таблица *Sales* содержит три поля (рисунок 2.3)



Рисунок 2.3 – Таблица *Sales*

Поле *salesId* хранит *id* продажи билета, поле *ticketId* хранит *id* билета, поле *userId* хранит *id* пользователя. Таблица *Sales* соединяется с таблицами *Users* и *Tickets* по внешним ключам *ticketId* и *userId*.

В таблице *Tickets* находятся данные о билете, так же таблица содержит четыре поля (рисунок 2.4)

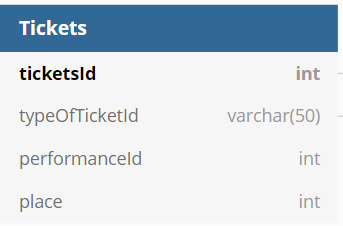


Рисунок 2.4 – Таблица *Tickets*

Поле *ticketsId* хранит *id* билета, поле *typeOfTicketId* хранит *id* тип билета, поле *performanceId* хранит *id* спектакля, поле *place* хранит посадочное место. Таблица *Tickets* соединяется с таблицами *ConcretePerformance* и *TypeOfTickets* по внешним ключам *typeOfTicketId* и *performanceId*.

В таблице *TypeOfTickets* находятся данные о типе билета, так же таблица содержит три поля (рисунок 2.5)

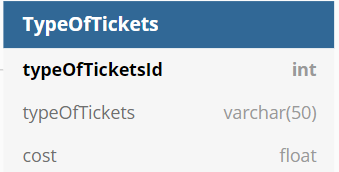


Рисунок 2.5 – Таблица *TypeOfTickets*

Поле *typeOfTicketsId* хранит *id* типа билета, поле *typeOfTickets* хранит тип билета, поле cost хранит стоимость билета. Таблица *TypeOfTickets* соединяется с таблицей *Tickets* по внешнему ключу *typeOfTicketsId*.

В таблице *ConcretePerformance* находятся данные о конкретном спектакле, так же таблица состоит из шести полей (рисунок 2.6)

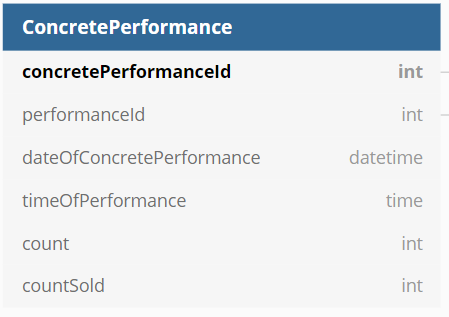


Рисунок 2.6 – Таблица *ConcretePerformance*

Поле *concretePerformanceId* хранит *id* конкретного спектакля, *performanceId* хранит *id* спектакля, поле *dateOfConcretePerformance* хранит дату конкретного спектакля, поле *timeOfPerformance* хранит время спектакля, поле *count* хранит количество мест на спектакль, поле *countSold* хранит количество проданных мест на спектакль. Таблица *ConcretePerformance* соединяется с таблицей *Perfomances* и *Tickets* по внешним ключам *perfomancesId* и *ticketsId*.

В таблице *Perfomances* находятся данные о спектакле, так же таблица состоит из четырех полей (рисунок 2.7)

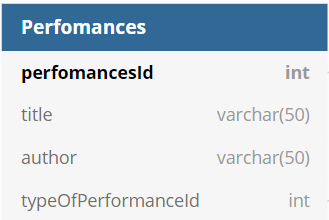


Рисунок 2.7 – Таблица *Perfomances*

Поле *perfomancesId* хранит *id* спектакля, поле *title* хранит заголовок спектакля, поле *author* хранит ФИО автора спектакля, поле *typeOfPerformanceId* хранит *id* типа спектакля. Таблица *Perfomances* соединяется с таблицей *ConcretePerformance* и *TypeOfPerformance* по внешним ключам *perfomancesId* и *typeOfPerformanceId*.

В таблице *TypeOfPerformance* находятся данные о типе спектакля, так же таблица содержит два поля (рисунок 2.8)

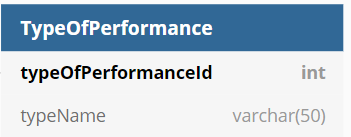


Рисунок 2.8 – Таблица *TypeOfPerformance*

Поле *typeOfPerformanceId* хранит *id* типа спектакля, поле *typeName* хранит название типа спектакля. Таблица *TypeOfPerformance* соединяется с таблицей *Perfomances* по внешнему ключу *typeOfPerformanceId*.

Все таблицы были подвергнуты процессу нормализации. Таблицы *TypeOfTickets* и *TypeOfPerformance* являются справочными таблицами, так же ни одна таблица не содержит повторяющихся строк.

## 2.3 Структура классов разработанного программного комплекса

Реализация задачи требует создание большого количества классов, именно по этому все классы нужно как-то сгруппировать, для этого используются папки (рисунок 2.9)

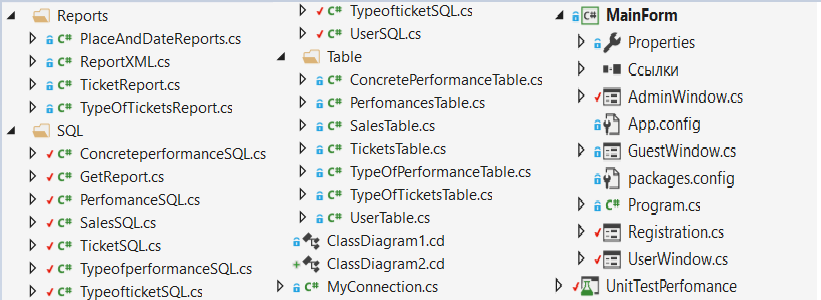


Рисунок 2.9 – Структура папок проекта

– папка *SQL* содержит классы которые реализуют операции к *CRUD;*

– папка *Table* содержит классы описывающие сущности;

– папка *Reports* содержит классы описывающие сущности отчетов.

Для сохранения целостности данных в базе данных во время выполнения программы, классы которые взаимодействуют с БД, реализуют слой *DAO*. Суть *DAO* заключается в инкапсуляции базы данных, т. Е. изменение данных осуществляется в программе, а не в самой БД, посредством реализации *CRUD* операций для заданной сущности. *CRUD* – обозначает следующие функции:

– создание;

– чтение;

– редактирование;

– удаление.

Таблица 2.1 – Таблица классов

|  |  |
| --- | --- |
| Классы | Функции |
| 1 | 2 |
| *MyConnection* | Класс, содержащий строку подключения |
| *UserTable* | Класс, описывающий сущность пользователей |
| *SalesTable* | Класс, описывающий сущность продажи билетов |
| *TypeOfTickets* | Класс, описывающий сущность типа билетов |
| *ConcretePerformance* | Класс, описывающий сущность конкретного спектакля |
| Продолжение таблицы 2.1 | |
| 1 | 2 |
| *Perfomances* | Класс, описывающий сущность спектакля |
| *TypeOfPerformance* | Класс, описывающий сущность типа спектакля |
| *UserSQL* | Содержит методы *CRUD* для таблицы *UserSQL* |
| *SalesSQL* | Содержит методы CRUD для таблицы *SalesSQL* |
| *TicketSQL* | Содержит методы CRUD для таблицы *TicketSQL* |
| *TypeofticketSQL* | Содержит методы CRUD для таблицы *TypeofticketSQL* |
| *ConcreteperformanceSQL* | Содержит методы CRUD для таблицы *ConcreteperformanceSQL* |
| *PerfomanceSQL* | Содержит методы CRUD для таблицы *PerfomanceSQL* |
| *TypeofperformanceSQL* | Содержит методы CRUD для таблицы *TypeofperformanceSQL* |
| *GetReport* | Содержит методы для реализации отчетов |
| *PlaceAndDateReports* | Класс, описывающий сущность отчета |
| *TicketReport* | Класс, описывающий сущность отчета |
| *TypeOfTicketsReport* | Класс, описывающий сущность отчета |
| *ReportXML* | Содержит методы сериализации отчетов |
| *GuestWindow* | Форма для гостя |
| *UserWindow* | Форма для зарегистрированного пользователя |
| *AdminWindow* | Форма для администратора |

Все классы содержат коллекции, где будут храниться данные конкретных сущностей БД. Заполнение коллекций осуществляется с помощью *CRUD*-операций. Все реализованные сущности в программе сопоставимы с сущностями в базе данных, что подразумевает реализацию слоя *ORM*. Это позволяет без проблем работать с данными локально в программе, а не напрямую в БД. Схема *ORM* представлена в приложении Е.

Для извлечения данных из коллекции используется технология *LINQ*. Технология *LINQ* представляет простой и удобный язык запросов к источнику данных. В качестве источника данных может выступать объект, реализующий интерфейс *IEnumerable*.

Описание классов, реализующие слой *ORM*, представлены в таблице 2.1

Для реализации поставленной задачи так же был использован один из методов ООП, наследование (рисунок 2.10)

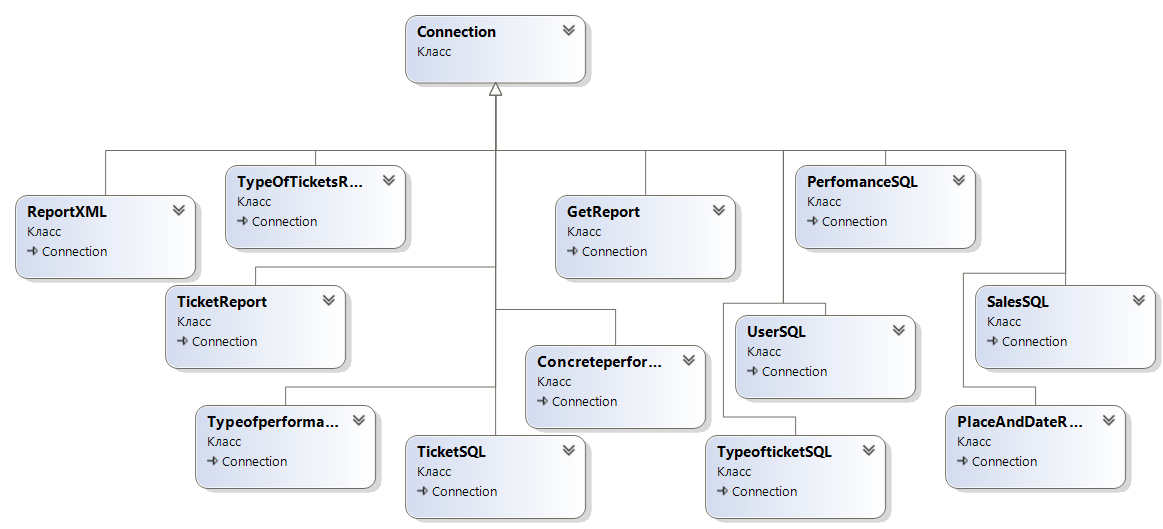


Рисунок 2.10 – Иерархия классов приложения

Класс My*Connection* представляет класс в котором содержится строка подключения к базе данных, которая передается дочерним классам для дальнейшей работы с базой данных. Описание остальных классов находится в таблице 2.1.

## 2.4 Графический интерфейс пользователя

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

*– GuestWindow;*

*– UserWindow;*

*– AdminWindow.*

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).

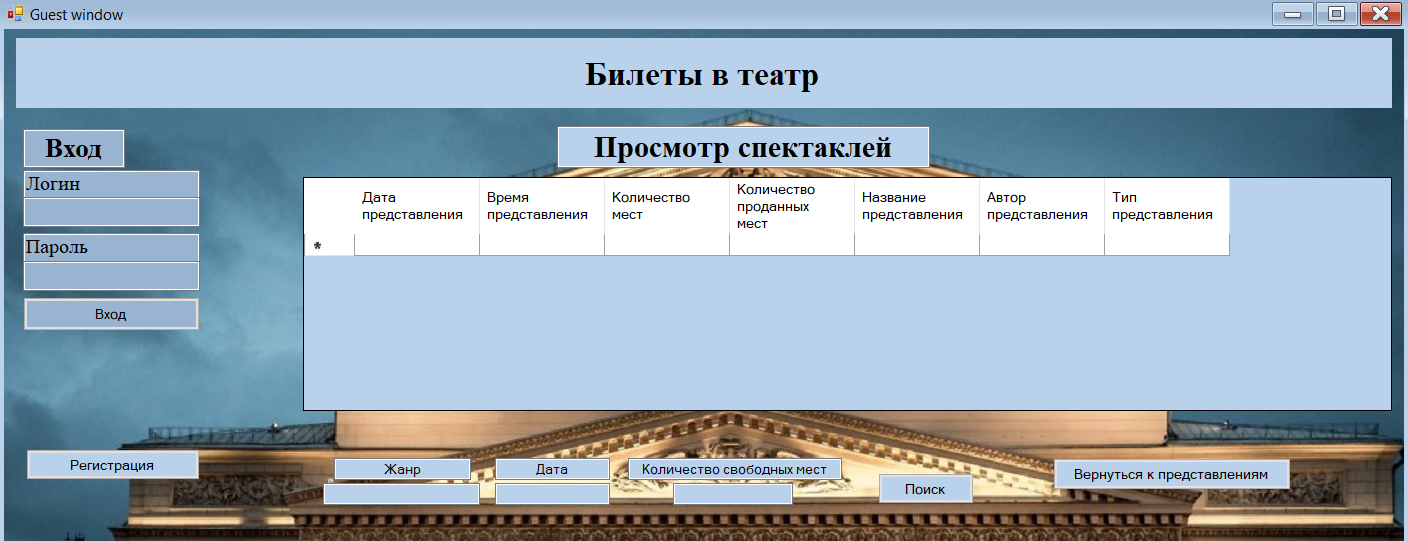


Рисунок 2.11 – Форма гостя

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)*.*

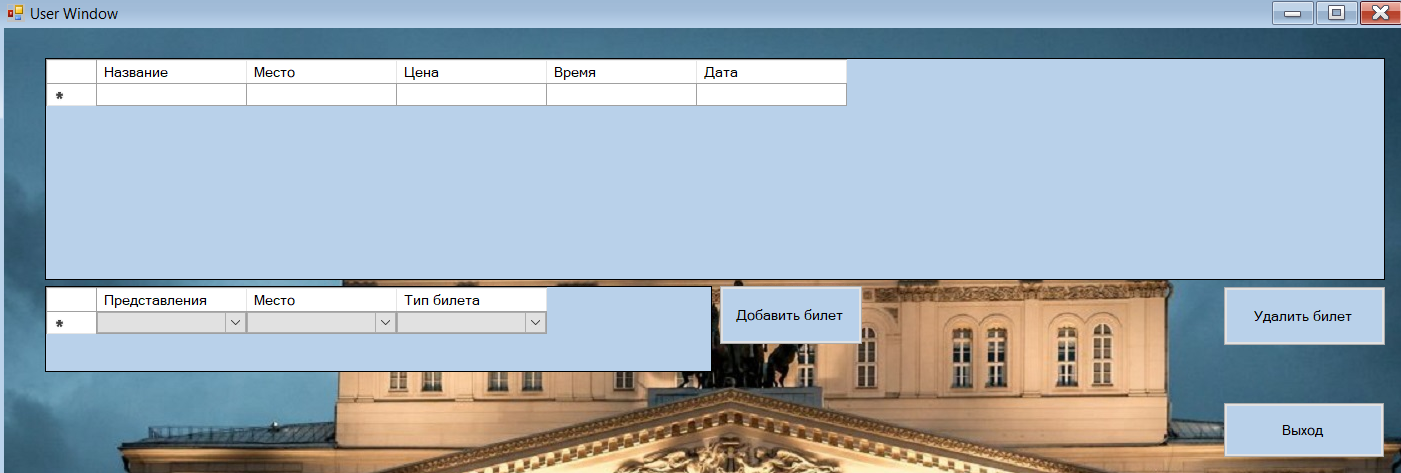


Рисунок 2.12 – Форма зарегистрированного пользователя

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).

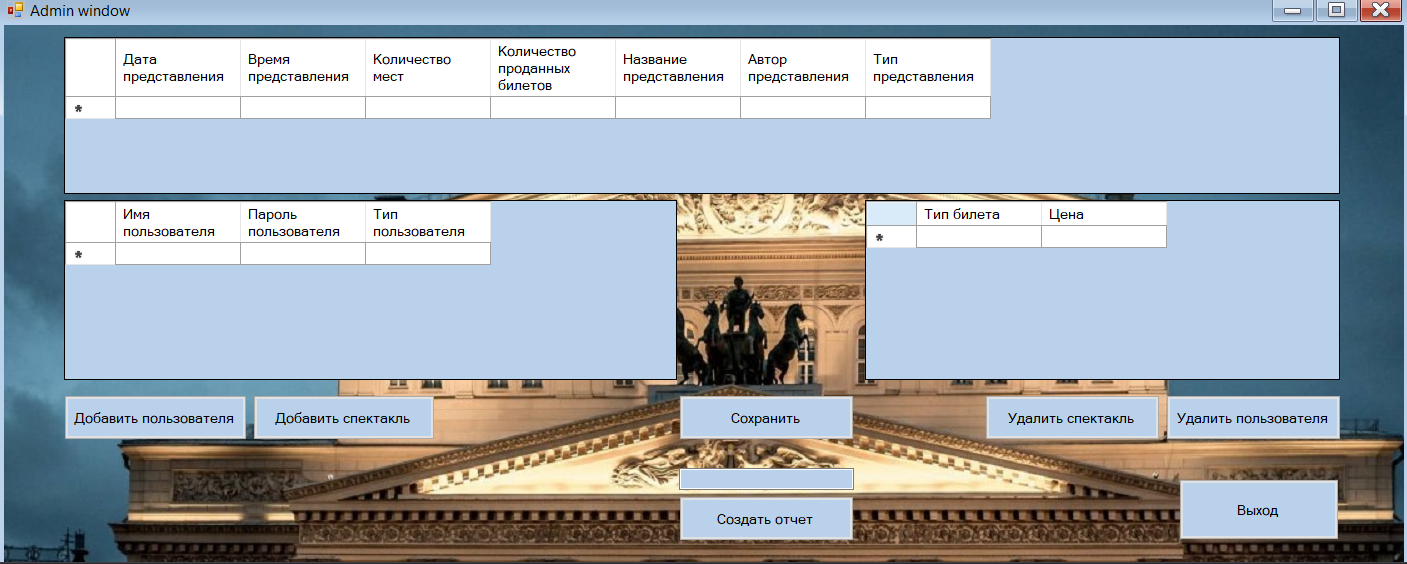


Рисунок 2.13 – Форма администратора

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).

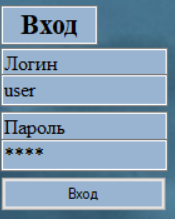


Рисунок 3.1 – Авторизация пользователя

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).

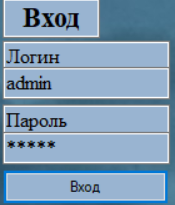


Рисунок 3.3 – Авторизация администратора

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).

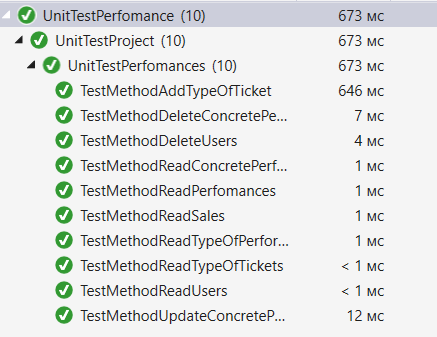


Рисунок 3.4 – Модульные тесты

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

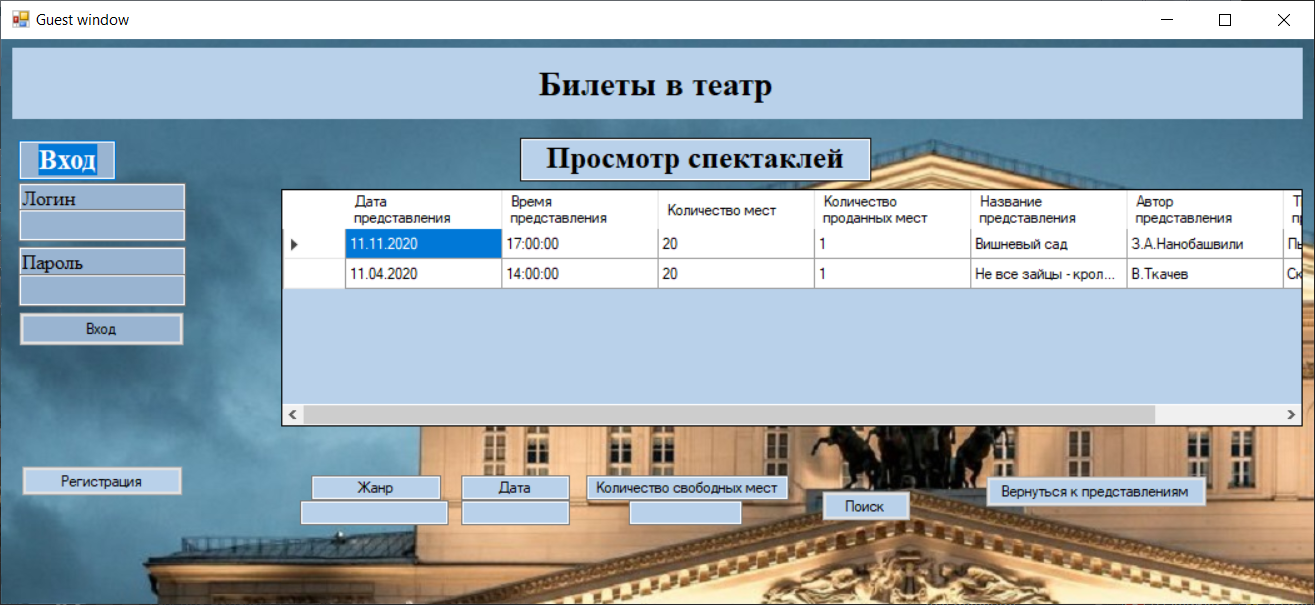


Рисунок А1 – Форма гостя

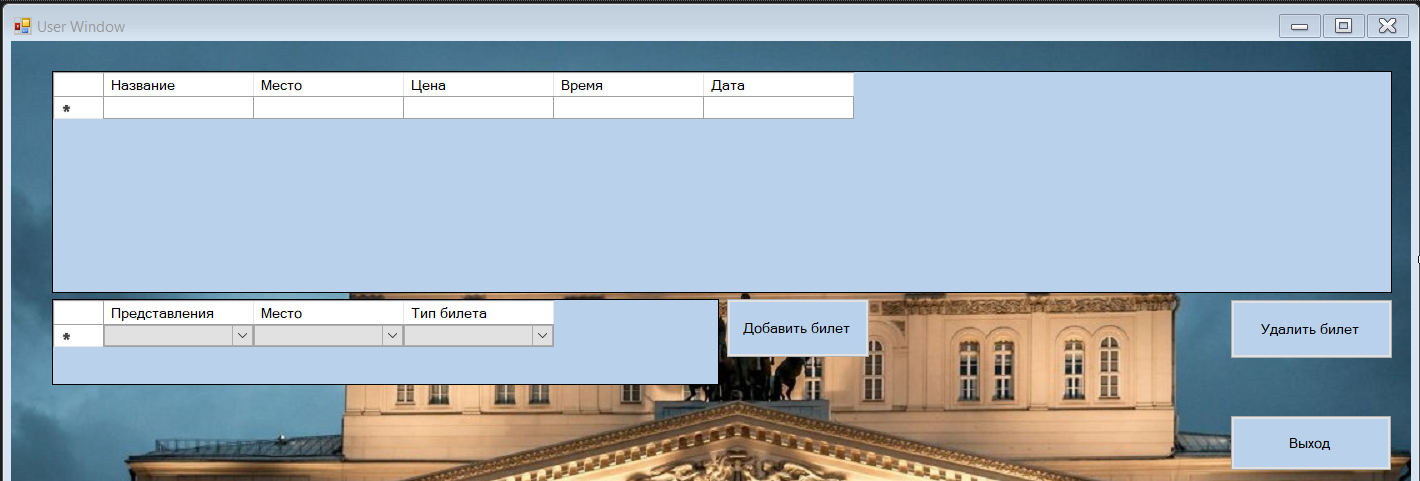


Рисунок А2 – Форма зарегистрированного пользователя

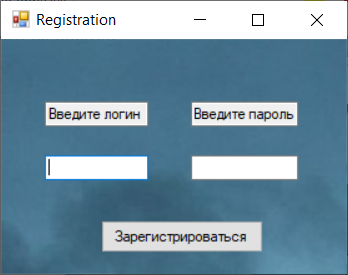


Рисунок А3 – Форма регистрации

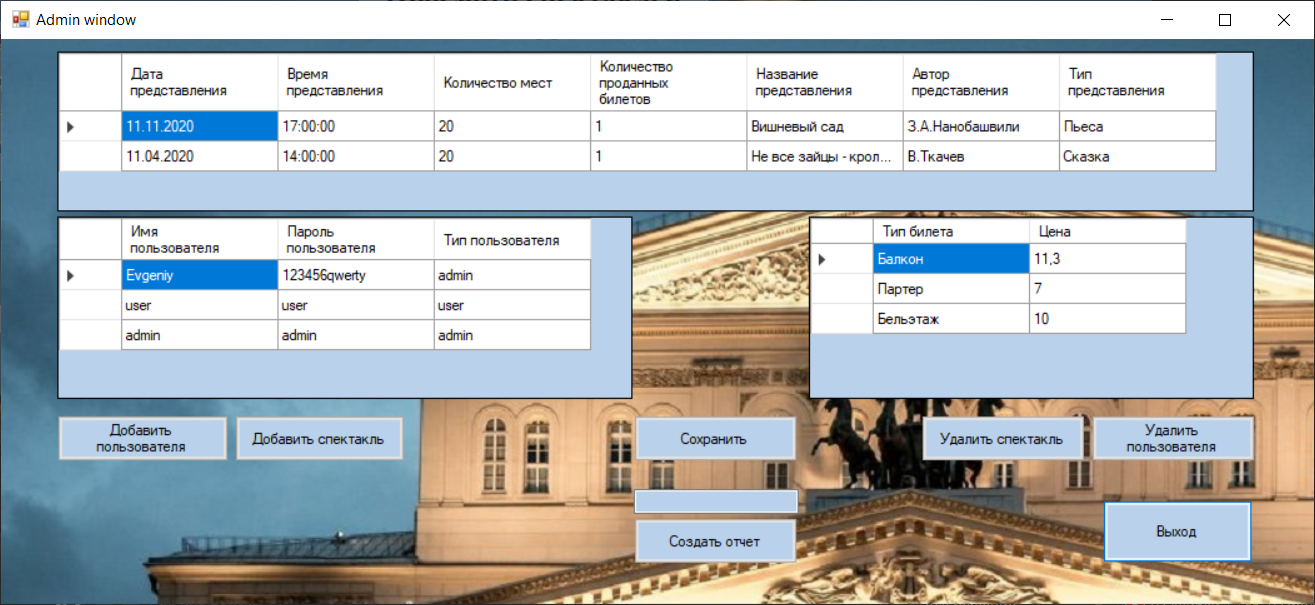


Рисунок А4 – Форма администратора

# 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