

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: «***WINDOWS FORM*** ПРИЛОЖЕНИЕ ДЛЯ СИСТЕМЫ РАСПРОСТРАНЕНИЯ БИЛЕТОВ В ТЕАТРЕ»

Выполнил: студент гр. ИТП-21

Бондарев Е.Ю.

Руководитель: доцент

Курочка К.С.

Дата проверки: _____

Дата допуска к защите: _____

Дата защиты: _____

Оценка работы: _____

Подписи членов комиссии

по защите курсового проекта: _____

Гомель 2023

СОДЕРЖАНИЕ

Введение	5
1 Технические подходы к реализации приложения для системы продажи билетов	6
1.1 Сравнительный анализ <i>C#</i> и <i>C++</i>	6
1.2 Различия <i>Windows Forms</i> и <i>WPF</i>	10
1.3 Технология языка <i>XML</i>	13
2 Алгоритмический анализ поставленной задачи и описание программного комплекса	16
2.1 Архитектура программного комплекса	16
2.2 Компоненты приложения для продажи билетов	16
2.3 Структура базы данных	19
2.4 Структура классов разработанного программного комплекса	20
3 Пользовательский интерфейс и тестирование	27
3.1 Описание пользовательского интерфейса приложения	27
3.2 Результаты тестирования пользовательского интерфейса	29
3.3 Результаты модульного тестирования	30
ошибка! Закладка не определена.	
Заключение	33
Список используемых источников	34
ПРИЛОЖЕНИЕ А Графические представления интерфейса программы	35
ПРИЛОЖЕНИЕ Б Листинг программы	43
ПРИЛОЖЕНИЕ В Руководство пользователя	84
ПРИЛОЖЕНИЕ Г Руководство программиста	85
ПРИЛОЖЕНИЕ Д Руководство системного программиста	86
ПРИЛОЖЕНИЕ Е Иерархическая схема классов приложения	87

ВВЕДЕНИЕ

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

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

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

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

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

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

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

1 ТЕХНИЧЕСКИЕ ПОДХОДЫ К РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ ДЛЯ СИСТЕМЫ ПРОДАЖИ БИЛЕТОВ

1.1 Сравнительный анализ C# и C++

C# является современным объектно-ориентированным языком программирования, разработанным компанией *Microsoft*, и является одним из основных языков для разработки приложений под платформу *.NET Framework*. Его синтаксис очень похож на язык программирования *Java*, что делает его более доступным для тех, кто уже знаком с *Java*. Он также имеет синтаксис, схожий с языками C и C++, что облегчает понимание для программистов, имеющих опыт работы с этими языками.

C# предлагает полную поддержку объектно-ориентированного подхода, что позволяет разрабатывать сложные приложения с использованием концепций, таких как наследование, полиморфизм, инкапсуляция и абстракция. Он основывается на тех же принципах объектно-ориентированного программирования (ООП), что и другие языки, включая инкапсуляцию, наследование и полиморфизм.

В C# классы представляют собой шаблоны для создания объектов, определяющие их свойства и методы. Язык поддерживает наследование, позволяющее создавать новые классы на основе существующих, наследуя их свойства и методы и добавляя свои собственные. Также C# поддерживает полиморфизм, который позволяет использовать один и тот же код для разных типов объектов. Для этого можно использовать абстрактные классы и интерфейсы. Например, в C# можно определить интерфейсы, которые указывают набор методов и свойств, которые должны быть реализованы классом, который реализует интерфейс.

C# является языком программирования с управляемым кодом, что означает, что он работает в среде *CLR (Common Language Runtime)*, которая обеспечивает автоматическое управление памятью, проверку безопасности и другие преимущества, связанные с управляемым кодом.

Код на C# будет скомпилирован в управляемый код при использовании среды выполнения *.NET Framework* или *.NET Core*. CLR будет управлять выполнением кода и автоматически освобождать память, когда объекты больше не нужны. Кроме того, CLR будет контролировать безопасность кода, проверяя его на наличие потенциально опасных операций, таких как доступ к несуществующей памяти или запуск небезопасного кода. Это обеспечивает более высокую безопасность и защиту от вредоносных программ.

Код на C# облегчает разработку приложений, так как разработчику не нужно самостоятельно управлять памятью или заботиться о безопасности кода. Вместо

этого *CLR* предоставляет средства для выполнения этих задач, что позволяет разработчику сосредоточиться на бизнес-логике приложения.

C# является платформо-независимым языком программирования, поэтому программы, написанные на *C#*, могут работать на различных платформах, таких как *Windows*, *Linux* и *MacOS*.

Кроссплатформенность *C#* обеспечивается благодаря использованию платформо-независимой виртуальной машины (*VM*), называемой *Common Language Runtime (CLR)*. *CLR* выполняет компиляцию *C#*-кода в промежуточный байт-код, называемый *Common Intermediate Language (CIL)*, который затем выполняется на виртуальной машине *CLR*. Это позволяет *C#*-коду работать на любой платформе, на которой есть *CLR*.

Вместе с тем, кроссплатформенность языка *C#* дополнительно обеспечивается использованием таких инструментов, как *.NET Core* и *Xamarin*. *.NET Core* является кроссплатформенной реализацией *.NET Framework* и поддерживает *Windows*, *macOS* и *Linux*. *Xamarin* представляет собой среду разработки для мобильных приложений на базе *C#*, которая позволяет разрабатывать приложения для *iOS*, *Android* и *Windows Phone*.

Кроссплатформенность *C#* позволяет разработчикам писать код один раз и запускать его на различных платформах, что упрощает процесс разработки и обеспечивает большую гибкость при создании приложений. Кроме того, *C#* является одним из наиболее популярных языков программирования в мире и имеет широкую поддержку сообщества разработчиков и библиотек, что делает его еще более удобным и привлекательным для кроссплатформенной разработки.

Также *C#* обеспечивает поддержку многопоточности, что позволяет разрабатывать приложения, которые могут выполнять несколько задач одновременно. *C#* тесно интегрирован с другими технологиями, созданными *Microsoft*, такими как *.NET Framework*, *Windows Forms*, *ASP.NET* и т.д., что позволяет создавать мощные и сложные приложения.

C# имеет богатую библиотеку классов, которая обеспечивает широкий диапазон функциональных возможностей, таких как работа с базами данных, ввод-вывод, работа с сетью и многое другое.

Библиотека классов *.NET (Class Library)* представляет собой набор готовых классов, интерфейсов и методов, которые разработчик может использовать в своих приложениях на языке *C#*. Она содержит множество классов, предназначенных для выполнения различных задач, таких как работа с файлами и директориями, работа с сетью, работа с базами данных, работа с графикой и многие другие.

Библиотека классов *.NET* включает в себя базовые классы, такие как классы *Object*, *String*, *Array*, а также классы, специфичные для различных областей программирования, такие как классы для работы с *XML*, классы для работы с *Windows Forms*, классы для работы с *ASP.NET* и т.д.

Разработчики могут использовать библиотеку классов *.NET* для упрощения и ускорения процесса разработки приложений. Вместо того, чтобы писать код с нуля, они могут использовать готовые классы из библиотеки, что позволяет сократить время разработки и уменьшить количество ошибок. Кроме того, библиотека классов *.NET* является кроссплатформенной и может использоваться для разработки приложений на *Windows*, *macOS* и *Linux*, используя *.NET Core*.

Примеры классов из библиотеки классов *.NET*:

- *System.IO.File* – класс для работы с файлами;
- *System.Net.Http.HttpClient* – класс для работы с *HTTP*-запросами;
- *System.Data.SqlClient.SqlConnection* – класс для работы с базой данных *Microsoft SQL Server*;
- *System.Drawing.Bitmap* – класс для работы с изображениями;
- *System.Xml.XmlDocument* – класс для работы с *XML*-документами.

Каждый класс из библиотеки классов *.NET* имеет свои свойства, методы и события, которые можно использовать в приложении. Например, класс *System.IO.File* имеет методы для чтения и записи файлов, а класс *System.Drawing.Bitmap* имеет методы для обработки изображений. Кроме того, библиотека классов *.NET* содержит документацию для каждого класса, которая описывает его свойства и методы, а также примеры их использования.

C++ – это мощный язык программирования, который широко используется для разработки сложных системного и прикладного программного обеспечения. *C++* является компилируемым языком программирования, что означает, что код на *C++* компилируется в машинный код, который выполняется быстрее, чем интерпретируемые языки программирования, такие как *Python* или *JavaScript*. *C++* предлагает полную поддержку объектно-ориентированного подхода, что позволяет разрабатывать сложные приложения с использованием концепций, таких как наследование, полиморфизм, инкапсуляция и абстракция. *C++* предназначен для системного программирования, что означает, что он может быть использован для написания ОС, драйверов, библиотек и других низкоуровневых компонентов системы.

Язык может работать на разных платформах, таких как *Windows*, *Linux*, *macOS* и другие, что позволяет разработчикам писать кроссплатформенный код. *C++* предоставляет программистам низкоуровневый доступ к памяти, что позволяет им управлять памятью напрямую, что особенно важно для системного программирования и оптимизации производительности.

Язык программирования имеет богатую стандартную библиотеку, которая обеспечивает широкий диапазон функциональных возможностей, таких как работа с файлами, потоками, сетевыми протоколами, шаблоны и многое другое.

Стандартная библиотека C++ – это набор классов и функций, которые предоставляются компилятором и являются частью стандарта языка. Она содержит множество полезных компонентов, таких как контейнеры, алгоритмы, потоки ввода/вывода, строки, время и др.

Контейнеры – это объекты, которые содержат коллекцию элементов. В стандартной библиотеке C++ предоставляются различные контейнеры, такие как вектор, список, множество, хэш-таблицы и др. Контейнеры могут быть использованы для хранения и обработки данных, а также для реализации структур данных, таких как стеки, очереди и т.д.

Алгоритмы – это функции, которые применяются к контейнерам для обработки их содержимого. В стандартной библиотеке C++ предоставляются различные алгоритмы, такие как сортировка, поиск, преобразование, удаление элементов и др. Алгоритмы могут быть использованы для упрощения обработки данных в контейнерах.

Потоки ввода/вывода – это объекты, которые предоставляют функциональность ввода и вывода данных в программе. В стандартной библиотеке C++ предоставляются различные потоки, такие как потоки ввода/вывода файлов, строк, стандартного ввода/вывода и др. Потоки могут быть использованы для чтения и записи данных из различных источников.

Строки – это последовательности символов, которые могут быть использованы для хранения текстовых данных. В стандартной библиотеке C++ предоставляются различные функции для работы со строками, такие как конкатенация, поиск, сравнение и др. Строки могут быть использованы для обработки текстовых данных в программе.

Стандартная библиотека C++ также содержит функции для работы с временем, такие как получение текущего времени, преобразование времени в другой формат, расчет временных промежутков и др. Функции работы со временем могут быть использованы для управления временем в программе.

C++ предоставляет программистам возможности, такие как управление памятью, многопоточность, *inline*-функции, возможность работы с ассемблером и другие, которые не доступны в других языках программирования.

Проанализировав сведения о языках программирования C# и C++ можно сделать вывод о том, что для решения задачи больше подходит язык C# по следующим причинам:

– простота – C# предлагает более высокий уровень абстракции и упрощенный синтаксис, что делает его более доступным для начинающих разработчиков.

В отличие от C++, который предлагает низкоуровневый доступ к памяти и более сложный синтаксис;

- быстрота разработки – C# имеет множество инструментов и библиотек, которые ускоряют разработку и сокращают время написания кода, в то время как C++ требует более многословного кода и большего количества времени на разработку;

- кроссплатформенность – C# может быть использован для разработки кроссплатформенных приложений, что означает, что один и тот же код может работать на разных платформах (например, *Windows*, *Linux* и *macOS*), в то время как C++ требует отдельной компиляции для каждой платформы;

- безопасность – C# имеет встроенную систему управления памятью, что делает его более безопасным и менее склонным к ошибкам, связанным с утечками памяти, чем C++, который предоставляет низкоуровневый доступ к памяти;

- поддержка новых технологий – C# тесно интегрируется с новыми технологиями, такими как *.NET Core* и *Xamarin*, что делает его более актуальным для разработки современных приложений, таких как веб-приложения, мобильные приложения и другие.

1.2 Различия *Windows Forms* и *WPF*

Windows Presentation Foundation (WPF) – это фреймворк пользовательского интерфейса для создания приложений настольного компьютера, представленный компанией *Microsoft* в 2006 году. *WPF* был создан для замены *WinForms*, которая была основным фреймворком пользовательского интерфейса для приложений настольного компьютера *Windows* с 2002 года. В то время как *WinForms* все еще используется сегодня, *WPF* предоставляет более современный, визуально привлекательный фреймворк пользовательского интерфейса, построенный поверх *DirectX*.

Windows Forms, с другой стороны, является фреймворком пользовательского интерфейса, использующим *Windows API* для создания приложений настольного компьютера в *Windows*. Он является традиционным фреймворком пользовательского интерфейса для приложений настольного компьютера *Windows* и до сих пор широко используется.

Windows Forms представляет собой набор готовых элементов управления и классов, которые упрощают создание и манипулирование окнами, кнопками, текстовыми полями, таблицами и другими элементами пользовательского интерфейса. Эта технология предоставляет богатый набор инструментов для работы с многопоточностью, работой с сетью и веб-сервисами, а также с базами данных.

Windows Forms предоставляет простой и интуитивно понятный объектно-ориентированный подход к созданию пользовательского интерфейса. Он также

обеспечивает удобный инструментарий для разработки, такой как конструктор форм и интегрированную справку.

Windows Forms содержит более 40 элементов управления:

- кнопка (*Button*) – элемент управления, который позволяет пользователю запускать какие-либо действия, щелкая по нему;
- поле ввода текста (*TextBox*) – элемент управления, предназначенный для ввода однострочного или многострочного текста;
- метка (*Label*) – элемент управления, который используется для отображения статической текстовой информации;
- список (*ListBox*) – элемент управления, который позволяет пользователю выбирать один или несколько элементов из списка;
- выпадающий список (*ComboBox*) – элемент управления, который позволяет пользователю выбирать один из нескольких predetermined элементов;
- флажок (*CheckBox*) – элемент управления, который позволяет пользователю выбирать один или несколько параметров из группы параметров;
- переключатель (*RadioButton*) – элемент управления, который позволяет пользователю выбрать один параметр из группы параметров;
- полоса прокрутки (*ScrollBar*) – элемент управления, который используется для прокрутки содержимого формы или других элементов управления;
- таблица (*DataGridView*) – элемент управления, который используется для отображения и редактирования табличных данных;
- меню (*MenuStrip*) – элемент управления, который позволяет пользователю выбирать команды из списка;
- панель инструментов (*ToolStrip*) – элемент управления, который предоставляет доступ к наиболее часто используемым командам и функциям;
- форма (*Form*) – элемент управления, который является основным контейнером для других элементов управления и определяет внешний вид окна приложения.

Данные элементы могут быть легко настроены и ими легко управлять. Кроме того, пользовательские элементы управления могут быть созданы и добавлены в приложение.

Windows Forms приложения могут быть созданы и запущены на разных операционных системах, таких как *Windows*, *Linux* и *macOS*. Это достигается через использование *.NET Framework*, который является кроссплатформенной платформой разработки приложений.

Windows Forms позволяет настроить внешний вид элементов управления с помощью встроенных свойств и стилей, а также использовать собственные ресурсы для создания пользовательского интерфейса.

Windows Forms предоставляет удобный механизм для работы с событиями и обработчиками, что позволяет легко реагировать на действия пользователя и изменения состояния приложения.

Windows Forms обеспечивает удобную работу с многопоточностью и позволяет легко создавать и управлять несколькими потоками внутри приложения.

WPF использует язык разметки *XAML* (*Extensible Application Markup Language*), который позволяет описывать пользовательский интерфейс в виде структурированного документа.

XAML использует декларативный подход к созданию пользовательского интерфейса. Вместо того чтобы программировать пользовательский интерфейс с помощью кода, вы можете определить его структуру и внешний вид в виде документа *XAML*.

XAML позволяет разделить представление пользовательского интерфейса и логику приложения на отдельные компоненты. Это упрощает создание и поддержку приложения.

XAML позволяет использовать векторную графику для создания элементов интерфейса. Векторная графика имеет множество преимуществ, включая возможность масштабирования без потери качества и улучшенную производительность.

XAML позволяет использовать стили и шаблоны для создания переиспользуемых элементов интерфейса. Это упрощает создание и поддержку приложения.

XAML интегрируется с кодом, написанным на *C#* или других языках *.NET*, что позволяет создавать более сложные и интерактивные пользовательские интерфейсы.

XAML может использоваться вместе с интегрированной средой разработки *Visual Studio*, что упрощает создание и отладку приложений.

XAML является расширяемым языком, который позволяет создавать собственные элементы управления и добавлять их в приложение [7, с. 43].

Язык *XAML* является мощным инструментом для создания пользовательского интерфейса в технологии *WPF*. Он предоставляет широкие возможности для создания красивого, гибкого и переиспользуемого пользовательского интерфейса в приложениях *Windows*.

WPF предоставляет широкие возможности для создания интерактивных элементов управления, включая анимацию, переходы, эффекты и т.д.

WPF предоставляет множество инструментов для создания анимации, включая возможность создания анимации с помощью *XAML* или кода:

- анимация свойств элемента (например, изменение его размера, положения, цвета и т.д.);

- переходы между состояниями элемента;
- анимация с помощью пути (например, перемещение элемента вдоль кривой);
- анимация с помощью кадров (т.е. анимация, созданная из набора изображений).

WPF также поддерживает множество переходов, которые можно использовать для создания эффектов перехода между различными состояниями элементов. Некоторые из доступных переходов включают:

- плавное появление/исчезновение;
- переходы между двумя различными изображениями;
- переходы между состояниями элементов, такими как нажатие кнопки или наведение курсора мыши.

WPF также предоставляет множество эффектов, которые можно использовать для изменения внешнего вида элементов. Вот некоторые из возможных эффектов:

- размытие;
- световое оформление;
- насыщенность;
- оттенки серого;
- декоративные края;
- искажение.

Как можно понять, *WPF* предоставляет множество возможностей для создания разнообразных эффектов, переходов и анимаций, что позволяет создавать интерактивные и динамичные пользовательские интерфейсы.

WPF использует модель *MVVM* (*Model-View-ViewModel*), которая позволяет разделить логику приложения и пользовательский интерфейс на отдельные компоненты.

Одно из ключевых отличий между *WPF* и *WinForms* заключается в способе работы с графикой. *WPF* использует систему векторной графики, что позволяет создавать графику высокого качества, масштабируемую и легко анимируемую и управляемую.

1.3 Технология языка *XML*

XML (*Extensible Markup Language*) – это один из самых распространенных языков разметки документов в мире. Его популярность объясняется тем, что *XML* является стандартным форматом для обмена информацией между различными системами и приложениями. Благодаря своей гибкости и расширяемости, *XML* используется во многих областях, включая веб-приложения, базы данных, настройки программного обеспечения, научные исследования и многое другое.

В отличие от *HTML*, который используется для создания веб-страниц, *XML* не определяет, как информация должна выглядеть на экране. Вместо этого, *XML* определяет структуру и семантику данных, что делает его более универсальным и удобным для использования в различных областях.

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

Одной из особенностей *XML* является его расширяемость. Разработчики могут определять свои собственные элементы, атрибуты и правила, которые будут использоваться в файле *XML*. Это делает *XML* очень гибким и удобным для использования в различных областях.

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

Кроме того, *XML* является кроссплатформенным и независимым от языка программирования форматом, что позволяет использовать его для обмена данными между различными операционными системами и языками программирования.

Одним из примеров применения *XML* является *SOAP* (*Simple Object Access Protocol*), протокол, используемый для обмена данными между веб-службами (*web services*). *SOAP* использует *XML* для передачи данных, что обеспечивает стандартизацию и возможность взаимодействия между различными веб-службами.

Также *XML* широко применяется для хранения и передачи данных в базах данных. Например, язык запросов *XQuery* позволяет извлекать данные из *XML*-документов, что делает его удобным для работы с данными, организованными в иерархическую структуру.

XML имеет несколько достоинств перед другими базами данных:

- универсальность и платформенная независимость: *XML* является открытым стандартом, который поддерживается практически всеми платформами и языками программирования. Это позволяет передавать и обмениваться данными между различными системами, независимо от используемых технологий;

- гибкость и расширяемость: *XML* позволяет создавать пользовательские схемы и определять собственные теги, что делает его гибким и адаптируемым к различным потребностям. Вы можете определить собственную структуру данных и создавать свои собственные правила для валидации и обработки этих данных;

- человекочитаемость: *XML* представляет данные в виде текстовых файлов, используя теги и атрибуты для организации информации. Это делает *XML* легко читаемым для людей и удобным для отладки и анализа данных;

- поддержка иерархической структуры: *XML* поддерживает иерархическую структуру данных, что позволяет организовывать информацию в виде дерева, состоящего из элементов и подэлементов. Это особенно полезно для представления сложных и связанных данных.

- интеграция с другими технологиями: *XML* может быть легко интегрирован с другими технологиями и форматами данных, такими как *XSLT* (*eXtensible Stylesheet Language Transformations*) для преобразования *XML* в другие форматы, *SOAP* (*Simple Object Access Protocol*) для веб-сервисов и другие.

- расширенная поддержка инструментов: *XML* имеет широкую поддержку инструментов для разработки, валидации, трансформации и анализа данных. Существуют различные библиотеки, среды разработки и утилиты, которые облегчают работу с *XML*.

- совместимость с существующими системами: Множество существующих систем и приложений используют *XML* для обмена данными. Использование *XML* позволяет легко интегрировать ваши данные с такими системами и использовать существующие стандарты и протоколы.

2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ И ОПИСАНИЕ ПРОГРАММНОГО КОМПЛЕКСА

2.1 Архитектура программного комплекса

Данное приложение написано на языке программирования *C#*, для создания визуального интерфейса были выбраны средства *C# Windows Forms*. Основными задачами при разработке приложения являются: разработка общей схемы приложения, создание классов и построение иерархии наследования для них. Классы построены таким образом, что при необходимости без особых проблем можно будет добавить новый тип билета, пользователя, спектакля.

В приложении существуют три типа пользователей: гость, авторизованный пользователь, администратор. Гость может просматривать афишу театра; авторизованный пользователь покупать или же отменять билеты на спектакль; администратор добавлять, редактировать, удалять спектакли (в один день должен идти один спектакль), также он управляет жанрами, авторами, может изменять пользователей, формировать различные статистические отчеты о проданных билетах, менять цены билетов, при этом количество билетов остается неизменным. При этом все данные хранятся в *XML*-файлах.

2.2 Компоненты приложения для продажи билетов

Приложение разделено на составные части, что значительно ускоряет и облегчает разработку программы. Так как логика приложения, интерфейс, взаимодействие с данными могут быть разделены и разрабатываться независимо друг от друга. Иными словами, приложение состоит из следующих блоков: пользовательский интерфейс, посредством которого пользователь будет взаимодействовать с программой; классы, отвечающие за работу приложения; классы, работающие с данными, выбираемыми из некоторого источника.

Если изменится способ хранения данных, например, данные будут храниться не в *XML*-файлах, а в базе данных, то нужно будет изменить лишь классы, отвечающие за работу с данными, при этом никак не затрагивая остальные компоненты приложения. Классы, которые будут работать с уже полученными данными, не знают, откуда они приходят, им главное их получить, для этого вызвав тот или иной метод класса, отвечающего за работу с данными. Это позволяет абстрагироваться от источника данных и таким образом разработанный алгоритм может применяться в различных системах без каких-либо серьезных изменений.

Интерфейс отделен от самой логики работы программы, так что в случае изменения интерфейса логика работы программы никак не изменится, но при

этом изменение самой логики программы, например, добавление какого-то нового функционала, такого как новый тип билета, интерфейс поменять так или иначе придется, если измениться способ получения данных или же в результате рефакторинга код классов будет улучшен, это опять-таки никак не повлияет на интерфейс.

При запуске приложения пользователь посредством элементов управления в графическом интерфейсе взаимодействует с программой. Графические компоненты интерфейса, например, кнопки, содержат в себе код, который работает с графическим интерфейсом, а также взаимодействует с классами, реализующими логику работы программы. Пользователь при нажатии на кнопку вызывает событие, которое содержит в себе код, обращающийся к классам, работающими с логикой программы. Эти классы в свою очередь обращаются к классам, работающими с источником данных. Таким образом можно сформировать логику работы приложения, абстрагировавшись от конкретной реализации данных классов (обобщенная функциональная схема приложения представлена на рисунке 2.1).

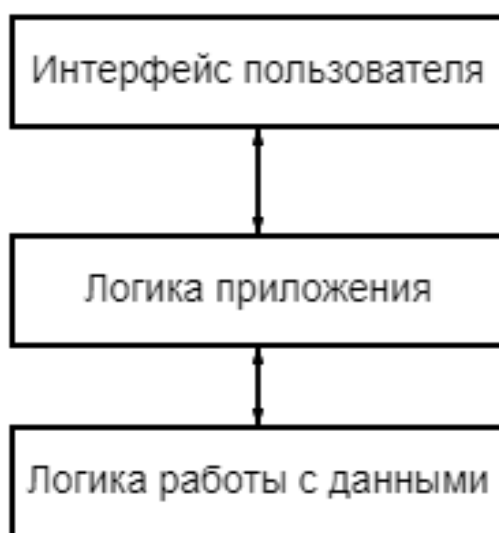


Рисунок 2.1 – Обобщённая функциональная схема приложения

При запуске приложения пользователь в зависимости от своего типа должен видеть то или иное содержимое в графическом окне, значит, необходимо разработать иерархию пользователей и класс, содержащий методы, в зависимости от типа пользователя определять, может ли пользователь исполнить некоторое действие или нет. Пользователи имеют возможность входа в систему, значит, нужен еще класс, содержащий в себе сведения о данных пользователя для входа в систему.

Необходимо создать класс, который будет содержать в себе методы пользователей, которому передается пользователь и его разрешения на определенные действия. Значит, важно еще создать класс, который создает пользователей и присваивает им определенные разрешения на выполнение определенных действий. Отсюда вытекает создание класса, представляющего из себя разрешение, и создание класса, выполняющего авторизацию, а также создание аккаунта.

Существуют различные типы билетов, а значит, нужно создать иерархию классов, представляющих функционал для билета. Заранее известна цена для билета, количество неизменяемо, значит, нужно создать класс, который будет создавать билеты.

Имеются авторы, жанры, спектакли, которые тоже будут иметь свои классы. Так как количество билетов ограничено и необходимо вести учет проданных билетов, то нужно создать класс, который будет учитывать продажи билетов, а также необходимо реализовать функционал, реализующий проверку существования спектакля в заданный день и проверяющий, есть ли еще свободные места на данный спектакль или нет.

Также наиболее важными будут классы, работающими с данными, и в зависимости от типа объектов, с которым работает класс, например, класс может работать с жанрами, спектаклями, авторами, билетами, нужно реализовать соответствующий функционал для работы с ними, иными словами, для каждого типа объекта нужно реализовать свой класс, работающими с объектами данного типа.

Таким образом, в приложении должны быть следующие объекты:

- билет, реализующий функционал билета;
- заказ, реализующий функционал заказа, содержит в себе сведения о купленном билете, дате покупки, сведения о пользователе, совершившего покупку;
- спектакль, реализующий функционал спектакля;
- пользователь, реализующий функционал пользователя, от которого будут наследоваться другие типы пользователей;
- аккаунт, содержит в себе сведения о пользователе, необходимые для входа в систему;
- типы пользователей (администратор, гость, авторизованный пользователь), реализующий функционал пользователя;
- действия пользователя, содержащий в себе методы пользователей;
- разрешение, содержащий в себе список действий, который пользователь определенного типа может выполнять;
- учет продажи билетов, объект, содержащий в себе сведения о спектакле и количество проданных на него билетов;
- создатель билетов, который будет создавать билеты определенного типа;
- создатель пользователей, который будет создавать пользователей определенного типа;

- отчет, содержащий в себе сведения о проданных билетах.

2.3 Структура базы данных

По условию курсовой работы необходимо разработать хранилище данных в виде базы данных основанной на *XML*-файлах. При разработке структуры базы данных необходимо выполнить следующие шаги:

- выделение сущностей и их атрибутов, которые будут храниться в базе данных, и формирование по ним таблиц;
- определение уникальных идентификаторов (первичных ключей) объектов, которые хранятся в строках таблицы;
- определение отношений между таблицами с помощью внешних ключей;
- нормализация данных в базе данных.

База данных состоит из четырех таблиц:

- *genres*;
- *spectacles*;
- *tickets*;
- *users*.

Для работы с таблицами существует три модели:

- *SpectacleModel*;
- *TicketModel*;
- *UserModel*.

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

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

Category и *Price* в таблице *TicketModel* содержат соответствующие данные о спектакле, на который был куплен билет.

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

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

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

2.4 Структура классов разработанного программного комплекса

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

- папка *img* содержит все картинки, содержащиеся в курсовом проекте;
- папка *Logs* содержит классы для работы и формирования статистических отчетов;
- папка *Models* содержит сущности, описанные выше;
- папка *Repositories* содержит три класса содержащих основные *CRUD* операции для работы со всеми базами данных в проекте;
- папка *Services* содержит классы для взаимодействия графического интерфейса с классами, находящимися в папке *Repositories*;

- папка *Users* содержит классы пользователей с определенными ролями;
- папка *XMLData* содержит *XML*-файлы содержащие данные о жанрах, спектаклях, билетах и пользователях, также в папке хранится информация о статистических отчетах (информационная модель проекта представлена на рисунке 2.1).

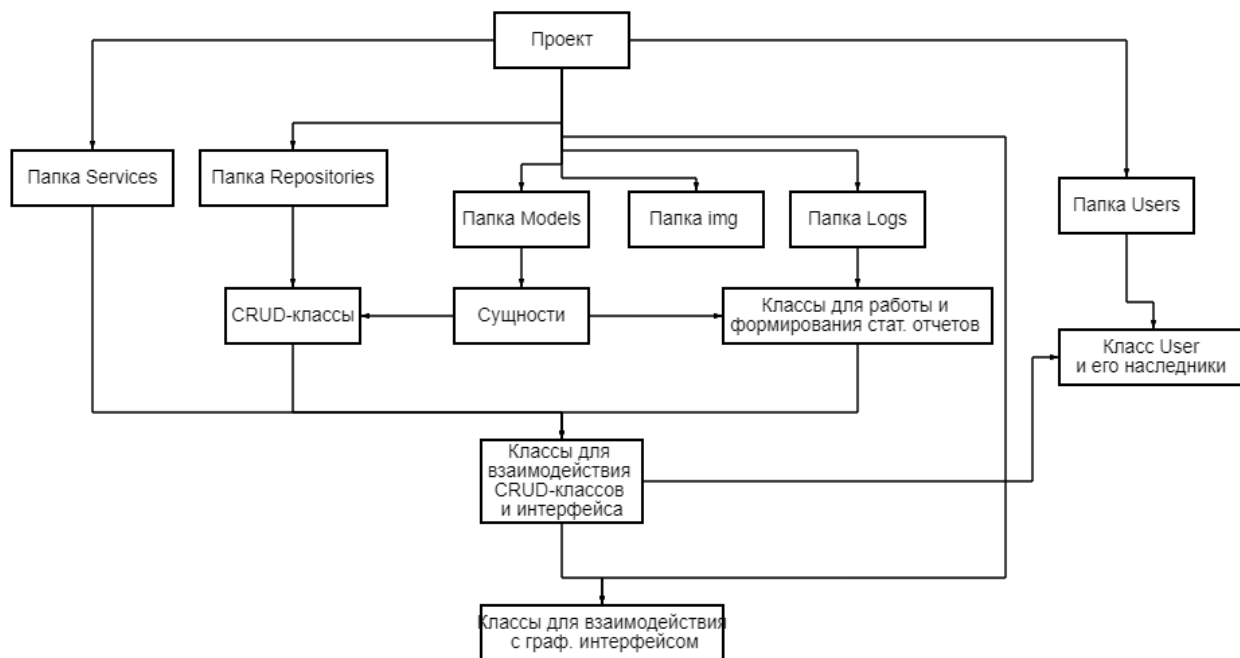


Рисунок 2.1 – информационная модель проекта

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

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

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

Метод *GetAll* возвращает коллекцию объектов типа *SpectacleModel*, которая содержит информацию о представлениях, полученную из *XML*-документа. Метод использует *LINQ to XML* для выбора всех элементов *spectacle* класса *SpectacleModel* из корневого элемента *XML*-документа и преобразует каждый элемент *spectacle* в объект *SpectacleModel*. Для каждого элемента *spectacle* метод

выбирает значения свойств объекта *SpectacleModel* из соответствующих элементов *XML*-документа. Кроме того, метод преобразует строковые значения элементов *XML* в соответствующие типы данных, такие как *DateTime* и *Int32*, используя методы *Parse*. Также, метод использует метод *ToDictionary* для преобразования элементов *category* в словарь, где ключом является значение атрибута *name*, а значением – десятичное число, полученное из значения элемента *category*.

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

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

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

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

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

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

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

Метод *GetAllGenres* возвращает список всех жанров, которые содержатся в файле с жанрами. В методе используется *LINQ to XML* для выборки элементов

с тегом *genre* из корневого элемента *XML*-документа *genreDoc*. Затем из каждого элемента выбирается значение *Value* и добавляется в список. В итоге метод возвращает список всех жанров, представленных в *XML*-файле.

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

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

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

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

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

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

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

предотвратить запись поврежденного файла, если что-то пойдет не так во время сохранения.

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

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

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

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

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

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

Класс *Registered* расширяет функциональность базового класса *User*, предоставляя методы для работы с данными пользователя, такие как *UpdateUserData* для обновления личных данных, а также методы из класса *TicketServices* для добавления, удаления и обновления данных о билетах. Кроме того, класс *Regis-*

tered предоставляет метод *GetThisSpectacle* для просмотра более подробной информации о спектаклях. Эти методы доступны только для зарегистрированных пользователей с соответствующими правами доступа.

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

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

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

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

3 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС И ТЕСТИРОВАНИЕ

3.1 Описание пользовательского интерфейса приложения

Помимо этого, приложение также должно обеспечивать возможность работы с различными объектами программы, такими как спектакли, авторы, пользователи, билеты и жанры. Каждый из этих объектов требует отдельного окна для работы с ним. Таким образом, в программе предусмотрены следующие графические окна:

- *MainForm* – главное окно приложения, которое содержит основную информацию о билетах и пользователях;
- *LoginForm* – окно, которое содержит в себя форму авторизации;
- *RegistrForm* – окно, предназначенное регистрации нового пользователя;
- *TicketBuyForm* – окно, предназначенное для покупки билетов и редактирования данных о спектаклях;
- *UserTicketsForm* – окно для просмотра и удаления билетов указанного пользователя;

При запуске приложения пользователь увидит приветственное окно *LoginForm*, которое является общим для всех типов пользователей и отображено на рисунке А.1. Это окно содержит два поля для ввода данных о пользователе, такие как имя пользователя и пароль, а также три кнопки: «Войти», «Войти как гость» и «Регистрация».

Если пользователь выберет кнопку «Регистрация», то откроется форма *RegistrForm*, которая также представлена на рисунке А.2. Форма содержит поля для ввода имени и пароля нового пользователя. После ввода данных, пользователь должен подтвердить регистрацию, и введенные данные будут проходить проверку. При успешной регистрации новый пользователь будет создан в XML-файле, а форма регистрации автоматически закроется.

Если же пользователь выберет кнопку «Войти как гость», то он попадет на форму *MainForm*, которая представлена на рисунке А.3. Данная форма содержит элементы, которые присущи гостю, включая таблицу с информацией о спектаклях и поля для поиска спектаклей по жанру и дате. В правом верхнем углу находится кнопка «Регистрация», при нажатии на которую пользователь сможет вернуться на форму *LoginForm*.

При успешной авторизации с ролью «Пользователь» пользователь увидит соответствующее окно *MainForm*. Кнопка «Регистрация» в правом верхнем углу изменится на «Выйти», и появится возможность покупки билета и просмотра информации о пользователе. Если пользователь выберет один из спектаклей, то на экране появится форма *TicketBuyForm*, которая представлена на рисунке А.4. Здесь пользователь сможет ознакомиться с информацией о выбранном спектакле и купить билет, нажав на кнопку «Купить». Также в правом верхнем углу отображается информация о пользователе, при нажатии на которую появится окно

UserTicketsForm, которое представлено на рисунке А.5. Здесь пользователь сможет просмотреть список приобретенных билетов и отменить покупку, если это необходимо.

При авторизации с ролью «Администратор» пользователь получает доступ к различным функциям. В частности, на главной форме (*MainForm*) ему становится доступно редактирование и удаление спектаклей. Также он может создавать отчеты о продажах билетов, которые будут представлены в виде *EXCEL*-таблицы.

Помимо этого, администратор может перейти на новую вкладку «Панель администрации», на которой расположена таблица со всеми пользователями и поля для добавления и изменения их данных. Если администратор выберет конкретного пользователя в таблице, то он сможет просмотреть список его билетов и удалить их через форму *UserTicketsForm*. Также администратор может редактировать данные этого пользователя. Если же при вводе имени пользователя в соответствующее поле окажется, что такого пользователя не существует, то программа предложит администратору создать его.

Таким образом, администратор получает расширенные возможности по управлению спектаклями и пользователями, что позволяет ему более эффективно управлять системой (диаграмма вариантов использования *UI* представлена на рисунке 3.1).

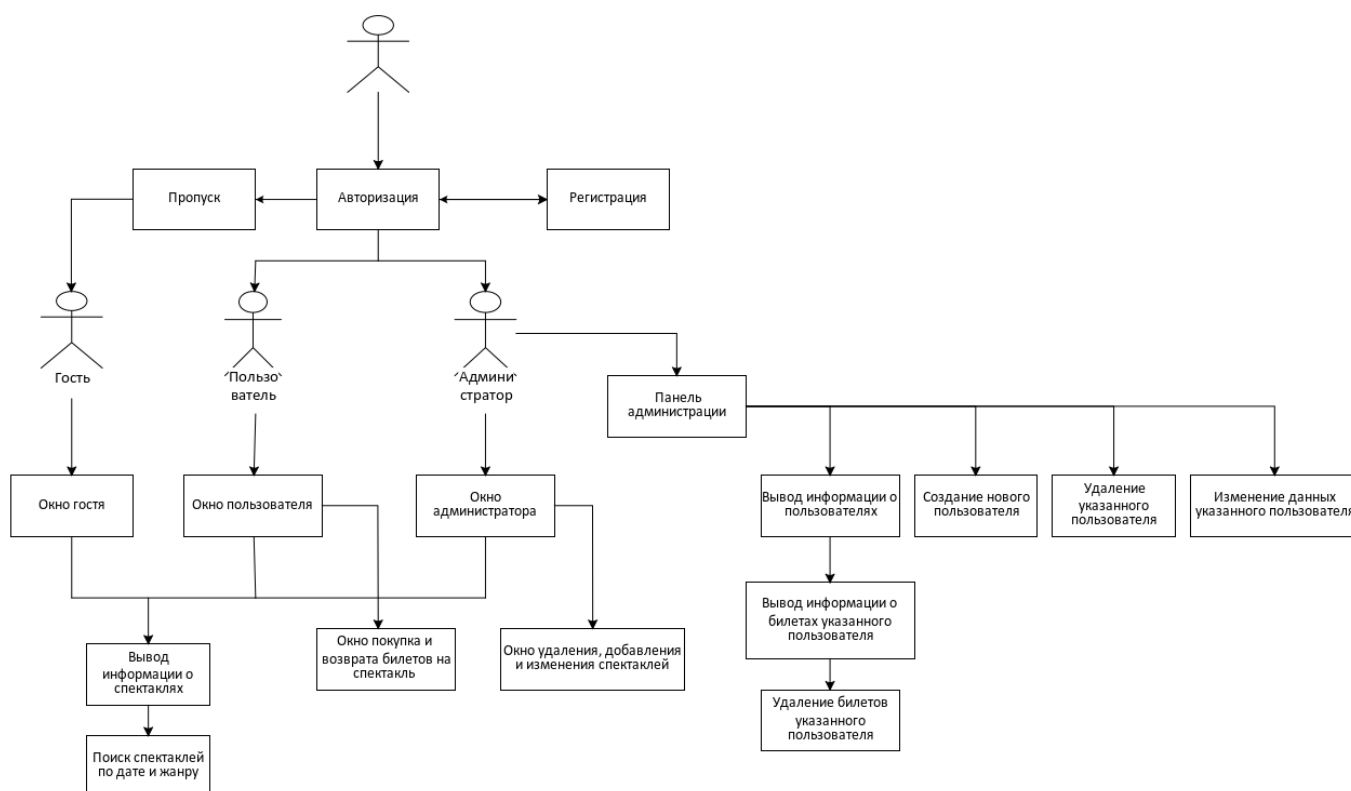


Рисунок 3.1 – Диаграмма вариантов использования *UI*

Графический интерфейс для каждого окна представлен в приложении А.

3.2 Результаты тестирования пользовательского интерфейса

Для начала работы с приложением необходимо открыть файл *App.exe*. После запуска приложения появится окно авторизации, где пользователь может войти в систему или продолжить работу в качестве гостя. Чтобы авторизоваться, пользователю нужно ввести свои данные и нажать кнопку «Войти». Если пользователь не зарегистрирован, он может зарегистрироваться, нажав соответствующую кнопку. Если пользователь вводит неверные данные, ему будет выведено сообщение об ошибке.

После входа в систему в качестве зарегистрированного пользователя будут доступны следующие функции:

- просмотр афиши театра;
- просмотр сведений о спектакле;
- просмотр всех приобретенных билетов;
- покупка билетов на интересующий пользователя спектакль;
- возможность отмены билета;
- поиск спектаклей по определенной дате;
- поиск спектаклей по наличию свободных мест;
- поиск спектаклей по жанру.

Для того чтобы купить билет на спектакль, пользователю необходимо нажать на выбранный спектакль, откроет окно, что содержит сведения о спектакле, в котором можно купить билет. В данном окне отображается информация о жанре дате и ценах на различные категории. Ниже находится кнопка «Купить», при нажатии на которую совершается покупка билета.

Когда пользователь приобретает билеты, он может в дальнейшем проверить список своих покупок. Для этого нужно перейти в главное окно программы, выбрать информацию о пользователе и затем выбрать опцию «Мои билеты». Это откроет окно, где пользователь может просмотреть свои приобретенные билеты. Если пользователь захочет вернуть билет, он может выбрать кнопку «Возврат билета». После этого билет будет удален из списка купленных и количество доступных билетов на спектакль увеличится.

Если пользователь вошел систему в качестве администратора, то ему предоставляется следующий функционал:

- добавление спектаклей;
- редактирование спектакля;
- удаление спектакля;
- добавление жанра;
- просмотр пользователей;
- редактирование пользователей;
- удаление пользователей;

- изменение цен билетов;
- формирование статистических отчетов о проданных билетах.

В окне управления спектаклями администратор может редактировать, добавлять и удалять спектакли, выбрав нужный спектакль из таблицы. Для добавления нового спектакля администратор может нажать на кнопку «Добавить» или на пустое поле таблицы. При добавлении нового спектакля администратор должен выбрать жанр из выпадающего списка, ввести дату, название и установить цены на каждую категорию билетов. Если спектакль с указанной датой уже существует, появится сообщение об ошибке.

Для редактирования спектакля администратор должен выбрать нужный спектакль в таблице. При нажатии на нужное поле появится окно для редактирования и удаления спектакля. Если необходимо добавить новый жанр, администратор может нажать на кнопку «Добавить» во всплывающем окне. Внешний вид окна для редактирования спектакля представлен на рисунке А.6.

Для работы с пользователями администратор должен перейти во вкладку «Администрация» в которой содержится таблица со всеми пользователями. Для редактирования информации о пользователе администратор должен выбрать нужного пользователя в списке и нажать на его имя. Тогда откроется окно, в котором администратор сможет изменить информацию о пользователе, а также увидеть данные о приобретенных им билетах. Если нужный пользователь отсутствует в списке, то администратор может добавить его, введя имя пользователя и нажав на кнопку «Добавить». Если такого пользователя нет в базе данных, программа предложит создать его. После добавления пользователь будет доступен в списке пользователей.

Администратор также может составлять различные статистические отчеты. Для работы с отчетами администратор должен нажать на кнопку «Отчеты» в подробной информации о пользователе. После сформируется отчет в виде *EXCEL*-таблицы.

3.3 Результаты модульного тестирования

Тестирование кода на *C#* является важным шагом в разработке программного обеспечения. Оно позволяет выявить ошибки и недочеты в коде, а также убедиться, что он работает корректно и соответствует требованиям.

В проекте было создано отдельный проект *CourseWorkTests* для тестов. Проект содержит три класса для тестов *SpectacleManagerTests*, *TicketManagerTests* и *UserManagerTests*. Тестируемые классы *SpectacleManager*, *TicketManager* и *UserManager* включают в себя все операции *CRUD*. Проведение тестирования этих классов обеспечивает покрытие всех основных функций программы.

В классе *SpectacleManagerTests* были созданы методы для тестирования методов: *AddAddsNewSpectacle*, *UpdateUpdatesSpectacle*, *TestDataValidateValidDataReturnsTrue* и *DeleteDeletesSpectacle*.

Метод *AddAddsNewSpectacle* представляет собой юнит-тест, который проверяет, что метод *Add* объекта *SpectacleManager* правильно добавляет новый элемент в коллекцию всех «спектаклей» и корректно обновляет ее размерность.

UpdateUpdatesSpectacle представляет собой юнит-тест, который проверяет, что метод *Update* объекта *SpectacleManager* правильно обновляет информацию о существующем спектакле в коллекции всех «спектаклей».

DeleteDeletesSpectacle – юнит-тест, который проверяет, что метод *Delete* объекта *SpectacleManager* правильно удаляет спектакль из коллекции всех «спектаклей».

Тест *TestDataValidateValidDataReturnsTrue* является юнит-тестом для метода *DataValidate* объекта *SpectacleManager*. Метод *DataValidate* принимает объект *SpectacleModel* и проверяет, что все его поля соответствуют определенным правилам валидации.

Класс *TicketManagerTests* содержит методы для тестирования класса *TicketManager*. Тест *GetAllReturnsTicketModels* проверяет, что метод *GetAll* объекта *TicketManager* возвращает коллекцию моделей *TicketModel*.

AddValidTicketModelAddsTicketToXml проверяет, что при добавлении допустимой модели билета в коллекцию билетов (*TicketManager*), количество элементов в коллекции увеличивается на 1.

Тест *DeleteExistingTicketIdRemovesTicketFromXml* проверяет функциональность удаления билета с помощью метода *Delete* класса *TicketManager*. Сначала создается спектакль, чтобы использовать его для добавления билета, а затем берется *ID* первого билета в коллекции, полученной из метода *GetAll* *TicketManager*, чтобы удалить его. Затем происходит проверка того, что кол-во билетов уменьшилось на 1, и что билет с указанным *ID* больше не присутствует в коллекции. В конце удаляется созданный спектакль, чтобы не повлиять на другие тесты.

В классе *UserManagerTests* содержатся методы для класса *UserManager*. Тест *GetAllReturnsAllUsers* проверяет, что метод *GetAll* класса *UserManager* возвращает всех пользователей системы. В блоке *Arrange* создаются ожидаемые пользователи и создаётся *XML*-документ, содержащий информацию об этих пользователях. В блоке *Act* вызывается метод *GetAll*, который должен вернуть всех пользователей из системы. В блоке *Assert* проверяется, что количество возвращенных пользователей больше нуля.

AddValidUserAddsUserToXmlDoc проверяет, что метод *Add* класса *UserManager* успешно добавляет нового пользователя в список пользователей и что этот пользователь может быть найден в этом списке по его имени пользователя. Также этот тест в конце удаляет добавленного пользователя из списка для того, чтобы он не остался в списке после выполнения теста.

Тест *UpdateValidUserUpdatesUserInXmlDoc* проверяет, что метод *Update* менеджера пользователей обновляет информацию о существующем пользователе в документе *XML*. Сначала в методе *Arrange* создается объект *existingUser*

класса *UserModel*, который представляет существующего пользователя, которого нужно обновить. Затем создается новый пользователь *updatedUser*, который будет содержать обновленную информацию, и добавляется в документ *XML* с помощью метода *Add* менеджера пользователей. После этого вызывается метод *Update* с объектом *updatedUser* в качестве аргумента. В методе *Assert* проверяется, что информация обновленного пользователя была сохранена в документе *XML* с помощью метода *GetAll*. В конце метода *updatedUser* удаляется из документа *XML* с помощью метода *Delete*.

Тест *DeleteUserExistsRemovesUser* проверяет, что метод *Delete* класса *UserManager* корректно удаляет пользователя из системы. Сначала создается новый пользователь *user*, и он добавляется в систему с помощью метода *Add*. Затем вызывается метод *Delete* для удаления этого пользователя.

Далее можно представлен результат выполнения всех тестов (рисунок 3.1).

▲ ✓ CourseW...	153 мс
▲ ✓ Course...	153 мс
▲ ✓ Spec...	79 мс
✓ Ad...	69 мс
✓ Del...	3 мс
✓ Tes...	< 1 мс
✓ Up...	7 мс
▲ ✓ Ticke...	35 мс
✓ Ad...	8 мс
✓ Del...	11 мс
✓ Del...	16 мс
✓ Ge...	< 1 мс
▲ ✓ User...	39 мс
✓ Ad...	8 мс
✓ Del...	20 мс
✓ Ge...	3 мс
✓ Up...	8 мс

Рисунок 3.4 – Модульные тесты

На скриншоте представлены все методы классов *SpectacleManagerTests*, *TicketManagerTests* и *UserManagerTests* с результатами их прохождения. Галочка рядом с каждым методом означает, что тест успешно прошел проверку и программа работает правильно. Всего приведено 12 тестов, и все они прошли проверку.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано приложение для системы распространения билетов в театре.

В соответствии с поставленными задачами была изучена сфера продажи билетов в театр. Выявлены типы билетов в театр, свойства спектаклей, процесс организации продажи билетов в театр. Различные типы билетов и пользователей включены в приложение для системы распространения билетов в театре.

К системе обеспечен доступ для пользователей различных типов, у каждого пользователя есть свой присущий лишь ему набор функций:

- администратор – может добавлять спектакли, авторов, жанры, управлять пользователями, изменять цены на билеты, формировать различные статистические отчеты о проданных билетах по различным параметрам: по дате, жанру, автору;

- авторизованный пользователь – просматривать спектакли, проводить поиск спектаклей по дате, жанру, наличию свободных мест, покупать билеты, отменять купленные билеты;

- гость – просматривать афишу театра.

В качестве источника данных для приложения использованы *XML*-файлы. Для реализации поставленной задачи использованы средства языка программирования *C#*. При реализации поставленной задачи были использованы паттерны проектирования. Для доступа и работы с данными была изучена и использована технология *LINQ*. Для верификации разработанного приложения были разработаны модульные тест.

Созданное приложение значительно облегчит автоматизацию процесса управления театром и полезно как для администрации театра, так и для рядового покупателя, так как позволяет быстро и удобно приобрести билеты на спектакль.

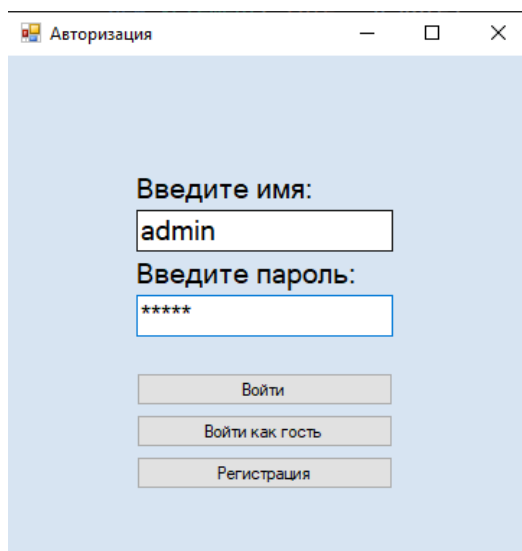
Список используемых источников

1. Metanit [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/tutorial/15.1.php> – Дата доступа: 20.04.2023.
2. Создание приложения Windows Forms на языке программирования C# [Электронный ресурс] – Режим доступа: <https://learn.microsoft.com/ru-ru/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022/> – Дата доступа: 28.04.2023.
3. Сайт с информацией по паттернам проектирования [Электронный ресурс] – Режим доступа: <https://www.dofactory.com/> – Дата доступа: 01.05.2023.
1. 4. Что такое *Windows Forms - Windows Forms .NET | Microsoft Docs* [Электронный ресурс] – Режим доступа: [https:// docs.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/](https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/) – Дата доступа: 05.05.2023.
5. Албахари, Дж. C# 7.0. Карманный справочник / Дж. Албахари, Б. Албахари. – пер. с английского – СПб.: ООО «Альфа-книга», 2017. – 224 с.
6. Колесников. А. Программирование на языке C# в среде .NET Framework / А. Колесников. – СПб: БХВ-Петербург, 2015. – 251 с.
7. Натан, А. WPF 4, подробное руководство / А. Натан. – пер. с англ. – СПб.: Символ-Плюс, 2018. – 880 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А (обязательное)

Графические представления интерфейса программы



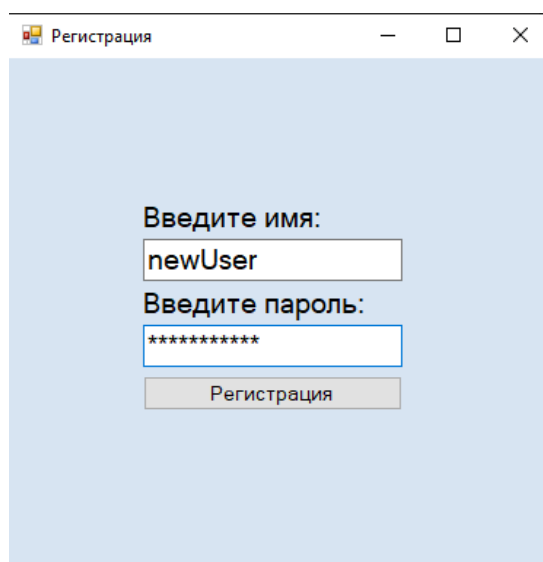
Авторизация

Введите имя:
admin

Введите пароль:

Войти
Войти как гость
Регистрация

Рисунок А1 – Окно авторизации



Регистрация

Введите имя:
newUser

Введите пароль:

Регистрация

Рисунок А2 – Окно регистрации

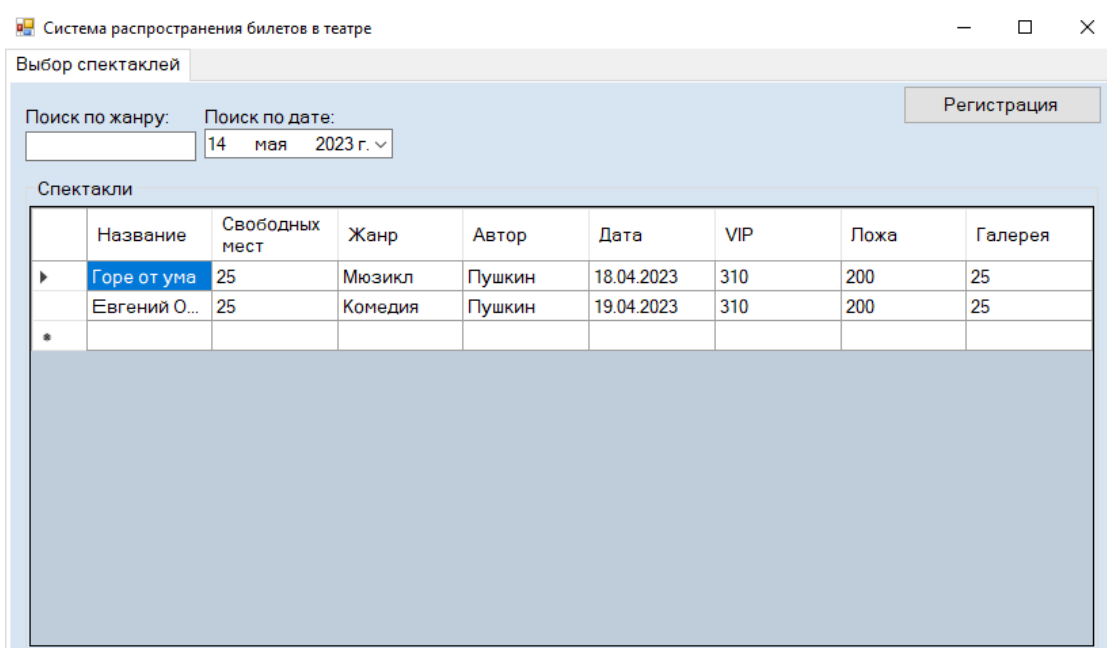


Рисунок А3 – Главное окно пользователя

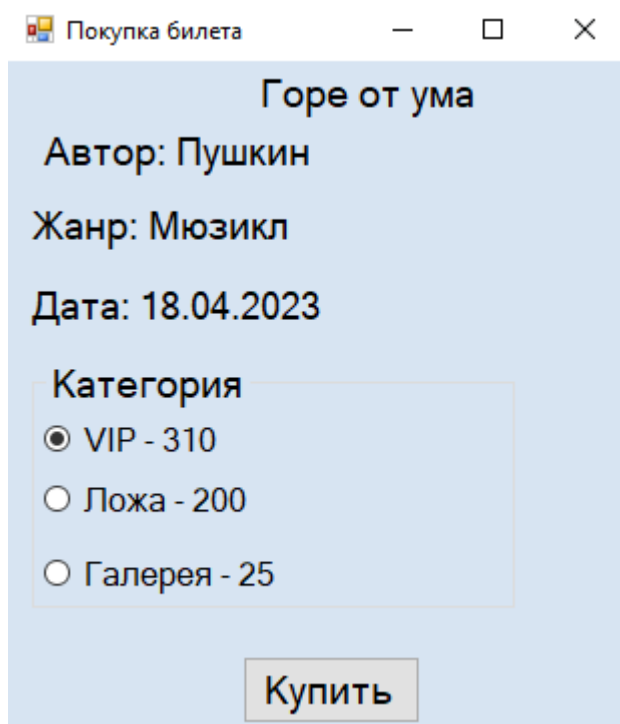


Рисунок А4 – Окно покупки билета на спектакль

Билеты пользователя 123

№	Название	Дата	Категория	Цена
1	Горе от ума	18.04.2023	Medium	200
2	Горе от ума	18.04.2023	VIP	310
3	Евгений Он...	19.04.2023	VIP	310
4	Шекспировс...	15.05.2023	VIP	500
5	Ревизор	15.02.2023	VIP	150
*				

Возврат билета

Рисунок А5 – Окно информации о билетах

Покупка билета

Название: Евгений Онегин

Автор: Пушкин

Жанр: Комедия

Дата: 19 апреля 2023 г.

Категория

VIP 310

Ложа 200

Галерея 25

Изменить Удалить

Рисунок А6 – Окно изменения информации о билетах

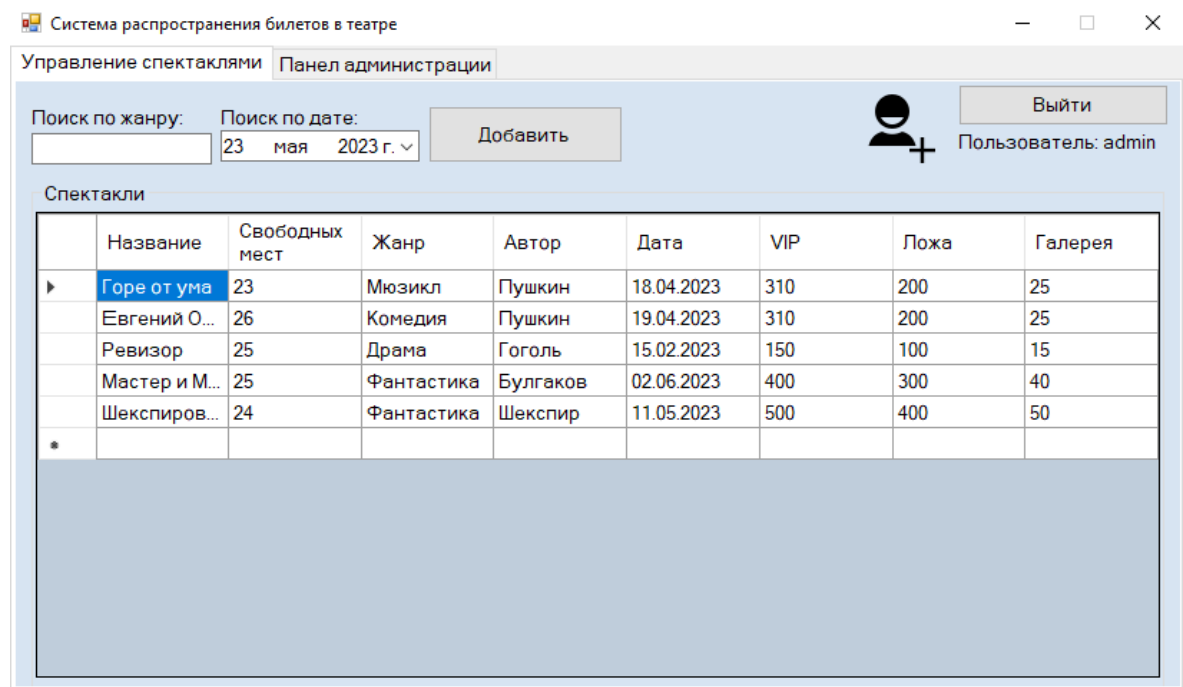


Рисунок А7 – Главное окно администратора

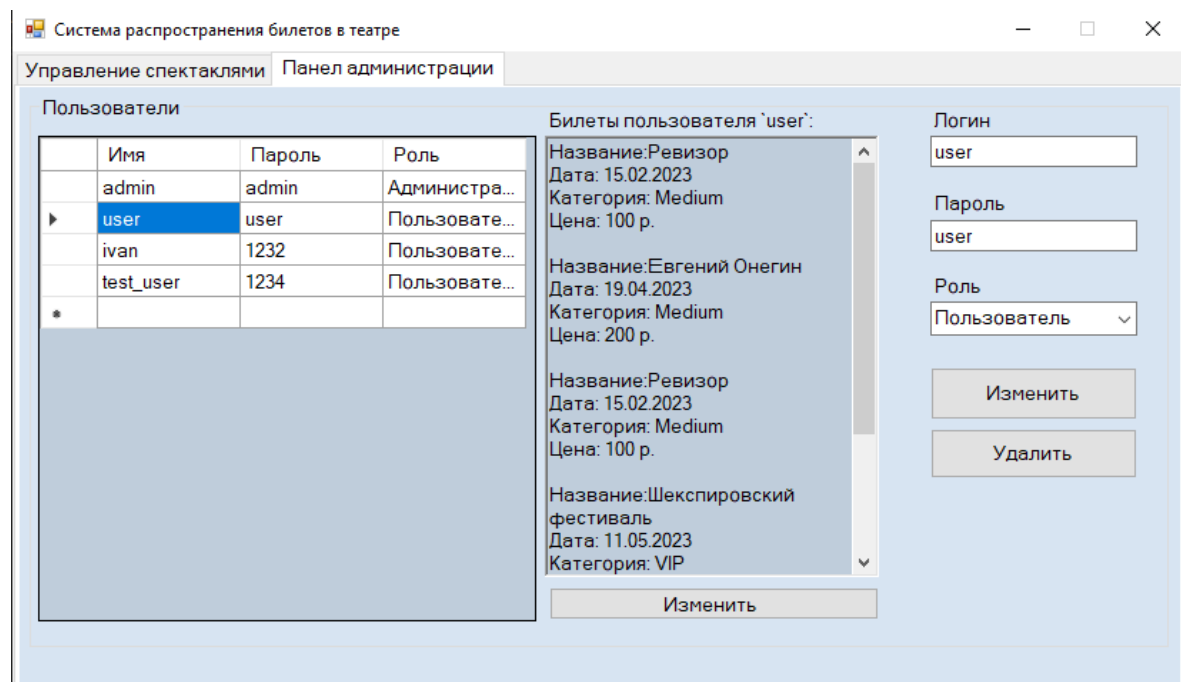


Рисунок А8 – Окно панели администрации

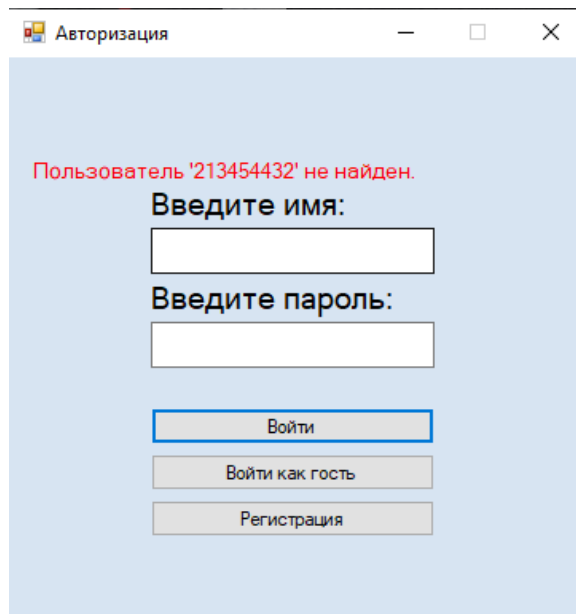


Рисунок А9 – Окно с ошибкой при авторизации

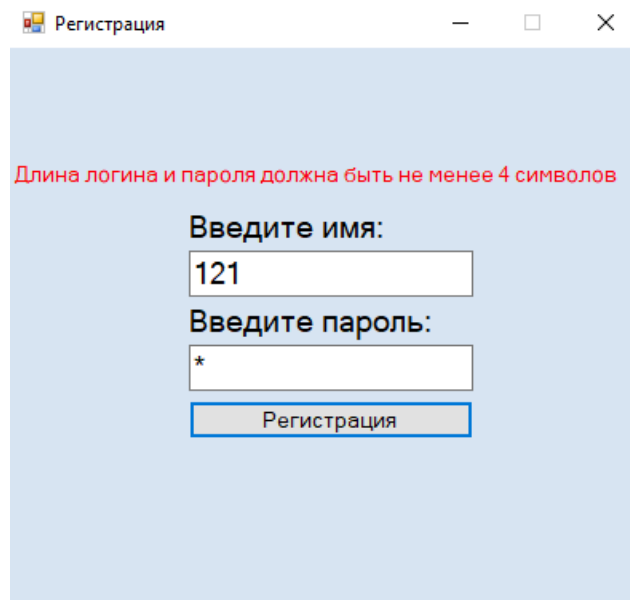


Рисунок А10 – Окно с ошибкой при регистрации

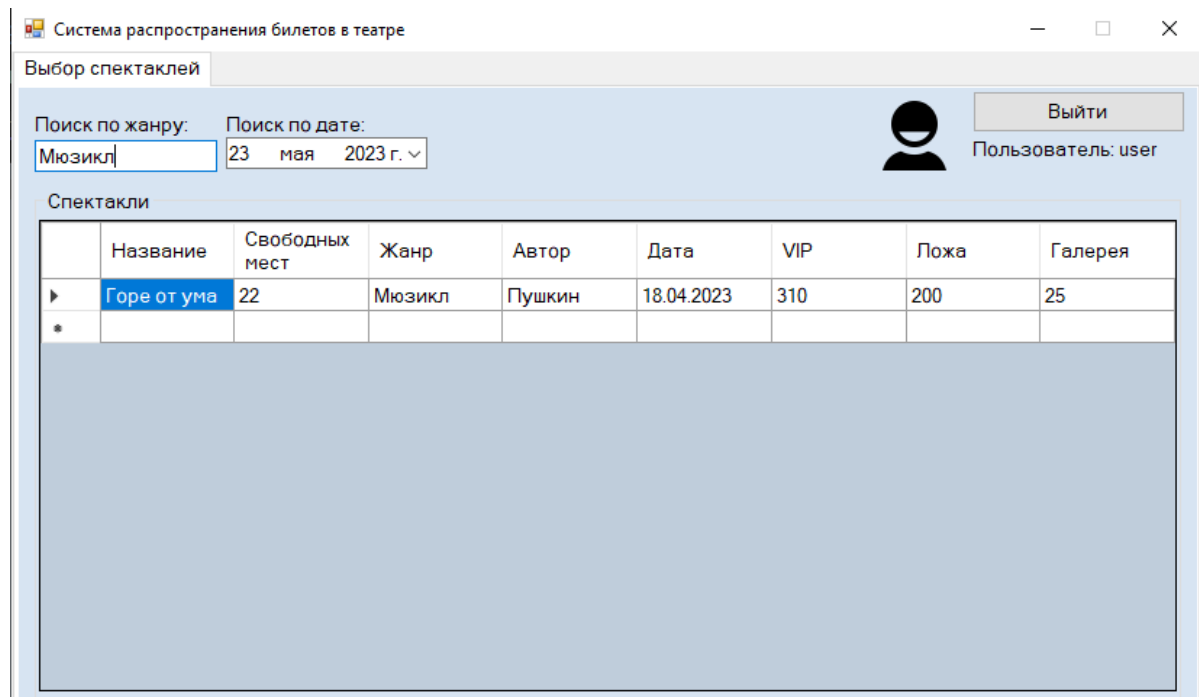


Рисунок А11 – Окно с примером поиска спектакля по жанру

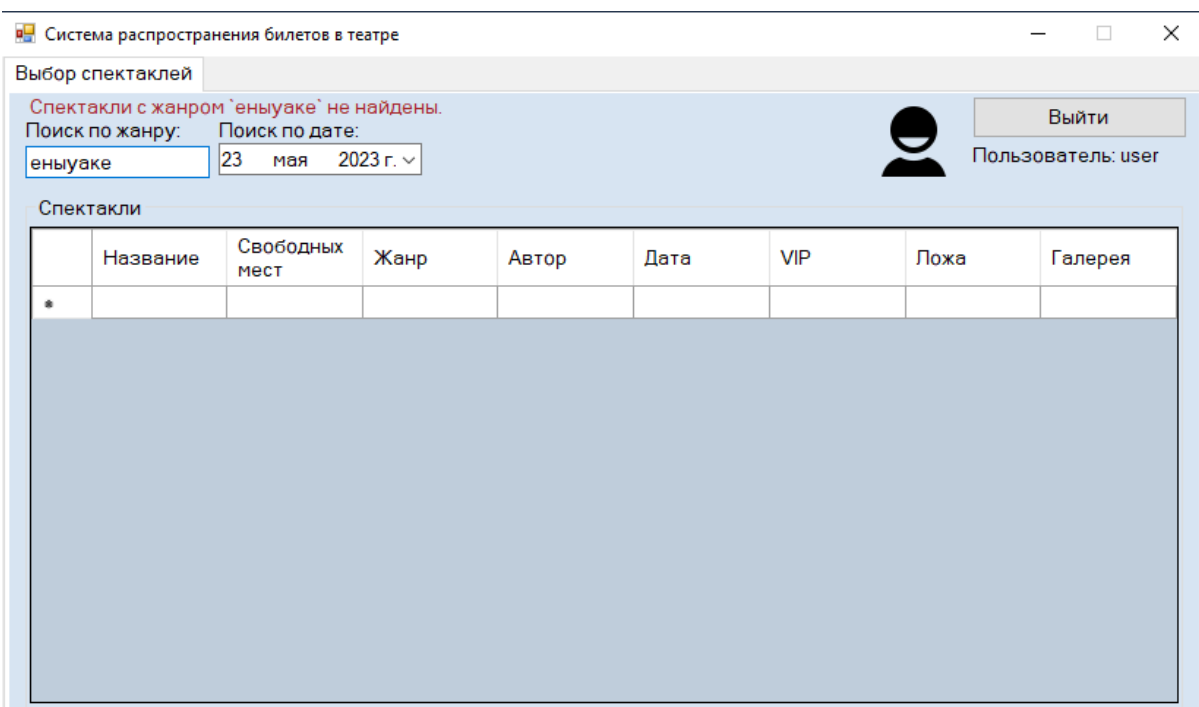


Рисунок А12 – Окно с ошибкой при поиске спектакля по жанру

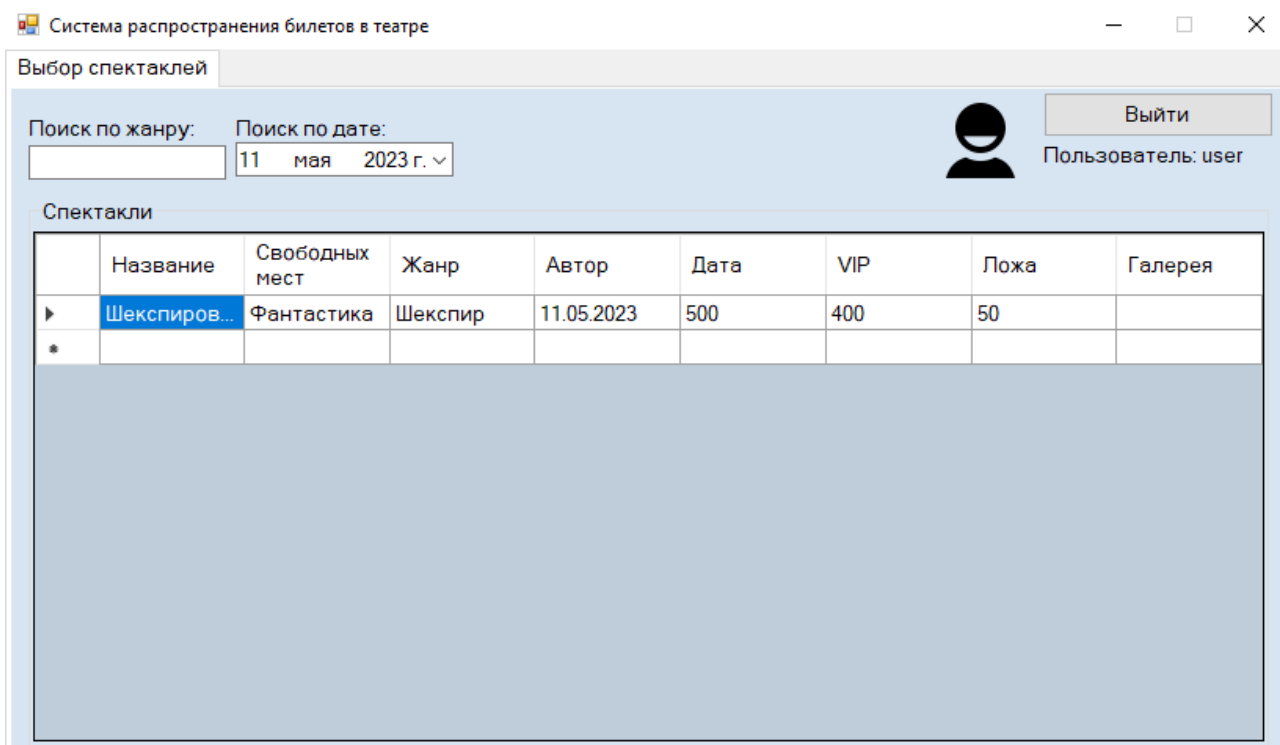


Рисунок А13 – Окно с примером поиска спектакля по дате

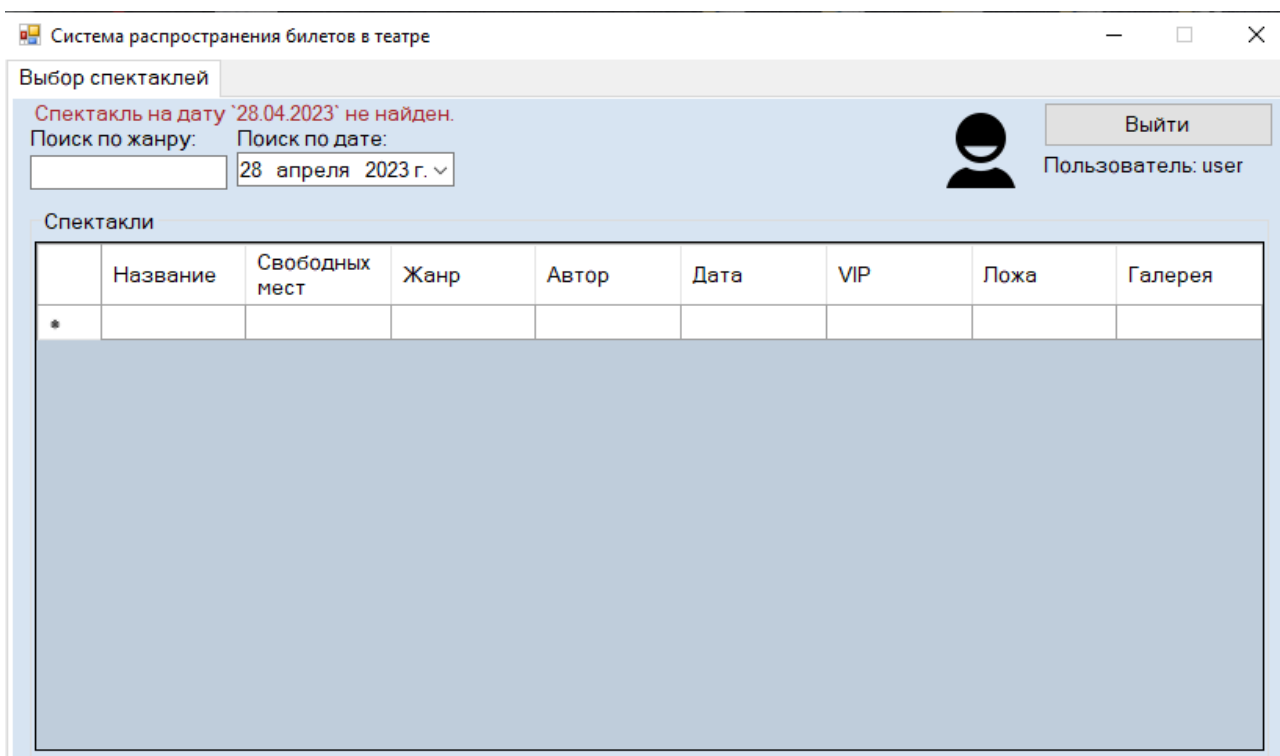


Рисунок А14 – Окно с ошибкой при поиске спектакля по дате

Система распространения билетов в театре

Выбор спектаклей

Поиск по жанру: Поиск по дате: 23 мая 2023 г. ▾

Регистрация

Спектакли

	Название	Свободных мест	Жанр	Автор	Дата	VIP	Ложа	Галерея
	Евгений О...	24	Комедия	Пушкин	19.04.2023	310	200	25
	Ревизор	24	Драма	Гоголь	15.02.2023	150	100	15
	Мастер и М...	23	Фантастика	Булгаков	02.06.2023	400	300	40
	Шекспиров...	22	Фантастика	Шекспир	11.05.2023	500	400	50
▶	Горе от ума	22	Мюзикл	Пушкин	18.04.2023	310	200	25
*								

Рисунок А15 – Окно с отсортированными по возрастанию спектаклями

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг программы

Листинг SpectacleModel.cs

```
using System.Collections.Generic;
using System;
using System.Windows.Forms;

public enum Categorias
{
    VIP,
    Medium,
    Standart
}

/// <summary>
/// Класс сущности спектакля
/// </summary>
public class SpectacleModel
{
    public string Title { get; set; }
    public string Author { get; set; }
    public string Genre { get; set; }
    public DateTime Date { get; set; }
    public Dictionary<Categorias, decimal> Categories { get; set; }
    public int FreePlace { get; set; }
}
```

Листинг TicketModel.cs

```
using System;

/// <summary>
/// Класс сущности билета
/// </summary>
public class TicketModel
{
    public int Id { get; set; }
    public string Owner { get; set; }
    public DateTime Date { get; set; }
    public string Title { get; set; }
    public Categorias Category { get; set; }
    public decimal Price { get; set; }
}
```

Листинг UserModel.cs

```
public enum Role
{
    admin,
    registered,
    guest
}

/// <summary>
/// Класс сущности пользователя
```

```

/// </summary>
public class UserModel
{
    public string Login { get; set; }
    public string Password { get; set; }
    public Role Role { get; set; }
}

```

Листинг SpectacleManager.cs

```

using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using System;
using System.Xml.Schema;
using System.Collections.ObjectModel;

/// <summary>
/// Предоставляет методы для работы с спектаклями в XML-документе.
/// </summary>
public static class SpectacleManager
{
    private static readonly string _xmlFilePath;
    private static readonly string _xmlGenrePath;

    private static readonly XDocument _xmlDoc;
    private static readonly XmlSchemaSet _schemas;
    private static readonly XDocument _genreDoc;
    static SpectacleManager()
    {
        _xmlFilePath = "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\spectacle.xml";
        _xmlGenrePath = "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\genres.xml";

        _xmlDoc = XDocument.Load(_xmlFilePath);
        _schemas = new XmlSchemaSet();
        _schemas.Add(null, "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\spectacle.xsd");
        _genreDoc = XDocument.Load(_xmlGenrePath);
    }
    /// <summary>
    /// Получить все спектакли.
    /// </summary>
    /// <returns>Список всех спектаклей.</returns>
    public static IEnumerable<SpectacleModel> GetAll()
    {
        return _xmlDoc.Root.Elements("spectacle").Select(x => new SpectacleModel
        {
            Title = x.Element("title").Value,
            Author = x.Element("author").Value,
            Genre = GetGenreNameById(Int32.Parse(x.Element("genre").Value)),
            Date = DateTime.Parse(x.Element("date").Value),
            Categories = x.Elements("category").ToDictionary(y => (Categorias)Enum.Parse(typeof(Categorias), y.Attribute("name").Value), y => decimal.Parse(y.Value)),
            FreePlace = Int32.Parse(x.Element("freePlase").Value)
        });
    }
    /// <summary>
    /// Добавить спектакль.
    /// </summary>
    /// <param name="item">Новый спектакль.</param>
    /// <exception cref="ArgumentException">Исключение, если спектакль на эту дату уже существует.</exception>

```

```

public static void Add(SpectacleModel item)
{
    if (DataValidate(item))
    {
        DateTime date = item.Date.Date;
        if (GetAll().Any(x => x.Date.Date == date))
        {
            throw new ArgumentException($"Спектакль на дату {date.ToShortDateString()} уже существует.");
        }

        int genreId = GetGenreIdByName(item.Genre);

        XElement newSpectacle = new XElement("spectacle",
            new XElement("title", item.Title),
            new XElement("author", item.Author),
            new XElement("genre", genreId),
            new XElement("date", item.Date.ToString("yyyy-MM-dd")),
            new XElement("freePlase", item.FreePlace)
        );

        foreach (var category in item.Categories)
        {
            newSpectacle.Add(new XElement("category", category.Value.ToString(),
                new XAttribute("name", category.Key)));
        }
        _xmlDoc.Root.Add(newSpectacle);
        _xmlDoc.Save(_xmlFilePath);
    }
}

/// <summary>
/// Работа с жанрами
/// </summary>

/// <summary>
/// Получить Id жанра по названию.
/// </summary>
/// <param name="name">Название жанра</param>
/// <returns>Id искомого жанра</returns>
public static int GetGenreIdByName(string name)
{
    var genreElement = _genreDoc.Descendants("genre")
        .FirstOrDefault(x => x.Value == name);
    if (genreElement != null && int.TryParse(genreElement.Attribute("id").Value, out int genreId))
    {
        return genreId;
    }
    throw new ArgumentException($"Жанр {name} не найден в базе данных.");
}

/// <summary>
/// Получить название жанра по Id.
/// </summary>
/// <param name="id">Id жанра</param>
/// <returns>Название искомого жанра</returns>
private static string GetGenreNameById(int id)
{
    var genreElement = _genreDoc.Descendants("genre")
        .FirstOrDefault(x => x.Attribute("id").Value == id.ToString());
    if (genreElement != null)
    {
        return genreElement.Value;
    }
    throw new ArgumentException($"Жанр с id {id} не найден в базе данных.");
}

```

```

}
/// <summary>
/// Добавить новый жанр
/// </summary>
/// <param name="name">Название жанра</param>
public static void AddGenre(string name)
{
    int maxId = _genreDoc.Root.Elements("genre").Select(g => (int)g.Attribute("id")).Max();
    XElement newGenre = new XElement("genre", new XAttribute("id", maxId + 1), name);
    _genreDoc.Root.Add(newGenre);
    _genreDoc.Save(_xmlGenrePath);
}
public static List<string> GetAllGenres()
{
    return _genreDoc.Root.Elements("genre").Select(x => x.Value).ToList();
}
/// <summary>
/// Обновить спектакль.
/// </summary>
/// <param name="item">Спектакль, который нужно обогнуть.</param>
public static void Update(SpectacleModel item)
{
    if (DataValidate(item))
    {
        XElement spectacleToUpdate = GetElement(item);

        spectacleToUpdate.SetElementValue("author", item.Author);
        spectacleToUpdate.SetElementValue("genre", GetGenreIdByName(item.Genre).ToString());
        spectacleToUpdate.SetElementValue("date", item.Date.ToString("yyyy-MM-dd"));
        spectacleToUpdate.SetElementValue("freePlase", item.FreePlace);

        foreach (var category in item.Categories)
        {
            var categoryToUpdate = spectacleToUpdate.Elements("category").FirstOrDefault(x =>
Enum.Parse(typeof(Categorias), x.Attribute("name").Value).ToString() == category.Key.ToString());

            if (categoryToUpdate != null)
            {
                categoryToUpdate.SetValue(category.Value.ToString());
            }
            else throw new Exception(categoryToUpdate.ToString());
        }
        _xmlDoc.Save(_xmlFilePath);
    }
}
/// <summary>
/// Удаляет спектакль.
/// </summary>
/// <param name="item">Спектакль, который нужно удалить.</param>
public static void Delete(SpectacleModel item)
{
    XElement spectacleToDelete = GetElement(item);
    spectacleToDelete.Remove();
    _xmlDoc.Save(_xmlFilePath);
}
/// <summary>
/// Получает элемент спектакля из XML-документа.
/// </summary>
/// <param name="item">Спектакль, элемент которого нужно получить.</param>
/// <returns>Элемент спектакля.</returns>
public static XElement GetElement(SpectacleModel item)
{

```

```

        DateTime date = item.Date.Date;
        XElement element = _xmlDoc.Root.Elements("spectacle").FirstOrDefault(x => DateTime.Parse(x.Element("date").Value).Date == date);
        if (element != null)
        {
            return element;
        }
        else throw new ArgumentException("Элемент не найден.");
    }
    /// <summary>
    /// Проверяет, что данные спектакля проходят валидацию.
    /// </summary>
    /// <param name="item">Спектакль, данные которого нужно проверить.</param>
    /// <returns>true, если данные проходят валидацию, false в противном случае.</returns>
    public static bool DataValidate(SpectacleModel item)
    {
        XElement tmpSpectacle = new XElement("spectacles",
            new XElement("spectacle",
                new XElement("title", item.Title),
                new XElement("author", item.Author),
                new XElement("genre", item.Genre),
                new XElement("date", item.Date.ToString("yyyy-MM-ddTHH:mm:ss"))
            ));

        foreach (var category in item.Categories)
        {
            tmpSpectacle.Element("spectacle").Add(new XElement("category", category.Value,
                new XAttribute("name", category.Key)));
        }
        var xdoc = new XDocument(tmpSpectacle);
        xdoc.Validate(_schemas, (o, e) =>
        {
            throw new ArgumentException(e.Message);
        });
        return true;
    }
}

```

Листинг TicketManager.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using System.Xml.Schema;
using App.Services;

/// <summary>
/// Предоставляет методы для управления билетами в документе XML.
/// </summary>
public static class TicketManager
{
    private static readonly string _xmlFilePath;
    private static readonly XDocument _xmlDoc;
    private static readonly XmlSchemaSet _schemas;
    private static SpectacleServices _spectacleServices;

    static TicketManager()
    {
        _xmlFilePath = "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\tickets.xml";
        _xmlDoc = XDocument.Load(_xmlFilePath);
        _schemas = new XmlSchemaSet();
    }
}

```

```

        _schemas.Add(null, "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\tickets.xsd");
        _spectacleServices = new SpectacleServices();
    }
    /// <summary>
    /// Получает все билеты.
    /// </summary>
    /// <returns>Список всех билетов.</returns>
    public static IEnumerable<TicketModel> GetAll()
    {
        return _xmlDoc.Root.Elements("ticket").Select(t =>
            new TicketModel
            {
                Id = int.Parse(t.Attribute("id").Value),
                Owner = t.Element("owner").Value,
                Date = DateTime.Parse(t.Element("date").Value),
                Title = _spectacleServices.ShowSpectacle(DateTime.Parse(t.Element("date").Value)).Title,
                Category = (Categorias)Enum.Parse(typeof(Categorias), t.Element("category").Value),
                Price = int.Parse(t.Element("price").Value)
            });
    }
    /// <summary>
    /// Добавляет новый билет.
    /// </summary>
    /// <param name="ticket">Новый билет для добавления.</param>
    /// <exception cref="ArgumentException">Выдается, когда данные недействительны.</exception>
    public static void Add(TicketModel ticket)
    {
        if (!DataValidate(ticket))
        {
            throw new ArgumentException("Данные не валидны.");
        }

        int id = _xmlDoc.Root.Elements("ticket").Max(t => int.Parse(t.Attribute("id").Value)) + 1;
        XElement newTicket = new XElement("ticket",
            new XAttribute("id", id),
            new XElement("owner", ticket.Owner),
            new XElement("date", ticket.Date.ToString("yyyy-MM-dd")),
            new XElement("category", ticket.Category),
            new XElement("price", ticket.Price)
        );
        _xmlDoc.Root.Add(newTicket);
        _xmlDoc.Save(_xmlFilePath);
    }
    /// <summary>
    /// Удаляет билет.
    /// </summary>
    /// <param name="ticketId">Идентификатор удаляемого билета.</param>
    /// <exception cref="ArgumentException">Выдается, когда билет с указанным идентификатором не суще-
    ствует.</exception>
    public static void Delete(int ticketId)
    {
        XElement ticketToDelete = _xmlDoc.Root.Elements("ticket")
            .SingleOrDefault(t => t.Attribute("id")?.Value == ticketId.ToString());

        if (ticketToDelete != null)
        {
            ticketToDelete.Remove();
            _xmlDoc.Save(_xmlFilePath);
        }
        else
        {
            throw new ArgumentException($"Ticket with id {ticketId} does not exist.");
        }
    }

```

```

    }
}
/// <summary>
/// Проверяет, что данные билета проходят валидацию.
/// </summary>
/// <param name="ticket">Спектакль, данные которого нужно проверить.</param>
/// <returns>true, если данные проходят валидацию, false в противном случае.</returns>
private static bool DataValidate(TicketModel ticket)
{
    bool isValid = true;

    var tmpTicket = new XElement("tickets",
        new XElement("ticket",
            new XAttribute("id", int.MinValue),
            new XElement("owner", ticket.Owner),
            new XElement("date", ticket.Date.ToString("yyyy-MM-dd")),
            new XElement("category", ticket.Category),
            new XElement("price", ticket.Price)
        ));

    var xdoc = new XDocument(tmpTicket);

    xdoc.Validate(_schemas, (o, e) =>
    {
        isValid = false;
    });

    return isValid;
}

```

} UserManager

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Linq;
using System.Xml.Schema;

/// <summary>
/// Предоставляет методы для работы с пользователями в файле XML.
/// </summary>
public static class UserManager
{
    private static readonly string _xmlFilePath;
    private static readonly XDocument _xmlDoc;
    private static readonly XmlSchemaSet _schemas;

    static UserManager()
    {
        _xmlFilePath = "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\users.xml";
        _xmlDoc = XDocument.Load(_xmlFilePath);
        _schemas = new XmlSchemaSet();
        _schemas.Add(null, "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\users.xsd");
    }

    /// <summary>
    /// Возвращает все объекты UserModel из файла XML.
    /// </summary>
    /// <returns>Все объекты UserModel из файла XML.</returns>
    public static IEnumerable<UserModel> GetAll()
    {

```

```

return _xmlDoc.Root.Elements("user").Select(u =>
    new UserModel
    {
        Login = u.Element("login").Value,
        Password = u.Element("password").Value,
        Role = (Role)Enum.Parse(typeof(Role), u.Element("role").Value)
    });
}
/// <summary>
/// Добавляет новый объект UserModel в файл XML.
/// </summary>
/// <param name="user">Объект UserModel для добавления.</param>
/// <exception cref="ArgumentException">Вызывается, когда данные не валидны или пользователь с таким ло-
гинном уже существует.</exception>
public static void Add(UserModel user)
{
    if (!DataValidate(user))
    {
        throw new ArgumentException("Данные не валидны.");
    }
    if (!UserValid(user))
    {
        throw new ArgumentException("Пользователь с таким логином уже существует.");
    }

    var newUser = new XElement("user",
        new XElement("login", user.Login),
        new XElement("password", user.Password),
        new XElement("role", user.Role)
    );
    _xmlDoc.Root.Add(newUser);
    using (var ms = new MemoryStream())
    {
        _xmlDoc.Save(ms);

        using (var file = new FileStream(_xmlFilePath, FileMode.Create, FileAccess.Write))
        {
            ms.WriteTo(file);
        }
    }
}
/// <summary>
/// Обновляет объект UserModel в файле XML.
/// </summary>
/// <param name="user">Объект UserModel для обновления.</param>
/// <exception cref="ArgumentException">Вызывается, когда данные не валидны или пользователь с таким ло-
гинном не существует.</exception>
public static void Update(UserModel user)
{
    if (!DataValidate(user))
    {
        throw new ArgumentException("Данные не валидны.");
    }
    if (UserValid(user))
    {
        throw new ArgumentException("Пользователя с таким логином не существует.");
    }

    XElement userToUpdate = _xmlDoc.Root.Elements("user")
        .SingleOrDefault(u => u.Element("login").Value == user.Login);

    if (userToUpdate != null)

```



```

    {
        userToUpdate.Element("password").Value = user.Password;
        userToUpdate.Element("role").Value = user.Role.ToString();
        _xmlDoc.Save(_xmlFilePath);
    }
}
// <summary>
/// Удаление объекта UserModel в файле XML.
/// </summary>
/// <param name="user">Объект UserModel для удаления.</param>
/// <exception cref="ArgumentException">Вызывается, когда пользователя не существует.</exception>
public static void Delete(UserModel user)
{
    if (UserValid(user))
    {
        throw new ArgumentException("Пользователя с таким логином не существует.");
    }

    XElement userToDelete = _xmlDoc.Root.Elements("user")
        .SingleOrDefault(u => u.Element("login").Value == user.Login);

    if (userToDelete != null)
    {
        userToDelete.Remove();
        _xmlDoc.Save(_xmlFilePath);
    }
}
// <summary>
/// Проверка уникальности пользователя.
/// </summary>
/// <param name="user">Объект UserModel для проверки валидации.</param>
private static bool UserValid(UserModel user)
{
    if (_xmlDoc.Root.Elements("user").Any(u => u.Element("login").Value == user.Login))
    {
        return false;
    }
    else return true;
}
// <summary>
/// Проверяет, что данные пользователя проходят валидацию.
/// </summary>
/// <param name="user">Пользователь которого нужно проверить.</param>
/// <returns>true, если данные проходят валидацию, false в противном случае.</returns>
private static bool DataValidate(UserModel user)
{
    bool isValid = true;

    var tmpSpectacle = new XElement("users",
        new XElement("user",
            new XElement("login", user.Login),
            new XElement("password", user.Password),
            new XElement("role", user.Role)
        ));

    var xdoc = new XDocument(tmpSpectacle);
    xdoc.Validate(_schemas, (o, e) =>
    {
        isValid = false;
    });
    return isValid;
}

```

```
}  
}
```

Листинг SpectacleServices.cs

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace App.Services  
{  
    public class SpectacleServices : ISpectacleServices<SpectacleModel>  
    {  
        public SpectacleServices()  
        {  
        }  
  
        public IEnumerable<SpectacleModel> ShowAllSpectacles()  
        {  
            IEnumerable<SpectacleModel> spectacles = SpectacleManager.GetAll();  
            return spectacles;  
        }  
        public SpectacleModel ShowSpectacle(DateTime date)  
        {  
            IEnumerable<SpectacleModel> spectacles = SpectacleManager.GetAll();  
            SpectacleModel spectacle = spectacles.FirstOrDefault(x => x.Date.ToString("d") == date.ToString("d"));  
  
            if (spectacle != null)  
            {  
                return spectacle;  
            }  
            else throw new ArgumentException($"Спектакль на дату `{date.ToString("d")}` не найден.");  
        }  
        public IEnumerable<SpectacleModel> ShowSpectacle(string genre)  
        {  
            IEnumerable<SpectacleModel> spectacles = SpectacleManager.GetAll();  
            IEnumerable<SpectacleModel> filteredSpectacles = spectacles.Where(x => x.Genre.ToLower() == genre.ToLower());  
            if (filteredSpectacles.Any())  
            {  
                return filteredSpectacles;  
            }  
            else  
            {  
                throw new ArgumentException($"Спектакли с жанром `{genre}` не найдены.");  
            }  
        }  
  
        public void AddNewSpectacle(string title, string author, string genre, DateTime date,  
            decimal vipPrise, decimal mediumPrice, decimal standartPrice)  
        {  
            SpectacleManager.Add(CreateSpectacleElement(title, author, genre, date, vipPrise, mediumPrice, standartPrice));  
        }  
  
        public void UpdateSpectacle(string newTitle, string newAuthor, string newGenre, DateTime date,  
            decimal newVipPrise, decimal newMediumPrice, decimal newStandartPrice)  
        {  
            SpectacleManager.Update(CreateSpectacleElement(newTitle, newAuthor, newGenre, date, newVipPrise, newMediumPrice, newStandartPrice));  
        }  
  
        public void AddGanre(string ganreName)  
        {  
            SpectacleManager.AddGenre(ganreName);  
        }  
    }  
}
```

```

    }
    public List<string> GetAllGenres()
    {
        return SpectacleManager.GetAllGenres();
    }
    public int GetGenreIdByName(string name)
    {
        return SpectacleManager.GetGenreIdByName(name);
    }
    public void DeleteSpectacle(DateTime date)
    {
        SpectacleModel spectacleToDelete = SpectacleManager.GetAll().FirstOrDefault(x => x.Date.Equals(date));
        if (spectacleToDelete != null)
        {
            SpectacleManager.Delete(spectacleToDelete);
        }
        else throw new ArgumentException($"Спектакль на дату `{date.ToString("d")}` не найден.");
    }
    private SpectacleModel CreateSpectacleElement(string title, string author, string genre, DateTime date,
        decimal vipPrice, decimal mediumPrice, decimal standardPrice)
    {
        if (string.IsNullOrEmpty(title))
        {
            throw new ArgumentException("Поле `Название` не должно быть пустым или состоять только из пробелов");
        }

        if (string.IsNullOrEmpty(author))
        {
            throw new ArgumentException("Поле `Автор` не должно быть пустым или состоять только из пробелов");
        }

        if (string.IsNullOrEmpty(genre))
        {
            throw new ArgumentException("Поле `Жанр` не должно быть пустым или состоять только из пробелов");
        }

        if (vipPrice <= 0 || mediumPrice <= 0 || standardPrice <= 0)
        {
            throw new ArgumentException("Цена должна быть положительной");
        }

        Dictionary<Categorias, decimal> thisCategories = new Dictionary<Categorias, decimal>()
        {
            { Categorias.VIP, vipPrice },
            { Categorias.Medium, mediumPrice },
            { Categorias.Standart, standardPrice }
        };

        return new SpectacleModel
        {
            Title = title,
            Author = author,
            Genre = genre,
            Date = date,
            Categories = thisCategories,
            FreePlace = 25
        };
    }
}
}

```

Листинг TicketServices.cs

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace App.Services
{
    public class TicketServices : ITicketServices<TicketModel>
    {
        private static SpectacleServices _spectacleServices;
        public TicketServices()
        {
            _spectacleServices = new SpectacleServices();
        }
        public void AddTicket(string userName, SpectacleModel spectacleModel, Categoria category)
        {
            if (spectacleModel.FreePlace > 0)
            {
                spectacleModel.FreePlace -= 1;
                SpectacleManager.Update(spectacleModel);
                TicketManager.Add(CreateTicketElement(userName, spectacleModel, category));
            }
            else throw new ArgumentException($"На спектакль {spectacleModel.Title} нет свободных мест.");
        }

        public void DeleteTicket(int id)
        {
            SpectacleModel spectacle = _spectacleServices.ShowSpectacle(GetTicket(id).Date);
            spectacle.FreePlace += 1;
            SpectacleManager.Update(spectacle);

            TicketManager.Delete(id);
        }

        public IEnumerable<TicketModel> GetTicket()
        {
            return TicketManager.GetAll();
        }

        public IEnumerable<TicketModel> GetTicket(string owner)
        {
            IEnumerable<TicketModel> tickets = TicketManager.GetAll().Where(x => x.Owner == owner);
            if (tickets != null)
            {
                return tickets;
            }
            else throw new ArgumentException($"Билет пользователя {owner} не найден.");
        }

        public TicketModel GetTicket(int id)
        {
            IEnumerable<TicketModel> tickets = TicketManager.GetAll();
            TicketModel ticket = tickets.FirstOrDefault(x => x.Id == id);
            if (ticket != null)
            {
                return ticket;
            }
            else throw new ArgumentException($"Билет на с id={id} не найден.");
        }
    }
}

```

```

private TicketModel CreateTicketElement(string userName, SpectacleModel spectacleModel, Categoria category)
{
    return new TicketModel
    {
        Owner = userName,
        Date = spectacleModel.Date,
        Title = spectacleModel.Title,
        Category = category,
        Price = spectacleModel.Categories[category]
    };
}
}
}

```

Листинг UserServices.cs

```

using System;
using System.Collections.Generic;
using System.Linq;

namespace App.Services
{
    public class UserServices : IUserServices<UserModel>
    {
        public UserServices()
        {
        }

        public IEnumerable<UserModel> GetUser()
        {
            return UserManager.GetAll();
        }

        public UserModel GetUser(string userLogin)
        {
            UserModel user = UserManager.GetAll().FirstOrDefault(x => x.Login == userLogin);
            if (user != null)
            {
                return user;
            }
            throw new ArgumentException($"Пользователь '{userLogin}' не найден.");
        }

        public void AddUser(string login, string password, Role role)
        {
            UserManager.Add(CreateUserElement(login, password, role));
        }

        public void UpdateUserByName(string name, string newPassword, Role newRole)
        {
            UserModel thisUser = UserManager.GetAll().FirstOrDefault(x => x.Login == name);
            if (thisUser != null)
            {
                UserManager.Update(CreateUserElement(name, newPassword, newRole));
            }
            else throw new ArgumentException($"Пользователь с логином '{thisUser.Login}' не найден.");
        }

        public void DeleteUser(string name)
        {
        }
    }
}

```

```

        UserModel userToDelete = UserManager.GetAll().FirstOrDefault(x => x.Login == name);
        if (userToDelete != null)
        {
            UserManager.Delete(userToDelete);
        }
        else throw new ArgumentException($"Пользователь с логином '{userToDelete.Login}' не найден.");
    }

    private UserModel CreateUserElement(string login, string password, Role role)
    {
        return new UserModel
        {
            Login = login,
            Password = password,
            Role = role
        };
    }
}

```

Листинг User.cs

```

using App.Services;
using System;
using System.Collections.Generic;
public abstract class User : UserModel
{
    public readonly SpectacleServices _spectacleService;

    public User(SpectacleServices spectacleService)
    {
        _spectacleService = spectacleService;
    }

    public IEnumerable<SpectacleModel> ViewSpectacle()
    {
        return _spectacleService.ShowAllSpectacles();
    }
    public IEnumerable<SpectacleModel> ViewSpectacle(string genre)
    {
        return _spectacleService.ShowSpectacle(genre);
    }
    public SpectacleModel ViewSpectacle(DateTime date)
    {
        return _spectacleService.ShowSpectacle(date);
    }
}

```

Листинг Registered.cs

```

using App.Services;
using System;
using System.Collections.Generic;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

public class Registered : User
{
    private TicketServices _ticketServices;
}

```

```

private UserServices _userServices;

public Registered(SpectacleServices spectacleService, UserServices userServices, TicketServices ticketServices) :
base(spectacleService)
{
    _ticketServices = ticketServices;
    _userServices = userServices;
}

public void UpdateUser(string userName, string newPassword)
{
    _userServices.UpdateUserByName(userName, newPassword, Role.registered);
}

public IEnumerable<TicketModel> GetTicket()
{
    return _ticketServices.GetTicket(this.Login);
}
public IEnumerable<TicketModel> GetTicket(string owner)
{
    return _ticketServices.GetTicket(owner);
}
public TicketModel GetTicket(int id)
{
    return _ticketServices.GetTicket(id);
}

public void AddTicket(string userName, DateTime spectacleDate, Categorias category)
{
    _ticketServices.AddTicket(userName, GetThisSpectacle(spectacleDate), category);
}

public void DeletTicket(int id)
{
    _ticketServices.DeletTicket(id);
}

private SpectacleModel GetThisSpectacle(DateTime spectacleDate)
{
    return _spectacleService.ShowSpectacle(spectacleDate);
}
}

```

Листинг Guest.cs

```

using App.Services;
using System;

public class Guest : User
{
    public Guest(SpectacleServices spectacleService) : base(spectacleService)
    {
    }
}

```

Листинг Administrator.cs

```

using App.Services;
using System;

```

```

using System.Collections.Generic;

public class Administrator : User
{
    private UserServices _userServices;
    private TicketServices _ticketServices;

    public Administrator(SpectacleServices spectacleService, UserServices userServices, TicketServices ticketServices) :
    base(spectacleService)
    {
        _userServices = userServices;
        _ticketServices = ticketServices;
    }

    public void AddSpectacle(string title, string author, string genre, DateTime date,
        decimal vipPrise, decimal mediumPrice, decimal standartPrice)
    {
        _spectacleService.AddNewSpectacle(title, author, genre, date, vipPrise, mediumPrice, standartPrice);
    }

    public void UpdateSpectacle(string newTitle, string newAuthor, string newGenre, DateTime date,
        decimal newVipPrise, decimal newMediumPrice, decimal newStandartPrice)
    {
        _spectacleService.UpdateSpectacle(newTitle, newAuthor, newGenre, date, newVipPrise, newMediumPrice,
        newStandartPrice);
    }

    public void DeleteSpectacle(DateTime date)
    {
        _spectacleService.DeleteSpectacle(date);
    }

    public void AddGanre(string name)
    {
        _spectacleService.AddGanre(name);
    }
    public List<string> GetAllGenres()
    {
        return _spectacleService.GetAllGenres();
    }
    public int GetGenreIdByName(string name)
    {
        return _spectacleService.GetGenreIdByName(name);
    }

    public void AddeUser(string login, string password, Role role)
    {
        _userServices.AddUser(login, password, role);
    }

    public IEnumerable<UserModel> GetUser()
    {
        return _userServices.GetUser();
    }

    public UserModel GetUser(string login)
    {
        return _userServices.GetUser(login);
    }

    public void UpdateUser(string login, string password, Role role)
    {

```



```

        _userServices.UpdateUserByName(login, password, role);
    }

    public void DeleteUser(string login)
    {
        _userServices.DeleteUser(login);
    }

    public IEnumerable<TicketModel> GetTicket(string owner)
    {
        return _ticketServices.GetTicket(owner);
    }
    public IEnumerable<TicketModel> GetTicket()
    {
        return _ticketServices.GetTicket();
    }

    public TicketModel GetTicket(int id)
    {
        return _ticketServices.GetTicket(id);
    }

    public void AddTicket(string userName, DateTime spectacleDate, Categorias category)
    {
        _ticketServices.AddTicket(userName, GetThisSpectacle(spectacleDate), category);
    }

    public void DeletTicket(int id)
    {
        _ticketServices.DeletTicket(id);
    }

    private UserModel GetThisUser(string userName)
    {
        return _userServices.GetUser(userName);
    }
    private SpectacleModel GetThisSpectacle(DateTime spectacleDate)
    {
        return _spectacleService.ShowSpectacle(spectacleDate);
    }
}

```

Листинг TicketServicesXmlLoggingDecorator.cs

```

using App.Services;
using System;
using System.Collections.Generic;
using System.IO;
using System.Reflection;
using System.Xml.Linq;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

namespace App
{
    public class TicketServicesXmlLoggingDecorator : TicketServices
    {
        private readonly TicketServices _ticketServices;
        private readonly XDocument _log;
        private readonly string _xmlFilePath = "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\Log.xml";

        public TicketServicesXmlLoggingDecorator(TicketServices inner)
    }
}

```

```

    {
        _ticketServices = inner;
        _log = XDocument.Load(_xmlFilePath);
    }

    public void AddTicket(string userName, SpectacleModel spectacleModel, Categorias category)
    {
        _ticketServices.AddTicket(userName, spectacleModel, category);

        LogOperation("Покупка билета", userName, spectacleModel, category);
    }

    public void DeletTicket(int id)
    {
        LogOperation("Возврат билета", _ticketServices.GetTicket(id));

        _ticketServices.DeletTicket(id);
    }

    public IEnumerable<TicketModel> GetTicket()
    {
        return _ticketServices.GetTicket();
    }

    public IEnumerable<TicketModel> GetTicket(string owner)
    {
        return _ticketServices.GetTicket(owner);
    }

    public TicketModel GetTicket(int id)
    {
        return _ticketServices.GetTicket(id);
    }

    private void LogOperation(string methodName, string userName, SpectacleModel spectacleModel, Categorias category)
    {
        var logEntry = new XElement("Operation",
            new XElement("Method", methodName),
            new XElement("Owner", userName),
            new XElement("SpectacleName", spectacleModel.Title),
            new XElement("SpectacleDate", spectacleModel.Date.ToString("d")),
            new XElement("Category", category),
            new XElement("Price", spectacleModel.Categories[category]),
            new XElement("Timestamp", DateTime.Now.ToString("o"))
        );
        _log.Root.Add(logEntry);
        _log.Save("C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\log.xml");
    }

    private void LogOperation(string methodName, TicketModel ticketModel)
    {
        var logEntry = new XElement("Operation",
            new XElement("Method", methodName),
            new XElement("Owner", ticketModel.Owner),
            new XElement("SpectacleName", ticketModel.Title),
            new XElement("SpectacleDate", ticketModel.Date.ToString("d")),
            new XElement("Category", ticketModel.Category),
            new XElement("Price", ticketModel.Price),
            new XElement("Timestamp", DateTime.Now.ToString("o"))
        );
    }

```

```

        _log.Root.Add(logEntry);
        _log.Save("C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\log.xml");
    }
}
}

```

Листинг XmlToExcel.cs

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using Excel = Microsoft.Office.Interop.Excel;

namespace App.Logs
{
    public static class XmlToExcel
    {
        static private string _filePath = "C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\Log.xml";
        static private XDocument _doc = XDocument.Load(_filePath);

        public static void CreateTable()
        {
            Excel.Application xlApp = new Excel.Application();
            Excel.Workbook xlWorkbook = xlApp.Workbooks.Add();
            Excel._Worksheet xlWorksheet = xlWorkbook.Sheets[1];
            Excel.Range xlRange = xlWorksheet.UsedRange;

            xlWorksheet.Cells[1, 1] = "Method";
            xlWorksheet.Cells[1, 2] = "Owner";
            xlWorksheet.Cells[1, 3] = "SpectacleName";
            xlWorksheet.Cells[1, 4] = "SpectacleDate";
            xlWorksheet.Cells[1, 5] = "Category";
            xlWorksheet.Cells[1, 6] = "Price";
            xlWorksheet.Cells[1, 7] = "Timestamp";

            int row = 2;
            foreach (XElement operation in _doc.Descendants("Operation"))
            {
                string method = operation.Element("Method").Value;
                string owner = operation.Element("Owner").Value;
                string spectacleName = operation.Element("SpectacleName").Value;
                string spectacleDate = operation.Element("SpectacleDate").Value;
                string category = operation.Element("Category").Value;
                string price = operation.Element("Price").Value;
                string timestamp = operation.Element("Timestamp").Value;

                xlWorksheet.Cells[row, 1] = method;
                xlWorksheet.Cells[row, 2] = owner;
                xlWorksheet.Cells[row, 3] = spectacleName;
                xlWorksheet.Cells[row, 4] = spectacleDate;
                xlWorksheet.Cells[row, 5] = category;
                xlWorksheet.Cells[row, 6] = price;
                xlWorksheet.Cells[row, 7] = timestamp;

                if (method == "Покупка билета")
                {
                    xlRange = xlWorksheet.Range[xlWorksheet.Cells[row, 1], xlWorksheet.Cells[row, 10]];
                    xlRange.Interior.Color = System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Green);
                }
            }
        }
    }
}

```

```

    }
    else if (method == "Возврат билета")
    {
        xlRange = xlWorksheet.Range[xlWorksheet.Cells[row, 1], xlWorksheet.Cells[row, 10]];
        xlRange.Interior.Color = System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Red);
    }

    row++;
}
string resultLog = $"C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\Logs\\Log.xlsx";
xlWorkbook.SaveAs(new FileInfo(resultLog));

xlWorkbook.Close();
xlApp.Quit();

Process.Start(resultLog);
}
}
}

```

Листинг LoginForm.cs

```

using App.Services;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace App
{
    public partial class LoginForm : Form
    {
        private MainForm _maiForm;
        private RegistrForm _regForm;
        private UserServices _userServices;

        public LoginForm(MainForm mainForm)
        {
            InitializeComponent();
            _userServices = new UserServices();
            _maiForm = mainForm;
            _regForm = new RegistrForm();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            LogIn();
        }

        private void registerButton_Click(object sender, EventArgs e)
        {
            _regForm.ShowDialog();
        }

        private void skipButton_Click(object sender, EventArgs e)
        {
            UserModel nullUser = new UserModel() { Login = null, Password = null, Role = Role.guest };
            _maiForm.CreateUser(nullUser);
            this.Close();
        }
    }
}

```

```

private void RegisterForm_Load(object sender, EventArgs e)
{
    userPasswordForm.PasswordChar = '*';
    exptLable.Text = "";

}
private void LogIn()
{
    string userName = userNameForm.Text;
    string userPassword = userPasswordForm.Text;

    if (userName == "" || userPassword == "")
    {
        exptLable.Text = "Неправильно введен логин или пароль.";
        userNameForm.Text = userPasswordForm.Text = "";
    }
    else
    {
        try
        {
            UserModel user = _userServices.GetUser(userName);

            if (user.Password == userPassword)
            {
                _maiForm.CreateUser(user);
                userNameForm.Text = userPasswordForm.Text = "";
                this.Close();
            }
            else
            {
                exptLable.Text = "Неверный пароль.";
                userPasswordForm.Text = "";
            }
        }
        catch (ArgumentException exp)
        {
            exptLable.Text = exp.Message;
            userNameForm.Text = userPasswordForm.Text = "";
        }
    }
}
}
}

```

MainForm

```

using App.Services;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.NetworkInformation;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using System.IO;
using App.Logs;

```

```

namespace App
{
    public partial class MainForm : Form
    {
        public SpectacleServices spectacleServices = new SpectacleServices();
        public UserServices userServices = new UserServices();
        public TicketServices ticketServices = new TicketServices();

        public LoginForm loginForm;
        public TicketBuyForm ticketBuyForm;
        public UserTicketsForm userTicketsForm;

        public dynamic _user;

        public MainForm()
        {
            InitializeComponent();

            loginForm = new LoginForm(this);
            ticketBuyForm = new TicketBuyForm();
            userTicketsForm = new UserTicketsForm();

            spectacleGridView.ReadOnly = true;
            spectacleGridView.Columns.Add("Название", "Название");
            spectacleGridView.Columns.Add("Свободных мест", "Свободных мест");
            spectacleGridView.Columns.Add("Жанр", "Жанр");
            spectacleGridView.Columns.Add("Автор", "Автор");
            spectacleGridView.Columns.Add("Дата", "Дата");
            spectacleGridView.Columns.Add("VIP", "VIP");
            spectacleGridView.Columns.Add("Ложа", "Ложа");
            spectacleGridView.Columns.Add("Галерея", "Галерея");

            userGridView.ReadOnly = true;
            userGridView.Columns.Add("Имя", "Имя");
            userGridView.Columns.Add("Пароль", "Пароль");
            userGridView.Columns.Add("Роль", "Роль");

            userTicketsListBox.ReadOnly = true;

            RemoveAdminPanel();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            loginForm.ShowDialog();
            spectacleGridView.CellClick += dataGridView1_CellClick;

            spectacleGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
            spectacleGridView.RowHeadersWidthSizeMode = DataGridViewRowHeadersWidthSizeMode.AutoSizeToAll-
Headers;

            userInfo.MouseDown += new MouseEventHandler(InitializeContextMenu);
            userRolePictureBox.MouseDown += new MouseEventHandler(InitializeContextMenu);
        }
    }
}

```

```

public void CreateUser(UserModel user)
{
    UserFactory userFactory = new UserFactory(userServices, spectacleServices, ticketServices);

    _user = userFactory.CreateUser(user);

    ShowWindowToUser(_user);
    ShowSpectacle();

}

public void ShowWindowToUser(Administrator user)
{
    userInfo.Visible = true;
    userInfo.Text = $"Пользователь: {_user.Login}";
    RegistrBut.Text = "Выйти";
    addSpectacle.Visible = true;
    permissionTabControl.TabPages[0].Text = "Управление спектаклями";
    userRolePictureBox.Image = Image.FromFile("C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\img\\icons\\ad-
min.png");

    ShowUsers();
    ShowAdminPanel();
}

public void ShowWindowToUser(Registered user)
{
    userInfo.Visible = true;
    userInfo.Text = $"Пользователь: {_user.Login}";
    RegistrBut.Text = "Выйти";
    addSpectacle.Visible = false;
    permissionTabControl.TabPages[0].Text = "Выбор спектаклей";
    RemoveAdminPanel();
    userRolePictureBox.Image = Image.FromFile("C:\\Users\\Evgeni\\Desktop\\Course-
Work\\App\\img\\icons\\user.png");
}

public void ShowWindowToUser(Guest user)
{
    userInfo.Visible = false;
    RegistrBut.Text = "Регистрация";
    addSpectacle.Visible = false;
    permissionTabControl.TabPages[0].Text = "Выбор спектаклей";
    RemoveAdminPanel();
    userRolePictureBox.Image = null;

}

private void InitializeContextMenu(object sender, MouseEventArgs e)
{
    if (_user is Registered || _user is Administrator)
    {
        userInfoContextMenu.Items.Clear();

        userInfoContextMenu.Items.Add($"Имя: {_user.Login}");

        if (_user is Registered)
        {
            userInfoContextMenu.Items.Add($"Роль: Пользователь");
        }
    }
}

```

```

        ToolStripButton showUserTicketsButton = new ToolStripButton();
        showUserTicketsButton.Text = "Мои билеты";
        showUserTicketsButton.Click += new EventHandler(ticketMenuItem_Click);

        showUserTicketsButton.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;

        userInfoContextMenu.Items.Add(showUserTicketsButton);
    }
    else if (_user is Administrator)
    {
        userInfoContextMenu.Items.Add($"Роль: Администратор");
        ToolStripButton showUserTicketsButton = new ToolStripButton();
        showUserTicketsButton.Text = "Сформировать отчет";
        showUserTicketsButton.Click += new EventHandler(ticketLog_Click);

        showUserTicketsButton.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;

        userInfoContextMenu.Items.Add(showUserTicketsButton);
    }

    userInfoContextMenu.Name = "Информация о пользователе";

    userInfoContextMenu.Show(userInfo, e.Location);
}
}
private void ticketMenuItem_Click(object sender, EventArgs e)
{
    if (_user is Registered)
    {
        userTicketsForm.ShowDialog(_user);
        ShowSpectacle();
    }
}
private void ticketLog_Click(object sender, EventArgs e)
{
    if (_user is Administrator)
    {
        DialogResult result = MessageBox.Show("Сформировать статистический отчёт о продажах билетов?",
"Формирование отчета",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question);

        if (result == DialogResult.Yes)
        {
            XmlToExcel.CreateTable();
        }
        else if (result == DialogResult.No)
        {
        }
    }
}

public void ShowSpectacle()
{
    if (_user is Administrator || _user is Registered || _user is Guest)
    {
        spectacleGridView.Rows.Clear();

        IEnumerable<SpectacleModel> spectacles = _user.ViewSpectacle();

        foreach (SpectacleModel spectacle in spectacles)

```



```

        {
            spectacleGridView.Rows.Add(spectacle.Title, spectacle.FreePlace, spectacle.Genre, spectacle.Author,
spectacle.Date.ToString("d"),
                $"{spectacle.Categories[Categories.VIP]}", $"{spectacle.Categories[Categories.Medium]}",
                $"{spectacle.Categories[Categories.Standart]}");
            mainExpString.Text = "";
        }
    }

}

public void ShowSpectacle(string ganre)
{
    if (_user is Administrator || _user is Registered)
    {
        spectacleGridView.Rows.Clear();

        try
        {
            IEnumerable<SpectacleModel> spectacles = _user.ViewSpectacle(ganre);

            foreach (SpectacleModel spectacle in spectacles)
            {
                spectacleGridView.Rows.Add(spectacle.Title, spectacle.FreePlace, spectacle.Genre, spectacle.Author,
spectacle.Date.ToString("d"),
                    $"{spectacle.Categories[Categories.VIP]}", $"{spectacle.Categories[Categories.Me-
dium]}",
                    $"{spectacle.Categories[Categories.Standart]}");

                mainExpString.Text = "";
            }
        }
        catch (ArgumentException e)
        {
            mainExpString.Text = e.Message;
        }
    }
}

public void ShowSpectacle(DateTime date)
{
    if (_user is Administrator || _user is Registered)
    {
        spectacleGridView.Rows.Clear();

        try
        {
            SpectacleModel spectacle = _user.ViewSpectacle(date);

            spectacleGridView.Rows.Add(spectacle.Title, spectacle.Genre, spectacle.Author, specta-
cle.Date.ToString("d"),
                $"{spectacle.Categories[Categories.VIP]}", $"{spectacle.Categories[Categories.Me-
dium]}",
                $"{spectacle.Categories[Categories.Standart]}");

            mainExpString.Text = "";
        }
        catch (ArgumentException e)
        {

```

```

        mainExpString.Text = e.Message;
    }
}
}
private void ShowUsers()
{
    if (_user is Administrator)
    {
        userGridView.Rows.Clear();

        IEnumerable<UserModel> users = _user.GetUser();

        foreach (UserModel user in users)
        {
            userGridView.Rows.Add(user.Login, user.Password, user.Role == Role.registered ? "Пользователь" :
"Администратор");
        }
    }
}
private void RemoveAdminPanel()
{
    if (permissionTabControl.TabPages.Count == 2) {
        permissionTabControl.TabPages.Remove(AdminTabPanel);
    }

}
private void ShowAdminPanel() {
    if (permissionTabControl.TabPages.Count == 1)
    {
        permissionTabControl.TabPages.Add(AdminTabPanel);
    }
}

private void searchSpectacleByGenre_TextChanged(object sender, EventArgs e)
{
    if (!string.IsNullOrEmpty(searchSpectacleByGenre.Text))
    {
        ShowSpectacle(searchSpectacleByGenre.Text);
    }
    else ShowSpectacle();
}

private void searchSpectacleByDate_ValueChanged(object sender, EventArgs e)
{
    DateTime selectedDate = searchSpectacleByDate.Value;
    ShowSpectacle(selectedDate);
}

private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
{
}

private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (_user is Registered || _user is Administrator)
    {
        if (e.RowIndex >= 0)
        {
            DataGridViewRow row = spectacleGridView.Rows[e.RowIndex];

            if (row.Cells[3].Value != null)

```

```

        {
            SpectacleModel thisSpectacle = _user.ViewSpectacle(DateTime.Parse(row.Cells[4].Value.ToString()));
            ticketBuyForm.ShowDialog(thisSpectacle, _user);
        }
        else ticketBuyForm.ShowDialog(_user);

    }
    else if (_user is Administrator)
    {
        ticketBuyForm.ShowDialog(_user);
    }
    ShowSpectacle();
}
}

private void userGridView_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (_user is Administrator)
    {
        if (e.RowIndex >= 0)
        {
            DataGridViewRow row = userGridView.Rows[e.RowIndex];

            if (row.Cells[0].Value != null)
            {
                UserModel thisUser = _user.GetUser(row.Cells[0].Value.ToString());
                userLoginBox.Text = thisUser.Login;
                userPasswordBox.Text = thisUser.Password;
                userRoleBox.Text = thisUser.Role == Role.registered ? "Пользователь" : "Администратор";
                delUserButton.Visible = true;

                ShowTicketsList(thisUser.Login);
            }

        }
        else if (_user is Administrator)
        {
            userLoginBox.Text = "";
            userPasswordBox.Text = "";
            userRoleBox.Text = "";
            delUserButton.Visible = false;
        }
    }
}

private void addSpectacle_Click(object sender, EventArgs e)
{
    if (_user is Administrator)
    {
        ticketBuyForm.ShowDialog(_user);
        ShowSpectacle();
    }
}

private void RegistrBut_Click(object sender, EventArgs e)
{
    loginForm.ShowDialog();
}

private void userLoginBox_TextChanged(object sender, EventArgs e)
{
    if (_user is Administrator)

```

```

{
    try
    {
        UserModel newUser = _user.GetUser(userLoginBox.Text);
        userPasswordBox.Text = newUser.Password;
        userRoleBox.Text = newUser.Role == Role.registered ? "Пользователь" : "Администратор";
        addUserButton.Text = "Изменить";
        delUserButton.Visible = true;

        ShowTicketsList(newUser.Login);

    }
    catch (ArgumentException userNull)
    {
        userPasswordBox.Text = "";
        userRoleBox.Text = "";
        addUserButton.Text = "Добавить";

        delUserButton.Visible = false;
        userTicketsLable.Visible = false;
        userTicketsListBox.Clear();
    }
}
}
private void ShowTicketsList(string userLogin)
{
    if (_user is Administrator || _user is Registered)
    {
        userTicketsLable.Visible = true;
        userTicketsLable.Text = $"Билеты пользователя `{userLogin}`:";

        userTicketsListBox.Clear();

        IEnumerable<TicketModel> tickets = _user.GetTicket(userLogin);
        foreach (TicketModel ticket in tickets)
        {
            userTicketsListBox.Text += ($"Название: {ticket.Title}\nДата: {ticket.Date.ToString("d")}\nКатегория: {ticket.Category}\nЦена: {ticket.Price} р.\n\n");
        }
    }
}

private void AddUserButton_Click(object sender, EventArgs e)
{
    if (_user is Administrator)
    {
        if (addUserButton.Text == "Добавить")
        {
            _user.AddUser(userLoginBox.Text, userPasswordBox.Text, userRoleBox.Text == "Пользователь" ?
Role.registered : Role.admin);
            ShowUsers();
        }
        else if (addUserButton.Text == "Изменить")
        {
            _user.UpdateUser(userLoginBox.Text, userPasswordBox.Text, userRoleBox.Text == "Пользователь" ?
Role.registered : Role.admin);
            ShowUsers();
        }
    }
}
}

```

```

    }
    private void delUserButton_Click(object sender, EventArgs e)
    {
        if (_user is Administrator)
        {
            try
            {
                if (userLoginBox.Text != _user.Login)
                {
                    _user.DeleteUser(userLoginBox.Text);
                    ShowUsers();
                }
                else MessageBox.Show("Невозможно удалить данного пользователя!");
            }
            catch (ArgumentException delUserExp)
            {
                MessageBox.Show(delUserExp.Message);
            }
        }
    }
    private void changeTicketsButton_Click(object sender, EventArgs e)
    {
        if (_user is Administrator || _user is Registered)
        {
            userTicketsForm.ShowDialog(_user, userLoginBox.Text);
            ShowTicketsList(userLoginBox.Text);
            ShowSpectacle();
        }
    }
}
}

```

Листинг RegistrForm.cs

```

using App.Services;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace App
{
    public partial class RegistrForm : Form
    {
        private UserServices userServices;
        public RegistrForm()
        {
            InitializeComponent();
            userServices = new UserServices();
        }

        private void logInButtog_Click(object sender, EventArgs e)
        {

```

```

        if(UserDataValid(newUserNameForm.Text, newUserPasswordForm.Text))
        {
            Registr();
        }
    }

    private void RegistrForm_Load(object sender, EventArgs e)
    {
        exptRegisterLable.Text = "";
        newUserNameForm.Text = "";
        newUserPasswordForm.PasswordChar = '*';
        newUserPasswordForm.Text = "";
    }

    private void Registr()
    {
        try
        {
            userServices.AddUser(newUserNameForm.Text, newUserPasswordForm.Text, Role.registered);
            this.Close();

        }
        catch (ArgumentException exp)
        {
            exptRegisterLable.Text = exp.Message;
            newUserNameForm.Text = newUserPasswordForm.Text = "";
        }
    }

    public bool UserDataValid(string username, string password)
    {
        if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
        {
            exptRegisterLable.Text = "Заполните поле логина и пароля";
            return false;
        }

        string usernameRegex = @"^.{4,}$";
        string passwordRegex = @"^.{4,}$";
        if (!Regex.IsMatch(username, usernameRegex) || !Regex.IsMatch(password, passwordRegex))
        {
            exptRegisterLable.Text = "Длина логина и пароля должна быть не менее 4 символов";
            return false;
        }

        if (username.Contains(" ") || password.Contains(" "))
        {
            exptRegisterLable.Text = "Логин и пароль не должны содержать пробелов";
            return false;
        }

        string invalidCharsRegex = @"[;']";
        if (Regex.IsMatch(username, invalidCharsRegex) || Regex.IsMatch(password, invalidCharsRegex))
        {
            exptRegisterLable.Text = "Логин и пароль содержат недопустимые символы `;'";
            return false;
        }
        exptRegisterLable.Text = "";
        return true;
    }
}

```

```
}
```

Листинг TicketBuyForm.cs

```
using App.Services;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Window;

namespace App
{
    public partial class TicketBuyForm : Form
    {
        private SpectacleModel _spectacleModel;
        private TicketServices _ticketServices;
        private TicketServicesXmlLoggingDecorator _ticketServicesLog;
        private dynamic _owner;

        public TicketBuyForm()
        {
            _ticketServices = new TicketServices();
            _ticketServicesLog = new TicketServicesXmlLoggingDecorator(_ticketServices);
            InitializeComponent();
        }

        private void SpectacleTitle_Click(object sender, EventArgs e)
        {
        }

        private void groupBox1_Enter(object sender, EventArgs e)
        {
        }

        private void TicketBuyForm_Load(object sender, EventArgs e)
        {
            this.ClientSize = new Size(311, 334);
            AdminBuyPanel.Location = new Point(0, 0);
            RegisterBuyPanel.Location = new Point(0, 0);
        }

        public new DialogResult ShowDialog(SpectacleModel thisSpectacle, Administrator user)
        {
            ClearAdminFields();

            AdminBuyPanel.Visible = true;
            RegisterBuyPanel.Visible = false;

            _spectacleModel = thisSpectacle;
            _owner = user;

            newSpectacleName.Text = _spectacleModel.Title;
        }
    }
}
```

```

newAuthorName.Text = _spectacleModel.Author;
GanreBox.Text = _spectacleModel.Genre;
newDateName.Value = _spectacleModel.Date;

GanreBox.Items.Clear();
GanreBox.Items.AddRange(_owner.GetAllGenres().ToArray());

newVIPPrice.Value = _spectacleModel.Categories[Categories.VIP];
newMediumPrice.Value = _spectacleModel.Categories[Categories.Medium];
newStandartPrice.Value = _spectacleModel.Categories[Categories.Standart];

return base.ShowDialog();
}
public new DialogResult ShowDialog(Administrator user)
{
    ClearAdminFields();

    AdminBuyPanel.Visible = true;
    RegisterBuyPanel.Visible = false;

    _owner = user;

    GanreBox.Items.Clear();
    GanreBox.Items.AddRange(_owner.GetAllGenres().ToArray());

    changeTiket.Text = "Добавить";
    changeTiket.Location = new Point((this.ClientSize.Width - changeTiket.Width) / 2, 297);
    delSpectacle.Visible = false;

    return base.ShowDialog();
}
public new DialogResult ShowDialog(SpectacleModel thisSpectacle, Registered user)
{
    ClearUserFields();

    AdminBuyPanel.Visible = false;
    RegisterBuyPanel.Visible = true;

    _spectacleModel = thisSpectacle;
    _owner = user;

    SpectacleTitle.Text = _spectacleModel.Title;
    SpectacleAuthor.Text = $"Автор: {_spectacleModel.Author}";
    SpectacleGenre.Text = $"Жанр: {_spectacleModel.Genre}";
    SpectacleDate.Text = $"Дата: {_spectacleModel.Date.ToString("d")}";

    CategoriesRadio1.Text = $"VIP - {_spectacleModel.Categories[Categories.VIP]}";
    CategoriesRadio2.Text = $"Люжа - {_spectacleModel.Categories[Categories.Medium]}";
    CategoriesRadio3.Text = $"Галерея - {_spectacleModel.Categories[Categories.Standart]}";

    return base.ShowDialog();
}
public new void ShowDialog(SpectacleModel thisSpectacle, Guest user)
{
}

public new void ShowDialog(User user)
{
}

public new void ShowDialog(SpectacleModel thisSpectacle, User user)
{
}

```



```

    }
    private void ClearUserFields()
    {
        SpectacleTitle.Text = string.Empty;
        SpectacleAuthor.Text = string.Empty;
        SpectacleGenre.Text = string.Empty;
        SpectacleDate.Text = string.Empty;

        CategoriesRadio1.Text = string.Empty;
        CategoriesRadio2.Text = string.Empty;
        CategoriesRadio3.Text = string.Empty;
    }
    private void ClearAdminFields()
    {
        newSpectacleName.Text = string.Empty;
        newAuthorName.Text = string.Empty;
        GanreBox.Text = string.Empty;
        newDateName.Value = DateTime.Today;

        newVIPPrice.Value = 0;
        newMediumPrice.Value = 0;
        newStandartPrice.Value = 0;
    }

    private void BuyTicket_Click(object sender, EventArgs e)
    {
        try{
            Categorias category = new Categorias();
            if (CategoriesRadio1.Checked) category = Categorias.VIP;

            else if (CategoriesRadio2.Checked) category = Categorias.Medium;

            else if (CategoriesRadio3.Checked) category = Categorias.Medium;

            else throw new ArgumentException("Ни одна категория не выбрана");

            _ticketServicesLog.AddTicket(_owner.Login, _spectacleModel, category);
            this.Close();
        }
        catch(ArgumentException exp)
        {
            MessageBox.Show(exp.Message);
        }
    }

    private void changeTiket_Click(object sender, EventArgs e)
    {
        try
        {
            _owner.GetGenreIdByName(GanreBox.Text);
        }
        catch (ArgumentException exp)
        {
            DialogResult addGanreMessage = MessageBox.Show($"{exp.Message}\nДобавить жанр?",
"Подтверждение", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            if (addGanreMessage == DialogResult.Yes)
            {

```

```

        _owner.AddGanre(GanreBox.Text);
    }
    return;
}
try {
    if (changeTiket.Text == "Изменить")
    {
        var q = new DateName.Value;
        _owner.UpdateSpectacle(new SpectacleName.Text, new AuthorName.Text, GanreBox.Text, new Date-
Name.Value,
                                new VIPPrice.Value, new MediumPrice.Value, new StandartPrice.Value);
    }
    else if (changeTiket.Text == "Добавить")
    {
        _owner.AddSpectacle(new SpectacleName.Text, new AuthorName.Text, GanreBox.Text, new Date-
Name.Value,
                                new VIPPrice.Value, new MediumPrice.Value, new StandartPrice.Value);
    }
    this.Close();
}
catch (ArgumentException exp)
{
    MessageBox.Show(exp.Message);
}
}
private void GanreBox_SelectedIndexChanged(object sender, EventArgs e)
{
}
private void delSpectacle_Click(object sender, EventArgs e)
{
    try
    {
        _owner.DeleteSpectacle(new DateName.Value);
        this.Close();
    }
    catch (ArgumentException exp)
    {
        MessageBox.Show(exp.Message);
    }
}
private void newDateName_ValueChanged(object sender, EventArgs e)
{
    try
    {
        _owner.ViewSpectacle(new DateName.Value);
        changeTiket.Text = "Изменить";
        changeTiket.Location = new Point(7, 297);
        delSpectacle.Visible = true;
    }
    catch
    {
        changeTiket.Text = "Добавить";
        changeTiket.Location = new Point((this.ClientSize.Width - changeTiket.Width) / 2, 297);
        delSpectacle.Visible = false;
    }
}
}
}

```

Листинг UserTicketsForm.cs

```

using App.Services;
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 App
{
    public partial class UserTicketsForm : Form
    {
        private TicketServices _ticketServices;
        private TicketServicesXmlLoggingDecorator _ticketServicesLog;
        private string _userName;

        private TicketModel _activeTicket;
        public UserTicketsForm()
        {
            InitializeComponent();
            _ticketServices = new TicketServices();
            _ticketServicesLog = new TicketServicesXmlLoggingDecorator(_ticketServices);

            userTicketsGridView.ReadOnly = true;

            userTicketsGridView.Columns.Add("№", "№");
            userTicketsGridView.Columns.Add("Название", "Название");
            userTicketsGridView.Columns.Add("Дата", "Дата");
            userTicketsGridView.Columns.Add("Категория", "Категория");
            userTicketsGridView.Columns.Add("Цена", "Цена");

            userTicketsGridView.Columns[0].Width = 30;
        }

        private void UserTicketsForm_Load(object sender, EventArgs e)
        {
        }

        public new DialogResult ShowDialog(Administrator user, string userName)
        {
            _userName = userName;
            userTicketGroupBox.Text = $"Билеты пользователя {_userName}";

            ShowTickets();
            return base.ShowDialog();
        }

        public new DialogResult ShowDialog(Registered user)
        {
            _userName = user.Login;

            userTicketGroupBox.Text = $"Билеты пользователя {_userName}";

            ShowTickets();
            return base.ShowDialog();
        }

        public new DialogResult ShowDialog(User user)
        {
        }
    }
}

```

```

        return base.ShowDialog();
    }

    public new void ShowTickets()
    {
        userTicketsGridView.Rows.Clear();

        IEnumerable<TicketModel> tickets = _ticketServicesLog.GetTicket(_userName);

        foreach (TicketModel ticket in tickets)
        {
            userTicketsGridView.Rows.Add(ticket.Id, ticket.Title, ticket.Date.ToString("d"), ticket.Category,
            ticket.Price);
        }
    }

    private void userTicketsGridView_CellClick(object sender, DataGridViewCellEventArgs e)
    {
        if (e.RowIndex >= 0)
        {
            DataGridViewRow row = userTicketsGridView.Rows[e.RowIndex];

            _activeTicket = _ticketServicesLog.GetTicket(Int32.Parse(row.Cells[0].Value.ToString()));
        }
    }

    private void delUserTicketButton_Click(object sender, EventArgs e)
    {
        if (_activeTicket != null)
        {
            DialogResult result = MessageBox.Show("Вы уверены, что хотите удалить билет на спектакль \"" + _activeTicket.Title + "\"?", "Подтверждение удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            if (result == DialogResult.Yes)
            {
                _ticketServicesLog.DeletTicket(_activeTicket.Id);

                ShowTickets();
            }
        }
        else
        {
            MessageBox.Show("Выберите спектакль для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}

```

Листинг SpectacleManagerTests.cs

```

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace CourseWorkTests
{
    [TestClass]
    public class SpectacleManagerTests
    {

        private readonly SpectacleModel _testSpectacle = new SpectacleModel
        {
            Title = "Test Spectacle",
            Author = "Test Author",
            Genre = "Драма",
            Date = DateTime.Now,
            Categories = new Dictionary<Categorias, decimal>()
            {
                { Categorias.VIP, 100 },
                { Categorias.Medium, 50 },
                { Categorias.Standart, 25 }
            },
            FreePlace = 25
        };

        [TestMethod]
        public void Add_AddsNewSpectacle()
        {
            // Arrange
            int initialCount = SpectacleManager.GetAll().Count();

            // Act
            SpectacleManager.Add(_testSpectacle);
            var result = SpectacleManager.GetAll();

            // Assert
            Assert.AreEqual(initialCount + 1, result.Count());
            Assert.IsTrue(result.Any(x => x.Title == _testSpectacle.Title));

            SpectacleManager.Delete(_testSpectacle);
        }

        [TestMethod]
        public void Update_UpdatesSpectacle()
        {
            // Arrange
            var updatedSpectacle = _testSpectacle;
            updatedSpectacle.Author = "Updated Author";
            SpectacleManager.Add(_testSpectacle);

            // Act
            SpectacleManager.Update(updatedSpectacle);
            var result = SpectacleManager.GetAll();

            // Assert
            Assert.IsTrue(result.Any(x => x.Title == _testSpectacle.Title && x.Author == updatedSpectacle.Author));

            SpectacleManager.Delete(_testSpectacle);
        }

        [TestMethod]
        public void Delete_DeletesSpectacle()
        {
            // Arrange
            SpectacleManager.Add(_testSpectacle);
            var initialCount = SpectacleManager.GetAll().Count();

```

```

        // Act
        SpectacleManager.Delete(_testSpectacle);
        var result = SpectacleManager.GetAll();

        // Assert
        Assert.AreEqual(initialCount - 1, result.Count());
        Assert.IsFalse(result.Any(x => x.Title == _testSpectacle.Title));
    }
    [TestMethod]
    public void TestDataValidate_ValidData_ReturnsTrue()
    {
        // Arrange
        var item = new SpectacleModel()
        {
            Title = "Title",
            Author = "Author",
            Genre = "Genre",
            Date = DateTime.Now,
            Categories = new Dictionary<Categorias, decimal>()
            {
                { Categorias.VIP, 100 },
                { Categorias.Medium, 50 },
                { Categorias.Standart, 25 }
            },
        };

        // Act
        var result = SpectacleManager.DataValidate(item);

        // Assert
        Assert.IsTrue(result);
    }
}
}

```

Листинг TicketManagerTests.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CourseWorkTests
{
    [TestClass]
    public class TicketManagerTests
    {
        private readonly SpectacleModel _testSpectacle = new SpectacleModel
        {
            Title = "Test Spectacle",
            Author = "Test Author",
            Genre = "Драма",
            Date = DateTime.Now,
            Categories = new Dictionary<Categorias, decimal>()
            {
                { Categorias.VIP, 100 },
                { Categorias.Medium, 50 },
                { Categorias.Standart, 25 }
            }
        };
    }
}

```

```

    },
    FreePlace = 25
};

```

```

[TestMethod]
public void GetAll_ReturnsTicketModels()
{
    // Arrange

    // Act
    var result = TicketManager.GetAll();

    // Assert
    Assert.IsNotNull(result);
    Assert.IsInstanceOfType(result, typeof(IEnumerable<TicketModel>));
}

```

```

[TestMethod]
public void Add_ValidTicketModel_AddsTicketToXml()
{
    // Arrange
    SpectacleManager.Add(_testSpectacle);
    int initialCount = TicketManager.GetAll().Count();
    var validTicket = new TicketModel
    {
        Owner = "Valid Owner",
        Date = DateTime.Now,
        Category = Categories.Medium,
        Price = 100
    };
    // Act
    TicketManager.Add(validTicket);

    // Assert
    Assert.AreEqual(initialCount + 1, TicketManager.GetAll().Count());
    SpectacleManager.Delete(_testSpectacle);
}

```

```

[TestMethod]
public void Delete_ExistingTicketId_RemovesTicketFromXml()
{
    // Arrange
    SpectacleManager.Add(_testSpectacle);
    int ticketIdToDelete = TicketManager.GetAll().First().Id;
    int initialCount = TicketManager.GetAll().Count();

    // Act
    TicketManager.Delete(ticketIdToDelete);

    // Assert
    Assert.AreEqual(initialCount - 1, TicketManager.GetAll().Count());
    Assert.IsNull(TicketManager.GetAll().SingleOrDefault(t => t.Id == ticketIdToDelete));
    SpectacleManager.Delete(_testSpectacle);
}

```

```

[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void Delete_NonExistingTicketId_ThrowsArgumentException()
{
    // Arrange

```

```

        int nonExistingTicketId = -1;

        // Act
        TicketManager.Delete(nonExistingTicketId);

        // Assert
        // ExpectedException
    }
}
}

```

Листинг UserManagerTests.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace CourseWorkTests
{
    [TestClass]
    public class UserManagerTests
    {
        XDocument xmlDoc = XDocument.Load("C:\\Users\\Evgeni\\Desktop\\CourseWork\\App\\XMLData\\users.xml");

        [TestMethod]
        public void GetAll_ReturnsAllUsers()
        {
            // Arrange
            var expectedUsers = new List<UserModel>
            {
                new UserModel { Login = "user1", Password = "password1", Role = Role.admin },
                new UserModel { Login = "user2", Password = "password2", Role = Role.registered },
                new UserModel { Login = "user3", Password = "password3", Role = Role.guest }
            };
            var xmlDoc = new XDocument(new XElement("users",
                expectedUsers.Select(u => new XElement("user",
                    new XElement("login", u.Login),
                    new XElement("password", u.Password),
                    new XElement("role", u.Role)
                ))));
            // Act
            var actualUsers = UserManager.GetAll();

            // Assert
            Assert.IsTrue(actualUsers.ToList().Count() > 0);
        }

        [TestMethod]
        public void Add_ValidUser_AddsUserToXmlDoc()
        {
            // Arrange
            var newUser = new UserModel { Login = "user4", Password = "password4", Role = Role.admin };

            // Act
            UserManager.Add(newUser);

            // Assert
            var actualUser = UserManager.GetAll().FirstOrDefault(x => x.Login == newUser.Login);

```



```

        Assert.IsNotNull(actualUser);

        UserManager.Delete(newUser);
    }

[TestMethod]
public void Update_ValidUser_UpdatesUserInXmlDoc()
{
    // Arrange
    var existingUser = new UserModel { Login = "user1", Password = "password1", Role = Role.admin };

    var updatedUser = new UserModel { Login = "user1", Password = "newPassword", Role = Role.registered };
    UserManager.Add(updatedUser);
    // Act
    UserManager.Update(updatedUser);

    // Assert
    var actualUser = UserManager.GetAll().FirstOrDefault(x => x.Login == updatedUser.Login);
    Assert.IsNotNull(actualUser);

    UserManager.Delete(updatedUser);
}
[TestMethod]
public void Delete_UserExists_RemovesUser()
{
    // Arrange
    var user = new UserModel
    {
        Login = "existinguser",
        Password = "password",
        Role = Role.admin
    };
    UserManager.Add(user);

    // Act
    UserManager.Delete(user);

    // Assert
    Assert.IsFalse(UserManager.GetAll().Any(u => u.Login == user.Login));
}
}
}

```

ПРИЛОЖЕНИЕ В

(обязательное)

Руководство пользователя

Для запуска приложения необходимо открыть файл «App.exe», для завершения программы необходимо нажать кнопку в правом верхнем углу программы.

Программа доступна на следующий операционных системах:

- Windows 7;
- Windows 8;
- Windows 10;
- Windows 11.

Гостю доступен следующий функционал:

- просмотр афиши театра;
- работа с системой в качестве гостя;
- возможность регистрации.

Для авторизации необходимо ввести логин и пароль в форме авторизации на главном окне приложения. В зависимости от типа пользователя функционал будет различаться.

Для зарегистрированного пользователя доступен следующий функционал:

- просмотр афиши театра;
- поиск спектаклей;
- просмотр сведений о спектакле;
- покупка билета на спектакль;
- просмотр купленных пользователем билетов.

Администратор имеет возможности редактировать жанры, авторов, спектакли, пользователей, формировать различные статистические отчеты о проданных билетах, менять цены на билеты.

В случае возникновения непредвиденных ситуаций, например, неудачной попытки входа или попытки добавления спектакля в уже занятую дату, пользователь будет уведомлен соответствующим сообщением об ошибке.

ПРИЛОЖЕНИЕ Г

(обязательное)

Руководство программиста

Приложение предназначено для продажи билетов на спектакли, учета проданных билетов за некоторый период времени, по жанру, по автору.

Основные функции программы:

- покупка билетов на спектакли;
- формирование различных статистических отчетов по проданным билетам за некоторый промежуток времени, по конкретному автору, жанру;
- считывание данных из *XML*-файлов;
- вывод в графический интерфейс информации, считанных из файлов.

Для функционирования приложения должен быть установлен *.NET Framework*. Разработанная программа написана на языке программирования *C#* в среде разработки *Microsoft Visual Studio 2022*, в качестве источника данных используются *XML*-файлы.

При работе программа оповещает пользователя о различных проблемах, которые могут возникнуть в результате функционирования программы. К таким проблемам относятся следующие проблемы:

- отсутствие аккаунта с вводим пользователем логином;
- неправильно введенный пользователем пароль;
- пользователь также не сможет отменить билет на спектакль, который уже прошел и на текущий момент не доступен.

При работе программа оповещает администратора о различных проблемах, которые могут возникнуть в результате функционирования программы. К таким проблемам относятся следующие проблемы:

- попытка добавить спектакль, когда имеется другой спектакль в текущую дату, в результате этого администратору выведется сообщение о неудачной попытке добавления спектакля;
- попытка добавить спектакль, в окне для добавления которого введены не все данные, выводится сообщение о неудачной попытке добавления спектакля;
- попытка добавить жанр, в окне для добавления которого введены не все данные, выводится сообщение о неудачной попытке добавления жанра;
- попытка добавить автора, в окне для добавления которого введены не все данные, выводится сообщение о неудачной попытке добавления автора;

В иных случаях, если ошибке устранить не удалось, требуется обратиться к разработчику программы.

ПРИЛОЖЕНИЕ Д (обязательное)

Руководство системного программиста

Приложение написано на языке программирования C#. В качестве источника данных применяются XML-файлы. Для взаимодействия с приложением необходимо предварительно установить на компьютер с официального сайта программу *Microsoft Visual Studio 2022*.

Для программы важна структура папок, необходимо, чтобы исполняемый файл «App.exe». Чтобы начать работу с программой, системный программист в XML-файле «users» должен прописать пароль и логин для администратора. Затем в XML-файле «tickets» системный программист должен прописать цены и количество билетов по каждому типу. Только лишь после этого программой можно пользоваться.

В приложении могут возникнуть различные ошибки, например, такие ошибки, как неудачная попытка добавления пользователя, ошибка при составлении статистического отчета о проданных билетах. Так как при удалении автора или жанра значения данного поля у спектакля становится равным пустому значению, и при учете проданных билетов из-за этого значения могут посчитаны неверно. Чтобы это исправить, необходимо проверить корректность данных для спектакля, для пустых значений жанра и автора присвоить новое значение или вернуть предыдущее.

Если ошибки так и не удалось устранить, требуется обратиться к разработчику программы.

ПРИЛОЖЕНИЕ Е
(обязательное)

Иерархическая схема классов приложения