

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**  
**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**  
**«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**ИМЕНИ П. О. СУХОГО»**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к курсовой работе  
по дисциплине «Объектно-ориентированное программирование»

на тему: **«WPF ПРИЛОЖЕНИЕ ДЛЯ СИСТЕМЫ  
РАСПРОСТРАНЕНИЯ БИЛЕТОВ В ЦИРКЕ»**

Исполнитель: студент гр. ИТП-21  
Бортновская В.В.

Руководитель: доцент  
Гуменников Е.Д.

Дата проверки: \_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_

Дата защиты: \_\_\_\_\_

Оценка работы: \_\_\_\_\_

Подписи членов комиссии  
по защите курсовой работы: \_\_\_\_\_

Гомель 2022

## СОДЕРЖАНИЕ

Введение .....	3
1 Средства разработки реализации системы распространения билетов в цирке.....	4
1.1 Парадигмы программирования .....	4
1.2 Основы объектно-ориентированного представления .....	5
1.3 Средство доступа к данным <i>LINQ to XML</i> .....	7
1.4 Расширяемый язык разметки <i>XML</i> .....	8
1.5 Паттерное проектирование .....	9
2 Реализация системы билетов в цирке .....	11
2.1 Понятие архитектуры.....	11
2.2 Используемые паттерны проектирования .....	12
2.3 Общая архитектура приложения .....	14
2.4 Общая структура приложения .....	16
3 Верификация и тестирования системы реализации билетов в цирке города .....	21
3.1 Модульное тестирование приложения.....	21
Заключение .....	26
Список используемых источников .....	27
Приложение А .....	28
Приложение Б.....	29
Приложение В .....	30
Приложение Г .....	31
Приложение Д.....	32

## ВВЕДЕНИЕ

Начиная с XX века люди начали пользоваться одним из величайших достижений человечества – компьютерами. Персональные компьютеры и прочие вычислительные машины, такими, какими люди их знают сейчас, появились относительно недавно и на сегодняшний день они стали неотъемлемой частью повседневной жизни. Без вычислительной техники на сегодняшний день очень сложно представить общество, да и без неё жизнь общества значительно бы замедлилась. С появлением вычислительной техники стали активно развиваться и информационные технологии. Информационные системы и технологии на сегодняшний день весьма востребованы и актуальны. Ежедневно их роль только возрастает.

Информационные технологии помогают значительно упростить или автоматизировать многие рабочие процессы, которые необходимо выполнять человеку. Поэтому они активно внедряются предприятиями и компаниями в свою бизнес среду. За счёт внедрения новейших информационных технологий компании способны уменьшить затраты труда в процессе производства и значительно увеличить свою производительность труда. Что благоприятно сказывается на прибыли, получаемой предприятием.

Так, например, компании могут внедрять информационные системы, которые позволяют полностью контролировать свою деятельность, при этом иметь возможность собирать и анализировать информацию по работе предприятия, а также предусмотрена возможность работы клиентами с данной системой, что помогает им самим принимать решения и получать информацию без вмешательства третьего лица.

Актуальность темы данной курсовой работы обусловлена важностью разработки приложения-помощника в структуризации данных, ведении учета проданных/доступных билетов, что значительно облегчает задачу сотрудников данной области продаж.

Тема данной курсовой работы не является чем-то новым в сфере услуг, так как потребность в упрощения в системе учета появилась давно из-за большого объема данных, которые необходимо хранить, постоянно обновлять и упорядочивать.

Информационная система должна обладать надёжностью и быть спроектирована с учетом всех особенностей того места, где она будет использоваться. Первым этапом при разработке информационной системы является выбор инструментов и среды разработки. Затем происходит моделирование объектов, устанавливается взаимосвязь между объектами информационной системы, после чего проектируется схема данных, предназначенная для хранения информации. После чего происходит моделирование программных комплексов и их взаимодействие между собой в системе.

# **1 СРЕДСТВА РАЗРАБОТКИ РЕАЛИЗАЦИИ СИСТЕМЫ РАСПРОСТРАНЕНИЯ БИЛЕТОВ В ЦИРКЕ**

## **1.1 Парадигмы программирования**

Существует большая численность всевозможных раскладов к созданию программного обеспечения. Любая организация, занимающаяся разработкой программных товаров, содержит свод правил, который применяется программистами при проектировании программных систем, при написании кода, при испытании готовых товаров. Есть определённые критерии и советы, которые считаются совместными и применяются не лишь только в какой-нибудь определенной фирмы, а по всему миру. Например, люди, как правило, выбирают дорогу меньшего сопротивления, разработчики программного обеспечения не исключение. Поэтому для написания кода были выдуманы и разработаны критерии, советы, и именуемые парадигмы программирования.

Парадигма программирования – это набор идей, понятий, определяющих стиль написания кода для создания компьютерных программ [1, с. 24]. В первый раз данным термином воспользовался Роберт Флойд в 1978 году в собственной лекции лауреата премии Тьюринга. Данный термин не содержит четкого определения, почти все создатели выделяют различные определения. Дэниел Бобров в собственной заметке определяет парадигму как «стиль программирования, как описания целей программиста». Брюс Шрайвер разъясняет парадигму программирования как «модель или же расклад к заключению проблемы».

Первой парадигмой программирования считается императивное программирование. Императивная программа представляет собой набор команд, который обязан исполнить компьютер [2, с. 63]. От сего и пришло заглавие, например, как слово «imperative» переводится с британского как веление. То есть в некоем значении программа приказывает компьютеру, собственно, что он обязан. В начальном коде программы записываются инструкции, которые обязаны производиться поочередно. При выполнении, данные могут сберегаться в память и могут быть получены надлежащими инструкциями из памяти. Для сего в языках программирования применяется присваивание значений именованным переменным.

В 1970-е годы с подъемом сложности решаемых задач нужен был иной расклад к решению трудных задач, в следствие этого понадобилась свежая парадигма и ей стало структурное программирование.

Значение структурного программирования заключается в том, что надо логически поделить программу на структурные блоки. В согласовании с парадигмой, каждая программа, состоит из трёх базисных структурных блоков: последовательность, ветвление, цикл.

Структурное программирование стало почвой всего последующего становления методов и подходов к конструированию программ. Гигантская доля передовых языков программирования поддерживает эту структурную парадигму.

Есть еще функциональное программирование, которое считается как парадигмой программирования, как и разделом дискретной математики. Сущность парадигмы в том, что процесс вычисления трактуется как вычисление значений всевозможных функций.

Функциональное программирование подразумевает обходиться лишь только вычислением результатов функций от начальных данных и не подразумевает какое-либо очевидное сбережение состояния программы. В сопоставлении с императивным подходом, в котором при вызове одной и той же функции с схожими аргументами имеет возможность выделяться результат в зависимости от состояния программы, в функциональном подходе ответ остается одинаковым.

Самой популярной на данный момент парадигмой программирования считается объектно-ориентированная. В связи увеличивающейся сложностью программного обеспечения необходима была свежая методология программирования, которая бы решала задачи на более высоком уровне, чем структурное программирование.

Главная мысль объектно-ориентированного подхода состоит в том, что программа представляется в виде совокупности объектов [3, с. 28]. Любой объект представлен экземпляром определённого класса, те же в собственную очередь могут создавать иерархии.

## **1.2 Основы объектно-ориентированного представления**

В связи с развитием информационных технологий, сильно возрастает сложность программного обеспечения. Внутренние архитектуры, на первый взор несложных программ, могут быть весьма огромных масштабов. Эти архитектуры могут разрабатываться и поддерживаться бесчисленными командами создателей и тестировщиков. С самого начала эпохи компьютеров всё усложняется и для борьбы с этим по части программного обеспечения есть объектно-ориентированное программирование, более обширно применяемое при разработке программных продуктов.

Объектно-ориентированное программирование – это парадигма программирования, в которой программы состоят из объектов. Объекты, в свою очередь, это сущности, обладающие конкретными свойствами и поведением [4, с. 42].

Объект – это владеющий именем комплект данных (полей и свойств), на физическом уровне оказавшихся в памяти компьютера, и методов, имеющих доступ к ним. Имя применяется для работы с полями и методами объекта. Объекты одного типа обязаны иметь одни свойства. В объектно-ориентированном программировании тип принято именовать классом.

Класс – это комплексный тип данных, состоящий из единственного комплекта полей (именованных переменных, являющихся собственностью класса) и методов (функций для работы с данными полями). Класс является моделью информационной сущности с интерфейсами для оперирования содержимым. То есть он описывает какими свойствами и поведением станет владеть объект этого класса.

Объектно-ориентированное программирование подключает в себя четыре ведущих принципа: абстракция, наследование, инкапсуляция, полиморфизм.

Абстракция – это более незатейливый и понятный принцип, означающий выделение более важных, ключевых данных предмета для решаемой задачи и отбрасывание лишнего. То есть происходит процесс абстрагирования от реального объекта для выделения лишь только подходящих данных.

Другой пример. В конце месяца вы сняли с банковской карты зарплату. Теперь ваша общая мысль – «деньги». Это тоже абстракция. В процессе абстрагирования вы мысленно исключили (отделили) несущественные связи объекта.

Наследование – это принцип, позволяющий описать новый (дочерний) класс на базе уже имеющегося (родительского) класса с абсолютной или же отчасти заимствующейся функциональностью. Благодаря наследованию уменьшается количество повторяющегося кода, потому что в классах потомках можно использовать поля и методы, описанные в родительских. Вследствие чего, классы становятся проще и понятнее. Недостающую функциональность можно выделить в отдельный класс на основе имеющегося и наоборот, можно взять существующую функциональность и добавить в имеющийся класс для расширения его возможностей.

Существуют беспроводные телефоны, которые работают от батареи и проводные телефоны, которым батарея не нужна, оба вида телефонов могут звонить, следовательно, их можно унаследовать от базового класса обычного телефона. Проводные телефоны по способу управления делятся на кнопочные, дисковые с ручной коммутацией. Беспроводные телефоны по виду связи делятся на сотовые, спутниковые и радиотелефоны. Сотовые же в свою очередь можно разделить на ещё несколько типов по пользовательским возможностям: базовый, с мультимедиа и смартфон. Каждый дочерний класс сохраняет все свойства родительского и добавляет что-то ещё.

Инкапсуляция – это принцип, означающий размещение данных и методов в одном классе и сокрытие деталей реализации с помощью ограничения доступа к данным. Другими словами, данные помещаются в какой-то корпус и получается «чёрный ящик», с которым можно взаимодействовать с помощью открытого интерфейса.

Вот представьте себе телевизор без корпуса. Человек хочет переключить канал, подходит к телевизору, и вместо того, чтобы нажать кнопку переключения канала, он выдергивает какой-то проводок (ну не имел он никогда дела с телеви-

зорами или пьяный). Канал не переключился, результат не тот. В программировании это равноценно вызову вспомогательного метода, непредназначенного для прямого вызова. Человеку свойственно ошибаться, инкапсуляция позволяет избавить программиста от допущения подобного рода ошибок.

Полиморфизм – это принцип, позволяющий использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Автор языка программирования C++ Бьёрн Страуструп определил полиморфизм как «один интерфейс – много реализаций».

Например, вверху иерархии классов стоит класс *Animal*. Его наследуют три класса – *Cat*, *Dog* и *Cow*. У класса *Animal* есть метод *voice*. Этот метод выводит на экран сообщение «Голос». Естественно, ни собака, ни кошка не говорят «Голос». Они гавкают и мяукают. Соответственно, Вам нужно задать другой метод для классов *Cat*, *Dog* и *Cow* – чтобы кошка мяукала, собака гавкала, а корова говорила «Муу».

### 1.3 Средство доступа к данным *LINQ to XML*

*LINQ to XML* обеспечивает интерфейс программирования для работы с *XML* в памяти на основе платформы *.NET LINQ FRAMEWORK*. Интерфейс *LINQ to XML* использует возможности *.NET* и может быть сравним с обновленным, переработанным программным интерфейсом *XML* модели *DOM*.

*XML* широко используется для форматирования данных в целом ряде контекстов. Примеры *XML* можно обнаружить в веб-среде, в файлах конфигурации, в файлах *Microsoft Office Word* и в базах данных.

*LINQ to XML* является обновленным, переработанным подходом к программированию с помощью *XML*. Он позволяет изменять документы в оперативной памяти на основе модели *DOM* и поддерживает выражения запросов *LINQ*. Хотя в синтаксическом отношении эти выражения запросов отличаются от *XPATH*, они предоставляют аналогичные функциональные возможности.

*LINQ to XML* похожа на модель *DOM* в том, что он переносит *XML*-документ в память. К такому документу можно направить запрос, его можно изменить, а после изменения его можно сохранить в файле или сериализовать и передать через Интернет, однако *LINQ to XML* отличается от *DOM*:

- она предоставляет новую объектную модель с более легким весом и простотой работы;
- он использует преимущества языковых функций *C#* и *Visual Basic*.

Наиболее важным преимуществом *LINQ to XML* является его интеграция с *Language-Integrated Query (LINQ)*. Эта интеграция дает возможность создавать запросы к загруженному в память *XML*-документу с целью получения коллекций элементов и атрибутов. Возможности запросов *LINQ to XML* сравнимы по функциональности (хотя и не в синтаксисе) с *XPath* и *XQuery*. Интеграция *LINQ* в *C#* и *Visual Basic* обеспечивает более надежную типизацию.

## 1.4 Расширяемый язык разметки XML

*XML (eXtensible Markup Language)* – расширяемый язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов как программами, так и человеком, с акцентом на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Расширение XML – это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих, какие атрибуты и элементы могут входить в состав других элементов. Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как, собственно, XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах.

С физической точки зрения документ состоит из сущностей, из которых каждая может ссылаться на другую сущность. Единственный корневой элемент – документная сущность. Содержание сущностей – символы.

С логической точки зрения документ состоит из комментариев, объявлений, элементов, ссылок на сущности, инструкций обработки. Всё это в документе структурируется разметкой.

Сущность – мельчайшая часть в документе. Все сущности что-нибудь содержат, и у всех них есть имя (существуют исключения, напр. документная сущность).

Документ состоит из сущностей, содержание которых – символы. Все символы разделены на два типа: символы данных и символы разметки. Часть документа, не принадлежащая разметке, составляет символьные данные документа.

Все составляющие части документа обобщаются в пролог и корневой элемент. Корневой элемент – обязательная часть документа, составляющая всю его суть (пролог, вообще говоря, может отсутствовать). Корневой элемент может включать (а может не включать) вложенные в него элементы, символьные данные и комментарии. Вложенные в корневой элемент элементы, в свою очередь, могут включать вложенные в них элементы, символьные данные и комментарии, и так далее. Пролог может включать объявления, инструкции обработки, комментарии. Его следует начинать с объявления XML, хотя в определённой ситуации допускается отсутствие этого объявления.

Элементы документа должны быть правильно вложены: любой элемент,



начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался.

Символьные данные могут встречаться внутри элементов как непосредственно, так и в специальных секциях «*CDATA*». Объявления, инструкции обработки и элементы могут иметь связанные с ними атрибуты. Атрибуты используются для связывания с логической единицей текста пар имя-значение.

В языке *XML* все имена должны начинаться с буквы, символа подчёркивания или двоеточия и продолжаться только допустимыми для имён символами, а именно: они могут содержать только буквы, входящие в секцию букв кодировки *Unicode*, арабские цифры, дефисы, знаки подчёркивания, точки и двоеточия. Однако имена не могут начинаться со строки «*xml*» в любом регистре. Имена, начинающиеся с этих символов, зарезервированы для использования консорциумом *W3C*. Нужно помнить, что так как буквы не ограничены исключительно символами *ASCII*, то в именах можно использовать слова из родного языка.

## 1.5 Паттерное проектирование

Паттерны – это набор готовых решений, рецепты, предлагающие к повторному использованию самое ценное для разработчика – сплав мирового опыта по созданию программного обеспечения. Паттерны позволяют разными способами решать ту или иную задачу, с которыми постоянно сталкиваются проектировщики объектно-ориентированных приложений.

Порождающие паттерны делают систему независимой от способа создания, композиции и представления объектов. Для порождающих паттернов актуальны две темы. Во-первых, эти паттерны инкапсулируют знания о конкретных классах, которые применяются в системе. А во-вторых, скрывают детали того, как эти классы создаются и стыкуются. Порождающие паттерны обеспечивают гибкость в вопросе, что создаётся, кто это создаёт, как и когда.

Фабричный метод – паттерн, порождающий классы.

Фабричный метод паттерн, который определяет интерфейс для создания объекта, но оставляет подклассам решение о том, объект такого класса создавать. Позволяет делегировать создание объектов классам-наследникам.

Применяется, когда:

- классу заранее неизвестно, объекты каких классов ему нужно создавать;
- класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами;
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и вы планируете локализовать знание о том, какой класс принимает эти обязанности на себя.

Плюсы паттерна фабричный метод: предоставляет подклассам операции-

зацепки, соединяет параллельные иерархии.

Минус паттерна в том, что для каждого нового продукта необходимо создавать свой класс создателя.

Одиночка – паттерн, который гарантирует создание только одного экземпляра для определенного класса, и предоставляет ему глобальную точку доступа.

Используется паттерн одиночка в тех случаях, когда:

- должен быть один единственный экземпляр некоторого класса, легко доступный всем клиентам;
- единственный экземпляр должен расширяться путем порождения подклассов, и клиентам нужно иметь возможность работать с расширенным экземпляром без модификации своего кода.

В результате обзора порождающих шаблонов проектирования для решения задачи курсовой работы выбраны такие паттерны, как фабричный метод и одиночка [2, с. 32].

Паттерны пытаются стандартизировать подходы, которые и так уже широко используются. Эта стандартизация кажется некоторым людям догмой и они реализуют паттерны «как в книжке», не приспособивая паттерны к реалиям проекта.

Но осознанное владение инструментом как раз и отличает профессионала от любителя. Вы можете забить гвоздь молотком, а можете и дрелью, если сильно постараетесь. Но профессионал знает, что главная фишка дрели совсем не в этом. Итак, зачем же знать паттерны?

- проверенные решения. Вы тратите меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений вы смогли бы додуматься и сами, но многие могут быть для вас открытием.

- стандартизация кода. Вы делаете меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.

- общий программистский словарь. Вы произносите название паттерна, вместо того, чтобы час объяснять другим программистам, какой крутой дизайн вы придумали и какие классы для этого нужны.

## 2 РЕАЛИЗАЦИЯ СИСТЕМЫ БИЛЕТОВ В ЦИРКЕ

### 2.1 Понятие архитектуры

Архитектура приложения разделена на три фундаментальные части:

- библиотека классов, описывающая слой доступа к данным;
- библиотека классов, описывающая бизнес-логику;
- проект классов для работы с графическим интерфейсом *WPF*.

Схема общей архитектуры приложения представлена на рисунке 2.1.1.

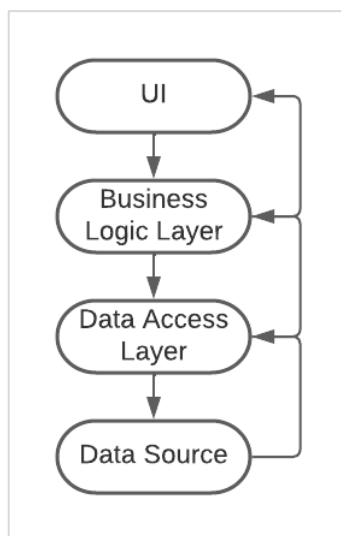


Рисунок 2.1 – Схема общей архитектуры приложения

Первым уровнем является *Data Access Layer* или уровень доступа к данным. Этот уровень обычно содержит все модели данных, хранящихся в БД, а также классы, через которые идет взаимодействие с БД.

В качестве структурных и порождающих шаблонов проектирования используются:

- доступ к данным *LINQ*;
- одиночка (*Singleton*);
- репозиторий (*Repository*).

Паттерн «Репозиторий» как правило реализует *CRUD*-интерфейс, то есть представляет операции по:

- извлечению;
- добавлению;
- редактированию;
- удалению данных.

Таким образом паттерн «Репозиторий» является посредником между слоем доступа к данным и доменным слоем, работая как *in-memory* коллекция доменных объектов. Клиенты создают декларативные описания запросов и передают их в репозиторий для выполнения.

В приложениях часто приходится оперировать множеством сущностей и моделей, для управления которыми создается также большое количество репозитория. Паттерн *Unit of Work* помогает упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст базы данных.

На рисунке 2.1.2 представлена схема проекта с использованием паттерна *Repository*.

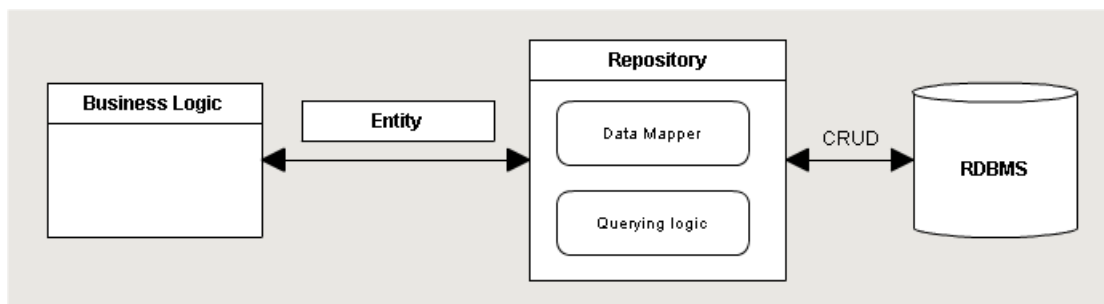


Рисунок 2.2 – Схема проекта с использованием паттерна *Repository*

В архитектурном шаблоне проектирования *MVC* компонент *Model* описывает классы, слой доступа к данным. В качестве слоя доступа к данным, выбрав шаблон проектирования *LINQ*, а для классов описывающие сущности создана цепочка наследования и получена иерархия.

Компонент *Services* содержит в себе бизнес-логику приложения по управлению авторизацией пользователя, взаимодействует с представлением, обрабатывая данные, полученные из слоя доступа и приводя их в пригодный формат для вывода. Для реализации этого получим классы-сервисы, реализующий шаблон проектирования «Одиночка», код созданных классов содержится в приложении Б.

Компонент *View* содержит представления видимое конечному пользователю. Взаимодействует с контроллерами, получая из них данные для вывода и передавая им данные о действиях пользователя. Технология, реализующая представление для данной информационной системе – *WPF*.

Приложение реализующие архитектурный шаблон проектирования *MVC*, является масштабируемым и поддерживаемым в дальнейшей разработке сторонними разработчиками.

## 2.2 Используемые паттерны проектирования

Первым этапом при создании программного комплекса, реализующего информационно систему, является создание приватного репозитория на *GitHub*, для контроля версий проекта и возможности комфортно ввести разработку вне зависимости от оборудования.

В качестве базы данных в данном проекте выступают *XML*-файлы, которые служат для хранения следующих сущностей:

- цирк;
- представление;
- роль;
- билет;
- категория;
- отчет;
- пользователь.

Для сущности «Билет» была выделена абстрактная сущность «Билет» и создана иерархия наследования:

- категория билета;
- идентификационный номер представления;
- стоимость;
- количество.

В сущности «Представление» были выделены следующие атрибуты:

- название;
- слоган;
- дата.

В сущности «Роль» выделен один атрибут: «Имя».

Так как информационная система имеет многопользовательский доступ. Для сущностей пользователь и администратор можно выделить абстракцию в виде сущности «Пользователь», так как эти сущности будут пользователями данной информационной системы. Они будут связаны общим отношением и иметь атрибуты (данные для авторизации в системе):

- логин;
- пароль;
- имя;
- идентификационный номер.

В информационной системе данные сущности будут иметь разные права доступа и возможности.

Для сущности «Цирк» выделено отношение с атрибутами:

- наименование цирка;
- категории билетов.

Для добавления представления в программу, требуется заполнить специальную форму, указав все требующие данные. Добавить в систему данные может только администратор. Зарегистрированный пользователь может просматривать афишу и заказать билеты. Пользователи, не прошедшие регистрацию, являются гостями и могут просматривать только афишу.

## 2.3 Общая архитектура приложения

Данное приложение состоит из трёх проектов:

- *CircusCourseWork* – проект графического интерфейса;
- *CircusDataAccessLibrary* – проект слоя доступа к данным;
- *CircusTests* – проект модульного теста.

Слой доступа к данным содержит модели, описывающие сущности, необходимые для корректной работы приложения. Каждая модель содержит уникальный идентификатор (*Id*). Описание классов моделей приведено ниже.

Класс *Perfomance* содержит следующие поля:

- *Name* – название представления;
- *Slogan* – слоган представления;
- *Date* – дата премьеры.

Класс *Ticket* содержит следующие поля:

- *Price* – стоимость билета (без учета категории);
- *PerformanceId* – идентификационный номер представления;
- *TicketCategoryInfoId* – идентификационный номер категории билета;
- *CustomerUserId* – идентификационный номер покупателя.

Класс *TicketCategoryInfo* содержит следующие поля:

- *Name* – название категории билета;
- *Price* – коэффициент, на который умножается цена;
- *Count* – количество.

Класс *User* содержит следующие поля:

- *Name* – имя;
- *Login* – логин;
- *Password* – пароль;
- *RoleId* – идентификационный номер роли.

*CRUD* – акроним, обозначающий четыре базовых функции, используемые при работе с хранилищами данных:

- создание;
- чтение;
- изменение;
- удаление.

В интерфейсах описываемы данным шаблоном проектирования помимо *CRUD* операций, могут быть описаны другие сигнатуры методов совершающие более сложные операции с данными в зависимости от требуемой задачи.

Обобщённый интерфейс *IRepository<T>* (где *T* – тип модели) содержит методы для реализации *CRUD*-операций:

- *Create(entity)* – создание сущности;
- *GetFirst(condition)* – получение первой сущности (по условию или без);
- *Get(condition, page, pageSize)* – получение списка сущностей (по условию или без, с возможностью пагинации);

- *Update(entity)* – обновление сущности;
- *Delete(id)* – удаление сущности по уникальному идентификатору;
- *Clear()* – очистка списка сущностей;
- *LoadAll(entities)* – загрузка всех сущностей из *XML*-файла.

Интерфейсы, относящиеся к конкретным сущностям, наследуются от интерфейса *IRepository<T>* и могут содержать дополнительные методы для определённых сущностей. Например, *IPerformanceRepository* наследуется от *IRepository<Performance>*. Однако в рамках данного приложения эти интерфейсы не содержат дополнительного функционала.

Интерфейс *XmlRepository* представляет паттерн *Unit Of Work*. Класс, реализующий этот интерфейс, содержит методы *Save()* для сохранения данных в *XML*-файлы и *Load()* для получения данных из файлов. Данный интерфейс также служит для координации работы репозиториях всех необходимых сущностей проекта. *XmlRepository* содержит следующие типы репозиториях:

- *CircusXmlRepository*;
- *PerformanceXmlRepository*;
- *RoleXmlRepository*;
- *TicketCategoryInfoXmlRepository*;
- *TicketXmlRepository*;
- *UserXmlRepository*.

В каждом из классов репозиториях для определённых сущностей содержится одинаковый методов, реализующих операции создания, чтения, обновления и удаления, а также в определённых репозиториях содержатся события и методы, с помощью которых реализуется каскадное удаление. Каскадное удаление реализовано для сущностей, связанных с другими сущностями.

Проект слоя доступа к данным содержит классы и интерфейсы, отвечающие за авторизацию и остальные действия, относящиеся к пользователям:

- *LoginViewModel* – класс, отвечающий за авторизацию пользователя и валидацию логина и пароля;
- *RegisterViewModel* – класс, отвечающий за создание, получение, обновление и удаление пользователей.

За атрибуты авторизации отвечает класс *Auth*, наследуемый от класса *IAuth*.

Проект графического интерфейса содержит формы для входа, регистрации, создания представления, покупки билетов, а также главную форму *MainForm*.

Также проект *CircusCourseWork* включает в себя пользовательские элементы управления для администратора, гостя и зарегистрированного пользователя.

Проект модульного тестирования включает в себя классы, содержащие методы для тестирования определённых сущностей, например, *RepositoriesCollection*, *XmlRepositoriesFixture*. Также проект *CircusTests* содержит папку *Data*, в которой хранятся *XML*-файлы, необходимые для корректного тестирования.

## 2.4 Общая структура приложения

Для создания пользовательского интерфейса использована технология *WPF*. Интерфейс обеспечивает взаимодействие пользователя с программной системой и служит переводчиком в диалоге пользователя и программы.

В реализованной программе используются такие элементы пользовательского интерфейса, которые показаны в таблице 2.1.

Таблица 2.1 – Элементы графического интерфейса пользователя

Элементы	Описание
Окна	Отображают на экране различную информацию
Меню	Текстовый ввод команд заменяется выбором команд из меню
Указатели	В качестве устройства указания команд в меню и отдельных элементов в окне используют мышшь
Кнопки	Реализуют определенные функции
Поля ввода	Реализуют текстовый ввод данных в поле

При запуске приложения в первую очередь открывается окно авторизации. При нажатии на кнопку «Зарегистрироваться» открывается окно, где пользователь может ввести свое имя, логин и пароль (рис. 2.3).

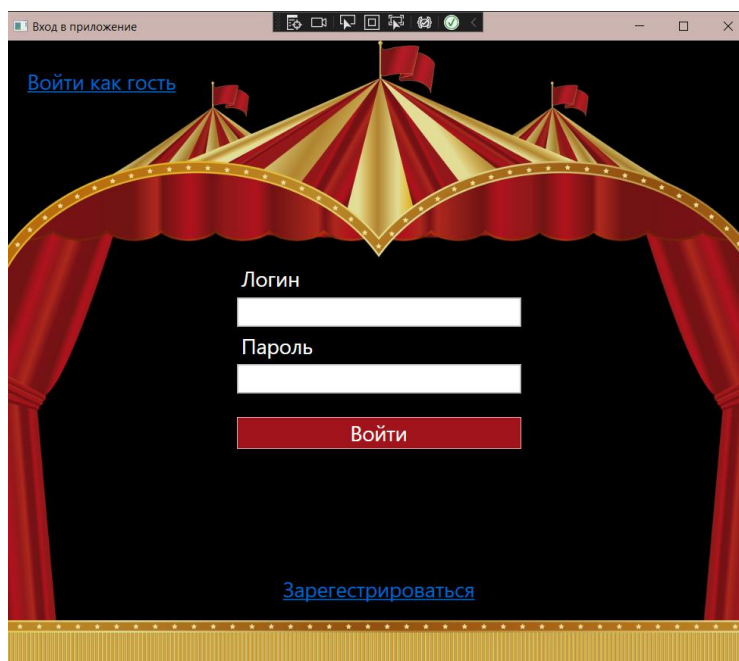


Рисунок 2.3 – Главное окно



На рисунке 2.4 изображено главное окно, где пользователь системы может ввести свой логин и пароль, тем самым авторизоваться в системе по определённой роли.

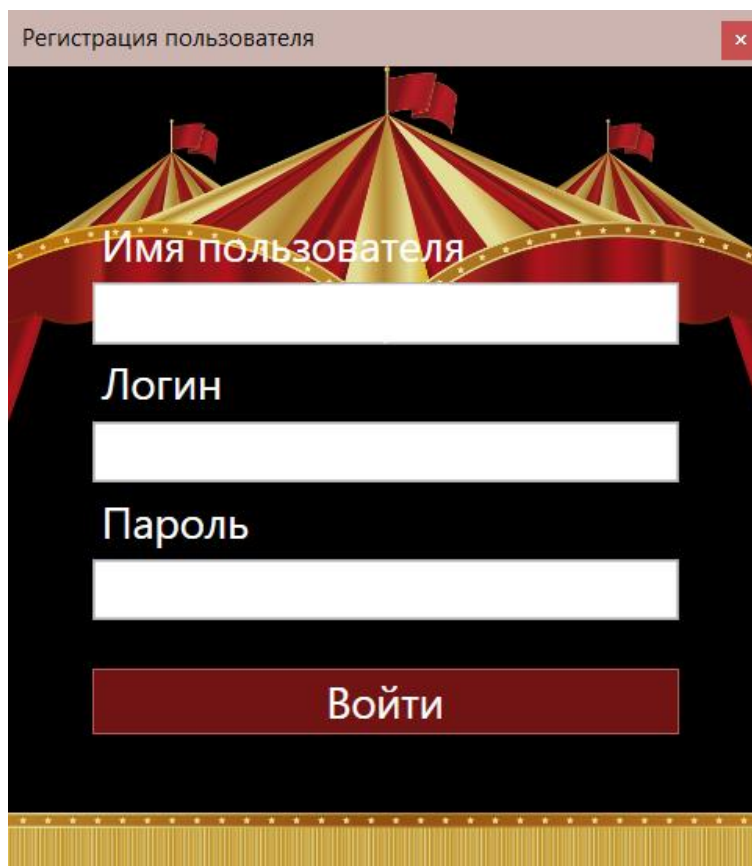
The image shows a software window titled "Регистрация пользователя" (User Registration). The window has a dark background with a decorative border featuring a red and yellow striped circus tent. Inside the window, there are three white input fields. The first field is labeled "Имя пользователя" (User Name), the second is labeled "Логин" (Login), and the third is labeled "Пароль" (Password). Below these fields is a red button with the text "Войти" (Login) in white. The window also has a standard Windows-style title bar with a close button (X) in the top right corner.

Рисунок 2.4 – Окно регистрации пользователя

После авторизации открывается окно с главным меню, где перечислены основные функции и возможности для данного пользователя. В зависимости от типа аккаунта (его роли) будут отличаться элементы меню, некоторые пункты будут недоступны для гостя и зарегистрированного пользователя. В окне для администратора доступны функции:

- добавление представления;
- просмотр афиши;
- изменение афиши;
- изменение количества билетов в каждой категории;
- удаление пользователей;
- просмотр отчетов по каждому представлению.

Шаблон интерфейса администратора изображен на рисунке 2.5.

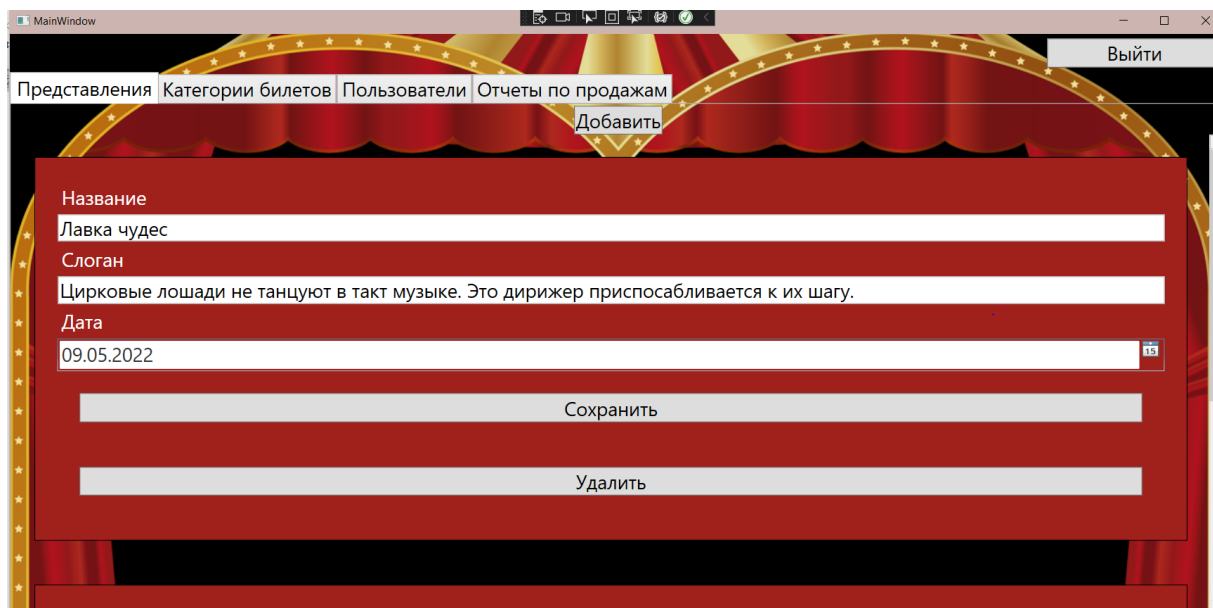


Рисунок 2.5 – Шаблон интерфейса главного меню для администратора

В окне пользователя доступны функции:

- просмотр афиши;
- покупка билетов;
- поиск билетов;
- просмотр купленных билетов.

Шаблон интерфейса пользователя изображен на рисунке 2.6.



Рисунок 2.6 – Шаблон интерфейса главного меню для пользователя

На странице «Представления» под надписью «Поиск» в выпадающем списке можно выбрать критерий поиска. В соответствии с выбранным критерием

ниже отобразится панель. Панели, предоставляющие функционал поиска по определённым критериям представлены на рисунке 2.7.

The image shows three separate panels, each with a red and gold circus tent graphic at the top. Each panel has a dropdown menu labeled 'Поиск по' (Search by) and a red 'Поиск' (Search) button. The first panel has a dropdown for 'Названию представл' (Presentation Name) and a text input field. The second panel has a dropdown for 'Дате представления' (Presentation Date) and a date input field showing '13.05.2022'. The third panel has a dropdown for 'Ценовая категория' (Price Category) and a dropdown menu showing 'Билеты категории №2' (Category #2 tickets).

Рисунок 2.7 – Панели для поиска по определённому критерию

На странице «Отчёты по продажам» отображаются отчеты, сформированные для администратора. Каждый отчёт содержит название представления, дату премьеры, количество купленных билетов в каждой категории и заработанную сумму. Страница «Отчёты по продажам» представлена на рисунке 2.8.

The screenshot shows a web application interface with a red and gold circus tent background. At the top, there is a navigation bar with tabs: 'Представления' (Presentations), 'Категории билетов' (Ticket Categories), 'Пользователи' (Users), and 'Отчёты по продажам' (Sales Reports). A 'Выйти' (Logout) button is in the top right corner. The 'Отчёты по продажам' tab is active, displaying a large red box with white text containing the following data:

Название представления	Лавка чудес
Дата представления	09.05.2022
Купленов билетов категории №1	2/10
Купленов билетов категории №2	0/40
Купленов билетов категории №3	3/50
Заработано	130 б.р.

Рисунок 2.8 – Страница «Отчёты по продажам»

Окно гостя отличается функциональностью и доступными возможностями. У гостя по сравнению с администратором возможности очень ограничены, он может только просматривать афишу. Шаблон интерфейса гостя изображён на рисунке 2.9.

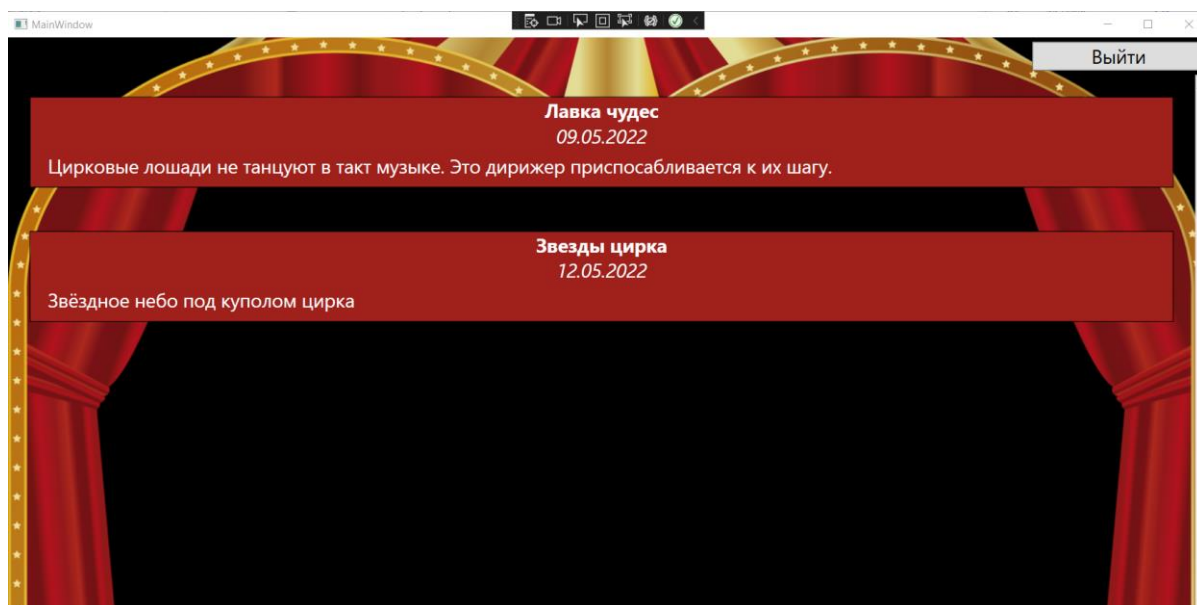


Рисунок 2.9 – Шаблон интерфейса главного меню для гостя

При вводе неверных данных в форме авторизации возникает сообщение об ошибке показанное на рисунке 2.10.

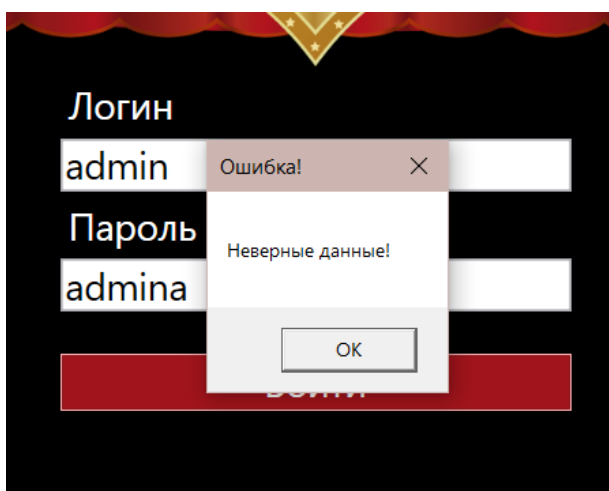


Рисунок 2.10 – Сообщение об ошибке

Помимо авторизации, в приложении могут возникнуть ошибки при попытке создания представления раньше текущего числа. Это поле подсвечивается красным цветом.

## 3 ВЕРИФИКАЦИЯ И ТЕСТИРОВАНИЯ СИСТЕМЫ РЕАЛИЗАЦИИ БИЛЕТОВ В ЦИРКЕ ГОРОДА

### 3.1 Модульное тестирование приложения

Модульное тестирование – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирования выполняется разработчиками на этапе кодирования приложения.

Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект.

После реализации основного функционала приложения, проведено модульное тестирование.

Для проведения тестирования основных методов классов были разработаны модульные тесты *CircusTests* (обязательное приложение А, код программы для *CircusTests*).

Результаты прохождения тестов изображены на рисунке 3.1.

Рисунок 3.1 – Результаты прохождения тестов

Для тестирования были разработаны следующие методы:

- *CallCrudMethodsWithNullArguments\_ThrowsNullArgumentException()* – тестирование вызова методов *Crud* с *Null* аргументами;
- *Create\_CheckIfCreatedIsEqualToCreating()* – проверка создания сущности;
- *Delete\_CheckIfDeletedIsEqualToDeleting()* – проверка удаления сущности;

- *Read\_CheckIfReadingDataContainsInsertedData()* – тестирование получения списка сущностей;
- *Update\_CheckIfUpdatedIsEqualToUpdating()* – проверка обновления сущности.

Проект модульного тестирования *CircusTests* реализован для возможности протестировать работоспособность созданного приложения. Предоставляется возможность протестировать каждый метод в приложении, а также возможность отработать все возможные исключительные ситуации, происходящие в коде при работе с ним.

Для работы с программой необходимо открыть файл «*CircusCourseWork.exe*». После запуска приложения, пользователь может авторизовываться в системе или зарегистрироваться.

Пример окна авторизации изображён на рисунке 3.2.

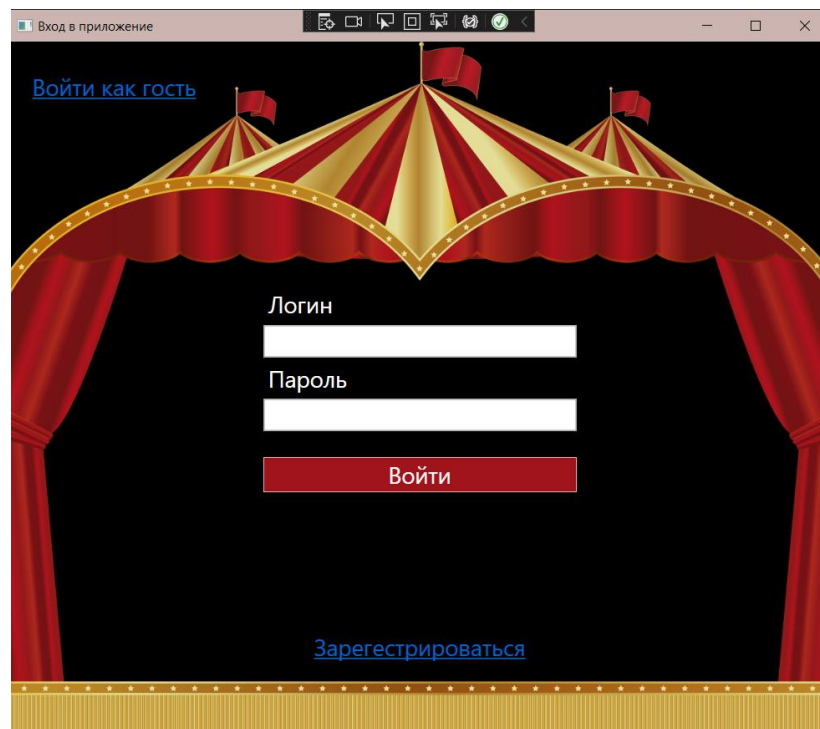


Рисунок 3.2 – Окно авторизации

Если авторизация не прошла, на экране будет выведен интерфейс гостя, в котором доступен только просмотр афиши.

Интерфейс гостя изображен на рисунке 3.3.

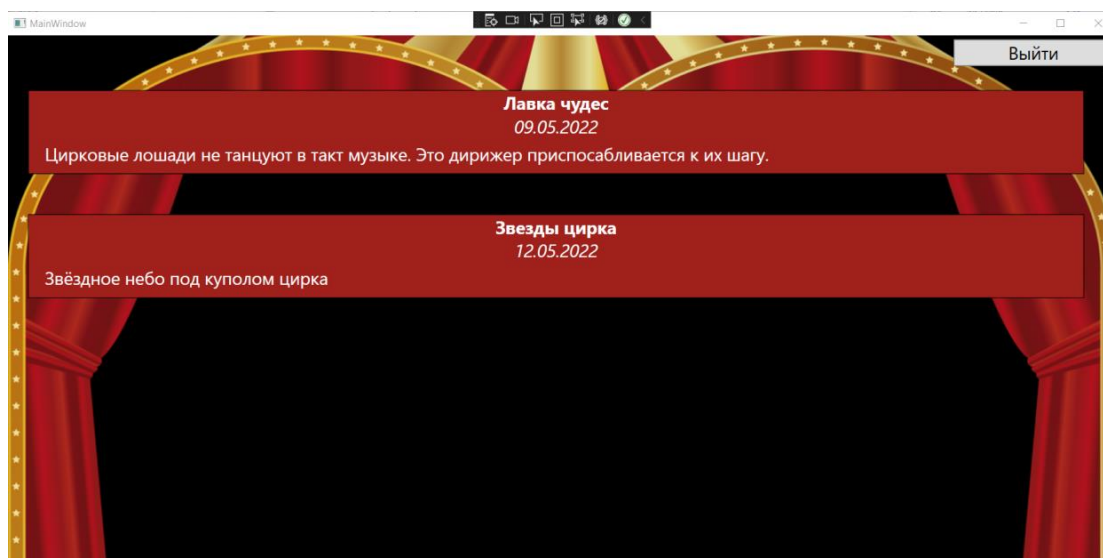


Рисунок 3.3 – Интерфейс гостя

При авторизации в системе администратора список его возможностей будет шире, главное меню для администратора изображено на рисунке 3.4. В главном меню для администратора есть пункты:

- добавление представления;
- просмотр афиши;
- изменение афиши;
- изменение количества билетов в каждой категории;
- удаление пользователей;
- просмотр отчетов по каждому представлению.



Рисунок 3.4 – Главное меню для администратора



Вкладка для добавления новых представлений доступна только администратору. Для того чтобы добавить записи требуется заполнить все поля формы корректно, рисунок 3.5, и в случае успешного добавления данных будет выведено соответствующее сообщение.



The screenshot shows a form with a red background. It contains the following fields and buttons:

- Название (Name): Шоу Славы Полунина
- Слоган (Slogan): Главный принцип Snow Show— никаких правил.
- Дата (Date): 11.05.2022
- Сохранить (Save) button
- Удалить (Delete) button

Рисунок 3.5 – Форма для добавления представления

Так же администратор может просматривать данные о представлениях и формировать отчёты по самым популярным и кассовым фильмам и по количеству зрителей в кинотеатрах. Так же при формировании отчётов имеются графические элементы, позволяющие пользователю проводить фильтрацию и сортировку данных по определённым критериям.

Примеры просмотра данных и формирования отчётов представлены на рисунке 3.6.



The screenshot shows a window with a red background and a gold border. It contains the following information:

- Название представления (Show Name): Лавка чудес
- Дата представления (Show Date): 09.05.2022
- Купленов билетов категории №1 (Category 1 tickets sold): 2/10
- Купленов билетов категории №2 (Category 2 tickets sold): 0/40
- Купленов билетов категории №3 (Category 3 tickets sold): 3/50
- Заработано (Revenue): 130 б.р.

Рисунок 3.6 – Окно со сформированными отчётами

При авторизации в системе как пользователя, появляются возможности



просмотра афиши и покупки билетов, рисунок 3.7.

The image shows a user interface for purchasing tickets. It has a red and gold striped background. The interface includes three categories of tickets, each with a label, a quantity input field, and a spinner control. The price is displayed as 130. A 'Купить' (Buy) button is at the bottom.

Категория	Количество
Количество билетов категории №1	2
Количество билетов категории №2	0
Количество билетов категории №3	3

Цена: 130

Кнопка: Купить

Рисунок 3.7 – Интерфейс пользователя

Подробная инструкция для пользователей, программиста и системного программиста описана в приложении В – Г. По итогу проведения интегрированного тестирования можно удостовериться в работоспособности приложения и его основных функций.

## ЗАКЛЮЧЕНИЕ

Был разработан программный продукт (*WPF* приложение), позволяющее распространять билеты в цирке города. Так же были выполнены все задачи, поставленные при разработке.

Процесс разработки программного средства закрепил, а также улучшил знания теоретического материала, в результате чего произошло структурирование имеющихся знаний и это поможет в дальнейшей разработке программных средств.

Потребность в реализации темы данного приложения со временем будет только увеличиваться, так как с помощью данной системы возможна покупка и бронирование билетов быстрым и понятным путем.

Недостатками данного приложения является отсутствие изображения афиш и бронирования билетов.

Достоинством приложения является многопользовательский доступ к системе с возможностью разграничения прав доступа, что значительно повышает надежность использования приложения. Пользователь может просматривать список фильмов в прокате, работать над заполнением справочных данных, оформлять продажи билетов, а также формировать отчеты.

Для программы также имеются руководство пользователя, программиста и системного программиста, которые помогут с легкостью разобраться в использовании программного средства.

Для хранения используемых приложением данных были созданы соответствующие *XML*-файлы. Разработанная структура данных спроектирована соответственно поставленной задаче.

Программа отлично подойдет в качестве основной системы для работы учета продаж билетов цирка города, за счет использования инновационных технологий она может спокойно развивать свой дальнейший функционал, а использование ООП позволяет сделать ее масштабируемой и удобной в сопровождении. За счет этого программное средство может стать конкурентно способным на рынке.

Проведенная верификация и модульное тестирование показывают, что программное средство является работоспособным на всех участках программы.

## 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. Мухортов В.В., Рылов В.Ю. Объектно-ориентированное программирование, анализ и дизайн: Методическое пособие. – Новосибирск, издательство «ИМ СО РАН», 2002. – 108 с.
3. Шилдт Герберт. С# 4.0: полное руководство: учебное пособие – ООО «И.Д. Вильямс», 2011. – 1056 с.
4. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# – ООО Издательство «Питер», 2013. – 896 с.
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж., Приемы объектно-ориентированного программирования. Паттерны проектирования. – Спб.: Питер, 2015. – 368 с.
6. Декоратор, Электронные данные. – Режим доступа: <https://refactoring.guru/ru/design-patterns/decorator>. Дата доступа: 13.12.2019.
7. Албахари, Д. С# 7.0. Карманный справочник. – СПб.: ООО «Альфа-книга», 2017. – 224 с.

## **ПРИЛОЖЕНИЕ А**

**(обязательное)**

### **Иерархия классов**

**ПРИЛОЖЕНИЕ Б**  
(обязательное)

**Код программы**

## ПРИЛОЖЕНИЕ В

(обязательное)

### Руководство системного программиста

#### 1. Назначение и условия применения

Разработанное программное приложение предназначено для организации распространения билетов в цирке. Оно предполагает три вида доступа в систему:

- гость;
- зарегистрированный пользователь;
- администратор.

Управление в приложении осуществляется с помощью клавиатуры и мыши. Для корректной работы программы необходимо соблюдение следующих средств:

- поддержка операционной системы *Windows XP* и выше;
- наличие стандартной клавиатуры и мыши;
- наличие экрана, как устройства вывода, подключенного по *HDMI*.

#### 2. Структура программы Структура приложения состоит из пяти форм *WFA*:

- *MainWindow* – главное окно;
- *RegisterWindow* – окно регистрации;
- *SignInWindow* – окно для входа в приложение;
- *SortTicketsWindow* – окно для сортировки билетов;
- *BuyTicketsWindow* – окно для покупки билетов.

А также из девяти пользовательских элементов управления:

- *AdminControl* – графический интерфейс для администратора;
- *GuestControl* – графический интерфейс для гостя;
- *RegisteredUserControl* – графический интерфейс для зарегистрированных пользователей;
- *ReportControl* – для отображения отчетов;
- *PerfomanceRegisteredUserControl* – форма для отображения циркового представления (для зарегистрированного пользователя);
- *PerfomanceGuestControl* – форма для отображения циркового представления (для гостя);
- *EditablePerformanceControl* – форма для изменения представления;
- *EditableUserControl* – форма для изменения пользователей;
- *TicketRegisteredUserControl* – форма для изменения билетов в заказе.

#### 3. Настройки программы

При запуске приложения никаких настроек производить не требуется.

#### 4. Проверка программы

Программа была проверена при помощи модульного тестирования. При прохождении тестов ошибок не обнаружено.

#### 5. Сообщение системному программисту

При появлении исключений не требуется перезапускать программу.

## ПРИЛОЖЕНИЕ Г

(обязательное)

### Руководство программиста

#### 1. Назначение и условия применения

Разработанное программное приложение предназначено для организации распространения билетов в цирке. Оно предполагает три вида доступа в систему:

- гость;
- зарегистрированный пользователь;
- администратор.

Управление в приложении осуществляется с помощью клавиатуры и мыши. Для корректной работы программы необходимо соблюдение следующих средств:

- поддержка операционной системы *Windows XP* и выше;
- наличие стандартной клавиатуры и мыши;
- наличие экрана, как устройства вывода, подключенного по *HDMI*.

#### 2. Характеристики программы

Приложение работает в автоматическом режиме. При запуске открывается главное окно, в котором отображается графический интерфейс для определенного пользователя. Пользователи имеют возможность зарегистрироваться, зайти и войти в аккаунт.

#### 3. Обращение к программе

Приложение запускается путем открытия файла *CircusCourseWork.exe* находящегося в каталге *bin/net5.0-windows/Debug*.

#### 4. Входные и выходные данные

В данной программе в качестве исходных данных используется язык программирования *C#*, *XML*-файлы, *WPF* и шаблон проектирования «*Repository*».

#### 5. Сообщения

При возникновении исключений не требуется перезапустить приложение.

## ПРИЛОЖЕНИЕ Д

(обязательное)

### Руководство пользователя

#### 1. Введение

Руководство пользователя обеспечивает получение пользовательских базовых навыков по эксплуатации приложения. Разработанная программа предназначена для распространения билетов в театре. В приложении есть три вида доступа в систему:

- гость;
- зарегистрированный пользователь;
- администратор.

Программное средство обладает следующим функционалом:

- графический интерфейс;
- ввод данных при помощи клавиатуры;
- нажатие на кнопки при помощи мыши.

Для использования программного средства пользователь должен ознакомиться с:

- настоящим руководством пользователя;
- правилами использования ЭВМ.

#### 2. Назначение и условия применения

Разработанное программное приложение предназначено для покупки и продажи билетов в цирке. Для корректной работы программы необходимо соблюдение следующих средств:

- поддержка операционной системы *Windows XP* и выше;
- наличие стандартной клавиатуры и мыши;
- наличие экрана, как устройства вывода, подключенного по *HDMI*.

#### 3. Подготовка в работе

Для установки приложение необходимо загрузить на компьютер каталог *CircusCourseWork*. Если все инструкции соблюдены, и приложение не выдает никаких сообщений об ошибках, следовательно, программа работает исправно.

#### 4. Описание операций для пользователей

Для гостя доступны следующие операции:

- просмотр афиш спектаклей;
- регистрация;
- вход как зарегистрированный пользователь.

Для зарегистрированного пользователя доступны следующие операции:

- просмотр афиши;
- поиск спектакля по разной категории;
- просмотр свободных мест на выбранный спектакль;
- заказ и отмена покупки билетов;
- выход из аккаунта.

Для администратора доступны следующие операции:

- редактирование, обновление и удаление спектаклей;
- редактирование коэффициента множителя цены для определенной категории



билетов;

- блокирование пользователя;
- создание отчетов по спектаклю;
- выход из аккаунта.

#### 5. Аварийные ситуации

Чтобы избежать ошибок при использовании данного приложения, необходимо соблюдать порядок действий и условий использования, описанных в пунктах выше данного руководства. В случае непредвидимых «зависаниях» программы, рекомендуется завершить процесс в диспетчере задач и запустить снова.

#### 6. Рекомендации по освоению

Приложение имеет простую реализацию, а все операции понятны на интуитивном уровне.