**ОГЛАВЛЕНИЕ**

[1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 7](#_Toc517229223)

[1.1. Исследование предметной области 7](#_Toc517229224)

[1.2. Обзор и анализ аналогичных решений 7](#_Toc517229225)

[1.3. Анализ средств программирования 8](#_Toc517229226)

[1.3.1. Обзор методов решения 10](#_Toc517229227)

[2. ПРОЕКТНАЯ ЧАСТЬ 12](#_Toc517229228)

[2.1. Техническое задание 12](#_Toc517229229)

[2.1.1. Постановка задачи 12](#_Toc517229230)

[2.1.2. Назначение программного продукта 12](#_Toc517229231)

[2.1.3. Основание для разработки 13](#_Toc517229232)

[2.1.4. Требования к программе 13](#_Toc517229233)

[2.1.4.1. Требования к функциональным характеристикам 13](#_Toc517229234)

[2.1.4.2. Требования к аппаратным и программным средствам 13](#_Toc517229235)

[2.1.4.3. Требования к надежности 14](#_Toc517229236)

[2.1.4.4. Требования к надежному функционированию системы 15](#_Toc517229237)

[2.1.4.5. Отказы из-за некорректных действий пользователей системы 15](#_Toc517229238)

[2.1.4.6. Требования к пользователям системы 15](#_Toc517229239)

[2.1.4.7. Защита сохранности данных 16](#_Toc517229240)

[2.2. Описание функционально-логической структуры программы 16](#_Toc517229241)

[2.3. Логическая модель данных 18](#_Toc517229242)

[2.4. Общий алгоритм работы программы 20](#_Toc517229243)

[2.5. Алгоритмы отдельных модулей 21](#_Toc517229244)

[2.6. Программная реализация 22](#_Toc517229245)

[2.6.1. Пользовательский интерфейс 22](#_Toc517229246)

[2.6.2. Разработка программного продукта 24](#_Toc517229247)

[2.6.3. Тестирование и отладка программы 24](#_Toc517229248)

[2.7. Инструкция пользователя 31](#_Toc517229249)

[2.8. Анализ результатов 45](#_Toc517229250)

[3. ЭКОНОМИЧЕСКАЯ ЧАСТЬ. РАСЧЕТ ПРЕДПОЛОГАЕМОЙ ПРИБЫЛИ ОТ РЕАЛИЗАЦИИ СОЗДАННОГО ПРОГРАММНОГО ПРОДУКТА 46](#_Toc517229251)

[3.1. Расчет затрат на выполнение проекта 46](#_Toc517229252)

[3.2. Расчет расходов на заработную плату 47](#_Toc517229253)

[3.3. Расчет затрат на материалы 48](#_Toc517229254)

[3.4. Расчет электроэнергии и амортизации оборудования 48](#_Toc517229255)

[3.5. Расчет сметы затрат 49](#_Toc517229256)

[3.6. Расчет экономического эффекта 49](#_Toc517229257)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 51](#_Toc517229258)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ 52](#_Toc517229259)

**ВВЕДЕНИЕ**

Каждый день более половины населения планеты используют компьютер для выполнения своих повседневных задач. Но почему? Во-первых, это удобно. Сейчас почти каждый имеет дома свой персональный компьютер или ноутбук, которые совмещают в себе печатную машинку, музыкальный и видеопроигрыватель, игровую приставку, картотеку, способную хранить очень большие объемы данных, а также множество других полезных функций. С их помощью люди работают, учатся, проводят досуг. Компьютер – это удобный сборник всего того, что может понадобится человеку для выполнения поставленных задач. Не менее удобным является ноутбук, так как его можно взять с собой и пользоваться им не находясь дома. Во-вторых, это быстро. Сейчас даже бюджетные компьютеры и ноутбуки показывают достаточно высокую скорость работы. Благодаря этому обработка больших документов, монтаж видео, редактирование фотографий требуют намного меньше времени. В-третьих, это легко. Сейчас любой пользовательский интерфейс выстроен так, чтобы он был интуитивно понятен и предсказуем даже для начинающего пользователя. Не менее важно, что компьютер умеет общаться с пользователем, например, если в системе произошел сбой, то будет выдано сообщение с описанием причины и вариантами ее исправления. Но ежедневное использование компьютера не было бы таким удобным, быстрым и легким, не будь программистов.

Компьютер – это устройство, которое выполняет четко заданную последовательность операций. Чтобы компьютер получил эту последовательность операций, её должен описать программист в программном коде.

Программист – это специалист, занимающийся разработкой программного обеспечения для компьютеров и различного рода вычислительно-операционных систем. Но для разработки программисту требуются специальные программы и средства, например, среда разработки, база данных, СУБД и другое. Также важно правильно выбрать язык, который подойдет для решения поставленной задачи.

Среда разработки (среда программирования) – это программа, которая позволяет программисту реализовать свой программный код с целью создания отдельного модуля нового программного продукта или полноценного приложения.   
В дипломе я использовал среду разработки Microsoft Visual Studio и язык программирования C#.

Microsoft Visual Studio – интегрированная среда разработки программного обеспечения под операционную систему семейства Windows.

C# – объектно-ориентированный язык программирования, разработанный компанией Microsoft как язык разработки приложений для платформы Microsoft .NET Framework.

База данных – это совокупность данных, систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью компьютерной системы. Чтобы создать базу данных и впоследствии использовать её требуется СУБД.

Система управления базой данных (СУБД) – специализированная программа или их совокупность, предназначенная для создания и использования базы данных. СУБД так же, как и базы данных, имеют обширную классификацию. В дипломе я использовал реляционную СУБД Microsoft Office Access.

Microsoft Office Access – реляционная система управления базами данных. Данная СУБД позволяет создавать базы данных, таблицы, запросы, формы, отчеты, указывать связь с внешними таблицами и базами данных, а также управлять ими.

С использованием перечисленных выше программ и средств была разработана система «Театральная касса», позволяющая упростить работу театральных касс. Разработанная система позволяет управлять билетами, списком театров, расписанием театров, создавать отчеты и производить фильтрацию и поиск по параметрам.

При написании проекта были применены навыки, приобретённые во время изучения профессиональных модулей:

ПМ 01. «Разработка программных модулей программного обеспечения для компьютерных сетей»:

* ПК 1.1. Выполнять разработку спецификаций отдельных компонент.
* ПК 1.2. Осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля.
* ПК 1.4. Выполнять тестирование программных модулей.
* ПМ 02. «Разработка и администрирование баз данных»:
* ПК 2.1. Разрабатывать объекты базы данных.
* ПК 2.2. Реализовывать базу данных в конкретной системе управления базами данных (СУБД).

ПМ 03. «Участие в интеграции программных модулей»

Дипломный проект подтверждает ряд профессиональных компетенций:

* ПК 3.1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения
* ПК 3.2. Выполнять интеграцию модулей в программную систему
* ПК 3.3. Выполнять отладку программного продукта с использованием специализированных программных средств
* ПК 3.4. Осуществлять разработку тестовых наборов и тестовых сценариев
* ПК 3.5. Производить инспектирование компонент программного продукта на предмет соответствия стандартам кодирования
* ПК 3.6. Разрабатывать технологическую документацию

1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

Театральная касса является большой системой, которая должна хранить в себе огромное количество разных данных: список театров, спектаклей, купленных билетов. Чтобы удобно управлять этими данными необходима программа, которая будет понятна даже неопытному пользователю. В связи с этим цель дипломного проекта – разработать информационную систему для учета билетов и их продаж, и составления отчетов, на основе полученных данных.

1.1. Исследование предметной области

Данная программа предназначена для использования сотрудниками театральной кассы. В данной версии программа больше необходима сотрудникам, которые занимаются продажей билетов.

1.2. Обзор и анализ аналогичных решений

На данный момент существует несколько аналогов подобных программ.   
Но многие из них созданы для определенного театра, а не для всех театров в целом. Для того, чтобы понять, какие достоинства и недостатки имеет каждая из существующих программ, стоит их сравнить.

1. Автоматизированная информационная система электронной продажи билетов в театре и учета финансовых результатов показа спектаклей «СУПЕРБИЛЕТ – ТЕАТР».

Достоинства:

* возможность быстрой адаптации программы к конкретным условиям;
* строгое разграничение прав доступа к информации;
* все основные настройки выполняет администратор системы;
* существует возможность добавления и изменения схем залов (дополнительные кресла и пр.);
* обеспечена возможность составления и использования ценовых схем;
* автоматизированное формирование различных отчетов;
* использование принтеров для печати билетов.

Недостатки:

* устаревший и загроможденный интерфейс;

2. Полнофункциональная билетная система «Radario».

Достоинства:

* продажа билетов онлайн и в кассе;
* возможность покупки абонементов;
* служба поддержки программного обеспечения;
* программа лояльности;
* оповещение об отмене спектакля;
* аналитика и отчетность;
* возможность проверки билетов.

Недостатки:

* отсутствие разграничения прав доступа к информации;

1.3. Анализ средств программирования

Среда разработки (среда программирования) – это программа, которая позволяет программисту реализовать свой программный код с целью создания отдельного модуля нового программного продукта или полноценного приложения. Говоря простым языком, среда программирования служит для разработки программ. Обычно среды разработки ориентируются на конкретный язык или языки программирования. Так называемые интегрированные среды программирования содержат все необходимые средства для разработки: редактор кода, компилятор и отладчик.

Рассмотрим отличия средств разработки на примере MonoDevelop и SharpDevelop.

1. MonoDevelop - свободная среда разработки, предназначенная для создания приложений C#, Java, Boo, Nemerle, Visual Basic .NET, Vala, CIL, C и C++. Интерфейс этой среды очень простой (см. рисунок 1). Она может использоваться для создания личных и коммерческих проектов. Является бесплатной средой программирования.

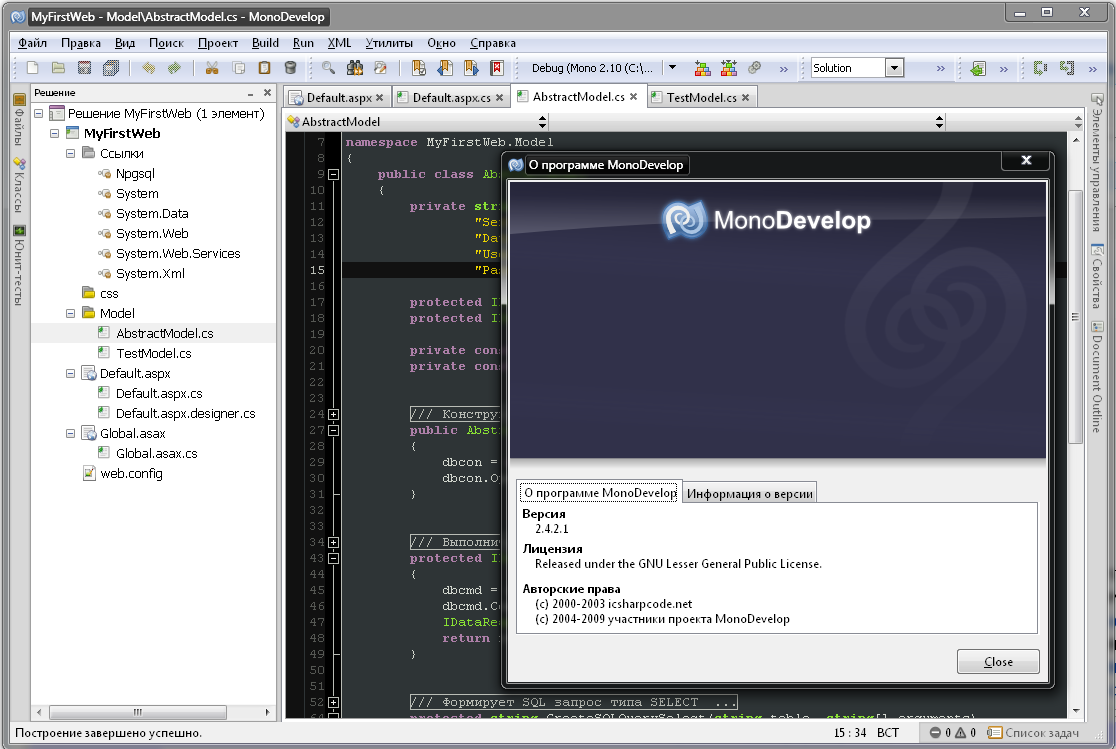


Рисунок 1 – Интерфейс MonoDevelop

2. SharpDevelop - свободная среда разработки для C#, Visual Basic .NET, Boo, IronPython, IronRuby, F#, C++. У этой среды просто и интуитивно понятный интерфейс (см. рисунок 2). Она может использоваться для создания личных и коммерческих проектов. Является бесплатной средой программирования.

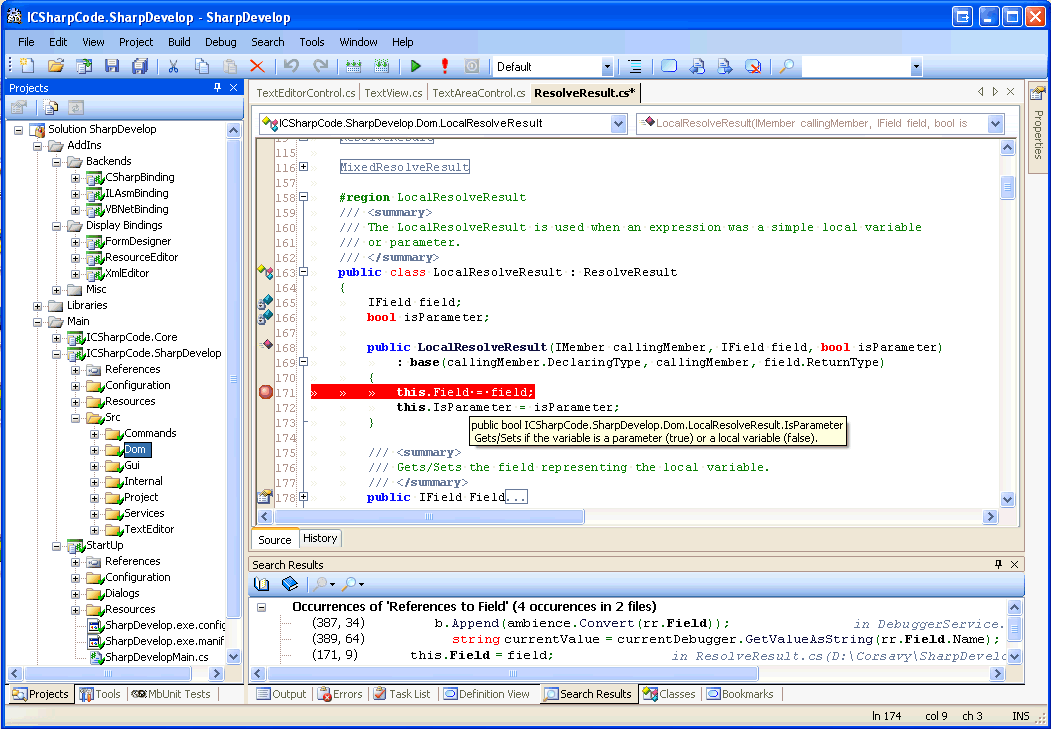


Рисунок 2 – Интерфейс SharpDevelop

1.3.1. Обзор методов решения

Существует несколько методов решения поставленной задачи. Для того, чтобы выбрать самый удобный, определим какие средства нам подходят.

Во-первых, надо выбрать подходящий для решения нашей задачи язык программирования. В нем должен присутствовать полный набор нужного для разработки функционала. Подойдут следующие языки: C++, C#.

Во-вторых, надо выбрать среду разработки, которая будет содержать все нужные для разработки проекта функции и поддерживать разработку на выбранном языке. Подойдут следующие среды программирования: Microsoft Visual Studio, MonoDevelop, Dev C++.

В-третьих, надо выбрать СУБД, с помощью которой будет создана база данных. Они отличаются способами создания и управления базами данных. Для работы с базой данных SQL надо знать язык запросов TransactSQL, умение устанавливать саму программу и создавать сервер для возможности работы с данной СУБД. Но есть более простые и понятные системы управления базами данных, такие как Microsoft Access, имеющие удобный графический интерфейс и возможность быстро создать и заполнить информацией базу данных.

**1.3.2. Обоснование выбора средств разработки**

В связи с тем, что программа разрабатывается для операционной системы Windows, можно сразу отсечь все языки, среды и СУБД, которые не могут работать с этой системой.

Поэтому, в качестве языка программирования был выбран C#. C# – объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework. Этот язык прекрасно подходит для разработки программы, содержащей формы, на которых будут расположены элементы интерфейса.

Самой популярной и удобной средой разработки на языке C# для операционной системы Windows является Microsoft Visual Studio, которая и будет использоваться для написания программы. В ней есть удобный редактор кода, быстрый компилятор и хороший отладчик, позволяющий отдельно пройти каждый шаг программы и посмотреть, как она себя ведет в той или иной ситуации.

Microsoft Office Access – данная СУБД была выбрана для написания дипломного проекта, так как она позволяет создать локальную базу данных и управлять ей с помощью пользовательского интерфейса, без надобности создавать сервер и описывать каждое действие с помощью языка запросов.

Указанные выше средства для разработки программного продукта были выбраны, потому что вместе они позволяют быстро и без особого труда создать работающую информационную систему.

Перед тем, как начать разработку программы, следует выполнить несколько шагов, а именно:

1. Создать базу данных, заполнить ее имеющимися данными и сохранить с расширением .mdb.

2. Создать C# проект в Visual Studio.

3. С помощью интерфейса или программно подключить к созданному проекту базу данных.

После выполнения описанных выше действий можно приступать к написанию, отладке и тестированию программы.

2. ПРОЕКТНАЯ ЧАСТЬ

В данной части дипломного проекта рассмотрим техническое задание, постановку задачи, назначение и основания для разработки программного продукта, его требования, защиту и планирование.

2.1. Техническое задание

Целью дипломного проекта является создание и разработка автоматизированной информационной системы для театральных касс, реализующих продажу театральных билетов. Данная система позволит осуществлять продажу билетов и вести учет проданных билетов.

2.1.1. Постановка задачи

Требуется разработать прикладную программу, имеющею следующий функционал:

* ведение списка театров и спектаклей с возможностью добавления, редактирования и удаления;
* осуществление продажи билетов на спектакли;
* ведение расписания спектаклей;
* осуществление поиска и фильтрации данных;
* осуществление вывода отчетов;
* авторизация.

Программа должна иметь ряд справочников: театры, репертуар (список спектаклей), расписание и позволять добавлять, изменять и удалять информацию из этих таблиц.

2.1.2. Назначение программного продукта

Данная программа создается для использования в театральных кассах всех городов России, так как является универсальной. С помощью своего функционала программа должна решать поставленные задачи. Также она должна быть удобна в использовании пользователям с любым уровнем владения компьютером и удобно и безопасно хранить данные.

Главной целью создания этой программы является потребность   
в автоматизации и ускорении ручного труда, упрощение процесса оформления театральных билетов и ведения отчетности.

2.1.3. Основание для разработки

Основание для разработки данного программного продукта является задание на преддипломную практику и задание на диплом.

2.1.4. Требования к программе

Программа должна иметь простой и интуитивно понятный интерфейс. Также программа должна иметь все нужные таблицы, уметь принимать, обрабатывать и выдавать информацию.

2.1.4.1. Требования к функциональным характеристикам

Основные функции программы:

* добавление новых записей;
* редактирование уже добавленных записей;
* удаление записей;
* сортировка записей;
* поиск записей;
* создание отчетов по определенным критериям.

Также интерфейс данной программы должен отвечать следующим требованиям:

* быть удобным в использовании;
* иметь понятные названия, которые не будут сбивать с толку;
* позволять легко ориентироваться в системе.

2.1.4.2. Требования к аппаратным и программным средствам

Минимальные требования для использования программы:

* процессор: Intel Pentium® IV 1.3 GHz или AMD Athlon™ от 1.5GHz;
* операционная система: Microsoft Windows 7 или выше;
* оперативная память: 1 Гб;
* устройства ввода: клавиатура и мышь.

Рекомендованные требования:

* процессор: Intel Core i3 2.4 GHz;
* оперативная память: 4 Гб;
* устройства ввода: клавиатура и мышь;
* принтер или МФУ;

2.1.4.3. Требования к надежности

Надежность программного обеспечения - способность программного продукта безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью. Контроль за входными и выходными данными, соблюдение условий эксплуатации программы - основное требование к обеспечению надежного функционирования программы.

Надежность является одним из важнейших факторов, определяющих общую производительность и эффективность систем. Надежность программного продукта определяется, как свойство системы выполнить заданную функцию, сохраняя во времени значения установленных эксплуатационных показателей в заданных пределах, соответствующих заданным режимам и условиям использования, технического обслуживания, ремонта, хранения, транспортирования. Свойство надежности проявляется в том, что система выполняет установленные перед ней задачи без проблем.

Если произойдет отказ, следует перезапустить программу, если же это не поможет, то обратиться к специалисту (инженеру, разработчику программного продукта).

Тестирование является еще более важным для программного обеспечения, чем для аппаратных средств. Как и аппаратные средства, программное обеспечение тестируется на нескольких уровнях, начиная с отдельными единицами, путем интеграции и тестирования полной системы контроля. В отличие от аппаратных средств, это нецелесообразно, чтобы пропустить уровень тестирования программного обеспечения. На всех этапах тестирования, ошибки программного обеспечения будут обнаружены, исправлены, и будет проведено повторное тестирование.

Оценки надежности обновляются на основе количества ошибок и других показателей. На системном уровне, среднее время между безотказной работой могут быть собраны и использованы для оценки надежности. В отличие от аппаратных средств, выполняя точно тот же самый тест на точно такую же конфигурацию, программного обеспечения не обеспечивает повышенную статистическую достоверность. Вместо этого, надежность программного обеспечения используют различные показатели, такие как покрытие кода.

2.1.4.4. Требования к надежному функционированию системы

В связи с тем, что вводимые в программу данные никаким образом не подготовлены для этого, то, чтобы избежать сбоев программы, стоит выполнять ряд простых правил:

* не стоит менять названия файлов и папок, находящихся в папке программой;
* нельзя удалять файлы и папки, находящиеся в папке с программой;
* ввод правильных данных.

2.1.4.5. Отказы из-за некорректных действий пользователей системы

Для избегания сбоев программы из-за некорректных действий пользователей системы должны быть созданы обработчики событий, которые будут:

* производить корректировку данных, вносимых пользователем в полях ввода;
* поля ввода данных должны соответствовать тому типу данных, которые требуются в полях таблицы.

2.1.4.6. Требования к пользователям системы

Система предназначена для пользователей, которые хотя бы немного разбираются в компьютерных технологиях и могут разобраться в том, как пользоваться этой программой.

2.1.4.7. Защита сохранности данных

Для обеспечения сохранности данных, хранящихся в базе данных, на неё установлен пароль. Таким образом получить доступ к этим данным может только пользователь, имеющий пароль. Для того чтобы установить пароль на базу данных следует выполнить несколько простых шагов:

1. Открыть базу данных в монопольном режиме (в диалоговом окне открытия выбрать пункт «Монопольно»).
2. Нажать Файл – Сведения и выбрать пункт «Установить пароль базы данных».
3. Откроется мастер настройки пароля базы данных. Следует ввести пароль для базы данных и повторить его. Пароль на базу данных успешно настроен.

2.2. Описание функционально-логической структуры программы

С помощью функционального моделирования (нотация IDEF0), можно провести систематический анализ предметной области, сосредоточившись на регулярно решаемых задачах (функциях), свидетельствующих об их правильном выполнении, показателях, необходимых для этого ресурсах, результатах и исходных материалах (сырье). В рамках методологии функционального моделирования IDEF0 бизнес-процесс представляется в виде набора функций, которые взаимодействуют между собой, а также показываются информационные, людские и производственные ресурсы, требуемые для каждой функции.

В данной дипломной работе на основе нотации IDEF0 была разработана контекстная диаграмма, которая показывает входные и выходные ресурсы, правила управления и механизм управления.

Контекстная диаграмма модели ИС представлена на рисунке 3. На рисунке 4 представлена детализированная контекстная диаграмма модели ИС. На рисунке 5 представлена диаграмма дерева узлов.



Рисунок 3 – Контекстная диаграмма модели ИС

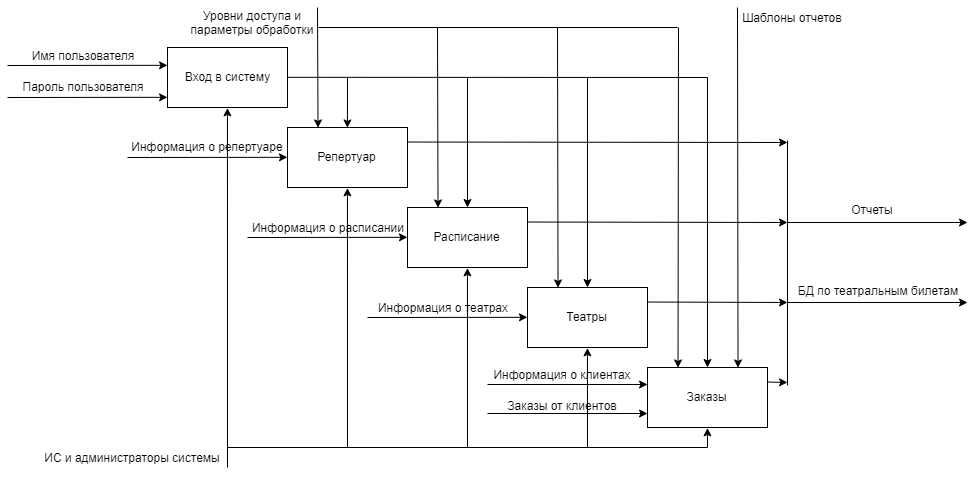


Рисунок 4 – Диаграмма декомпозиции первого уровня

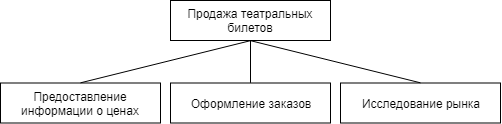


Рисунок 5 – Диаграмма дерева узлов

2.3. Логическая модель данных

На рисунке 6 предоставлена логическая модель базы данных, а ниже описание каждой таблицы.

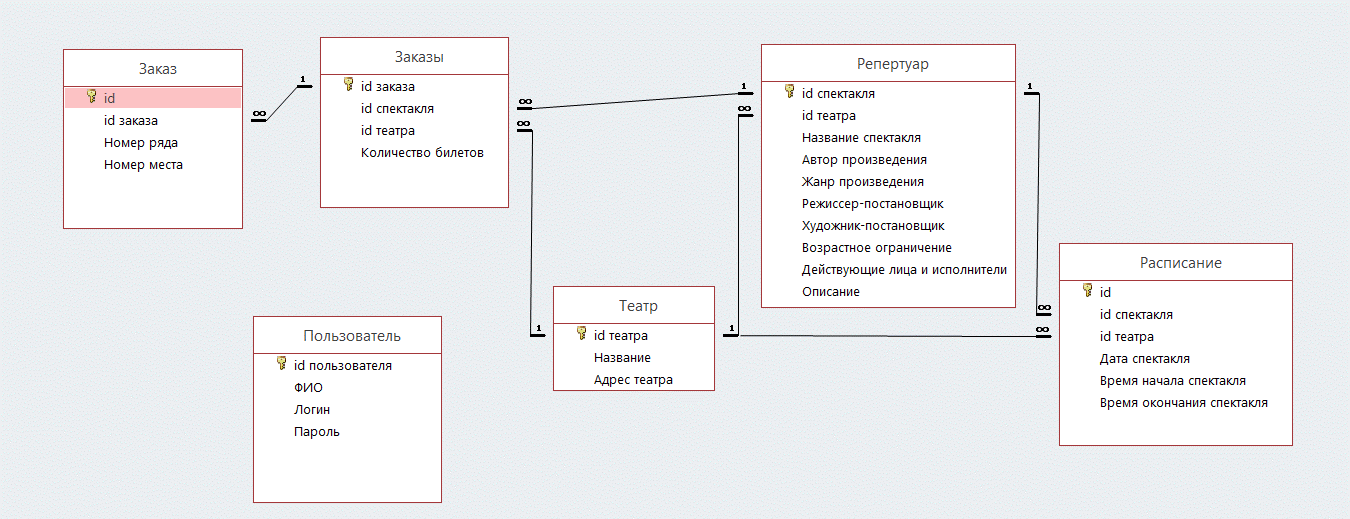


Рисунок 6 – Логическая модель данных

**Таблица 1 – Таблица «Заказы»**

|  |  |  |
| --- | --- | --- |
| Поле таблицы | Тип данных | Описание |
| id заказа | Целое число | Порядковый номер заказа |
| id спектакля | Целое число | Номер спектакля |
| id театра | Целое число | Номер театра |
| Количество билетов | Целое число | Количество билетов |

**Таблица 2 – Таблица «Заказ»**

|  |  |  |
| --- | --- | --- |
| Поле таблицы | Тип данных | Описание |
| id | Целое число | Порядковый номер записи |
| id заказа | Целое число | Номер заказа |
| Номер ряда | Целое число | Номер ряда |
| Номер места | Целое число | Номер места |

**Таблица 3 – Таблица «Репертуар»**

|  |  |  |
| --- | --- | --- |
| Поле таблицы | Тип данных | Описание |
| id спектакля | Целое число | Номер спектакля |
| id театра | Целое число | Номер театра |
| Название спектакля | Текстовый | Название спектакля |
| Автор произведения | Текстовый | Автор произведения, по которому написан спектакль |
| Жанр произведения | Текстовый | Жанр произведения, по которому написан спектакль |
| Режиссер-постановщик | Текстовый | Режиссер-постановщик спектакля |
| Художник-постановщик | Текстовый | Художник-постановщик спектакля |
| Возрастное ограничение | Текстовый | Возрастное ограничение спектакля |
| Действующие лица и исполнители | Текстовый | Действующие лица и исполнители ролей |
| Описание | Текстовый | Описание спектакля |

**Таблица 4 – Таблица «Пользователь»**

|  |  |  |
| --- | --- | --- |
| Поле таблицы | Тип данных | Описание |
| id пользователя | Целое число | Номер пользователя |
| ФИО | Текстовый | ФИО пользователя |
| Логин | Текстовый | Логин пользователя |
| Пароль | Текстовый | Пароль пользователя |

**Таблица 5 – Таблица «Театр»**

|  |  |  |
| --- | --- | --- |
| Поле таблицы | Тип данных | Описание |
| id театра | Целое число | Номер театра |
| Название | Текстовый | Название театра |
| Адрес театра | Текстовый | Адрес театра |

**Таблица 6 – Таблица «Расписание»**

|  |  |  |
| --- | --- | --- |
| Поле таблицы | Тип данных | Описание |
| id | Целое число | Порядковый номер |
| id спектакля | Целое число | Номер спектакля |
| id театра | Целое число | Номер театра |
| Дата спектакля | Дата | Дата спектакля |
| Время начала спектакля | Время | Время начала спектакля |
| Время окончания спектакля | Время | Время окончания спектакля |

2.4. Общий алгоритм работы программы

На рисунке 7 представлен общий алгоритм работы программы. На рисунке 8 представлен алгоритм интерфейсной части.

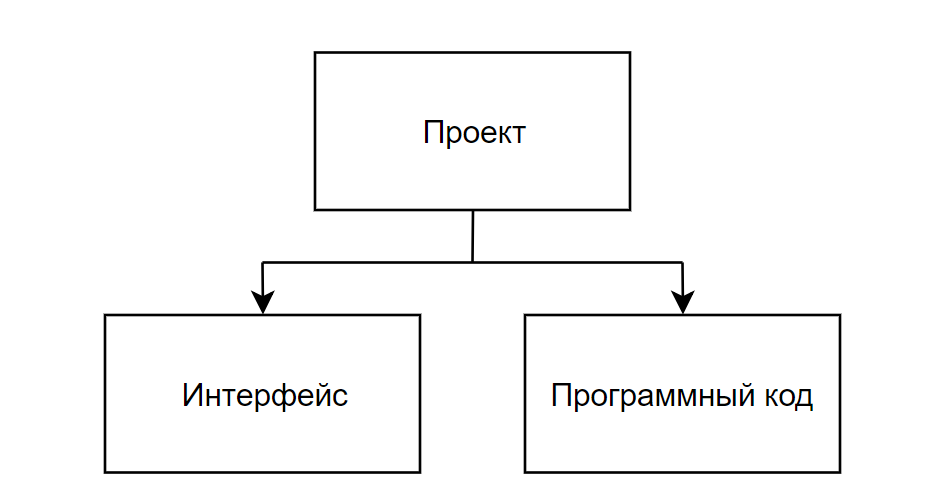


Рисунок 7 – Общий алгоритм программы

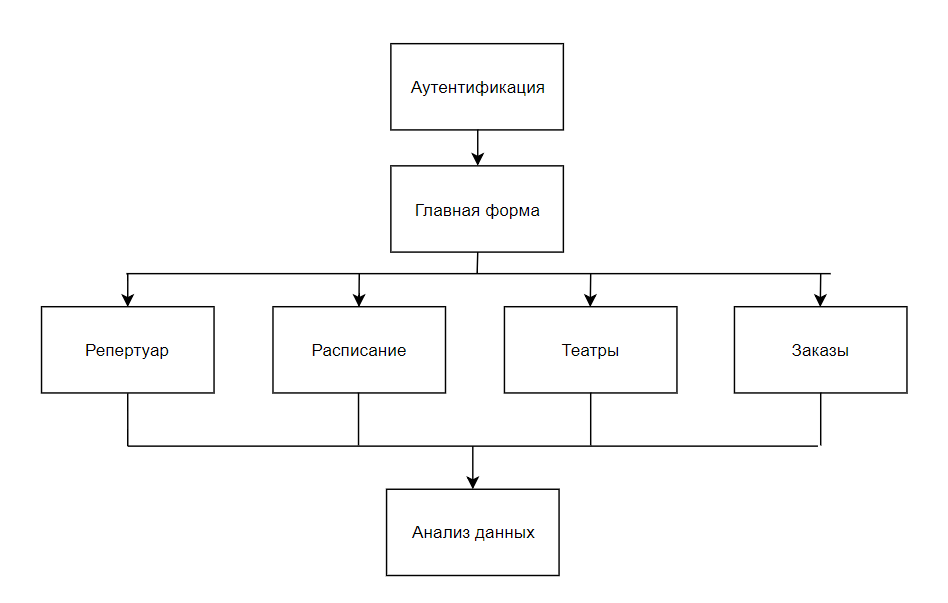


Рисунок 8 – Интерфейсная часть

2.5. Алгоритмы отдельных модулей

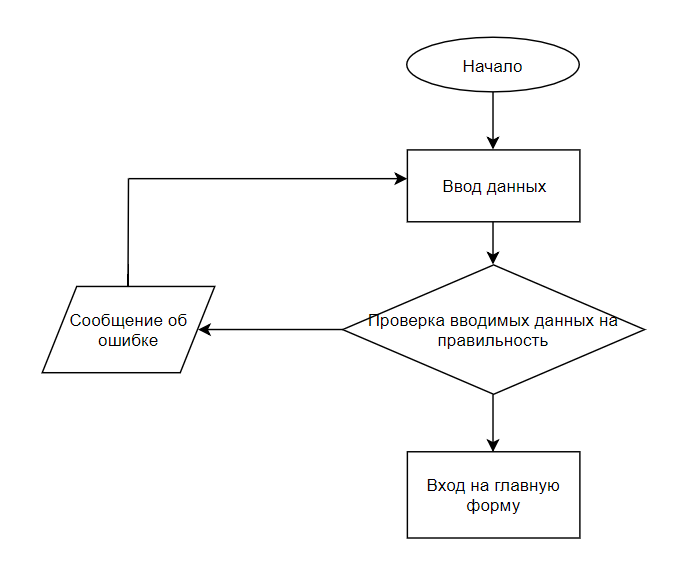


Рисунок 9 – Алгоритм входа в систему

2.6. Программная реализация

С помощью конструктора форм был создан интерфейс программы.

2.6.1. Пользовательский интерфейс

Были реализованы принципы обратной связи и толерантности. Программа отвечает на действия пользователя. При входе в случае корректного ввода логина и пароля будет выдано сообщение с приветствием (см. рисунок 10).

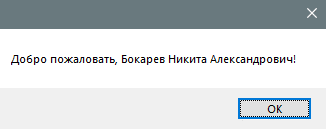


Рисунок 10 – Окно приветствия

Если же будет введен некорректный логин и/или пароль, то будет выдано сообщение об ошибке и вход в программу не будет произведен (см. рисунок 11).

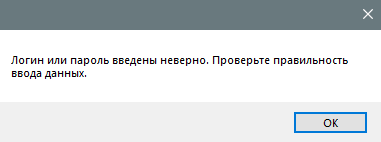


Рисунок 11 – Ошибка входа

Также сообщения будут показываться при успешном добавлении   
(см. рисунок 12), редактировании (см. рисунок 13) или удалении данных   
(см. рисунок 14) и в случае возникновения ошибок (см. рисунок 15).

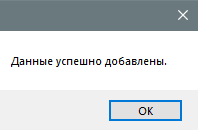


Рисунок 12 – Сообщение об успешном добавлении данных

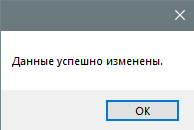


Рисунок 13 – Сообщение об успешном редактировании данных

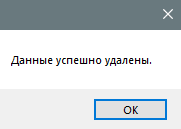


Рисунок 14 – Сообщение об успешном удалении данных

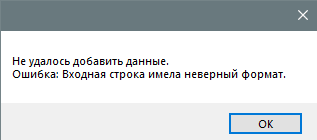


Рисунок 15 – Сообщение об ошибке

В системе реализовано подтверждение удаления записи из таблицы, для предотвращения незапланированного удаления данных (см. рисунок 16).

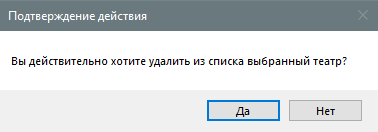


Рисунок 16 – Подтверждение удаления записи

Для удобства ввода данных переключаться между полями можно с помощью клавиши Tab.

В программе каждый модуль отвечает за отдельную часть системы, которые связаны между собой. Например, для того, чтобы добавить новый спектакль в расписание, сначала надо добавить театр, в котором проходит спектакль, потом добавить сам спектакль в модуль «Репертуар», и только после этого добавить спектакль в расписание.

Для управления элементами интерфейса используется мышь, клавиатура же используется для ввода данных. Но есть возможность использовать клавиатуру для управления интерфейсом: клавиша Tab для переключения между полями и клавиша Enter для подтверждения выбора.

Все кнопки на форме выделены более темным цветом, элементы сгруппированы так, чтобы пользователь смог их быстро найти. Также были реализованы всплывающие подсказки для некоторых элементов управления.

2.6.2. Разработка программного продукта

В программе у каждого модуля есть свое назначение.

**Модуль «Репертуар»**

Данный модуль содержит информацию о репертуаре театров, которые добавлены в модуле «Театры».

**Модуль «Расписание»**

Данный модуль содержит информацию о расписании спектаклей, которые добавлены в модуле «Репертуар».

**Модуль «Театры»**

Данный модуль содержит информацию о театрах.

**Модуль «Заказы»**

Данный модуль содержит информацию о заказах (купленных билетах) на спектакли.

2.6.3. Тестирование и отладка программы

Тестирование программы – это процесс выполнения программы на некотором наборе данных, для которого заранее известен результат или известны правила поведения программы.

Отладка программы – работа, направленная на обнаружение и исправление ошибок в программе.

Существует несколько видов тестирования программного обеспечения. Более подробно самые часто используемые из них описан ниже.

Функциональное тестирование – или проверка функциональности, проверка соответствия программного обеспечения требованиям, заявленным в спецификации. Может проводиться как полное тестирование заявленной функциональность, так и проверка только базовой функциональности. Существует два вида функционального тестирования: тестирование «белого ящика» на соответствие программного продукта требованиям со знанием внутренней структуры реализации системы и тестирование на соответствие программного продукта требованиям без знания внутренней структуры реализации системы.

Системное тестирование – высокоуровневая проверка функционала всей программы или системы в целом.

Тестирование производительности – проводится с целью определения как быстро работает система или её часть под определенной нагрузкой. Существует два вида тестирования производительности: нагрузочное тестирование – предназначено для проверки работоспособности системы при стандартных нагрузках и для определения максимально возможного пика, при котором система работает правильно и стресс тестирование – предназначено для проверки работоспособности системы при нестандартных нагрузках и для определения максимально возможного пика, при котором система работает правильно. Так же предназначено для выявления результатов, при которых система переходит в нерабочее состояние.

Регрессивное тестирование – проводится с целью проверить, не влияют ли новые функции, улучшения и исправленные дефекты на существующую функциональность продукта и не возникают ли старые дефекты.

Модульное тестирование – тестирование отдельных модулей программы. Каждая сложная программная система состоит из отдельных частей - модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе всей системы, необходимо вначале протестировать каждый модуль системы по отдельности. В случае возникновения проблем при тестировании системы в целом это позволяет проще выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них. Такое тестирование модулей по отдельности получило называние модульного тестирования.

Тестирование безопасности – это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Тестирование локализации – это процесс тестирования локализованной версии программного продукта. Проверка правильности перевода элементов интерфейса пользователя, проверка правильности перевода системных сообщений и ошибок, проверка перевода раздела "Помощь"/"Справка" и сопроводительной документации.

Тестирование удобства пользования (юзабилити тестирование) – это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. Выявлять проблемы, связанные со специфическим механизмом интерфейса определять, существуют ли проблемы с удобностью интерфейса для навигации, использования основного функционала.

Альфа-тестирование – имитация реальной работы с системой штатными разработчиками, либо реальная работа с системой потенциальными пользователями/заказчиком. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приёмочного тестирования. Иногда альфа-тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Обнаруженные ошибки могут быть переданы тестировщикам для дополнительного исследования в окружении, подобном тому, в котором будет использоваться программа.

Бета-тестирование – в некоторых случаях выполняется распространение предварительной версии (в случае проприетарного программного обеспечения иногда с ограничениями по функциональности или времени работы) для некоторой большей группы лиц с тем, чтобы убедиться, что продукт содержит достаточно мало ошибок. Иногда бета-тестирование выполняется для того, чтобы получить обратную связь о продукте от его будущих пользователей.

Для подтверждения работоспособности программы проведем тестирование одного из модулей программы на контрольном примере.

**Тестирование модуля «Театры»**

Добавление новой записи в таблицу.

Для добавления новой записи в таблицу театров надо нажать на кнопку «Добавить» на форме «Театры». Откроется панель, содержащая поля для ввода данных (см. рисунок 17). После заполнения всех полей нажимаем на кнопку «Добавить». В итоге в таблицу будет добавлена новая запись (см. рисунок 18).

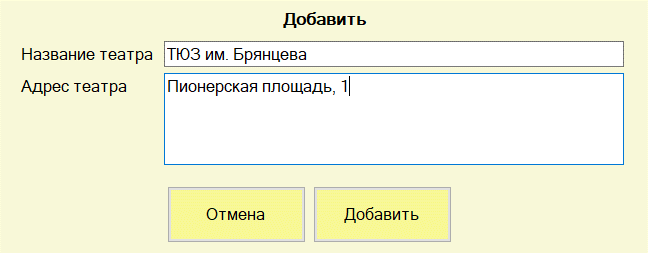


Рисунок 17 – Форма добавления новой записи

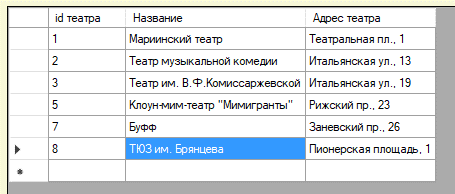


Рисунок 18 – Результат добавления новой записи

Запись в таблицу добавлена. Система работает правильно.

Существующую в таблице запись можно изменить, для этого надо выбрать запись и нажать на кнопку «Редактировать». Откроется панель редактирования, идентичная панели ввода. После редактирования надо нажать на кнопку «Изменить». В итоге запись в таблице будет изменена (см. рисунок 19).

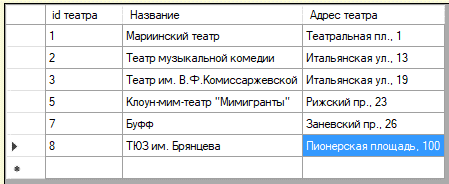


Рисунок 19 – Результат редактирования данных

Также возможно удаление существующей записи. Для этого надо выбрать нужную запись и нажать на кнопку «Удалить», появится окно подтверждения удаления записи (см. рисунок 20). После подтверждения удаления записи она будет удалена (см. рисунок 21).

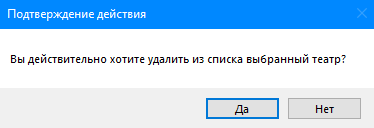


Рисунок 20 – Подтверждение удаления записи

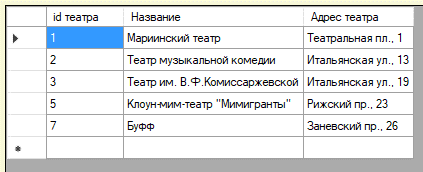


Рисунок 21 – Результат удаления записи

**Тестирование с помощью тест-кейсов**

|  |  |
| --- | --- |
| **Название проекта** | Театральная касса |
| **Номер версии** | 1.0 |
| **Имя тестера** | Никита |
| **Даты тестирования** | **05.06.18** |

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Наименование проекта** | Театральная касса |
| **Номер версии** | 1.0 |
| **Имя тестера** | Никита |
| **Даты тестирования** | 05.06.18 |
| **Test Case #** | TC\_1 |
| **Приоритет тестирования** *(Малы /Средний /высокий)* | Высокий |
| **Название тестирования/Имя** | Проверка формы входа с вводом логина и пароля |
| **Резюме испытания** | При нажатии на кнопку «Вход» должно отобразиться приветственное окно, подтверждающее вход в систему. |
| **Шаги тестирования** | 1. Действие: запустить программу.  2. Событие: открытие окна входа в систему.  3. Действие: ввод данных для входа (логин и пароль). Нажатие на кнопку входа.  4. Событие: вход в систему. Отображение приветственного окна. |
| **Данные тестирования** | Логин: nikita.bokarev  Пароль: 1111 |
| **Ожидаемый результат** | Должен быть произведен вход в систему, показ окна приветствия с кнопкой «ОК». |
| **Фактический результат** | Выполнен вход в систему, показано окно приветствия с кнопкой «ОК». |
| **Предпосылки** | Предварительных действий нет |
| **Постусловия** | Открыто окно входа в систему.  Открыто окно с приветствием пользователя. |
| **Статус** *(Pass/Fail)* | Pass |
| **Комментарии** | Комментариев нет |

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Наименование проекта** | Театральная касса |
| **Номер версии** | 1.0 |
| **Имя тестера** | Никита |
| **Даты тестирования** | 05.06.18 |
| **Test Case #** | TC\_2 |
| **Приоритет тестирования** *(Малы /Средний /высокий)* | Средний |
| **Название тестирования/Имя** | Проверка оформления заказа |
| **Резюме испытания** | В таблице «Заказы» должна появиться новая запись, содержащая информацию о заказе. Также должны появиться записи в таблице с билетами, в соответствии с их количеством. |
| **Шаги тестирования** | 1. Действие: нажать на кнопку «Добавить заказ».  2. Событие: открытие панели с полями для добавления заказа.  3. Действие: заполнение всех полей нужными данными (Дата спектакля, Спектакль, Количество, Ряд, Место) и нажатие на кнопку «Добавить».  4. Событие: Отображение окна с подтверждением добавления записи. |
| **Данные тестирования** | Дата спектакля: 30.06.2018  Спектакль: Лебединое озеро  Количество: 1  Ряд: 10  Место: 15 |
| **Ожидаемый результат** | Успешное добавление нового заказа.  Добавление в таблицу «Заказы» записи с информацией о сделанном заказе. Добавление в таблицу с билетами записей, в соответствии с их количеством. |
| **Фактический результат** | Успешное добавление нового заказа.  Добавление в таблицу «Заказы» записи с информацией о сделанном заказе. Добавление в таблицу с билетами записей, в соответствии с их количеством. |
| **Предпосылки** | 1. Запуск программы.  2. Вход в систему.  3. Нажатие на кнопку «Заказы», которая откроет форму с заказами. |
| **Постусловия** | Открыта форма с главным меню.  Открыта форма «Заказы».  Открыто окно с подтверждением об успешном добавлении заказа. |
| **Статус** *(Pass/Fail)* | Pass |
| **Комментарии** | Комментариев нет |

2.7. Инструкция пользователя

Для начала работы запустите файл boxOffice.exe.

**Форма входа**

После запуска программы открывается форма входа в систему. На ней есть два поля для ввода логина и пароля. Это окно показано на рисунке 22.

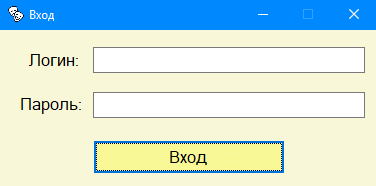


Рисунок 22 – Форма входа

В случае неправильного ввода пары логин-пароль будет выведено соответствующее сообщение (см. рисунок 23).

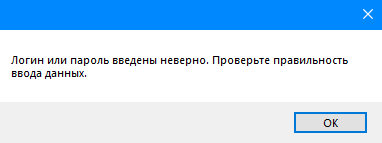


Рисунок 23 – Результат неправильного ввода данных

В случае корректного ввода данных для входа программа покажет приветственное сообщение и откроет главное меню (см. рисунок 24).

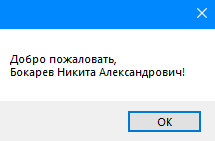


Рисунок 24 – Окно приветствия

**Главная форма**

На главной форме отображается меню, дающее доступ к другим частям программы и кнопка «Выход», позволяющая совершить выход из программы. Вид главной формы показан на рисунке 25.

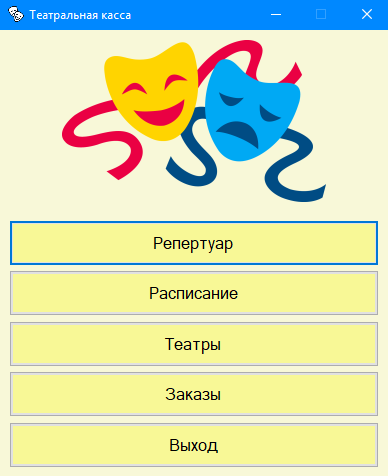


Рисунок 25 – Главная форма

**Форма «Репертуар»**

Форма «Репертуар» содержит таблицу с репертуаром и четыре кнопки: «Добавить», «Редактировать», «Удалить» и «Создать отчет о всех спектаклях»,   
а также поля для поиска и фильтрации (см. рисунок 26).

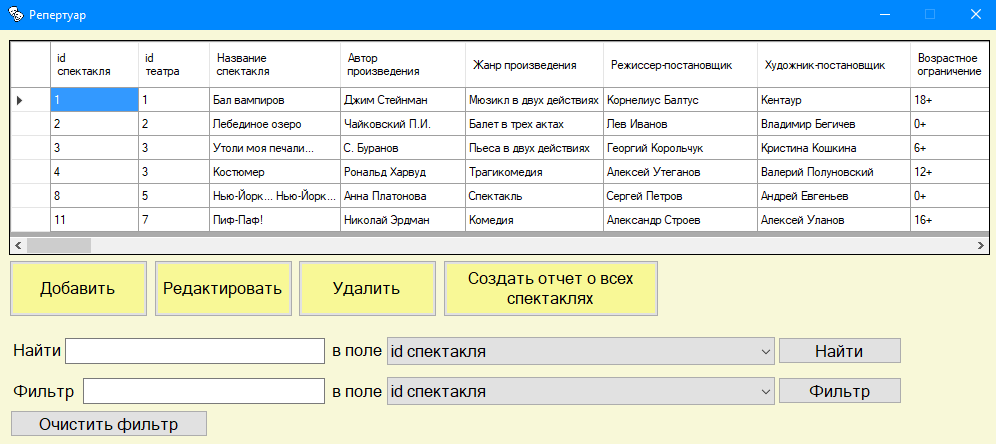


Рисунок 26 – Форма «Репертуар»

При нажатии на кнопку «Добавить» открывается панель с полями для добавления данных в таблицу репертуара (см. рисунок 27).

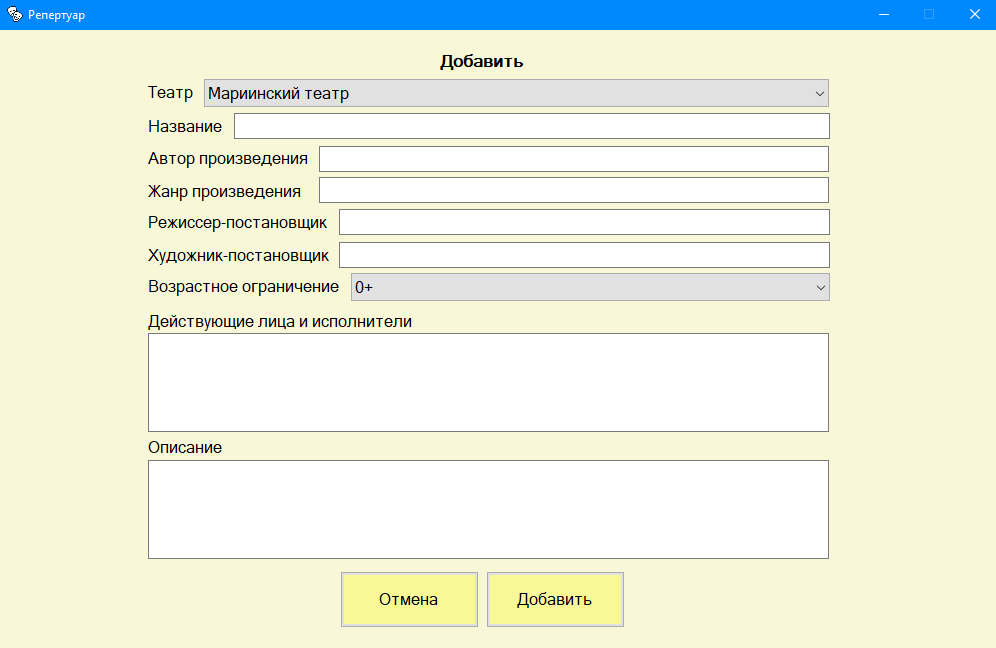


Рисунок 27 – Добавление данных в таблицу репертуара

Если заполнить все поля и нажать на кнопку «Добавить» отобразится сообщение об успешном добавлении записи в таблицу (см. рисунок 28).

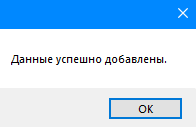


Рисунок 28 – Сообщение об успешном добавлении данных

Если при нажатии на кнопку «Добавить» будут заполнены не все поля, то отобразиться сообщение с просьбой заполнить все поля (см. рисунок 29).

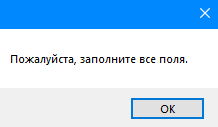


Рисунок 29 – Сообщение с просьбой заполнить все поля

При нажатии на кнопку «Отмена» закроется панель и отобразится форма «Репертуар».

При нажатии на кнопку «Редактировать» открывается панель с полями для редактирования данных выбранной записи из таблицы репертуара (см. рисунок 30).

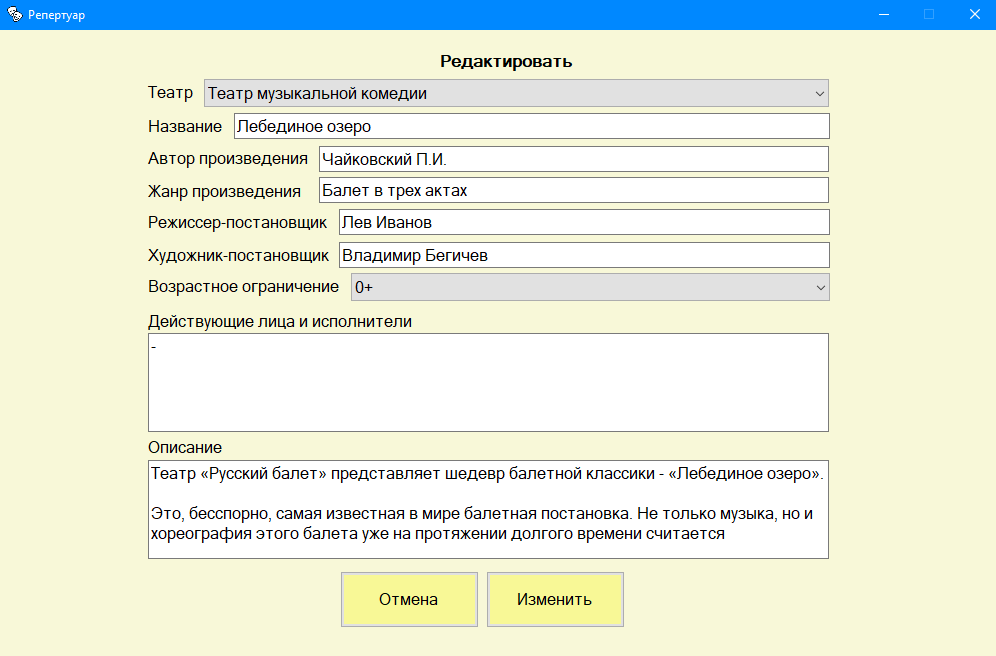


Рисунок 30 – Редактирование данных

Если нажать на кнопку «Изменить» отобразиться сообщение об успешном изменении записи и запись будет отредактирована (см. рисунок 31).

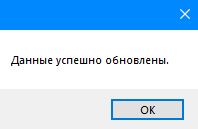


Рисунок 31 – Сообщение об успешном изменении данных

Если нажать на кнопку «Отмена» закроется панель и отобразится форма «Репертуар».

Если нажать на кнопку «Удалить» выведется сообщение о подтверждении удаления выбранной записи (см. рисунок 32).

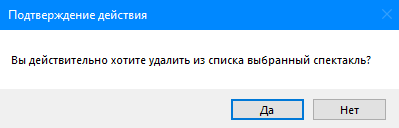


Рисунок 32 – Подтверждение удаления записи

Для подтверждения удаления записи надо выбрать «Да», в противном случае выбрать «Нет».

Если нажать на кнопку «Создать отчет о всех спектаклях», то откроется полный отчет в программе Excel с полным списком всех спектаклей и информацией о них (см. рисунок 33).

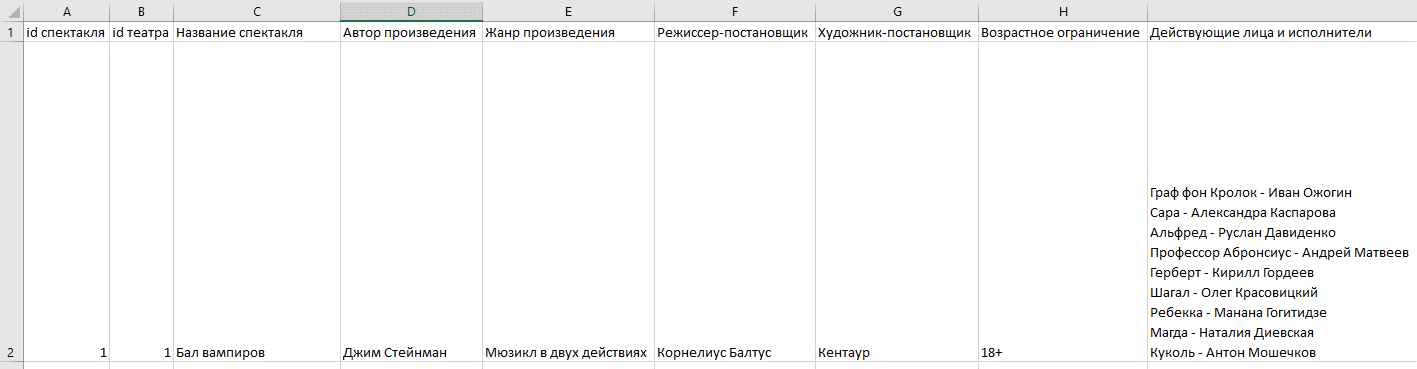


Рисунок 33 – Отчет о всех спектаклях

**Форма «Расписание»**

Форма «Расписание» содержит таблицу с расписанием и четыре кнопки: «Добавить», «Редактировать», «Удалить» и «Создать полный отчет о расписании», а также поля для поиска и фильтрации (см. рисунок 34).

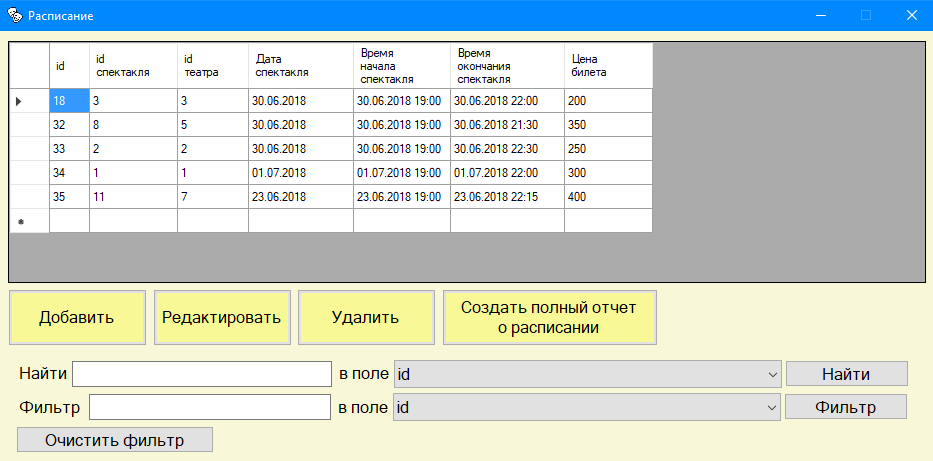


Рисунок 34 – Форма «Расписание»

При нажатии на кнопку «Добавить» открывается панель с полями для добавления данных в таблицу расписания (см. рисунок 35).

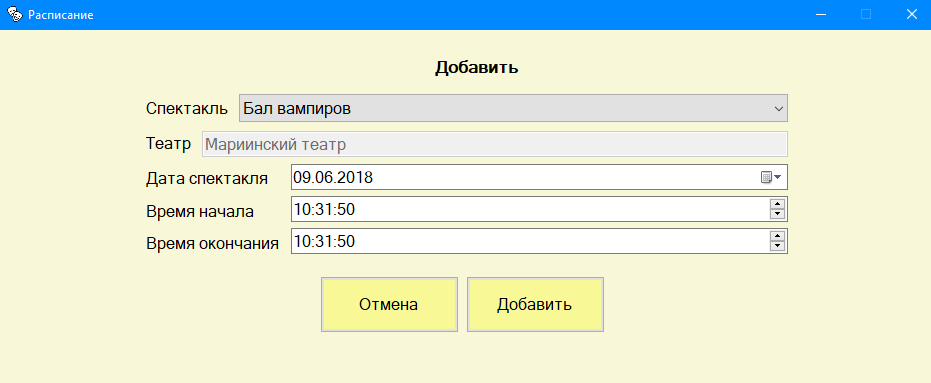


Рисунок 35 – Добавление данных в таблицу расписания

Если заполнить все поля и нажать на кнопку «Добавить» отобразится сообщение об успешном добавлении записи в таблицу (см. рисунок 36).

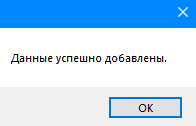


Рисунок 36 – Сообщение об успешном добавлении данных

Если при нажатии на кнопку «Добавить» будут заполнены не все поля, то отобразиться сообщение с просьбой заполнить все поля (см. рисунок 37).

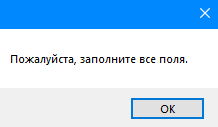


Рисунок 37 – Сообщение с просьбой заполнить все поля

При нажатии на кнопку «Отмена» закроется панель и отобразится форма «Расписание».

При нажатии на кнопку «Редактировать» открывается панель с полями для редактирования данных выбранной записи из таблицы расписания (см. рисунок 38).

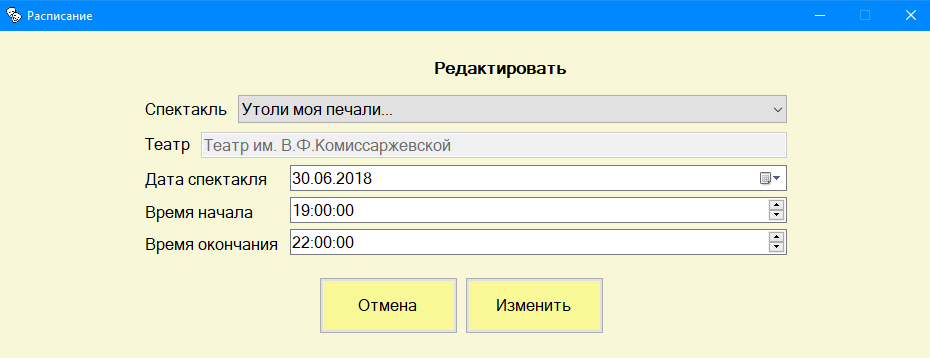


Рисунок 38 – Редактирование данных

Если нажать на кнопку «Изменить» отобразиться сообщение об успешном изменении записи и запись будет отредактирована (см. рисунок 39).

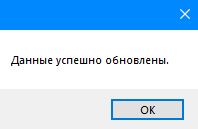


Рисунок 39 – Сообщение об успешном изменении данных

Если нажать на кнопку «Отмена» закроется панель и отобразится форма «Расписание».

Если нажать на кнопку «Удалить» выведется сообщение о подтверждении удаления выбранной записи (см. рисунок 40).

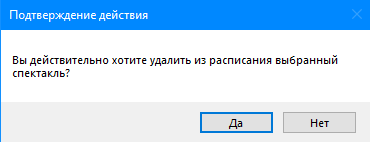


Рисунок 40 – Подтверждение удаления записи

Для подтверждения удаления записи надо выбрать «Да», в противном случае выбрать «Нет».

Если нажать на кнопку «Создать полный отчет о расписании», то откроется полный отчет в программе Excel с полным списком всего расписания и информацией о нем (см. рисунок 41).

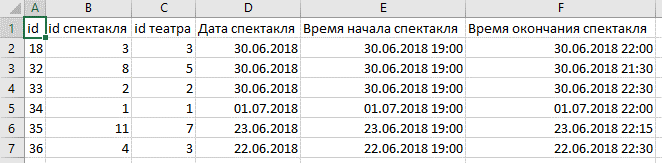


Рисунок 41 – Отчет о расписании

**Форма «Театры»**

Форма «Театры» содержит таблицу со списком театров и четыре кнопки: «Добавить», «Редактировать», «Удалить» и «Создать полный отчет о театрах»,   
а также поля для поиска и фильтрации (см. рисунок 42).

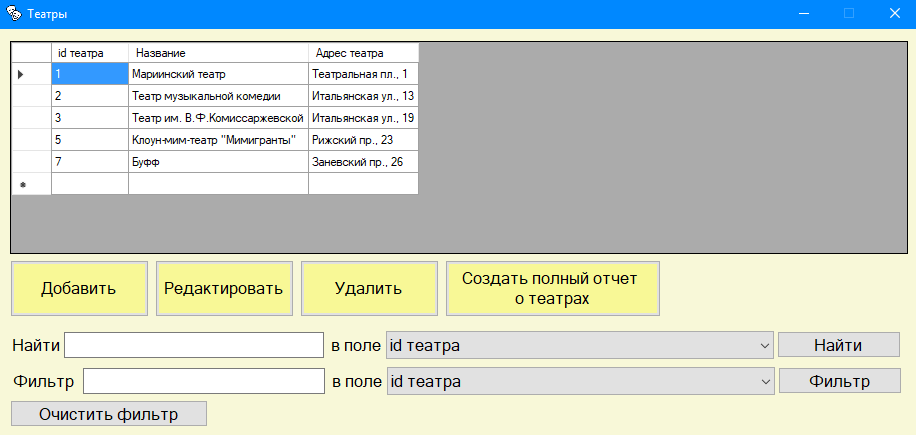


Рисунок 42 – Форма «Театры»

При нажатии на кнопку «Добавить» открывается панель с полями для добавления данных в таблицу театров (см. рисунок 43).

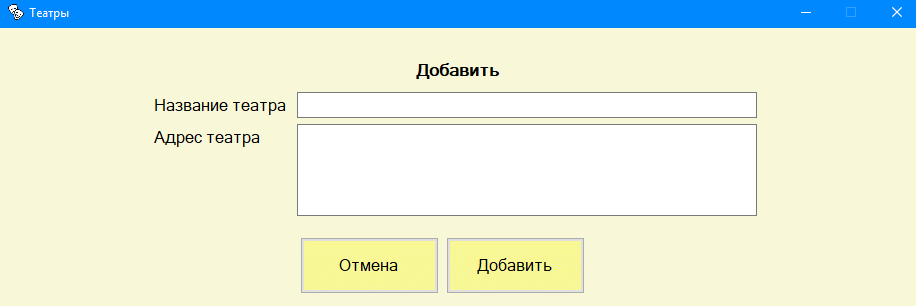


Рисунок 43 – Добавление данных в таблицу театров

Если заполнить все поля и нажать на кнопку «Добавить» отобразится сообщение об успешном добавлении записи в таблицу (см. рисунок 44).

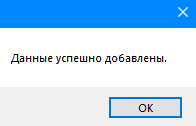


Рисунок 44 – Сообщение об успешном добавлении данных

Если при нажатии на кнопку «Добавить» будут заполнены не все поля, то отобразиться сообщение с просьбой заполнить все поля (см. рисунок 45).

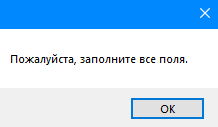


Рисунок 45 – Сообщение с просьбой заполнить все поля

При нажатии на кнопку «Отмена» закроется панель и отобразится форма «Театры».

При нажатии на кнопку «Редактировать» открывается панель с полями для редактирования данных выбранной записи из таблицы театров (см. рисунок 46).

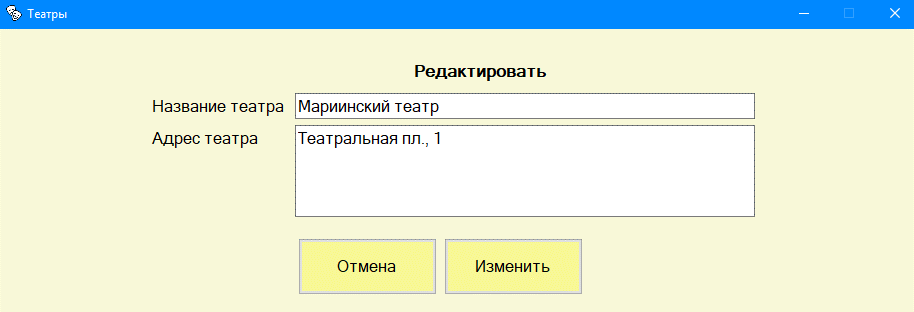


Рисунок 46 – Редактирование данных

Если нажать на кнопку «Изменить» отобразиться сообщение об успешном изменении записи и запись будет отредактирована (см. рисунок 47).

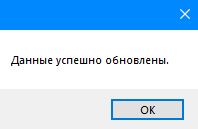


Рисунок 47 – Сообщение об успешном изменении данных

Если нажать на кнопку «Отмена» закроется панель и отобразится форма «Театры».

Если нажать на кнопку «Удалить» выведется сообщение о подтверждении удаления выбранной записи (см. рисунок 48).

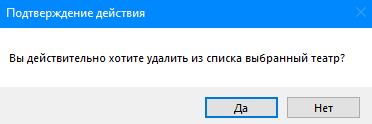


Рисунок 48 – Подтверждение удаления записи

Для подтверждения удаления записи надо выбрать «Да», в противном случае выбрать «Нет».

Если нажать на кнопку «Создать полный отчет о театрах», то откроется полный отчет в программе Excel с полным списком всех театров и информации о них (см. рисунок 49).

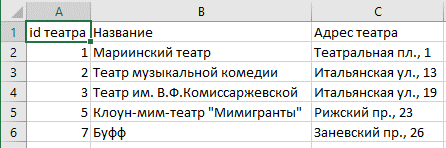


Рисунок 49 – Отчет о театрах

**Форма «Заказы»**

Форма «Заказы» содержит таблицу со списком заказов и четыре кнопки: «Добавить», «Редактировать», «Удалить» и «Создать полный отчет о приобретенных билетах», а также поля для поиска и фильтрации (см. рисунок 50).

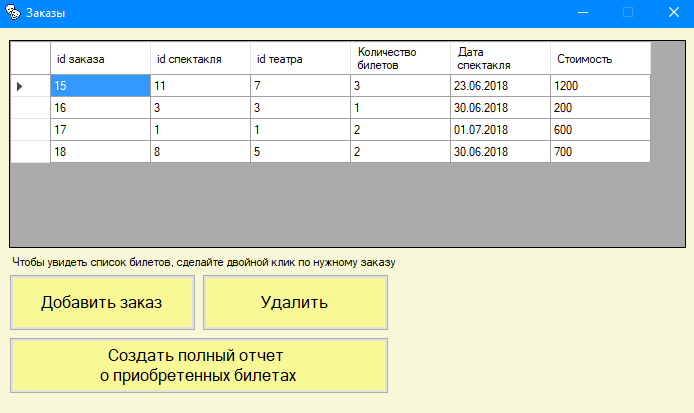


Рисунок 50 – Форма «Заказы»

При нажатии на кнопку «Добавить» открывается панель с полями для добавления данных в таблицу заказов (см. рисунок 51).

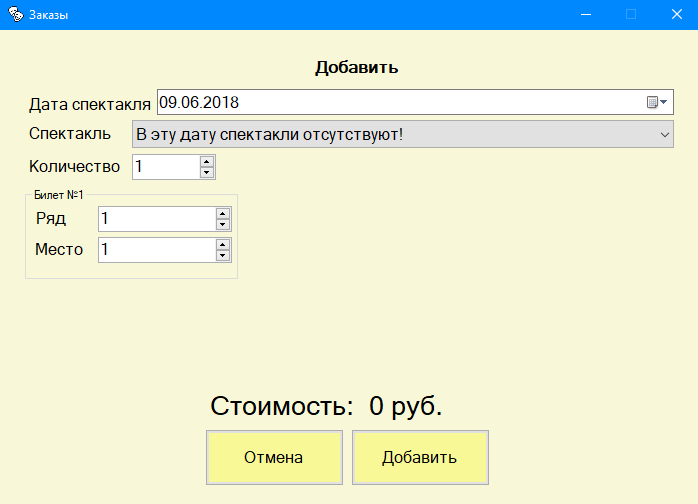


Рисунок 51 – Добавление данных в таблицу репертуар

Если заполнить все поля и нажать на кнопку «Добавить» отобразится сообщение об успешном добавлении записи в таблицу (см. рисунок 52).

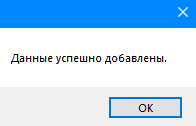


Рисунок 52 – Сообщение об успешном добавлении данных

Если при нажатии на кнопку «Добавить» будут заполнены не все поля, то отобразиться сообщение с просьбой заполнить все поля (см. рисунок 53).

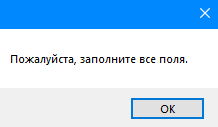


Рисунок 53 – Сообщение с просьбой заполнить все поля

При нажатии на кнопку «Отмена» закроется панель и отобразится форма «Заказы».

Если нажать на кнопку «Удалить» выведется сообщение о подтверждении удаления выбранной записи (см. рисунок 54).

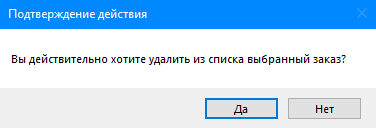


Рисунок 54 – Подтверждение удаления записи

Для подтверждения удаления записи надо выбрать «Да», в противном случае выбрать «Нет».

Если выполнить двойной клик на строке с нужным заказом, то откроется список билетов, оформленных в этом заказе и появятся две кнопки (см. рисунок 55): «Назад» для возвращения к списку заказов и «Распечатать билет» для открытия билета в программе Microsoft Office Excel и его последующей печати   
(см. рисунок 56).

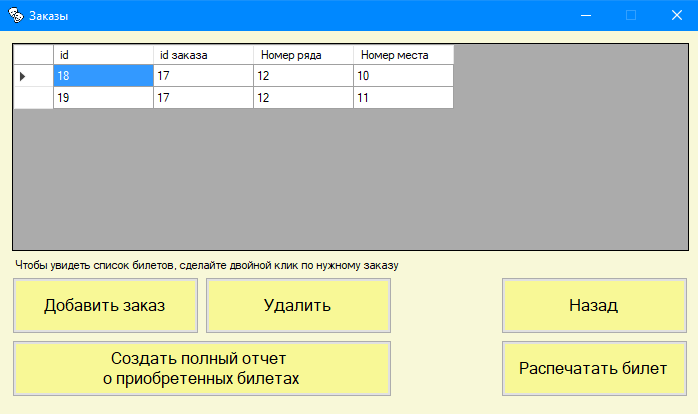


Рисунок 55 – Список билетов в заказе

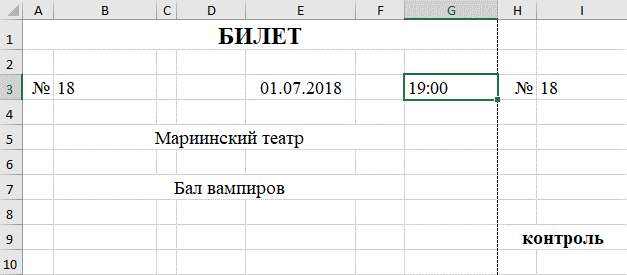


Рисунок 56 – Вывод билета в Excel

Если нажать на кнопку «Создать полный отчет о приобретенных билетах», то откроется полный отчет в программе Excel с расчетом итоговой суммы в конце отчета (см. рисунок 57).

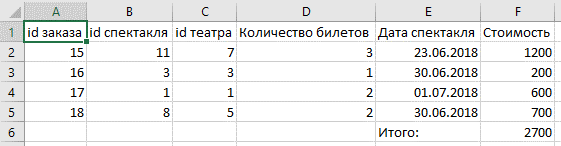


Рисунок 57 – Отчет о приобретенных билетах

2.8. Анализ результатов

В ходе разработки дипломного проекта была создана АИС «Театральная касса». Функционал данной программы предназначен для автоматизированного решения повседневных задач сотрудника отдела продаж театральной кассы, в обязанности которого входит продажа театральных билетов, консультация по спектаклям, а также ведение отчетности о проделанной работе.

В программе есть несколько справочников, информация о билетах, а также возможность создания отчетов, помогающих с анализом работы театральной кассы. Данная программа может быть использована в работе любой театральной кассы.

Программа обладает простым и понятным интерфейсом, в ее использовании без труда разберется как студент, так и взрослый человек.

Во время разработки дипломного проекта был получен опыт в работе по данной теме, так же были освоены основные приемы работы с базами данных в среде программирования Microsoft Visual Studio 2017. Важным аспектом было изучение новых компонентов, с помощью которых осуществлялась связь и отображение информации базы данных, а также вывод информации в другие программы в виде отчетов, позволяющие отправить данные на печать или послать их в дальнейшую работу, что значительно расширит функционал данной программы.

Были реализованы основные способы манипуляции с базами данных, а именно: просмотр, добавление, редактирование, поиск и фильтрация по заданным критериям.

В ходе работы были реализованы все необходимые операции с таблицами баз данных.

1. ЭКОНОМИЧЕСКАЯ ЧАСТЬ. РАСЧЕТ ПРЕДПОЛОГАЕМОЙ ПРИБЫЛИ ОТ РЕАЛИЗАЦИИ СОЗДАННОГО ПРОГРАММНОГО ПРОДУКТА

Данный проект был разработан для кассиров в театральных кассах. Работа программы начинается с авторизации, где сотрудник вводит данные для входа в систему. Далее открывается главная форма, откуда начинается работа с данной программой. В ней можно открыть другие формы системы, такие как: репертуар, расписание, театры, заказы.

3.1. Расчет затрат на выполнение проекта

Разработка данной программы, так же, как и любой другой, требует определенных материальных, временных и трудовых затрат, а, следовательно, должна соответственно окупаться. С экономической точки зрения затраты, связанные с выполнением проекта, должны быть покрыты доходами от реализации конечного продукта.

Трудоемкость выполнения программного продукта характеризуется перечнем основных этапов и видов работ, которые должны быть выполнены. Упорядочен данный перечень в соответствии со смысловым содержанием каждого вида работ и взаимосвязями между всеми видами работ.

Форма расположения работ по этапам:

1. Разработка технического задания (ТЗ):

* получение ТЗ.

2. Подготовительный этап:

* сбор информации;
* выбор объектного построения программы;
* разработка общей методики создания продукта.

3. Основной этап:

* разработка основного алгоритма;
* создание интерфейса;
* отладка.

4. Завершающий этап:

* подготовка технической документации;
* сдача продукта.

Трудоемкость выполнения работы оценивается в человеко-часах и носит вероятностный характер.

Расчет трудоемкости по проекту приведен в таблице 7.

**Таблица 7 – Расчет трудоемкости по проекту**

|  |  |
| --- | --- |
| Виды работ | Трудоемкость, чел/час. |
| 1. Получение ТЗ | 2 |
| 2. Сбор информации и ознакомление с предметной областью | 24 |
| 3. Выбор объектного построения программы | 12 |
| 4. Разработка общей методики создания продукта | 18 |
| 5. Разработка основного алгоритма | 60 |
| 6. Создание интерфейса | 26 |
| 7. Отладка | 18 |
| 8. Подготовка технической документации | 14 |
| 9. Сдача продукта | 1 |
| Итого | 175 |

Суммарная трудоемкость составила 175 чел./час.

3.2. Расчет расходов на заработную плату

На основе данных о трудоемкости и средней заработной плате по отрасли рассчитываем основную заработную плату. Предположим, что заработная плата программиста без опыта работы составляет 18000 тыс. руб. в месяц. (21 рабочий день, 8 часовой рабочий день) или 107,14 руб./час.

Таким образом, расходы на заработную плату по нашему проекту составляют:

ЗП = 107,14 \* 175 = 18749,50 руб.

Отчисления на ЗП (Пенсионный Фонд, Фонд соц. страхования, Фонд обязательного мед. страхования, территориальные фонды мед. страхования) составляют 30%.

В денежном выражении составляют: 18749,50 \* 0,3 = 5624,85 руб.

3.3. Расчет затрат на материалы

Расчет затрат на материалы приведен в таблице 8.

**Таблица 8 – Расчет затрат на материалы**

|  |  |  |  |
| --- | --- | --- | --- |
| Расходные материалы | Количество | Цена за единицу (руб.) | Сумма (руб.) |
| 1. Интернет | 1 месяц | 510 | 510 |
| 2. Бумага А4 для принтера | 1 упаковка | 220 | 220 |
| 3. Картридж для принтера | 1 штука | 970 | 970 |
| 4. Папка | 1 штука | 180 | 180 |
| 5. Файлы | 200 штук | 1 | 200 |
| Итого |  |  | 2080 |

3.4. Расчет электроэнергии и амортизации оборудования

Стоимость компьютера 25 тыс. рублей. Используем его в течении 5 лет   
(60 мес.). Годовая амортизация составит в месяц 416,6 рублей. В час амортизация составит: 2,85 рублей. Умножив на трудоемкость, определим:

АО = 2,85 \* 175 = 498,75 руб.

Компьютер в среднем потребляет 0,5 кВт/ч. 0,5 \* 175 = 87,5 кВт. Примерная стоимость кВт/ч составляет 4,23 руб.

Итого: 87,5 \* 4,23 = 370,13 руб.

3.5. Расчет сметы затрат

Смета затрат приведена в таблице 9.

**Таблица 9 – Смета всех затрат**

|  |  |
| --- | --- |
| Наименование статей затрат | Сумма (руб.) |
| 1. Расходные материалы | 2080 |
| 2. Основная заработная плата | 18749,50 |
| 3. Расходы на электроэнергию | 370,13 |
| 4. Амортизационные отчисления | 498,75 |
| 5. Отчисления на ЗП | 5624,85 |
| Итого | 27323,23 |

3.6. Расчет экономического эффекта

Экономическим эффектом данного проекта является предполагаемая прибыль, полученная в результате реализации программного продукта.

Суммарная стоимость затрат на создание данного программного продукта составляет 27323,23 рублей, следовательно, эту программу ниже этой цены продать нельзя так, как в этом случае программа не будет являться экономически эффективной. Созданный программный продукт актуален для организаций, занимающихся продажей театральных билетов.

Главными достоинствами созданного программного продукта являются:

* удобный и простейший интерфейс, в котором приятно и очень просто работать;
* разработан поиск и фильтрация определенных данных.

Рыночная цена аналогичного программного продукта составляет 35000 руб. Таким образом, реализовав наш программный продукт, мы можем получить прибыль в размере 7676,77 руб.

Данная программа при сравнительно небольших затратах 27323,23 руб., позволяет автоматизировать и максимально упростить работу кассира театральной кассы.

**ЗАКЛЮЧЕНИЕ**

В результате работы над дипломным проектом была разработана информационная система для театральных касс. В программе разработано 6 основных модулей:

* модуль «Вход»;
* модуль «Главное меню»;
* модуль «Репертуар»;
* модуль «Расписание»;
* модуль «Театры»;
* модуль «Заказы».

Программа позволяет систематизировать деятельность кассира театральной кассы, снизить к минимуму ошибки во время оформления заказов на билеты, а также ускорить консультирование клиентов по спектаклям, театрам и репертуару, благодаря наглядной демонстрации описания спектакля.

Достоинствами программы являются:

* простой и интуитивно понятный интерфейс;
* минимальное количество ручного ввода информации;
* занимает небольшое пространство на жестком диске;
* возможность создавать и выводить на печать отчеты;
* обеспечивает фильтрацию и поиск данных таблиц.

Недостатками программы являются:

* отсутствие разделения прав доступа;
* отсутствие возможности оформить в одном заказе более 6 билетов.

В ходе написания дипломного проекта были изучены свойства и методы программирования на языке C# в среде программирования Microsoft Visual Studio 2017, которые позволяют создавать приложения, работающие с полноценными базами данных, обращающиеся к данным через ADO технологию.

Разработанный проект может использоваться театральными кассами для консультирования клиентов и реализации билетов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

**Основная**

1. Биллинг В.А. Основы объектного программирования на C#. М.: Бином. Лаборатория знаний, Интернет-университет информационных технологий, 2013. – 584 с.
2. Джон Шарп Microsoft Visual C#. Подробное руководство. СПб.: Питер, 2017. – 848 с.
3. Хорев П.Б. Объектно-ориентированное программирование с примерами на C#. СПб.: Форум, 2016. – 200 с.
4. Виссер Джуст Разработка обслуживаемых программ на языке C#. М.: ДМК Пресс, 2017. – 192 с.
5. Г. Шилдт Полный справочник по С. М.: Вильямс, 2014. – 800 с.
6. Павловская Т.А.: С. Структурное и объектно-ориентированное программирование. СПб.: Питер, 2010. – 352 с.

**Дополнительная**

1. https://support.office.com/ru-ru/article/%D0%A1%D0%BE%D0%B7%D0%  
   B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B1%D0%B0%D0%B7%D1%8B-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-32a1ea1c-a155-43d6-aa00-f08cd1a8f01e – руководство по созданию базы данных в СУБД Access.
2. <https://msdn.microsoft.com/library/microsoft.office.interop.excel.aspx> – информация о пространстве имен Microsoft.Office.Interop.Excel.
3. <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/interop/walkthrough-office-programming> – пошаговое руководство. Программирование приложений Office.
4. <https://msdn.microsoft.com/ru-ru/library/system.windows.forms.datagridview(v=vs.110).aspx> – элемент DataGrid View.
5. http://bigcamagan.ru – информация о свойствах компонетов.
6. <https://msdn.microsoft.com/ru-ru/library/system.data.oledb(v=vs.110).aspx> – информация о пространстве имен System.Data.OleDb.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace boxOffice

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new loginForm());

}

}

}

**staticVariables.cs – класс для хранения публичных данных**

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace boxOffice

{

public static class staticVariables

{

public static OleDbConnection con = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=..\..\boxOfficeDB.mdb;Jet OLEDB:Database Password=boxOfficeDB");

public static int userId;

public static string userFio = "";

public static string userLogin = "";

}

}

**loginForm.cs – вход в систему**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace boxOffice

{

public partial class loginForm : Form

{

public loginForm()

{

InitializeComponent();

}

private void loginButton\_Click(object sender, EventArgs e) // действия при клике на кнопку "Вход"

{

string login = loginTextbox.Text;

login = login.Trim();

string password = passwordTextbox.Text;

OleDbConnection con = staticVariables.con;

try

{

con.Open();

OleDbCommand cmd = new OleDbCommand("select [id пользователя], ФИО, Логин, Пароль from Пользователь where Логин=:login and Пароль=:password", con);

cmd.Parameters.Add("login", OleDbType.VarChar).Value = login;

cmd.Parameters.Add("password", OleDbType.VarChar).Value = password;

OleDbDataReader reader = cmd.ExecuteReader();

if(reader.Read())

{

staticVariables.userId = reader.GetInt32(0);

staticVariables.userFio = reader.GetString(1);

staticVariables.userLogin = reader.GetString(2);

MessageBox.Show("Добро пожаловать, " + staticVariables.userFio + "!");

reader.Close();

con.Close();

loginTextbox.Text = "";

passwordTextbox.Text = "";

this.Hide();

}

else

{

MessageBox.Show("Логин или пароль введены неверно. Проверьте правильность ввода данных.");

reader.Close();

con.Close();

return;

}

}

catch

{

MessageBox.Show("Логин или пароль введены неверно. Проверьте правильность ввода данных.");

con.Close();

return;

}

mainForm mainForm1 = new mainForm();

mainForm1.Show();

}

}

}

**mainForm.cs – главная форма с меню**

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 boxOffice

{

public partial class mainForm : Form

{

public mainForm()

{

InitializeComponent();

}

private void mainForm\_Load(object sender, EventArgs e)

{

Bitmap theatreLogo = new Bitmap(@"../../resources/logo.png");

theatreLogoPB.Image = theatreLogo;

}

private void mainForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Application.Exit();

}

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

{

Application.Exit();

}

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

{

repertoireForm repertoireForm1 = new repertoireForm();

repertoireForm1.ShowDialog();

}

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

{

scheduleForm scheduleForm1 = new scheduleForm();

scheduleForm1.ShowDialog();

}

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

{

theatresForm theatresForm1 = new theatresForm();

theatresForm1.ShowDialog();

}

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

{

ordersForm ordersForm1 = new ordersForm();

ordersForm1.ShowDialog();

}

}

}

**repertoireForm.cs – форма «Репертуар»**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

namespace boxOffice

{

public partial class repertoireForm : Form

{

public repertoireForm()

{

InitializeComponent();

}

public void base\_load()

{

DataSet ds = new DataSet();

OleDbConnection con = staticVariables.con;

OleDbCommand com = new OleDbCommand("select \* from Репертуар", con);

con.Open();

OleDbDataAdapter adapter = new OleDbDataAdapter();

adapter.SelectCommand = com;

adapter.Fill(ds);

System.Data.DataTable dt = ds.Tables[0];

repertoireDataGridView.DataSource = dt;

con.Close();

repertoireDataGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

}

private void repertoireForm\_Load(object sender, EventArgs e)

{

repertoirePanel.Hide();

this.Height = 455;

List<string> theatres = new List<string>();

OleDbConnection con = staticVariables.con;

try

{

con.Open();

OleDbCommand cmd = new OleDbCommand("select Название from Театр", con);

OleDbDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

theatres.Add(reader.GetString(0));

}

reader.Close();

con.Close();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

for (int i = 0; i < theatres.Count; i++)

{

theatresCombobox.Items.Add(theatres[i]);

}

theatresCombobox.SelectedIndex = 0;

string[] ageLimit = { "0+", "6+", "12+", "16+", "18+" };

ageLimitCombobox.Items.AddRange(ageLimit);

ageLimitCombobox.SelectedIndex = 0;

base\_load();

for(int i = 0; i < repertoireDataGridView.Columns.Count; i++)

{

fieldsCombobox.Items.Add(repertoireDataGridView.Columns[i].HeaderText);

filterCombobox.Items.Add(repertoireDataGridView.Columns[i].HeaderText);

}

fieldsCombobox.SelectedIndex = 0;

filterCombobox.SelectedIndex = 0;

ToolTip tooltip = new ToolTip();

tooltip.SetToolTip(this.repertoireReportButton, "Отчет откроется в программе Excel");

}

int theatreId = 0;

bool isPerformed = false;

public void getTheatreID()

{

OleDbConnection con = staticVariables.con;

theatreId = 0;

try

{

OleDbCommand cmd = new OleDbCommand("select [id театра] from Театр where [Название]=@name", con);

cmd.Parameters.Add("@name", OleDbType.VarChar).Value = theatresCombobox.Text;

if (!isPerformed)

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

theatreId = reader.GetInt32(0);

}

reader.Close();

if (!isPerformed)

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось получить id выбранного театра.\nОшибка: " + ex.Message);

if (!isPerformed)

con.Close();

}

}

string theatreName = "";

public void getTheatreName(int id)

{

OleDbConnection con = staticVariables.con;

theatreId = 0;

try

{

OleDbCommand cmd = new OleDbCommand("select [Название] from Театр where [id театра]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = id;

if (!isPerformed)

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

theatreName = reader.GetString(0);

}

reader.Close();

if (!isPerformed)

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось получить id выбранного театра.\nОшибка: " + ex.Message);

if (!isPerformed)

con.Close();

}

}

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

{

headerLabel.Text = "Добавить";

submitButton.Text = "Добавить";

this.Height = 750;

repertoirePanel.Show();

}

public void clearFields()

{

theatresCombobox.SelectedIndex = 0;

perfomanceNameTextbox.Text = "";

authorOfWorkTextbox.Text = "";

genreOfWorkTextbox.Text = "";

directorTextbox.Text = "";

artDirectorTextbox.Text = "";

ageLimitCombobox.SelectedIndex = 0;

actorsTextbox.Text = "";

descriptionTextbox.Text = "";

}

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

{

repertoirePanel.Hide();

this.Height = 455;

clearFields();

}

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

{

OleDbConnection con = staticVariables.con;

getTheatreID();

if (submitButton.Text == "Добавить")

{

if(perfomanceNameTextbox.Text == "" || authorOfWorkTextbox.Text == "" ||

genreOfWorkTextbox.Text == "" || directorTextbox.Text == "" ||

artDirectorTextbox.Text == "" || actorsTextbox.Text == "" || descriptionTextbox.Text == "")

{

MessageBox.Show("Пожалуйста, заполните все поля.");

return;

}

try

{

OleDbCommand cmd = new OleDbCommand("insert into Репертуар ([id театра], [Название спектакля], [Автор произведения], [Жанр произведения], [Режиссер-постановщик], [Художник-постановщик], [Возрастное ограничение], [Действующие лица и исполнители], Описание) " +

"values (@theatreid, @name, @author, @genre, @director, @artdirector, @agelimit, @actors, @description)", con);

cmd.Parameters.Add("@theatreid", OleDbType.Integer).Value = theatreId;

cmd.Parameters.Add("@name", OleDbType.VarChar).Value = perfomanceNameTextbox.Text;

cmd.Parameters.Add("@author", OleDbType.VarChar).Value = authorOfWorkTextbox.Text;

cmd.Parameters.Add("@genre", OleDbType.VarChar).Value = genreOfWorkTextbox.Text;

cmd.Parameters.Add("@director", OleDbType.VarChar).Value = directorTextbox.Text;

cmd.Parameters.Add("@artdirector", OleDbType.VarChar).Value = artDirectorTextbox.Text;

cmd.Parameters.Add("@agelimit", OleDbType.VarChar).Value = ageLimitCombobox.Text;

cmd.Parameters.Add("@actors", OleDbType.VarChar).Value = actorsTextbox.Text;

cmd.Parameters.Add("@description", OleDbType.VarChar).Value = descriptionTextbox.Text;

con.Open();

cmd.ExecuteNonQuery();

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

repertoirePanel.Hide();

this.Height = 455;

clearFields();

con.Close();

base\_load();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

}

int currentCellColumn = 0;

int currentCellRow = 0;

int perfomanceID = 0;

if(submitButton.Text == "Изменить")

{

currentCellColumn = repertoireDataGridView.CurrentCell.ColumnIndex;

currentCellRow = repertoireDataGridView.CurrentCell.RowIndex;

perfomanceID = Convert.ToInt32(repertoireDataGridView[0, repertoireDataGridView.CurrentRow.Index].Value);

try

{

OleDbCommand cmd = new OleDbCommand("update Репертуар set [id театра]=@theatreid, [Название спектакля]=@perfomancename, [Автор произведения]=@author, " +

"[Жанр произведения]=@genre, [Режиссер-постановщик]=@director, [Художник-постановщик]=@artdirector, [Возрастное ограничение]=@agelimit, " +

"[Действующие лица и исполнители]=@actors, Описание=@description where [id спектакля]=@perfomanceid", con);

cmd.Parameters.Add("@theatreid", OleDbType.Integer).Value = theatreId;

cmd.Parameters.Add("@perfomancename", OleDbType.VarChar).Value = perfomanceNameTextbox.Text;

cmd.Parameters.Add("@author", OleDbType.VarChar).Value = authorOfWorkTextbox.Text;

cmd.Parameters.Add("@genre", OleDbType.VarChar).Value = genreOfWorkTextbox.Text;

cmd.Parameters.Add("@director", OleDbType.VarChar).Value = directorTextbox.Text;

cmd.Parameters.Add("@artdirector", OleDbType.VarChar).Value = artDirectorTextbox.Text;

cmd.Parameters.Add("@agelimit", OleDbType.VarChar).Value = ageLimitCombobox.Text;

cmd.Parameters.Add("@actors", OleDbType.VarChar).Value = actorsTextbox.Text;

cmd.Parameters.Add("@description", OleDbType.VarChar).Value = descriptionTextbox.Text;

cmd.Parameters.Add("@perfomanceid", OleDbType.Integer).Value = perfomanceID;

con.Open();

cmd.ExecuteNonQuery();

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

repertoirePanel.Hide();

this.Height = 455;

clearFields();

con.Close();

base\_load();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

}

repertoireDataGridView.CurrentCell = repertoireDataGridView[currentCellColumn, currentCellRow];

}

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

{

headerLabel.Text = "Редактировать";

submitButton.Text = "Изменить";

int perfomanceID = Convert.ToInt32(repertoireDataGridView[0, repertoireDataGridView.CurrentRow.Index].Value);

int theatreID = Convert.ToInt32(repertoireDataGridView[1, repertoireDataGridView.CurrentRow.Index].Value);

getTheatreName(theatreID);

OleDbConnection con = staticVariables.con;

try

{

OleDbCommand cmd = new OleDbCommand("select \* from Репертуар where [id спектакля]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = perfomanceID;

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if(reader.Read())

{

theatresCombobox.Text = theatreName;

perfomanceNameTextbox.Text = reader.GetString(2);

authorOfWorkTextbox.Text = reader.GetString(3);

genreOfWorkTextbox.Text = reader.GetString(4);

directorTextbox.Text = reader.GetString(5);

artDirectorTextbox.Text = reader.GetString(6);

ageLimitCombobox.Text = reader.GetString(7);

actorsTextbox.Text = reader.GetString(8);

descriptionTextbox.Text = reader.GetString(9);

}

reader.Close();

con.Close();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

this.Height = 750;

repertoirePanel.Show();

}

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

{

if (MessageBox.Show("Вы действительно хотите удалить из списка выбранный спектакль?", "Подтверждение действия", MessageBoxButtons.YesNo) == DialogResult.No)

return;

OleDbConnection con = staticVariables.con;

int perfomanceID = Convert.ToInt32(repertoireDataGridView[0, repertoireDataGridView.CurrentRow.Index].Value);

try

{

OleDbCommand cmd = new OleDbCommand("delete from Репертуар where [id спектакля]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = perfomanceID;

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

base\_load();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

}

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

{

try

{

for (int i = 0; i < repertoireDataGridView.Rows.Count; i++)

{

if (repertoireDataGridView[fieldsCombobox.SelectedIndex, i].Value.ToString() == findTextbox.Text)

{

repertoireDataGridView.CurrentCell = repertoireDataGridView[fieldsCombobox.SelectedIndex, i];

return;

}

}

}

catch

{

MessageBox.Show("Не удалось найти введенное значение.");

}

}

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

{

Excel.Application excelApp = new Excel.Application();

Excel.Workbook workbook;

Excel.Worksheet worksheet;

workbook = excelApp.Workbooks.Add(System.Reflection.Missing.Value);

worksheet = (Excel.Worksheet)workbook.Sheets[1];

worksheet.Cells[1, 1] = repertoireDataGridView.Columns[0].HeaderText;

worksheet.Cells[1, 2] = repertoireDataGridView.Columns[1].HeaderText;

worksheet.Cells[1, 3] = repertoireDataGridView.Columns[2].HeaderText;

worksheet.Cells[1, 4] = repertoireDataGridView.Columns[3].HeaderText;

worksheet.Cells[1, 5] = repertoireDataGridView.Columns[4].HeaderText;

worksheet.Cells[1, 6] = repertoireDataGridView.Columns[5].HeaderText;

worksheet.Cells[1, 7] = repertoireDataGridView.Columns[6].HeaderText;

worksheet.Cells[1, 8] = repertoireDataGridView.Columns[7].HeaderText;

worksheet.Cells[1, 9] = repertoireDataGridView.Columns[8].HeaderText;

worksheet.Cells[1, 10] = repertoireDataGridView.Columns[9].HeaderText;

for (int i = 0; i < repertoireDataGridView.Rows.Count; i++)

{

worksheet.Cells[i + 2, 1] = repertoireDataGridView[0, i] .Value;

worksheet.Cells[i + 2, 2] = repertoireDataGridView[1, i].Value;

worksheet.Cells[i + 2, 3] = repertoireDataGridView[2, i].Value;

worksheet.Cells[i + 2, 4] = repertoireDataGridView[3, i].Value;

worksheet.Cells[i + 2, 5] = repertoireDataGridView[4, i].Value;

worksheet.Cells[i + 2, 6] = repertoireDataGridView[5, i].Value;

worksheet.Cells[i + 2, 7] = repertoireDataGridView[6, i].Value;

worksheet.Cells[i + 2, 8] = repertoireDataGridView[7, i].Value;

worksheet.Cells[i + 2, 9] = repertoireDataGridView[8, i].Value;

worksheet.Cells[i + 2, 10] = repertoireDataGridView[9, i].Value;

}

worksheet.Columns.AutoFit();

excelApp.Visible = true;

excelApp.UserControl = true;

}

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

{

(repertoireDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Format("[" + filterCombobox.Text.ToString() + "]" + " = '{0}'", filterTextbox.Text);

}

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

{

(repertoireDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Empty;

}

}

}

**scheduleForm.cs – форма «Расписание»**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

namespace boxOffice

{

public partial class scheduleForm : Form

{

public scheduleForm()

{

InitializeComponent();

}

int theatreId;

public void base\_load()

{

DataSet ds = new DataSet();

OleDbConnection con = staticVariables.con;

OleDbCommand com = new OleDbCommand("select \* from Расписание", con);

con.Open();

OleDbDataAdapter adapter = new OleDbDataAdapter();

adapter.SelectCommand = com;

adapter.Fill(ds);

DataTable dt = ds.Tables[0];

scheduleDataGridView.DataSource = dt;

con.Close();

scheduleDataGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

}

public void clearFields()

{

perfomancesCombobox.SelectedIndex = 0;

perfomanceDateTimePicker.Value = DateTime.Today;

perfomanceStartsDateTimePicker.Value = DateTime.Now;

perfomanceEndsDateTimePicker.Value = DateTime.Now;

}

private void scheduleForm\_Load(object sender, EventArgs e)

{

base\_load();

schedulePanel.Hide();

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

OleDbConnection con = staticVariables.con;

try

{

OleDbCommand cmd = new OleDbCommand("select [Название спектакля] from Репертуар", con);

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

while(reader.Read())

{

perfomances.Add(reader.GetString(0));

}

reader.Close();

con.Close();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось получить список спектаклей.\nОшибка: " + ex.Message);

con.Close();

}

for(int i = 0; i < perfomances.Count; i++)

{

perfomancesCombobox.Items.Add(perfomances[i]);

}

perfomancesCombobox.SelectedIndex = 0;

for (int i = 0; i < scheduleDataGridView.Columns.Count; i++)

{

fieldsCombobox.Items.Add(scheduleDataGridView.Columns[i].HeaderText);

filterCombobox.Items.Add(scheduleDataGridView.Columns[i].HeaderText);

}

fieldsCombobox.SelectedIndex = 0;

filterCombobox.SelectedIndex = 0;

}

bool isPerformed = false;

string perfomanceName;

public void getPerfomanceName(int id)

{

OleDbConnection con = staticVariables.con;

theatreId = 0;

try

{

OleDbCommand cmd = new OleDbCommand("select [Название спектакля] from Репертуар where [id спектакля]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = id;

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

perfomanceName = reader.GetString(0);

}

reader.Close();

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось получить название спектакля.\nОшибка: " + ex.Message);

con.Close();

}

}

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

{

isPerformed = true;

headerLabel.Text = "Редактировать";

submitButton.Text = "Изменить";

OleDbConnection con = staticVariables.con;

int scheduleID = Convert.ToInt32(scheduleDataGridView[0, scheduleDataGridView.CurrentRow.Index].Value);

int perfomance\_id = Convert.ToInt32(scheduleDataGridView[2, scheduleDataGridView.CurrentRow.Index].Value);

getPerfomanceName(perfomance\_id);

try

{

OleDbCommand cmd = new OleDbCommand("select \* from Расписание where id=:id", con);

cmd.Parameters.Add("id", OleDbType.Integer).Value = scheduleID;

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

perfomancesCombobox.Text = perfomanceName;

perfomanceDateTimePicker.Value = reader.GetDateTime(4);

perfomanceStartsDateTimePicker.Value = reader.GetDateTime(5);

perfomanceEndsDateTimePicker.Value = reader.GetDateTime(6);

}

reader.Close();

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось получить данные.\nОшибка: " + ex.Message);

con.Close();

}

schedulePanel.Show();

isPerformed = false;

}

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

{

headerLabel.Text = "Добавить";

submitButton.Text = "Добавить";

schedulePanel.Show();

}

public void getTheatreID()

{

OleDbConnection con = staticVariables.con;

theatreId = 0;

try

{

OleDbCommand cmd = new OleDbCommand("select [id театра], [Название спектакля] from Репертуар", con);

if (!isPerformed)

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

if (reader.GetString(1) == perfomancesCombobox.Text)

{

theatreId = reader.GetInt32(0);

}

}

reader.Close();

if (!isPerformed)

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось найти выбранный театр.\nОшибка: " + ex.Message);

if (!isPerformed)

con.Close();

}

}

private void perfomancesCombobox\_SelectedIndexChanged(object sender, EventArgs e)

{

OleDbConnection con = staticVariables.con;

getTheatreID();

try

{

OleDbCommand cmd = new OleDbCommand("select Название from Театр where [id театра]=:theatreid", con);

cmd.Parameters.Add("theatreid", OleDbType.Integer).Value = theatreId;

if (!isPerformed)

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

while(reader.Read())

{

theatreTextbox.Text = reader.GetString(0);

}

reader.Close();

if (!isPerformed)

con.Close();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось определить театр, в котором проходит выбранный спектакль.\nОшибка: " + ex.Message);

if (!isPerformed)

con.Close();

}

}

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

{

schedulePanel.Hide();

clearFields();

}

int perfomanceID = 0;

public void getPerfomanceID()

{

OleDbConnection con = staticVariables.con;

perfomanceID = 0;

try

{

OleDbCommand cmd = new OleDbCommand("select [id спектакля] from Репертуар where [Название спектакля]=@name", con);

cmd.Parameters.Add("@name", OleDbType.VarChar).Value = perfomancesCombobox.Text;

if (!isPerformed)

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

perfomanceID = reader.GetInt32(0);

}

reader.Close();

if (!isPerformed)

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось получить id выбранного спектакля.\nОшибка: " + ex.Message);

if (!isPerformed)

con.Close();

}

}

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

{

OleDbConnection con = staticVariables.con;

if(submitButton.Text == "Добавить")

{

if(theatreTextbox.Text == "")

{

MessageBox.Show("Пожалуйста, заполните все поля.");

return;

}

try

{

getPerfomanceID();

getTheatreID();

OleDbCommand cmd = new OleDbCommand("insert into Расписание ([Название спектакля], [id спектакля], [id театра], [Дата спектакля], [Время начала спектакля], [Время окончания спектакля], [Цена билета]) " +

"values (@perfomancename, @perfomanceid, @theatreid, @perfomancedate, @starttime, @endtime, @cena)", con);

cmd.Parameters.Add("@perfomancename", OleDbType.VarChar).Value = perfomancesCombobox.Text;

cmd.Parameters.Add("@perfomanceid", OleDbType.Integer).Value = perfomanceID;

cmd.Parameters.Add("@theatreid", OleDbType.Integer).Value = theatreId;

cmd.Parameters.Add("@perfomancedate", OleDbType.DBDate).Value = perfomanceDateTimePicker.Value.ToShortDateString();

cmd.Parameters.Add("@starttime", OleDbType.DBTimeStamp).Value = perfomanceDateTimePicker.Value.Date + perfomanceStartsDateTimePicker.Value.TimeOfDay;

cmd.Parameters.Add("@endtime", OleDbType.DBTimeStamp).Value = perfomanceDateTimePicker.Value.Date + perfomanceEndsDateTimePicker.Value.TimeOfDay;

cmd.Parameters.Add("@cena", OleDbType.Integer).Value = Convert.ToInt32(ticketCostTextbox.Text);

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

schedulePanel.Hide();

clearFields();

base\_load();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось добавить данные.\nОшибка: " + ex.Message);

con.Close();

}

}

if(submitButton.Text == "Изменить")

{

int currentCellRow = scheduleDataGridView.CurrentCell.RowIndex;

int currentCellColumn = scheduleDataGridView.CurrentCell.ColumnIndex;

int scheduleID = Convert.ToInt32(scheduleDataGridView[0, scheduleDataGridView.CurrentRow.Index].Value);

try

{

getPerfomanceID();

getTheatreID();

OleDbCommand cmd = new OleDbCommand("update Расписание set [Название спектакля]=@perfomancename, [id спектакля]=@perfomanceid, [id театра]=@theatreid, [Дата спектакля]=@perfomacedate, [Время начала спектакля]=@perfomancestarts, [Время окончания спектакля]=perfomanceends where id=@id", con);

cmd.Parameters.Add("@perfomancename", OleDbType.VarChar).Value = perfomancesCombobox.Text;

cmd.Parameters.Add("@perfomanceid", OleDbType.Integer).Value = perfomanceID;

cmd.Parameters.Add("@theatreid", OleDbType.Integer).Value = theatreId;

cmd.Parameters.Add("@perfomancedate", OleDbType.DBDate).Value = perfomanceDateTimePicker.Value.ToShortDateString();

cmd.Parameters.Add("@perfomancestarts", OleDbType.VarChar).Value = perfomanceStartsDateTimePicker.Value.ToShortTimeString();

cmd.Parameters.Add("@perfomanceends", OleDbType.VarChar).Value = perfomanceEndsDateTimePicker.Value.ToShortTimeString();

cmd.Parameters.Add("@id", OleDbType.Integer).Value = scheduleID;

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

schedulePanel.Hide();

clearFields();

base\_load();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось изменить данные.\nОшибка: " + ex.Message);

con.Close();

}

}

}

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

{

if (MessageBox.Show("Вы действительно хотите удалить из расписания выбранный спектакль?", "Подтверждение действия", MessageBoxButtons.YesNo) == DialogResult.No)

return;

OleDbConnection con = staticVariables.con;

int scheduleID = Convert.ToInt32(scheduleDataGridView[0, scheduleDataGridView.CurrentRow.Index].Value);

try

{

OleDbCommand cmd = new OleDbCommand("delete from Расписание where [id]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = scheduleID;

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

base\_load();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

}

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

{

try

{

for (int i = 0; i < scheduleDataGridView.Rows.Count; i++)

{

if (scheduleDataGridView[fieldsCombobox.SelectedIndex, i].Value.ToString() == findTextbox.Text)

{

scheduleDataGridView.CurrentCell = scheduleDataGridView[fieldsCombobox.SelectedIndex, i];

return;

}

}

}

catch

{

MessageBox.Show("Не удалось найти введенное значение.");

}

}

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

{

Excel.Application excelApp = new Excel.Application();

Excel.Workbook workbook;

Excel.Worksheet worksheet;

workbook = excelApp.Workbooks.Add(System.Reflection.Missing.Value);

worksheet = (Excel.Worksheet)workbook.Sheets[1];

worksheet.Cells[1, 1] = scheduleDataGridView.Columns[0].HeaderText;

worksheet.Cells[1, 2] = scheduleDataGridView.Columns[1].HeaderText;

worksheet.Cells[1, 3] = scheduleDataGridView.Columns[2].HeaderText;

worksheet.Cells[1, 4] = scheduleDataGridView.Columns[3].HeaderText;

worksheet.Cells[1, 5] = scheduleDataGridView.Columns[4].HeaderText;

worksheet.Cells[1, 6] = scheduleDataGridView.Columns[5].HeaderText;

for (int i = 0; i < scheduleDataGridView.Rows.Count; i++)

{

worksheet.Cells[i + 2, 1] = scheduleDataGridView[0, i].Value;

worksheet.Cells[i + 2, 2] = scheduleDataGridView[1, i].Value;

worksheet.Cells[i + 2, 3] = scheduleDataGridView[2, i].Value;

worksheet.Cells[i + 2, 4] = scheduleDataGridView[3, i].Value;

worksheet.Cells[i + 2, 5] = scheduleDataGridView[4, i].Value;

worksheet.Cells[i + 2, 6] = scheduleDataGridView[5, i].Value;

}

worksheet.Columns.AutoFit();

excelApp.Visible = true;

excelApp.UserControl = true;

}

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

{

(scheduleDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Format("[" + filterCombobox.Text.ToString() + "]" + " = '{0}'", filterTextbox.Text);

}

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

{

(scheduleDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Empty;

}

}

}

**theatresForm.cs – форма «Театры»**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

namespace boxOffice

{

public partial class theatresForm : Form

{

public theatresForm()

{

InitializeComponent();

}

public void base\_load()

{

DataSet ds = new DataSet();

OleDbConnection con = staticVariables.con;

OleDbCommand com = new OleDbCommand("select \* from Театр", con);

con.Open();

OleDbDataAdapter adapter = new OleDbDataAdapter();

adapter.SelectCommand = com;

adapter.Fill(ds);

DataTable dt = ds.Tables[0];

theatresDataGridView.DataSource = dt;

con.Close();

theatresDataGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

}

public void clearFields()

{

theatreNameTextbox.Text = "";

theatreAdressTextbox.Text = "";

}

int theatreID;

private void theatresForm\_Load(object sender, EventArgs e)

{

base\_load();

theatresPanel.Hide();

for (int i = 0; i < theatresDataGridView.Columns.Count; i++)

{

fieldsCombobox.Items.Add(theatresDataGridView.Columns[i].HeaderText);

filterCombobox.Items.Add(theatresDataGridView.Columns[i].HeaderText);

}

fieldsCombobox.SelectedIndex = 0;

filterCombobox.SelectedIndex = 0;

}

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

{

headerLabel.Text = "Добавить";

submitButton.Text = "Добавить";

theatresPanel.Show();

}

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

{

OleDbConnection con = staticVariables.con;

if (submitButton.Text == "Добавить")

{

if(theatreNameTextbox.Text == "" || theatreAdressTextbox.Text == "")

{

MessageBox.Show("Пожалуйста, заполните все поля.");

return;

}

try

{

OleDbCommand cmd = new OleDbCommand("insert into Театр (Название, [Адрес театра]) values (@name, @adress)", con);

cmd.Parameters.Add("@name", OleDbType.VarChar).Value = theatreNameTextbox.Text;

cmd.Parameters.Add("@adress", OleDbType.VarChar).Value = theatreAdressTextbox.Text;

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

theatresPanel.Hide();

base\_load();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось добавить данные.\nОшибка: " + ex.Message);

con.Close();

}

clearFields();

}

if (submitButton.Text == "Изменить")

{

theatreID = Convert.ToInt32(theatresDataGridView[0, theatresDataGridView.CurrentRow.Index].Value);

int currentRow = theatresDataGridView.CurrentCell.RowIndex;

int currentColumn = theatresDataGridView.CurrentCell.ColumnIndex;

try

{

OleDbCommand cmd = new OleDbCommand("update Театр set Название=@name, [Адрес театра]=@adress where [id театра]=@id", con);

cmd.Parameters.Add("@name", OleDbType.VarChar).Value = theatreNameTextbox.Text;

cmd.Parameters.Add("@adress", OleDbType.VarChar).Value = theatreAdressTextbox.Text;

cmd.Parameters.Add("@id", OleDbType.Integer).Value = theatreID;

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

base\_load();

theatresPanel.Hide();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось изменить данные.\nОшибка: " + ex.Message);

con.Close();

}

clearFields();

}

}

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

{

if(MessageBox.Show("Вы действительно хотите удалить из списка выбранный театр?", "Подтверждение действия", MessageBoxButtons.YesNo) == DialogResult.No)

return;

OleDbConnection con = staticVariables.con;

theatreID = Convert.ToInt32(theatresDataGridView[0, theatresDataGridView.CurrentRow.Index].Value);

try

{

OleDbCommand cmd = new OleDbCommand("delete from Театр where [id театра]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = theatreID;

con.Open();

cmd.ExecuteNonQuery();

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

con.Close();

base\_load();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

}

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

{

headerLabel.Text = "Редактировать";

submitButton.Text = "Изменить";

theatreID = Convert.ToInt32(theatresDataGridView[0, theatresDataGridView.CurrentRow.Index].Value);

OleDbConnection con = staticVariables.con;

try

{

OleDbCommand cmd = new OleDbCommand("select \* from Театр where [id театра]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = theatreID;

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if(reader.Read())

{

theatreNameTextbox.Text = reader.GetString(1);

theatreAdressTextbox.Text = reader.GetString(2);

}

reader.Close();

con.Close();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось получить данные.\nОшибка: " + ex.Message);

con.Close();

}

theatresPanel.Show();

}

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

{

theatresPanel.Hide();

clearFields();

}

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

{

try

{

for (int i = 0; i < theatresDataGridView.Rows.Count; i++)

{

if (theatresDataGridView[fieldsCombobox.SelectedIndex, i].Value.ToString() == findTextbox.Text)

{

theatresDataGridView.CurrentCell = theatresDataGridView[fieldsCombobox.SelectedIndex, i];

return;

}

}

}

catch

{

MessageBox.Show("Не удалось найти введенное значение.");

}

}

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

{

Excel.Application excelApp = new Excel.Application();

Excel.Workbook workbook;

Excel.Worksheet worksheet;

workbook = excelApp.Workbooks.Add(System.Reflection.Missing.Value);

worksheet = (Excel.Worksheet)workbook.Sheets[1];

worksheet.Cells[1, 1] = theatresDataGridView.Columns[0].HeaderText;

worksheet.Cells[1, 2] = theatresDataGridView.Columns[1].HeaderText;

worksheet.Cells[1, 3] = theatresDataGridView.Columns[2].HeaderText;

for (int i = 0; i < theatresDataGridView.Rows.Count; i++)

{

worksheet.Cells[i + 2, 1] = theatresDataGridView[0, i].Value;

worksheet.Cells[i + 2, 2] = theatresDataGridView[1, i].Value;

worksheet.Cells[i + 2, 3] = theatresDataGridView[2, i].Value;

}

worksheet.Columns.AutoFit();

excelApp.Visible = true;

excelApp.UserControl = true;

}

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

{

(theatresDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Format("[" + filterCombobox.Text.ToString() + "]" + " = '{0}'", filterTextbox.Text);

}

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

{

(theatresDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Empty;

}

}

}

**ordersForm.cs – форма «Заказы»**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

namespace boxOffice

{

public partial class ordersForm : Form

{

public ordersForm()

{

InitializeComponent();

}

public void base\_load()

{

DataSet ds = new DataSet();

OleDbConnection con = staticVariables.con;

OleDbCommand com = new OleDbCommand("select \* from Заказы", con);

con.Open();

OleDbDataAdapter adapter = new OleDbDataAdapter();

adapter.SelectCommand = com;

adapter.Fill(ds);

DataTable dt = ds.Tables[0];

ordersDataGridView.DataSource = dt;

con.Close();

}

List<int> perfomaceIDs = new List<int>();

List<int> theatreIDs = new List<int>();

public void clearFields()

{

perfomanceDateTimePicker.Value = DateTime.Today.Date;

perfomanceCombobox.SelectedIndex = 0;

countNumericUpDown.Value = 1;

perfomaceIDs.Clear();

rowTicket1NUD.Value = 1;

rowTicket2NUD.Value = 1;

rowTicket3NUD.Value = 1;

rowTicket4NUD.Value = 1;

rowTicket5NUD.Value = 1;

rowTicket6NUD.Value = 1;

seatTicket1NUD.Value = 1;

seatTicket2NUD.Value = 1;

seatTicket3NUD.Value = 1;

seatTicket4NUD.Value = 1;

seatTicket5NUD.Value = 1;

seatTicket6NUD.Value = 1;

}

int orderID;

int value = 0;

GroupBox[] tickets = new GroupBox[5];

NumericUpDown[] rows = new NumericUpDown[6];

NumericUpDown[] seats = new NumericUpDown[6];

private void ordersForm\_Load(object sender, EventArgs e)

{

ticketsDataGridView.Hide();

base\_load();

ordersPanel.Hide();

ticket2.Hide();

ticket3.Hide();

ticket4.Hide();

ticket5.Hide();

ticket6.Hide();

tickets[0] = ticket2;

tickets[1] = ticket3;

tickets[2] = ticket4;

tickets[3] = ticket5;

tickets[4] = ticket6;

rows[0] = rowTicket1NUD;

rows[1] = rowTicket2NUD;

rows[2] = rowTicket3NUD;

rows[3] = rowTicket4NUD;

rows[4] = rowTicket5NUD;

rows[5] = rowTicket6NUD;

seats[0] = seatTicket1NUD;

seats[1] = seatTicket2NUD;

seats[2] = seatTicket3NUD;

seats[3] = seatTicket4NUD;

seats[4] = seatTicket5NUD;

seats[5] = seatTicket6NUD;

perfomanceDateTimePicker.Value = DateTime.Today.Date;

perfomanceDateTimePicker.MinDate = DateTime.Today.Date;

perfomanceCombobox.SelectedIndex = 0;

backButton.Hide();

printTicketButton.Hide();

for (int i = 0; i < ordersDataGridView.Columns.Count; i++)

{

fieldsCombobox.Items.Add(ordersDataGridView.Columns[i].HeaderText);

filterCombobox.Items.Add(ordersDataGridView.Columns[i].HeaderText);

}

fieldsCombobox.SelectedIndex = 0;

filterCombobox.SelectedIndex = 0;

}

int count = 0;

public void countOfOrders()

{

OleDbConnection con = staticVariables.con;

try

{

con.Open();

OleDbCommand cmd1 = new OleDbCommand("select count(\*) from Заказ", con);

OleDbDataReader reader = cmd1.ExecuteReader();

while (reader.Read())

{

count = reader.GetInt32(0);

}

reader.Close();

con.Close();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось связаться с базой данных.\nОшибка: " + ex);

con.Close();

}

}

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

{

if (MessageBox.Show("Вы действительно хотите удалить из списка выбранный заказ?", "Подтверждение действия", MessageBoxButtons.YesNo) == DialogResult.No)

return;

OleDbConnection con = staticVariables.con;

orderID = Convert.ToInt32(ordersDataGridView[0, ordersDataGridView.CurrentRow.Index].Value);

try

{

OleDbCommand cmd = new OleDbCommand("delete from Заказы where [id заказа]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = orderID;

countOfOrders();

con.Open();

cmd.ExecuteNonQuery();

for (int i = 0; i < count; i++)

{

OleDbCommand cmd2 = new OleDbCommand("delete from Заказ where [id заказа]=@id", con);

cmd2.Parameters.Add("@id", OleDbType.Integer).Value = orderID;

cmd2.ExecuteNonQuery();

}

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

con.Close();

base\_load();

}

catch

{

MessageBox.Show("Не удалось связаться с базой данных.");

con.Close();

}

}

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

{

headerLabel.Text = "Добавить";

submitButton.Text = "Добавить";

ordersPanel.Show();

}

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

{

ordersPanel.Hide();

clearFields();

}

bool Checked = true;

public void ticketsCheck()

{

List<int> order\_id = new List<int>();

countOfOrders();

OleDbConnection con = staticVariables.con;

con.Open();

try

{

OleDbCommand cmd1 = new OleDbCommand("select [id заказа] from Заказы where [id спектакля]=@id and [Дата спектакля]=@date", con);

cmd1.Parameters.Add("@id", OleDbType.Integer).Value = perfomaceIDs[perfomanceCombobox.SelectedIndex];

cmd1.Parameters.Add("@date", OleDbType.DBDate).Value = perfomanceDateTimePicker.Value.Date;

OleDbDataReader reader1 = cmd1.ExecuteReader();

while(reader1.Read())

{

order\_id.Add(reader1.GetInt32(0));

}

int row = 0;

int seat = 0;

for(int i = 0; i < countNumericUpDown.Value; i++)

for(int z = 0; z < order\_id.Count(); z++)

for(int j = 0; j < count; j++)

{

OleDbCommand cmd = new OleDbCommand("select [Номер ряда], [Номер места] from Заказ where [id заказа]=@orderid", con);

cmd.Parameters.Add("@orderid", OleDbType.Integer).Value = order\_id[z];

OleDbDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

row = reader.GetInt32(0);

seat = reader.GetInt32(1);

}

if (rows[i].Value == row && seats[i].Value == seat)

{

MessageBox.Show("Одно или несколько выбранных мест уже заняты.");

Checked = false;

con.Close();

return;

}

}

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось проверить билеты.\nОшибка: " + ex.Message);

con.Close();

Checked = false;

}

}

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

{

OleDbConnection con = staticVariables.con;

if(submitButton.Text == "Добавить")

{

if(perfomanceCombobox.Text == "В эту дату спектакли отсутствуют!")

{

MessageBox.Show("Заказ билетов невозможен!\nВ эту дату спектакли отсутствуют!");

return;

}

ticketsCheck();

if (Checked == false)

return;

try

{

OleDbCommand cmd = new OleDbCommand("insert into Заказы ([id спектакля], [Название спектакля], [id театра], [Количество билетов], [Дата спектакля], Стоимость) values (@perfomanceid, @perfomancename, @theatreid, @count, @date, @stoim)", con);

cmd.Parameters.Add("@perfomanceid", OleDbType.Integer).Value = perfomaceIDs[perfomanceCombobox.SelectedIndex];

cmd.Parameters.Add("@perfomancename", OleDbType.VarChar).Value = perfomanceCombobox.Text;

cmd.Parameters.Add("@theatreid", OleDbType.Integer).Value = theatreIDs[perfomanceCombobox.SelectedIndex];

cmd.Parameters.Add("@count", OleDbType.Integer).Value = countNumericUpDown.Value;

cmd.Parameters.Add("@date", OleDbType.DBDate).Value = perfomanceDateTimePicker.Value.Date;

string[] cost = costNumLabel.Text.Split(' ');

int costInt = Convert.ToInt32(cost[0]);

cmd.Parameters.Add("@stoim", OleDbType.Integer).Value = costInt;

con.Open();

cmd.ExecuteNonQuery();

int lastOrderID = 0;

OleDbCommand cmd1 = new OleDbCommand("select top 1 [id заказа] from Заказы order by [id заказа] desc", con);

OleDbDataReader reader = cmd1.ExecuteReader();

if(reader.Read())

{

lastOrderID = reader.GetInt32(0);

}

for(int i = 0; i < countNumericUpDown.Value; i++)

{

OleDbCommand cmd2 = new OleDbCommand("insert into Заказ ([id заказа], [Номер ряда], [Номер места]) values (@orderID, @rowNumber, @seatNumber)", con);

cmd2.Parameters.Add("@orderID", OleDbType.Integer).Value = lastOrderID;

cmd2.Parameters.Add("@rowNumber", OleDbType.Integer).Value = rows[i].Value;

cmd2.Parameters.Add("@seatNumber", OleDbType.Integer).Value = seats[i].Value;

cmd2.ExecuteNonQuery();

}

con.Close();

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

base\_load();

ordersPanel.Hide();

clearFields();

}

catch(Exception ex)

{

MessageBox.Show("Не удалось добавить данные.\nОшибка: " + ex.Message);

con.Close();

}

}

}

private void countNumericUpDown\_ValueChanged(object sender, EventArgs e)

{

foreach(GroupBox element in tickets)

{

element.Hide();

}

for(int i = 0; i < countNumericUpDown.Value - 1; i++)

{

tickets[i].Show();

}

if(perfomanceCombobox.Text == "В эту дату спектакли отсутствуют!")

{

costNumLabel.Text = "0 руб.";

return;

}

costNumLabel.Text = (value \* countNumericUpDown.Value).ToString() + " руб.";

}

private void ordersDataGridView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

int order\_id = Convert.ToInt32(ordersDataGridView[0, ordersDataGridView.CurrentRow.Index].Value);

DataSet ds = new DataSet();

OleDbConnection con = staticVariables.con;

con.Open();

OleDbCommand cmd = new OleDbCommand("select \* from Заказ where [id заказа]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = order\_id;

OleDbDataAdapter adapter = new OleDbDataAdapter();

adapter.SelectCommand = cmd;

adapter.Fill(ds);

DataTable dt = ds.Tables[0];

ticketsDataGridView.DataSource = dt;

con.Close();

ticketsDataGridView.Show();

backButton.Show();

printTicketButton.Show();

}

public void getPerfomancesNames()

{

perfomanceCombobox.Items.Clear();

OleDbConnection con = staticVariables.con;

try

{

con.Open();

for(int i = 0; i < perfomaceIDs.Count(); i++)

{

OleDbCommand cmd = new OleDbCommand("select [Название спектакля] from Репертуар where [id спектакля]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = perfomaceIDs[i];

OleDbDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

perfomanceCombobox.Items.Add(reader.GetString(0));

}

reader.Close();

}

con.Close();

if(perfomanceCombobox.Items.Count == 0)

{

perfomanceCombobox.Items.Add("В эту дату спектакли отсутствуют!");

costNumLabel.Text = "0 руб.";

}

perfomanceCombobox.SelectedIndex = 0;

}

catch(Exception ex)

{

MessageBox.Show("При выполение команды произошла ошибка.\nОшибка: " + ex);

con.Close();

}

}

private void perfomanceDateTimePicker\_ValueChanged(object sender, EventArgs e)

{

perfomaceIDs.Clear();

OleDbConnection con = staticVariables.con;

try

{

OleDbCommand cmd = new OleDbCommand("select [id спектакля], [id театра] from Расписание where [Дата спектакля]=@date", con);

cmd.Parameters.Add("@date", OleDbType.DBDate).Value = perfomanceDateTimePicker.Value;

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

while(reader.Read())

{

perfomaceIDs.Add(reader.GetInt32(0));

theatreIDs.Add(reader.GetInt32(1));

}

reader.Close();

con.Close();

getPerfomancesNames();

}

catch(Exception ex)

{

MessageBox.Show("При выполение команды произошла ошибка.\nОшибка: " + ex);

con.Close();

}

}

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

{

ticketsDataGridView.Hide();

backButton.Hide();

printTicketButton.Hide();

}

private void perfomanceCombobox\_SelectedIndexChanged(object sender, EventArgs e)

{

if (perfomanceCombobox.Text == "В эту дату спектакли отсутствуют!")

return;

OleDbConnection con = staticVariables.con;

try

{

OleDbCommand cmd = new OleDbCommand("select [Цена билета] from Расписание where [id спектакля]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = perfomaceIDs[perfomanceCombobox.SelectedIndex];

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

while(reader.Read())

{

value = reader.GetInt32(0);

}

reader.Close();

con.Close();

}

catch(Exception ex)

{

MessageBox.Show("При выполение команды произошла ошибка.\nОшибка: " + ex);

con.Close();

}

costNumLabel.Text = "0 руб.";

costNumLabel.Text = (value \* countNumericUpDown.Value).ToString() + " руб.";

}

string theatrename = "";

string perfomancename = "";

DateTime time;

public void getNames(int theatreID, int perfomanceID)

{

OleDbConnection con = staticVariables.con;

try

{

OleDbCommand cmd = new OleDbCommand("select Название from Театр where [id театра]=@id", con);

cmd.Parameters.Add("@id", OleDbType.Integer).Value = theatreID;

OleDbCommand cmd1 = new OleDbCommand("select [Название спектакля] from Репертуар where [id спектакля]=@id", con);

cmd1.Parameters.Add("@id", OleDbType.Integer).Value = perfomanceID;

con.Open();

OleDbDataReader reader = cmd.ExecuteReader();

if(reader.Read())

{

theatrename = reader.GetString(0);

}

reader.Close();

OleDbDataReader reader1 = cmd1.ExecuteReader();

if(reader1.Read())

{

perfomancename = reader1.GetString(0);

}

reader1.Close();

OleDbCommand cmd2 = new OleDbCommand("select [Время начала спектакля] from Расписание where [Дата спектакля]=@date and [id спектакля]=@id", con);

cmd2.Parameters.Add("@date", OleDbType.DBDate).Value = ordersDataGridView[4, ordersDataGridView.CurrentRow.Index].Value;

cmd2.Parameters.Add("@id", OleDbType.Integer).Value = perfomanceID;

OleDbDataReader reader2 = cmd2.ExecuteReader();

if(reader2.Read())

{

time = reader2.GetDateTime(0);

}

con.Close();

}

catch (Exception ex)

{

MessageBox.Show("При выполение команды произошла ошибка.\nОшибка: " + ex);

con.Close();

}

}

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

{

Excel.Application excelApp = new Excel.Application();

Excel.Workbook workbook = excelApp.Workbooks.Open(@"C:\Users\Никита\Desktop\Учеба\diplom\Диплом\boxOffice\boxOffice\resources\ticket.xlsx");

Excel.Worksheet worksheet = excelApp.ActiveSheet;

worksheet.Cells[3, 2] = ticketsDataGridView[0, ticketsDataGridView.CurrentRow.Index].Value;

worksheet.Cells[3, 9] = ticketsDataGridView[0, ticketsDataGridView.CurrentRow.Index].Value;

getNames(Convert.ToInt32(ordersDataGridView[2, ordersDataGridView.CurrentRow.Index].Value), Convert.ToInt32(ordersDataGridView[1, ordersDataGridView.CurrentRow.Index].Value));

worksheet.Cells[5, 2] = theatrename;

worksheet.Cells[7, 2] = perfomancename;

worksheet.Cells[3, 5] = ordersDataGridView[4, ordersDataGridView.CurrentRow.Index].Value;

worksheet.Cells[3, 7] = time.ToShortTimeString();

excelApp.Visible = true;

excelApp.UserControl = true;

}

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

{

Excel.Application excelApp = new Excel.Application();

Excel.Workbook workbook;

Excel.Worksheet worksheet;

workbook = excelApp.Workbooks.Add(System.Reflection.Missing.Value);

worksheet = (Excel.Worksheet)workbook.Sheets[1];

worksheet.Cells[1, 1] = ordersDataGridView.Columns[0].HeaderText;

worksheet.Cells[1, 2] = ordersDataGridView.Columns[1].HeaderText;

worksheet.Cells[1, 3] = ordersDataGridView.Columns[2].HeaderText;

worksheet.Cells[1, 4] = ordersDataGridView.Columns[3].HeaderText;

worksheet.Cells[1, 5] = ordersDataGridView.Columns[4].HeaderText;

worksheet.Cells[1, 6] = ordersDataGridView.Columns[5].HeaderText;

worksheet.Cells[1, 7] = ordersDataGridView.Columns[6].HeaderText;

int i = 0;

int summ = 0;

for (i = 0; i < ordersDataGridView.Rows.Count; i++)

{

worksheet.Cells[i + 2, 1] = ordersDataGridView[0, i].Value;

worksheet.Cells[i + 2, 2] = ordersDataGridView[1, i].Value;

worksheet.Cells[i + 2, 3] = ordersDataGridView[2, i].Value;

worksheet.Cells[i + 2, 4] = ordersDataGridView[3, i].Value;

worksheet.Cells[i + 2, 5] = ordersDataGridView[4, i].Value;

worksheet.Cells[i + 2, 6] = ordersDataGridView[5, i].Value;

worksheet.Cells[i + 2, 7] = ordersDataGridView[6, i].Value;

summ += Convert.ToInt32(ordersDataGridView[6, i].Value);

}

worksheet.Cells[i + 2, 6] = "Итого:";

worksheet.Cells[i + 2, 7] = summ;

worksheet.Columns.AutoFit();

excelApp.Visible = true;

excelApp.UserControl = true;

}

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

{

try

{

for (int i = 0; i < ordersDataGridView.Rows.Count; i++)

{

if (ordersDataGridView[fieldsCombobox.SelectedIndex, i].Value.ToString() == findTextbox.Text)

{

ordersDataGridView.CurrentCell = ordersDataGridView[fieldsCombobox.SelectedIndex, i];

return;

}

}

}

catch

{

MessageBox.Show("Не удалось найти введенное значение.");

}

}

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

{

(ordersDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Format("[" + filterCombobox.Text.ToString() + "]" + " = '{0}'", filterTextbox.Text);

}

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

{

(ordersDataGridView.DataSource as DataTable).DefaultView.RowFilter = string.Empty;

}

}

}