МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

Гомельский государственный технический университет имени п. о. сухого

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальность 1-40 05 01-01 Информационные системы и технологии

(в проектировании и производстве)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Объектно-ориентированное программирование  
 и проектирование»

на тему: «*WPF* система учета продаж мебели»

Исполнитель: студент гр. ИТП-21

Бурлаков А.О.

Руководитель: преподаватель

Курочка К.С.

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

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

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

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

Подписи членов комиссии

по защите курсовой работы: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc166184399)

[1 Основные средства и технологии в разработке приложения 5](#_Toc166184400)

[1.1 Основы объектно-ориентированного программирования 5](#_Toc166184401)

[1.2 Средства и технологии разработки 6](#_Toc166184402)

[1.3 Принципы *SOLID* 10](#_Toc166184403)

[1.4 Платформа *WPF* 12](#_Toc166184404)

[1.5 Техническое задание проекта 13](#_Toc166184405)

[2 Архитектура разработанного приложения 16](#_Toc166184406)

[2.1 Проектирование архитектуры информационной системы 16](#_Toc166184407)

[2.2 Описание сущностей информационной системы 17](#_Toc166184408)

[2.3 База данных приложения 17](#_Toc166184409)

[2.4 Иерархическая схема интерфейса 24](#_Toc166184410)

[2.5 Структура приложения 25](#_Toc166184411)

[3 Этапы проведения верификации и тестирования приложения 27](#_Toc166184412)

[3.1 Результаты проведения модульного тестирования 27](#_Toc166184413)

[3.2 Результаты проведения интегрированного тестирования 28](#_Toc166184414)

[Заключение 34](#_Toc166184415)

[Список использованных источников 35](#_Toc166184416)

[Приложение А Функциональная схема приложения 36](#_Toc166184417)

[Приложение Б Листинг программы 37](#_Toc166184418)

[Приложение В Листинг модульных тестов 82](#_Toc166184419)

[Приложение Г Руководство пользователя 87](#_Toc166184420)

[Приложение Д Руководство системного программиста 90](#_Toc166184421)

[Приложение Е Руководство программиста 91](#_Toc166184422)

ВВЕДЕНИЕ

Повсеместное использование компьютерных технологий и техники в повседневной жизни людей позволяет автоматизировать любые производственные процессы в любой отрасли, что упрощает учет в организации и позволяет формировать отчеты за любой период и любой сложности за считанные секунды. Основная задача автоматизации бизнес-процессов – вывод качественных показателей процессов на принципиально новый уровень. Это достигается за счет того, что основным преимуществом автоматизированного труда над ручным является его надежность.

Современный рынок мебельной индустрии характеризуется высокой конкуренцией и необходимостью эффективного управления бизнесом. Учет продаж мебели играет важную роль в организации производства, оптимизации запасов, а также в принятии стратегических решений по развитию компании. Разработка приложения для учета продаж мебели на языке *C*# представляет собой актуальную задачу, которая позволит автоматизировать процессы управления и повысить эффективность работы предприятия.

Поэтому возникает необходимость в использовании программных продуктов, автоматизирующих основные бизнес-процессы предприятия. Но не всегда на рынке таковые можно найти, учитывая специфику каждой конкретной организации, поэтому приходится прибегать к разработке собственной информационной системы под нужды предприятия.

Практическая значимость исследования заключается в разработке информационной системы учета деятельности магазина по продаже мебели

Результатом исследования является разработанная информационная система учета деятельности магазина по продаже мебели, которая может применяться в любом розничном магазине с единичной продукцией и гарнитурами.

1 ОСНОВНЫЕ СРЕДСТВА И ТЕХНОЛОГИИ В РАЗРАБОТКЕ ПРИЛОЖЕНИЯ

* 1. Основы объектно-ориентированного программирования

Объектно-ориентированное программирование (*object-oriented programming*) (далее – ООП) основано на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного типа, использующая механизм пересылки сообщений и классы, организованные в иерархию наследования. ООП представляет собой эффективный подход к программированию.

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

Программа, реализованная с помощью объектно-ориентированного программирования, состоит из взаимодействующих объектов.

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

При написании программы необходимо соблюдать основные принципы объектно-ориентированного программирования:

* абстракция;
* инкапсуляция;
* наследование;
* полиморфизм.

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

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

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

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

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

На этапе разработки информационных систем выполняются следующие основные действия:

– кодирование;

– первоначальное тестирование;

– разработка справочной системы.

Кодирование представляет собой процесс преобразования результатов проектирования в готовый программный продукт. При кодировании происходит описание составленной модели информационной системы средствами языка программирования и СУБД.

1.2 Средства и технологии разработки

Любая информационная система предназначена для хранения, поиска и обработки информации, а также для своевременного обеспечения ее пользователей надлежащей информацией.

Существуют различные методы реализации информационных систем, классифицируемые по степени автоматизации:

– ручные («без компьютера»);

– автоматические;

– автоматизированные.

Автоматические и автоматизированные системы предусматривают обязательное наличие в их составе аппаратно-программных средств, которым отводится главная роль по обработке информации. Основным отличием автоматизированных информационных систем является необходимость постоянного участия в процессе обработки информации человека. Именно этот класс систем соответствует современному представлению об информационных системах.

В данном курсовом проекте используется автоматизированный метод реализации информационной системы, а именно создается автоматизированный программный комплекс на платформе *.NET Framework*, написанный на языке программирования *C#* при помощи объектно-ориентированного подхода в среде разработки *Visual Studio 2022* с использованием шаблонов проектирования и слоя доступа к данным, который взаимодействует с СУБД *MS SQL Server.* Для реализации пользовательского интерфейса использована технология *WPF* (*Windows Presentation Foundation*).

*.NET Framework* – программная платформа, выпущенная компанией *Microsoft* в 2002 году. Основой платформы является общеязыковая среда исполнения *CLR (Common Language Runtime)*, которая подходит для разных языков программирования. Функциональные возможности *CLR* доступны в любых языках программирования, использующих эту среду. Платформа *.NET* *Framework* поддерживает множество языков программирования, таких как *C#*, *Visual Basic*, *Iron Python*, *C*, *C++* и прочие благодаря тому, что программа сначала переводится компилятором в единый для *.NET* байт-код *CIL* (*Common Intermediate Language*), затем код либо исполняется виртуальной машиной *CLR*, либо транслируется утилитой *NGen.exe* в исполняемый код для конкретного целевого процессора. Благодаря *CLR* встроенный в неё *JIT* (*Just–in–time*)компилятор преобразует промежуточный байт-код в машинные коды нужного процессора *[*1*]*.

Язык программирования *С#* – это один из языков, который предназначен для написания приложений для программной платформы *.NET Framework* (одной из основных платформ разработки, развертывания и исполнения распределенных приложений, которая разработана компанией *Microsoft*). Язык *C#* разработан в 1998-2001 годах группой инженеров компании *Microsoft*.

Язык *С#* обладает рядом достоинств: статическая типизация, перегрузка операторов, свойства, делегаты, *LINQ*, обобщённые типы. Также является профессиональным языком программирования, предназначенным для решения широкого спектра задач. Задачами могут выступать разработка приложений без или с графическим интерфейсом на *Windows*, на *Linux*, создание игр на базе *Unity*, создание веб-сервера на *ASP.NET*.

*WPF* (*Windows* *Presentation* *Foundation*) – это часть экосистемы .*NET*, которая представляет собой подсистему построения графических пользовательских интерфейсов. Преимуществом *WPF* является использование *DirectX* для отрисовки интерфейса, благодаря чему можно создавать более сложные и красивые представления, так как используется аппаратное ускорение графического процессора *[*2*]*.

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

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

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

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

При всех своих изменениях *SQL* остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов *[*3*]*.

*LINQ to SQL* – наименование, присвоенное *API*–интерфейсу *IQueryable<T>*, который позволяет запросам *LINQ* (*Language Integrated Query –* язык интегрированных запросов) работать с базой данных *Microsoft SQL Server.*

Аббревиатура *LINQ* обозначает целый набор технологий, создающих и использующих возможности интеграции запросов непосредственно в язык *C#*. Традиционно запросы к данным выражаются в виде простых строк без проверки типов при компиляции или поддержки *IntelliSense*. Кроме того, разработчику приходится изучать различные языки запросов для каждого типа источников данных: баз данных *SQL*, *XML*-документов, различных веб-служб и т.д. Технологии *LINQ* превращают запросы в удобную языковую конструкцию, которая применяется аналогично классам, методам и событиям.

Наиболее очевидной частью *LINQ* является интегрированное выражение запроса. Выражения запроса используют декларативный синтаксис запроса. С помощью синтаксиса запроса можно выполнять фильтрацию, упорядочение и группирование данных из источника данных, обходясь минимальным объемом программного кода. Одни и те же базовые выражения запроса позволяют одинаково легко получать и преобразовывать данные из баз данных *SQL*, наборов данных *ADO* .*NET*, *XML*-документов, *XML*-потоков и коллекций .*NET*.

*LINQ to SQL* представляет технологию доступа и управления реляционными данными. Данная технология позволяет составлять запросы к базам данных в удобной форме с помощью операторов *LINQ*, которые затем трансформируются в *sql*-выражения. Ключевыми объектами здесь являются сущности, которые хранятся в базе данных, контекст данных и запрос *LINQ*.

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

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

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

*PostgreSQL* является мощной и распространенной объектно-реляционной системой управления базами данных (СУБД). Она обладает богатым набором функций, поддерживает множество типов данных и обеспечивает высокую производительность и надежность. Использование *PostgreSQL* позволяет эффективно хранить и управлять данными о продажах мебели, обеспечивая высокий уровень безопасности и отказоустойчивости.

Технология доступа *ADO*: *ADO* (*ActiveX* *Data* *Objects*) представляет собой набор компонентов и объектов, которые позволяют приложениям на платформе *Windows* взаимодействовать с базами данных. С помощью технологии *ADO* можно осуществлять подключение к базе данных, выполнение *SQL*-запросов, получение и обновление данных. Это обеспечивает удобство работы с данными и интеграцию с различными источниками данных, в том числе с СУБД *PostgreSQL*.

Таким образом, использование указанных технологий в процессе разработки приложения для учета продаж мебели на языке *C*# обеспечивает высокую производительность, надежность и удобство работы с данными, что в конечном итоге способствует повышению эффективности управления бизнесом.

1.3 Принципы *SOLID*

Принципы *SOLID* - это набор основных принципов объектно-ориентированного программирования, которые помогают создавать гибкие, расширяемые и поддерживаемые приложения. Давайте рассмотрим каждый из принципов подробнее и как их можно применить в разработке приложения для учета продаж мебели:

1. Принцип единственной ответственности (*Single* *Responsibility* *Principle* - *SRP*):

Этот принцип гласит, что каждый класс должен иметь только одну причину для изменения. Он устанавливает, что класс должен быть ответственным только за одну часть функциональности.

Применение в приложении: Классы в приложении для учета продаж мебели должны быть разделены на отдельные модули, каждый из которых отвечает только за определенную функциональность, например, управление данными о продажах, генерация отчетов, аутентификация пользователей и т.д.

1. Принцип открытости/закрытости (*Open*/*Closed* *Principle* - *OCP*):

Согласно этому принципу, классы должны быть открыты для расширения, но закрыты для изменения. Это означает, что изменения в поведении класса должны происходить путем добавления нового кода, а не изменения существующего.

Применение в приложении: Для добавления новых функций или изменения поведения приложения, предпочтительно создавать новые классы или модули, которые расширяют функциональность приложения, не затрагивая существующий код.

1. Принцип подстановки Барбары Лисков (*Liskov* *Substitution* *Principle* - *LSP*):

Этот принцип утверждает, что объекты в программе должны быть заменяемыми на экземпляры их подтипов без изменения корректности выполнения программы.

Применение в приложении: При разработке приложения следует учитывать, что все подклассы должны быть совместимы с базовым классом, и их поведение не должно нарушать ожидаемую функциональность базового класса.

1. Принцип разделения интерфейса (*Interface* *Segregation* *Principle* - *ISP*):

Согласно этому принципу, клиенты не должны зависеть от методов, которые они не используют. Интерфейсы должны быть разделены на более мелкие, специфические для клиентов, чтобы минимизировать зависимости.

Применение в приложении: Интерфейсы в приложении должны быть разделены на более мелкие, специализированные интерфейсы, чтобы клиенты могли использовать только те методы, которые им нужны, без необходимости реализации всех методов большого интерфейса.

1. Принцип инверсии зависимостей (*Dependency* *Inversion* *Principle* - *DIP*):

Этот принцип гласит, что модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракций. Конкретные классы должны зависеть от абстракций, а не наоборот.

Применение в приложении: Вместо прямой зависимости между классами, предпочтительно использовать абстракции и интерфейсы для уменьшения связанности компонентов и облегчения тестирования и поддержки кода.

Применение принципов *SOLID* в разработке приложения для учета продаж мебели поможет создать более гибкий, расширяемый и легко поддерживаемый код, что повысит его качество и эффективность.

1.4 Платформа *WPF*

Платформа *Windows Presentation Foundation* (*WPF*) позволяет создавать клиентские приложения для настольных систем *Windows* с привлекательным пользовательским интерфейсом.

В основе *WPF* лежит независимый от разрешения векторный модуль визуализации, использующий возможности современного графического оборудования. Возможности этого модуля расширяются с помощью 11 комплексного набора функций разработки приложений, которые включают в себя язык *XAML*, элементы управления, привязку к данным, макет, двумерную и трехмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции. *WPF* является частью .*NET*.

Преимущества *WPF*:

– использование традиционных языков .*NET*-платформы - *C#* и *VB.NET* для создания логики приложения;

– возможность декларативного определения графического интерфейса с помощью специального языка разметки *XAML*, основанном на *xml* и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать *XAML* и *C#/VB.NET*;

– независимость от разрешения экрана: поскольку в *WPF* все элементы измеряются в независимых от устройства единицах, приложения на *WPF* легко масштабируются под разные экраны с разным разрешением;

– новые возможности, которых сложно было достичь в *WinForms*, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др;

– хорошее взаимодействие с *WinForms*, благодаря чему, например, в приложениях *WPF* можно использовать традиционные элементы управления из *WinForms*;

– богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое;

– аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2*D* или 3*D*, графикой или текстом, все компоненты приложения транслируются в объекты, понятные *Direct*3*D*, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной;

– создание приложений под множество *ОС* семейства *Windows* - от *Windows XP* до *Windows* 10.

Однако стоит учитывать, что по сравнению с приложениями на *WindowsForms* объем программ на *WPF* и потребление ими памяти в процессе работы в среднем несколько выше. Но это с лихвой компенсируется более широкими графическими возможностями и повышенной производительностью при отрисовке графики.

1.5 Техническое задание проекта

Цель проекта: Разработать приложение для учета продаж мебели.

Исходные данные: для каждой единицы товара известны ее наименование, категория, розничная цена и производитель. Мебель может быть представлена как поштучно, так и в составе гарнитуров.

Функциональные требования:

– Возможность добавления, удаления и редактирования информации о продажах мебели: Пользователь должен иметь возможность добавлять новые записи о продажах мебели, удалять существующие записи и редактировать информацию о продажах, включая наименование товара, категорию, цену и производителя. Для добавления новой записи пользователь должен заполнить обязательные поля формы, такие как наименование товара и его цена, а также может указать категорию и производителя. При редактировании существующей записи пользователь может изменять любые данные о продаже мебели. Для удаления записи пользователь должен подтвердить свое намерение удалить запись.

– Ведение базы данных о проданных товарах: Приложение должно вести базу данных о проданных товарах, включая информацию о наименовании товара, его категории, цене и производителе. Для каждой продажи мебели должна быть создана соответствующая запись в базе данных с указанием всех сопутствующих данных. База данных должна быть структурирована и удобна для поиска и фильтрации данных по различным критериям.

– Поддержка различных типов мебели: Приложение должно поддерживать различные типы мебели, включая как отдельные единицы товара, так и гарнитуры. Для каждого типа мебели должна быть предусмотрена соответствующая структура данных и интерфейс взаимодействия с пользователем.

– Генерация отчетов о продажах за определенный период времени: Пользователь должен иметь возможность генерировать отчеты о продажах мебели за определенный период времени. Отчет должен содержать информацию о каждой продаже, включая наименование товара, категорию, цену и производителя, а также дату и время продажи. Отчет может быть сохранен в различных форматах, включая *XML*, *CSV* или *PDF*, с возможностью открытия в электронных таблицах для анализа и обработки данных.

– Интерфейс пользователя с удобным и интуитивно понятным интерфейсом: Приложение должно иметь удобный и интуитивно понятный интерфейс, позволяющий пользователям легко и быстро освоить его функционал. Интерфейс должен быть доступен на русском и английском языках, с возможностью выбора пользователем предпочтительного языка. Пользовательский интерфейс должен быть адаптивным и поддерживать различные разрешения экранов, а также быть доступным для использования как на стационарных, так и на мобильных устройствах.

Нефункциональные требования:

– Производительность: Приложение должно обеспечивать высокую производительность при работе с базой данных, обработке запросов и генерации отчетов. Время ответа на запросы пользователя должно быть минимальным, не превышающим 1-2 секунды. Приложение должно быть способно обрабатывать большие объемы данных без замедления производительности.

– Надежность: Приложение должно быть стабильным и надежным, минимизируя возможность возникновения сбоев и ошибок. В случае возникновения ошибок приложение должно предоставлять информативные сообщения об ошибках и возможные способы их устранения. Все данные, вводимые пользователем, должны быть корректно валидированы, чтобы избежать ошибочного ввода и сохранить целостность данных.

– Безопасность: Доступ к приложению и данным должен быть защищен от несанкционированного доступа. Пользовательская аутентификация должна быть реализована с использованием сильных паролей и методов шифрования. Приложение должно обеспечивать разграничение прав доступа пользователей в зависимости от их ролей и полномочий.

– Масштабируемость: Приложение должно быть легко масштабируемым и способным обрабатывать рост числа пользователей и объема данных. В случае необходимости приложение должно поддерживать горизонтальное и вертикальное масштабирование без простоя в работе.

– Удобство использования: Интерфейс пользователя должен быть интуитивно понятным и легким в освоении даже для пользователей без технического образования. Для удобства пользователей приложение должно предоставлять контекстную справку и подсказки о функциональности и использовании различных элементов интерфейса.

– Отказоустойчивость: Приложение должно быть отказоустойчивым и способным восстанавливаться после сбоев и аварийных ситуаций. Резервное копирование данных и регулярное обслуживание базы данных должны быть организованы для предотвращения потери данных в случае сбоев.

– Эффективное использование ресурсов: Приложение должно эффективно использовать ресурсы компьютера, включая процессор, память и сетевое соединение. Приложение не должно потреблять излишние ресурсы компьютера и должно обеспечивать оптимальную производительность при минимальном потреблении ресурсов.

Таким образом, выполнение функциональных требований позволит разработать полнофункциональное приложение для учета продаж мебели, обеспечивающее удобство использования, точность и надежность в учете данных о продажах.

2 АРХИТЕКТУРА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

2.1 Проектирование архитектуры информационной системы

Анализ характеристик программного обеспечения помогает понять требования и ожидания, связанные с его функциональными и техническими аспектами. Чтобы обеспечить легкое расширение функциональности, программное обеспечение должно быть организовано в виде блоков и быть легким в обслуживании. Ключевыми характеристиками являются качество работы, высокая отказоустойчивость, масштабируемость и надежность.

Приложение должно выполнять следующие функции:

– вывод данных в корректной форме;

– обработка данных, полученных от пользователя;

– получение/обновление данных из источника.

Исходя из вышеописанных функций, можно выделить три основные части приложения:

– пользовательский интерфейс. Это та часть приложения, с которой взаимодействует пользователь, реализуется в виде графического пользовательского интерфейса, отображающего разрешенные в зависимости от роли пользователя элементы интерфейса;

– бизнес-логика. Содержит набор компонентов, с помощью которых обрабатывает полученные от уровня представлений данные, а также реализует всю необходимую логику приложения;

– доступ к данным. содержит компоненты для доступа к данным, в качестве источника данных.

Схема архитектуры разрабатываемого приложения представлена на рисунке 2.1



Рисунок 2.1 – Схема архитектуры разрабатываемого приложения

Каждый уровень приложения должен взаимодействовать со своим «соседом». Это позволить обеспечить масштабируемость приложения.

2.2 Описание сущностей информационной системы

Исходя из требований задания можно выделить следующие сущности:

– пользователь

– роли пользователя

– мебель

– тип мебели

– производитель

– гарнитур мебели

Для реализации многопользовательского доступа в систему необходимо разработать класс *User*, который будет содержать поля:

– имя пользователя *Username*

– пароль пользователя *Password*

– роль пользователя в приложении *Rolename*

Так как по заданию необходимо реализовать несколько видов мебели, то был разработан класс *FurnitureType*, содержащий поля:

– название типа *Typename*

– наценка на данный тип *Typemarkup*

Также по заданию необходимо описать сущность «производителя», которую описывает класс *Manufacturer* с полями:

– название производителя *Manufacturername*

– наценка производителя *Manufacturermarkup*

Сущность «мебели» отражает класс *Furniture*, который содержит следующие поля:

– название мебели *Furniturename*

– стоимость мебели *Furnitureprice*

– количество мебели *Furniturequantity*

– тип мебели *Furnituretype*

– производитель *Manufacturer*

Сущность «гарнитура» отражает класс *SetItems*, который содержит следующие поля:

– список мебели *Furnitures*

– название гарнитура *Setname*

2.3 База данных приложения

Приложение обеспечивает реализацию продаж и учет проданной мебели в магазине мебели.

В магазине мебели реализуются несколько видов категорий мебели: кухонная, спальная, гостиная, офисная. На каждую категорию мебели известны следующие параметры: название категории и наценка на категорию. Данные категории хранятся в таблице *furnituretypes.* Эта таблица предназначена для последующего добавления новых категорий.

В магазине мебели для каждой единицы мебели определен ее производитель. Для каждого производителя имеются следующие параметры: наименование производителя и наценка, устанавливаемая производителем. Данные объекты хранятся в таблице *manufacturers.* Эта таблица предназначена для последующего добавления новых производителей.

В системе магазина мебели реализован многопользовательский доступ, представленный тремя ролями для персонала: администратор, менеджер, продавец и гость. Гость не имеет параметров, потому в базе данных не хранится. Остальные объекты ролей представлены в таблице *roles.* Эта таблица предназначена для последующего добавления новых ролей.

Каждый сотрудник магазина, имеющий доступ к системе, имеет параметры: имя пользователя, пароль и информацию о роли. Объекты пользователей хранятся в таблице *users*. При вводе в базу данных нового сотрудника, во избежание появления пользователя с уже существующим логином, поле имени пользователя было сделано уникальным.

В магазине мебели по каждой единичной мебели следующие характеристики: название, стоимость, количество, категория и производитель. Модель мебели в программе содержит данные характеристики, а объекты хранятся в таблице *furnitures*. При реализации продажи мебели, количество данного экземпляра в базе данных будет уменьшаться на количество, равное общей совокупности данной мебели, купленной пользователем.

В магазине мебели для каждого гарнитура мебели определены следующие параметры: название гарнитура и перечень мебели, которые реализуют данный гарнитур. Модель гарнитура хранится в таблице *furnituresetitems.* Для реализации гарнитуров был использован фабричный метод, чтобы избежать ошибок при формирования гарнитура пользователем.

Для хранения отчетов по совершенным продажам в системе магазина существует модель «продаж» со следующими параметрами: название мебели, розничная цена, категория, производитель, проданное количество, дата покупки. Данная модель хранится в таблице *sales*. Эта таблица предназначена для хранения истории продаж и формирования *XML*-отчетов.

Созданные таблицы объединены в общую схему и связаны между собой каскадными связями по *ID* объектов. Таким образом при удалении или изменении объектов соответственно удаляются или изменяются объекты в других таблицах.

База данных, составленная таким образом, позволяет обеспечить удобный и простой доступ к данным, выполнять *CRUD* операции, комфортно осуществлять поиск и фильтрацию данных.

Так как в составленной схеме базы данных исключены повторяющиеся группы в отдельных таблицах, созданы отдельные таблицы для каждого набора связанных данных, созданы таблицы для наборов значений, применимых к нескольким записям, а также устранены поля, которые не зависят от ключа, то созданная база данных находится в третьей нормальной форме.

Схема базы данных *FurntureDB* представлена на рисунке 2.2.

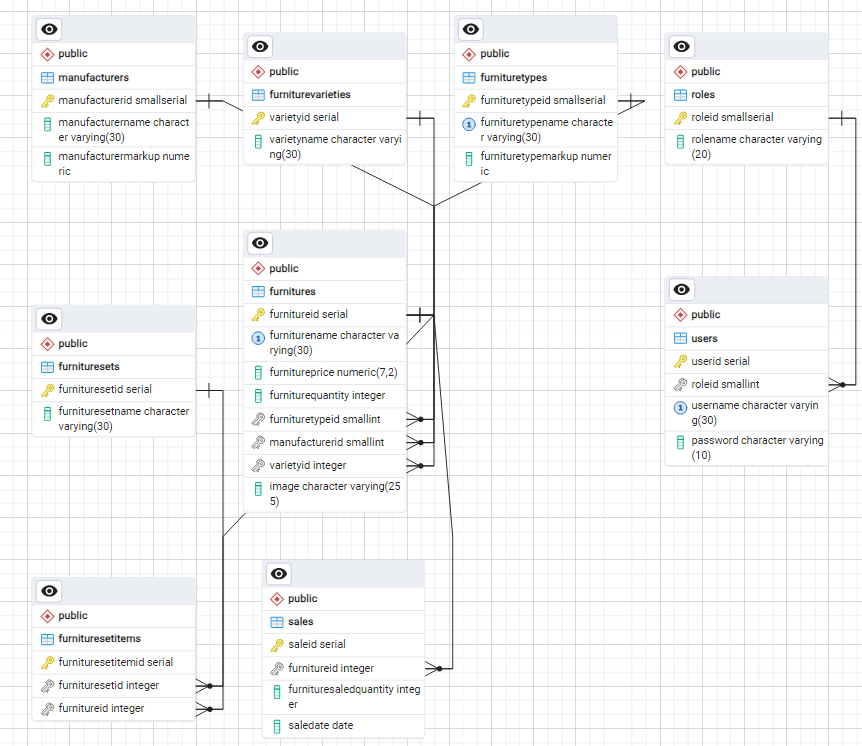


Рисунок 2.2 – Схема базы данных *FurntureDB*

Таблица *furnitures* является сущностью мебели и хранит в себе:

1. *ID* мебели – *furnitureId*. Данный столбец имеет тип *INTEGER* и является первичным ключом таблицы.
2. Названия мебели – *furnitureName*. Данный столбец имеет тип *VARCHAR(30)* и не может принимать значение *NULL* или пустую строку.
3. Цена мебели – *furnitureprice*. Данный столбец имеет тип *NUMERIC*(7,2) и не может принимать значение *NULL* или пустую строку.
4. Количество мебели – *furniturequantity*. Данный столбец имеет тип *INTEGER* и не может принимать значение *NULL* или пустую строку.
5. *ID* категории мебели – *furnituretypeid*. Данный столбец имеет отношение к таблице *furnituretypes* на каскадном удалении.
6. *ID* производителя мебели – *manufacturerid*. Данный столбец имеет отношение к таблице *manufacturers* на каскадном удалении.
7. *ID* вида мебели – *varietyid*. Данный столбец имеет отношение к таблице *furniturevarieties*.
8. Изображение мебели – *image*. Данный столбец имеет тип *VARCHAR*(255) и не может принимать значение *NULL* или пустую строку.

Заполнение таблицы *furnitures* представлено на рисунке 2.3.

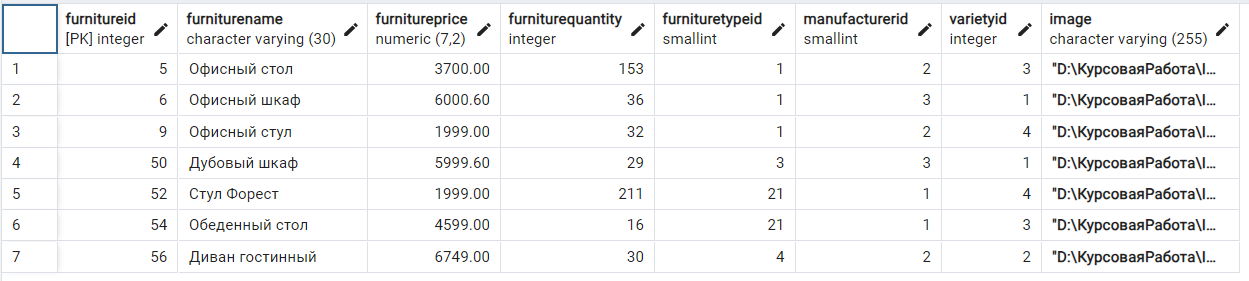


Рисунок 2.3 –­ Таблица *Furnitures*

Таблица *furnituretypes* является сущностью разных категорий мебели и хранит в себе:

1. *ID* категории мебели – *furnituretypeid*. Данный столбец имеет тип *SMALLINT* и является первичным ключом таблицы.
2. Название категории мебели – *furnituretypename*. Данный столбец имеет тип *VARCHAR(30)*, имеет атрибут *UNIQUE* и не может принимать значение *NULL* или пустую строку.
3. Наценка для категории – *furnituretypemarkup*. Данный столбец имеет тип *NUMERIC* и не может принимать значение *NULL*.

Заполнение таблицы *furnituretypes* представлено на рисунке 2.4.

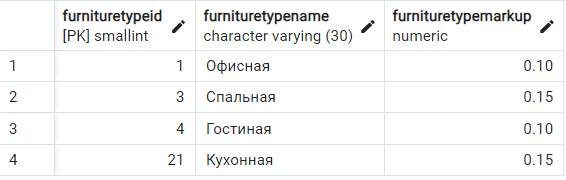


Рисунок 2.4 ­– Таблица *Furnituretypes*

Таблица *roles* является сущностью роли пользователя и хранит в себе:

1. *ID* роли – *RoleId*. Данный столбец имеет тип *SMALLINT* и является первичным ключом таблицы.
2. Название роли ­– *RoleName*. Данный столбец имеет тип *VARCHAR(20)*, имеет атрибут *UNIQUE* и не может принимать значение *NULL* или пустую строку.

Заполнение таблицы *Roles* представлено на рисунке 2.5.

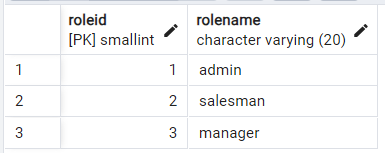


Рисунок 2.5 ­– Таблица *Roles*

Таблица *users* является сущностью пользователя и хранит в себе:

1. *ID* пользователя – *UserId*. Данный столбец имеет тип *INTEGER* и является первичным ключом таблицы.
2. *ID* роли пользователя – *RoleId*. Данный столбец имеет тип *TINYINT* *UNSIGNED*, не может принимать значение *NULL* и по умолчанию имеет значение равное 2.
3. Никнейм пользователя – *UserName*. Данный столбец имеет тип *VARCHAR(30)*, имеет атрибут *UNIQUE* и не может принимать значение *NULL* или пустую строку.
4. Пароль пользователя – *Password*. Данный столбец имеет тип *VARCHAR(10)* и не может принимать значение *NULL* или пустую строку.

Таблица *users* имеет внешний ключ *RoleId* и зависит от таблицы *roles*. Внешний ключ имеет атрибут *ON* *DELETE* *CASCADE*.

Заполнение таблицы *Users* представлено на рисунке 2.6.

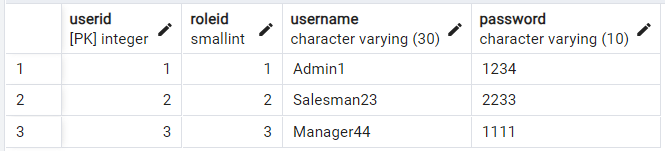


Рисунок 2.6 ­– Таблица *Users*

Таблица *manufacturers* является сущностью для разных производителей мебели и хранит в себе:

1. *ID* производителя мебели – *manufacturerid*. Данный столбец имеет тип *SMALLINT* и является первичным ключом таблицы.
2. Наименование производителя мебели – *manufacturername*. Данный столбец имеет тип *VARCHAR(30)*, имеет атрибут *UNIQUE* и не может принимать значение *NULL* или пустую строку.
3. Наценка производителя– *manufacturermarkup*. Данный столбец имеет тип *NUMERIC* и не может принимать значение *NULL*.

Заполнение таблицы *manufacturers* представлено на рисунке 2.7.

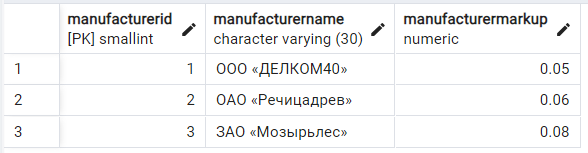


Рисунок 2.7 ­– Таблица *Manufacturers*

Таблица *sales* является сущностью для продаж мебели и хранит в себе:

1. *ID* продажи – *saleid*. Данный столбец имеет тип *INTEGER* и является первичным ключом таблицы.
2. *ID* мебели – *furnitureid*. Данный столбец имеет отношение к таблице *furnitures* на каскадном удалении.
3. Количество проданной мебели – *furnituresaledquantity*. Данный столбец имеет тип *INTEGER* и не может принимать отрицательные значения.
4. Дата продажи – *saledate.* Данный столбецимеет тип *DATE* и не может принимать значения *NULL*.

Заполнение таблицы *sales* представлено на рисунке 2.8.

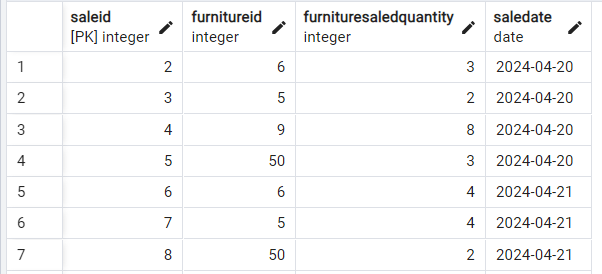


Рисунок 2.8– Таблица *Sales*

Таблица *furniturevarieties* является сущностью видов мебели и хранит в себе:

1. *ID* вида мебели – *varietyid*. Данный столбец имеет тип *SMALLINT* и является первичным ключом таблицы.
2. Наименование производителя мебели – *varietyname*. Данный столбец имеет тип *VARCHAR(30)*, имеет атрибут *UNIQUE* и не может принимать значение *NULL* или пустую строку.

Заполнение таблицы *furniturevarieties* представлено на рисунке 2.9.

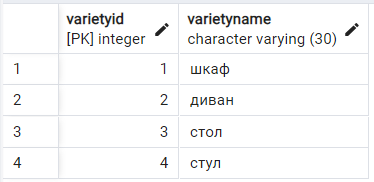


Рисунок 2.9 ­– Таблица *Furniturevarieties*

Таблица *furnituresets* является сущностью для разных категорий гарнитуров мебели и хранит в себе:

1. *ID* вида гарнитура мебели – *furnituresetid*. Данный столбец имеет тип *INTEGER* и является первичным ключом таблицы.
2. Наименование гарниутра мебели – *furnituresetname*. Данный столбец имеет тип *VARCHAR(30)*, имеет атрибут *UNIQUE* и не может принимать значение *NULL* или пустую строку.

Заполнение таблицы *furnituresets* представлено на рисунке 2.10.



Рисунок 2.10 ­– Таблица *Furnituresets*

Таблица *furnituresetitems* является сущностью перечня мебели, реализуемой гарнитуром и хранит в себе:

1. *ID* единица мебели в гарнитуре– *furnituresetitemid*. Данный столбец имеет тип *INTEGER* и является первичным ключом таблицы.
2. *ID* гарнитура мебели – *furnituresetid*. Данный столбец имеет отношение к таблице *furnituresets* на каскадном удалении.
3. *ID* мебели– *furnitureid*. Данный столбец имеет отношение к таблице *furnitures* на каскадном удалении.

Заполнение таблицы *furnituresetitems* представлено на рисунке 2.11.

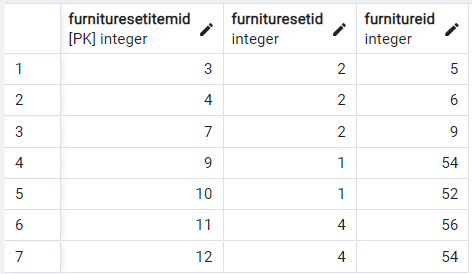


Рисунок 2.11 ­– Таблица *Furnituresetitems*

2.4 Иерархическая схема интерфейса

Основа пользовательского интерфейса базируется на том, что приложение должно иметь систему пользователей. Поэтому при проектировании интерфейса были разработаны окна с разным функционалом для администраторов, операторов и водителей. Все эти окна находятся в проекте *FurnitureShopWPF*. Иерархия окон интерфейса изображена на рисунке 2.2.

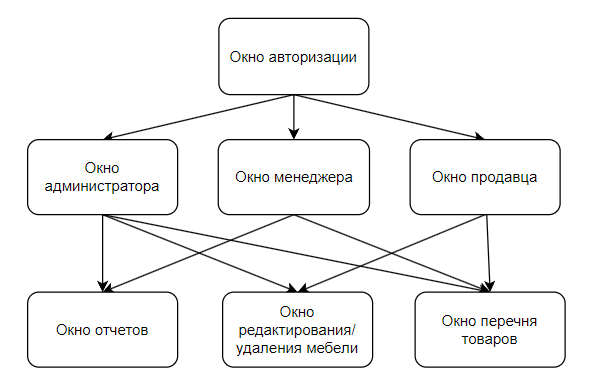


Рисунок 2.2 – Иерархия окон интерфейса

Окно авторизации отвечает за авторизацию пользователей. В нем находятся поля для ввода логина и пароля. Из этого окна, в зависимости от роли авторизированного пользователя, необходимо реализовать перенаправления на разные окна.

Администратор после авторизации перенаправляется в окно администратора, в котором находятся вкладки для перехода в такие окна, как:

– Окно перечней единичной мебели и гарнитуров мебели

– Окно отчетов

– Окно редактирования мебели

Менеджер после авторизации в системе направляется в окно менеджера, содержащее такие окна, как:

– Окно перечней единичной мебели и гарнитуров мебели

– Окно отчетов

Продавец после авторизации в системе направляется в окно продавца, содержащее такие окна, как:

– Окно перечней единичной мебели и гарнитуров мебели

– Окно редактирования мебели

Программа завершает работу в момент, когда все окна пользовательского интерфейса оказываются закрытыми.

2.5 Структура приложения

Структура классов решения изображена на рисунке 2.8.

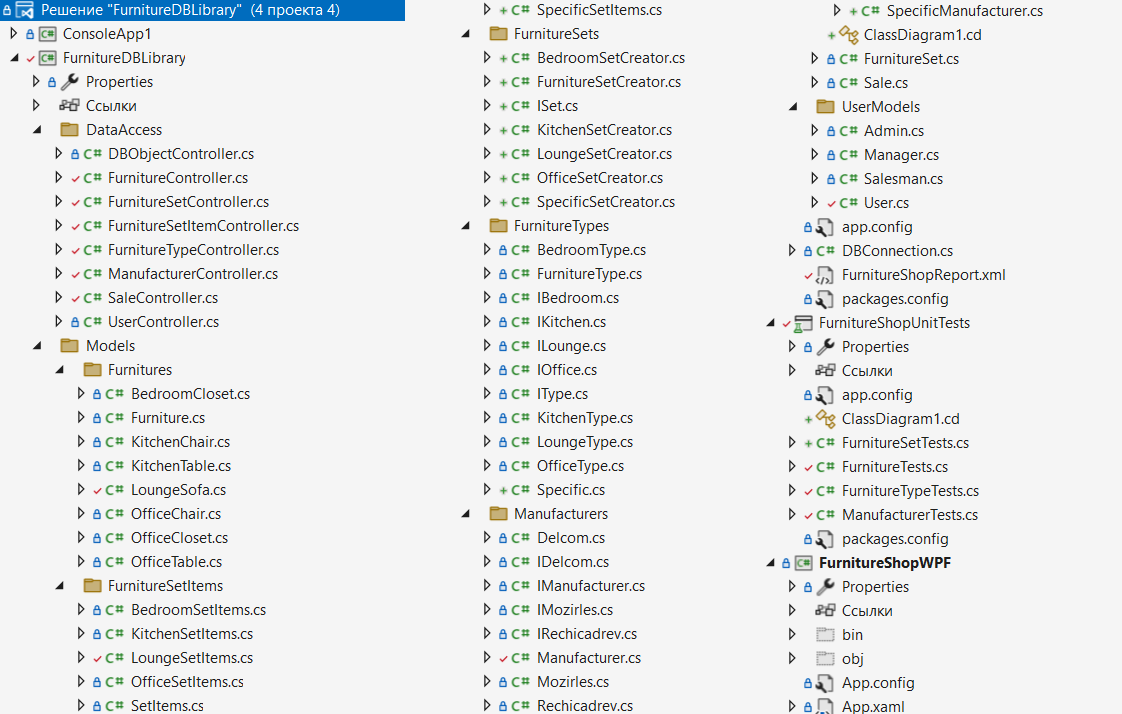


Рисунок 2.8 – Структура классов приложения

Реализация поставленной задачи требует разработки большого количества классов. Таким образом разработаны классы для реализации соединения с базой данных; классы, а также интерфейсы, отвечающие за *CRUD* операции, которые составляют *DAO* слой; классы, содержащие информацию о сущностях, которые необходимы для правильной работы приложения; классы, которые позволяют администратору создавать статистические необходимые статистические отчёты; классы, которые реализуют графический интерфейс; классы модульных тестов. Эти классы объединены в библиотеки классов: *FurnitureDBLIbrary*, *FurnitureShopUnitTests*, и *FurnitureShopWPF*.

3 ЭТАПЫ ПРОВЕДЕНИЯ ВЕРИФИКАЦИИ И ТЕСТИРОВАНИЯ  
 ПРИЛОЖЕНИЯ

3.1 Результаты проведения модульного тестирования

Одной из главных составляющих программы является ее тестирование на работоспособность.

Модульное тестирование – это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования [8].

Модульное тестирование, или юнит-тестирование (англ. *unit testing*) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного программного кода программы.

Концепция заключается в том, чтобы писать тесты для каждой нетривиальной функции или метода, что позволяет довольно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в ранее оттестированных местах программы, кроме того, облегчает обнаружение и устранение подобных ошибок.

На рисунке 3.1 приведены результаты проведения модульного тестирования.

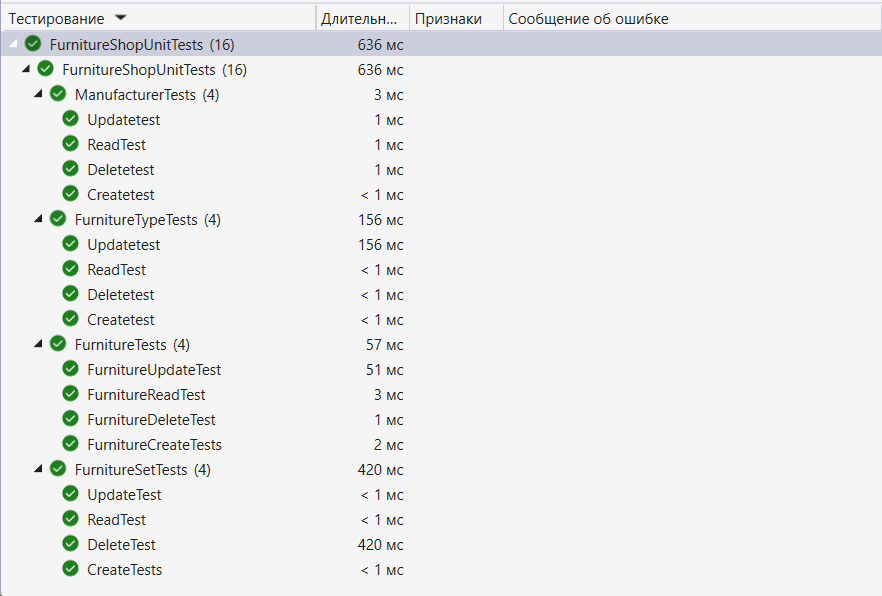


Рисунок 3.1 – Результат выполнения модульного тестирования

Модульное тестирование выполнено с помощью пакета *Moq*, листинг классов с модульными тестами находится в приложении В.

По результатам проведения модульного тестирования можно удостоверится в том, что программа работает корректно и выполняет все возложенные на неё функции.

На рисунке 3.2 приведена иерархия созданных классов для модульного тестирования.

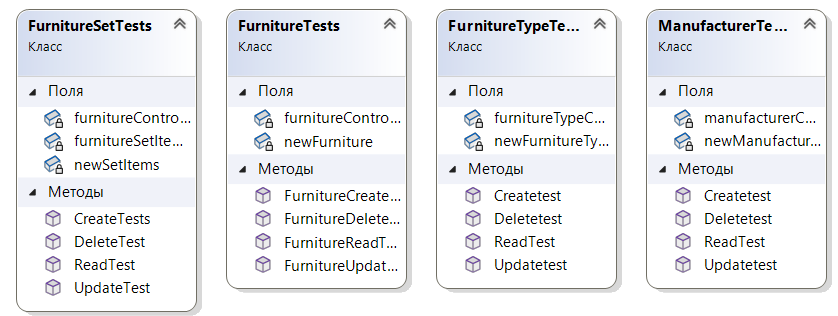


Рисунок 3.2 – Иерархия классов *Unit* тестов

3.2 Результаты проведения интегрированного тестирования

Интегрированное тестирование – одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интегрированное тестирование проводится после модульного тестирования.

Интегрированное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Целью интегрированного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц – объединения, множества или группы модулей – выполняется через их интерфейс, с использованием тестирования «чёрного ящика». [9]

Тестирование таким образом позволяет убедиться, что приложение полностью и правильно отображает данные, верно их изменяет и сохраняет. Такая проверка помогает найти слабые места и возможные ошибки, которые не выявило модульное тестирование.

Взаимодействие пользователя с приложением не должно вызывать ошибок или неполадок. Все непоследовательные действия пользователя должны быть предусмотрены и пользователю должно выводиться соответствующее сообщение о неверных действиях. Особое внимание уделено текстовым полям для ввода информации. Ввод некорректных данных проверяется и выводится соответствующее сообщение.

Для работы с программой необходимо открыть файл «*FurnitureShopWPF»*.

После появления оконного приложения пользователю предлагается один из нескольких вариантов:войти как гость или ввести логин и пароль и авторизоваться в системе. Окно авторизации представлено на рисунке 3.3. Логин и пароль проверяется на наличие в базе данных. Исходными данными в поставленной задаче являются предварительно сгенерированный набор данных, которые понадобятся для верификации приложения и проверки на функциональность.

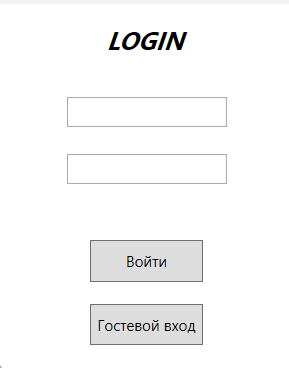


Рисунок 3.3 – Окно авторизации

Система предупреждает пользователя, если тот ввёл некорректные данные при входе, то есть, при попытке войти под логином и паролем пользователя, которого не существует в базе данных, будет выведено соответствующее сообщение. Пример некорректного ввода представлен на рисунке 3.4.

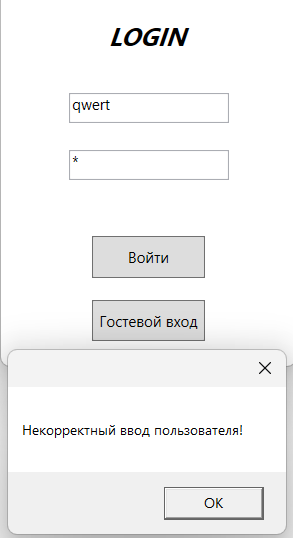


Рисунок 3.4 – Пример некорректного ввода данных

После успешной авторизации пользователя, текущее окно закрывается и открывается окно с главным меню в зависимости от типа аккаунта.

У разных типов аккаунтов отличаются доступные им возможности. После авторизации в системе администратора, главное окно с доступными ему функциями изображено на рисунке 3.7. Кроме доступных функций, администратор может формировать отчёты с возможность открытия в электронных таблицах о продажах мебели. Отчеты также доступны для менеджера.

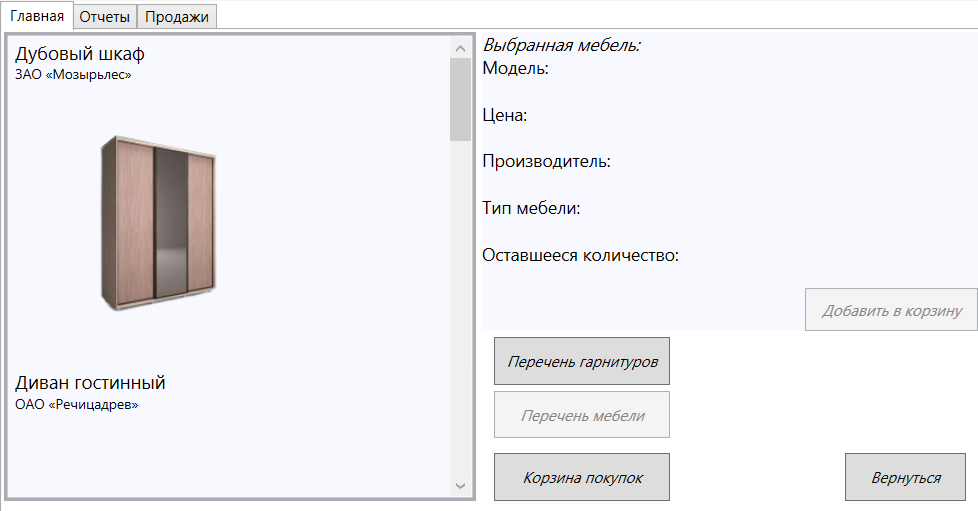


Рисунок 3.7 – Окно для администратора

При совершении приобретения товара, например, при покупке мебели, данные проверяются на наличие в базе данных (т.е. проверяется уникальность ведёной даты сделки). При успешном редактировании, либо удалении, информация загружается в базу данных. Аналогичным образом работа администратор выполняется на вкладках «Главная» и «Продажи».

При нажатии на вкладку «Отчёты» администратор может просмотреть сформированные отчёты по количеству проданных товаров по каждому производителю и выручку с них, это представлено на рисунке 3.8.

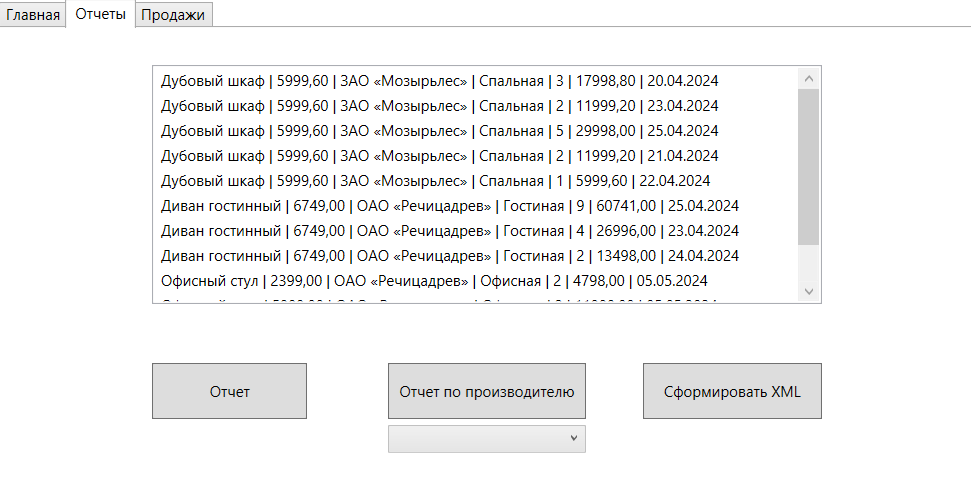


Рисунок 3.8 – Вкладка «Отчёты» для администратора

Во вкладке «Отчеты» для администратора и менеджера имеется функция формирования отчетов в формате *XML* для последующего открытия отчета в электронной таблице. На рисунке 3.9 представлен отчет в электронной таблице *Excel.*

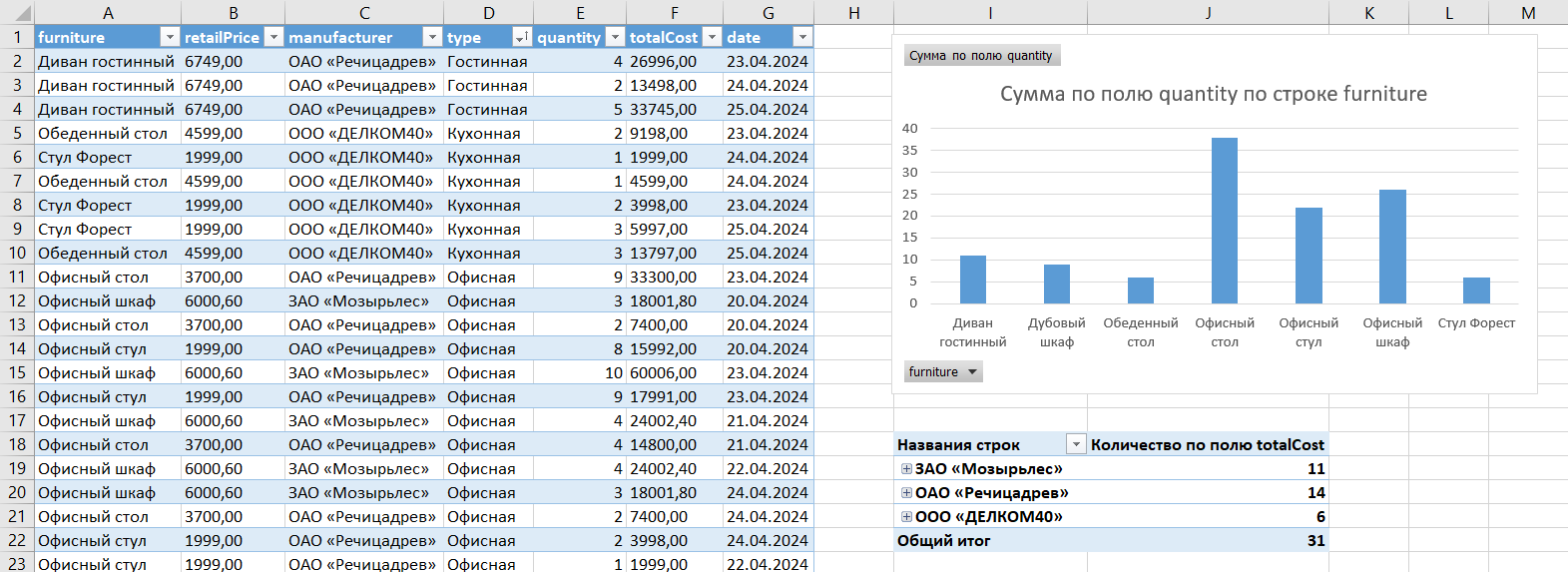


Рисунок 3.9 – Таблица отчетов по продажам мебели

Если пользователь хочет купить конкретную единицу мебели, то ему необходимо нажать на нужный товар, затем на кнопку «Добавить в корзину». После нажатия на кнопку выбранный товар будет добавлен в корзину покупок, оставаясь в корзину до момента покупки. В зависимости от производителя и категории мебели, устанавливается соответствующая наценка. Пример покупки мебели представлен на рисунке 3.10.

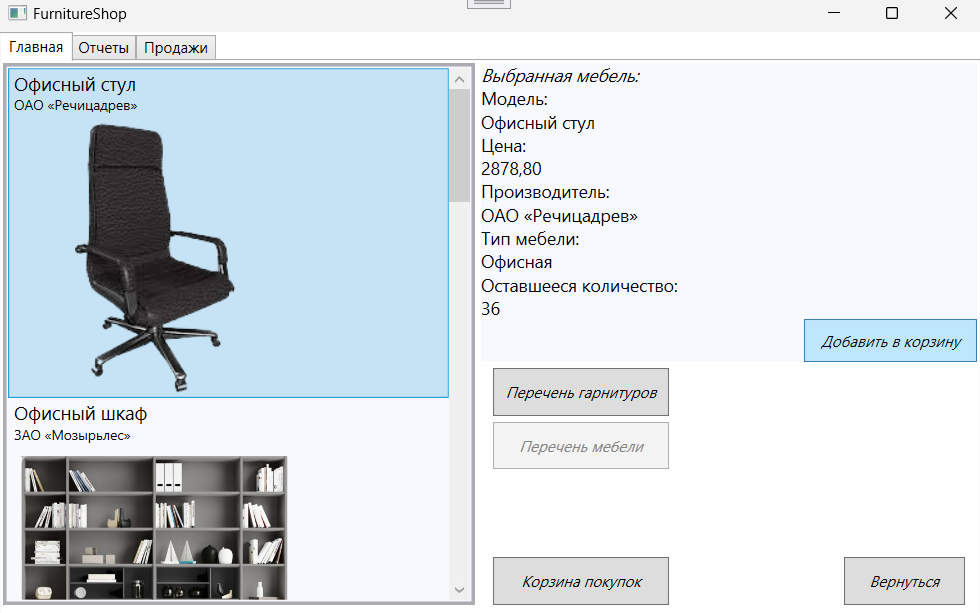


Рисунок 3.10 – Покупка мебели

Перейдя на «Корзину покупок», пользователь может ознакомиться со своими купленными товарами, рисунок 3.11. У пользователя есть возможность отменить покупку товара при необходимости.

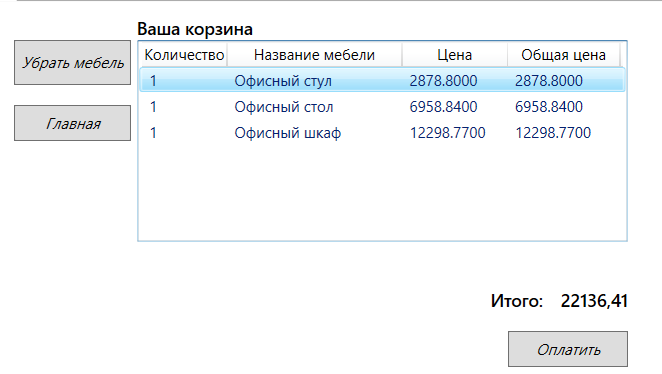


Рисунок 3.11 – Корзина покупок пользователя

Окно продавца предоставляет возможность редактирования, добавления и удаления заданных реализуемых товаров. При манипулировании характеристиками уже реализуемого магазином товара, измененный параметры также отразятся и в базе данных. Окно продаж представлено на рисунке 3.12

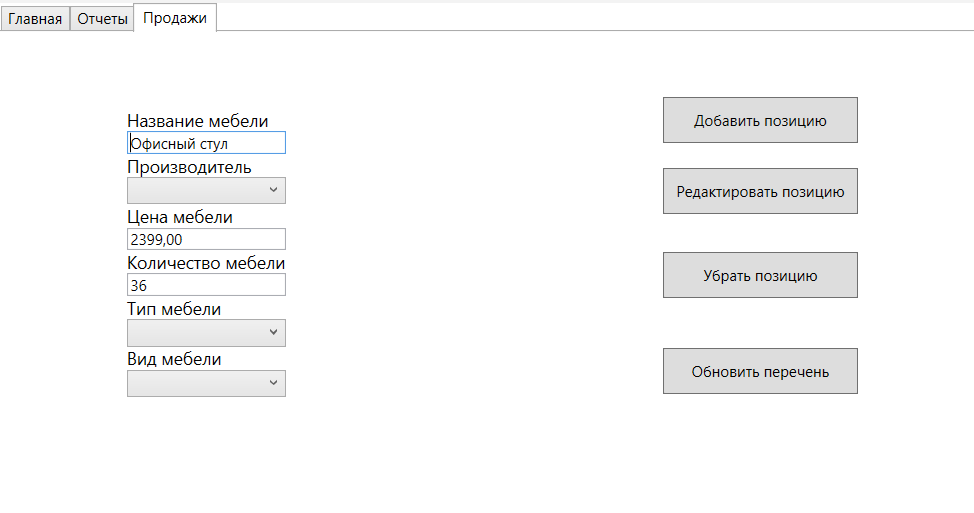


Рисунок 3.12 – Окно редактирования продаж

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

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта был разработан программный комплекс для реализации продаж мебели, который также является информационной системой продажи мебели.

Проведен аналитический обзор средств разработки, достаточных для освоения заданной темы. Затем были выделены несколько основных задач и составлен алгоритм разработки программного обеспечения. Основным этапом являлась разработка основных классов и логики программы. Для последующей расширяемости использованы шаблоны проектирования.

В разработке активно применялись современные технологии и подходы к решениям задач: для хранения данных использовалась база данных *PostgreSql, LINQ* *to Objects* для доступа к данным, средства языка программирования *C# WPF Application* и такие шаблоны проектирования, как: *MVC, singleton, factory method*.

Результатом разработки курсового проекта, является десктопное приложение с графическим интерфейсом пользователя с использованием фреймворка *WPF Application*. Приложение имеет многопользовательский доступ к системе, что решает проблему с целостностью данных, а также с безопасностью системы.

Архитектура приложения достаточно проста, чтобы её было по силам изменить под определённые требования заказчика, при этом не теряя в качестве и функциональности самой информационной системы.

Проведенные модульные и интегрированные тесты доказывают работоспособность данной программы и функционирование всех требуемых ролей и методов. Созданная информационная система полностью соответствует представленным требованиям и выполняет все необходимые задачи. Полученное приложение готово для работы с пользователем.

Курсовой проект был проверен в системе «Антиплагиат». Процент оригинальности составил 93,31%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вайсфельд, М. Объектно-ориентированное мышление в программировании / М.Вайсфельд – СПб.: Питер, 2014. – 304 с.
2. Объектно-ориентированный подход к программированию: Файловый архив для студентов. *Studfiles.* – Электрон. данные. – Режим доступа: https://studfile.net/preview/2059940/page:16/. – Дата доступа: 12.03.2023.
3. Введение в базы данных. Основные понятия и определения. – Электрон. данные. – Режим доступа: <https://siblec.ru/informatika-i-vychislitelnaya-tekhnika/bazy-dannykh>. – Дата доступа: 15.03.2023.
4. Обзор языка *C#* – Руководство по *C#*: *Microsoft Docs*. – Электрон. данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/>. – Дата доступа: 16.03.2023.
5. Принципы *SOLID* в *C#*. – Электрон. данные. – Режим доступа: <https://professorweb.ru/my/it/blog/net/solid.php>. – Дата доступа: 17.03.2023.
6. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# – ООО Издательство «Питер», 2013. – 896 с.
7. Шаблоны проектирования: Библиотека программиста. – Электрон. данные. – Режим доступа: https://proglib.io/p/creational-patterns/. – Дата доступа: 12.14.2023.
8. Основы тестирования программного обеспечения: Файловый архив для студентов. *Studfiles*. – Электрон. данные. – Режим доступа: https://studfile.net/preview/4494386/. – Дата доступа: 18.03.2023.
9. Дипломное проектирование: учеб.-метод. пособие для студентов специальностей 1-40 04 01 «Информатика и технологии программирования» и 1-40 05 01 «Информационные системы и технологии (по направлениям)» днев. формы обучения / сост.: Т.А. Трохова, И.А. Мурашко, К.С. Курочка. – Гомель: ГГТУ им. П.О. Сухого, 2019. – 55 с. – Режим доступа: <https://elib.gstu.by>. – Дата доступа: 20.04.2023.

ПРИЛОЖЕНИЕ А

(обязательное)

**ФУНКЦИОНАЛЬНАЯ СХЕМА ПРИЛОЖЕНИЯ**

ПРИЛОЖЕНИЕ Б

(обязательное)

**ЛИСТИНГ ПРОГРАММЫ**

**Код класса *LoginWindow.xaml.cs***

using FurnitureDBLibrary.DataAccess;

using FurnitureDBLibrary.UserModels;

using System;

using System.Collections.Generic;

using System.Windows;

namespace FurnitureShopWPF

{

/// <summary>

/// Логика взаимодействия для LoginWindow.xaml

/// </summary>

public partial class LoginWindow : Window

{

private UserController \_userController = new UserController();

private List<User> \_users;

public LoginWindow()

{

try

{

\_users = \_userController.Read();

InitializeComponent();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void EnterButton\_Click(object sender, RoutedEventArgs e)

{

try

{

if (\_users.Find(u => u.UserName == UserNameTextBox.Text && u.Password == Passwordbox.Password) != null)

{

\_users = \_userController.Read();

User currentUser = \_users.Find(u => u.UserName == UserNameTextBox.Text && u.Password == Passwordbox.Password);

MainWindow mainWindow = new MainWindow(currentUser);

this.Close();

mainWindow.Show();

}

else

throw new Exception();

}

catch (Exception)

{

MessageBox.Show("Некорректный ввод пользователя!");

}

}

private void GuestEnterButton\_Click(object sender, RoutedEventArgs e)

{

MainWindow mainWindow = new MainWindow();

this.Close();

mainWindow.Show();

}

}

}

**Код класса *MainWindow.xaml.cs***

using FurnitureDBLibrary;

using FurnitureDBLibrary.DataAccess;

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.UserModels;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Data.SqlClient;

using System.Linq;

using System.Runtime.Remoting.Messaging;

using System.Windows;

using System.Windows.Controls;

namespace FurnitureShopWPF

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

readonly User \_user;

Furniture \_currentFurniture;

FurnitureSet \_currentFurnitureSet;

List<Furniture> \_furnitures;

List<Furniture> \_furnitureList = new List<Furniture>();

List<Manufacturer> \_manufacturers;

List<FurnitureType> \_furnitureTypes;

List<FurnitureSet> \_furnitureSets;

List<SetItems> \_furnitureSetItems;

SetItems \_currentFurnitureSetItems;

List<Sale> \_sales;

bool isReLogined = false;

readonly SaleController \_saleController = new SaleController();

readonly FurnitureSetItemController \_furnitureSetItemController = new FurnitureSetItemController();

readonly ManufacturerController \_manufacturerController = new ManufacturerController();

readonly FurnitureTypeController \_furnitureTypeController = new FurnitureTypeController();

readonly FurnitureController \_furnitureController = new FurnitureController();

public MainWindow(User user)

{

try

{

\_user = user;

InitializeComponent();

switch (\_user.RoleName.ToLower())

{

case "admin":

InitializeAdminInterface();

break;

case "salesman":

InitializeSalesmanInterface();

break;

case "manager":

InitializeManagerInterface();

break;

default:

throw new Exception();

}

FurnitureSetListBox.Visibility = Visibility.Hidden;

FurnitureSetStackPanel.Visibility = Visibility.Hidden;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public MainWindow()

{

InitializeComponent();

FurnitureSetListBox.Visibility = Visibility.Hidden;

FurnitureSetStackPanel.Visibility = Visibility.Hidden;

}

public void FurnitureListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (BuyFurnitureButton.IsEnabled == false)

BuyFurnitureButton.IsEnabled = true;

if (FurnitureListBox.SelectedItem != null)

{

\_currentFurniture = (Furniture)FurnitureListBox.Items[FurnitureListBox.SelectedIndex];

NameTextBox.Text = \_currentFurniture.FurnitureName;

PriceTextBox.Text = \_currentFurniture.GetRetailPrice().ToString("#.00");

ManufacturerTextBox.Text = \_currentFurniture.ManufacturerName;

TypeTextBox.Text = \_currentFurniture.TypeName;

QuantityTextBox.Text = \_currentFurniture.FurnitureQuantity.ToString();

}

}

private void BuyButton\_Click(object sender, RoutedEventArgs e)

{

\_furnitureList.Add(\_currentFurniture);

--\_currentFurniture.FurnitureQuantity;

}

private void LoginWindow\_Click(object sender, RoutedEventArgs e)

{

LoginWindow loginWindow = new LoginWindow();

isReLogined = true;

this.Close();

loginWindow.Show();

}

private void FurnitureButton\_Click(object sender, RoutedEventArgs e)

{

FurnitureListBox.Visibility = Visibility.Visible;

FurnitureStackPanel.Visibility = Visibility.Visible;

FurnitureSetListBox.Visibility = Visibility.Hidden;

FurnitureSetStackPanel.Visibility = Visibility.Hidden;

\_furnitures = \_furnitureController.Read();

FurnitureButton.IsEnabled = false;

FurnitureSetButton.IsEnabled = true;

FurnitureListBox.ItemsSource = \_furnitures;

}

private void FurnitureSetButton\_Click(object sender, RoutedEventArgs e)

{

FurnitureSetController furnitureSetController = new FurnitureSetController();

FurnitureSetListBox.Visibility = Visibility.Visible;

FurnitureSetStackPanel.Visibility = Visibility.Visible;

FurnitureListBox.Visibility = Visibility.Hidden;

FurnitureStackPanel.Visibility = Visibility.Hidden;

BuyFurnitureButton.IsEnabled = false;

\_furnitureSets = furnitureSetController.Read();

\_furnitureSetItems = \_furnitureSetItemController.Read();

\_furnitures = \_furnitureController.Read();

FurnitureButton.IsEnabled = true;

FurnitureSetButton.IsEnabled = false;

FurnitureSetListBox.ItemsSource = \_furnitureSets;

}

private void FurnitureSetListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (BuyFurnitureSetButton.IsEnabled == false)

BuyFurnitureSetButton.IsEnabled = true;

if (FurnitureSetListBox.SelectedItem != null)

{

try

{

\_furnitureSetItems = \_furnitureSetItemController.Read();

decimal totalCost = 0;

\_currentFurnitureSet = (FurnitureSet)FurnitureSetListBox.Items[FurnitureSetListBox.SelectedIndex];

\_currentFurnitureSetItems = \_furnitureSetItems.Find(items => items.SetName == \_currentFurnitureSet.FurnitureSetName);

if (\_currentFurnitureSetItems.FurnitureList != null)

{

foreach (var item in \_currentFurnitureSetItems.FurnitureList)

totalCost += item.GetRetailPrice();

SetItemNameListBox.ItemsSource = \_currentFurnitureSetItems.FurnitureList;

TotalCostTextBlock.Text = "Общая стоимость: "+ totalCost.ToString("#.00");

}

else

throw new Exception();

}

catch (Exception)

{

MessageBox.Show("Данный гарнитур неукомплектован");

}

}

}

private void BuySetButton\_Click(object sender, RoutedEventArgs e)

{

foreach (Furniture furnitureSetItem in \_currentFurnitureSetItems.FurnitureList)

{

\_furnitureList.Add(furnitureSetItem);

}

}

private void InitializeAdminInterface()

{

MainTabItem.Visibility = Visibility.Visible;

ReportsTabItem.Visibility = Visibility.Visible;

SalesTabItem.Visibility = Visibility.Visible;

}

private void InitializeManagerInterface()

{

MainTabItem.Visibility = Visibility.Visible;

ReportsTabItem.Visibility = Visibility.Visible;

}

private void InitializeSalesmanInterface()

{

MainTabItem.Visibility = Visibility.Visible;

SalesTabItem.Visibility = Visibility.Visible;

}

private void AddFurnitureButton\_Click(object sender, RoutedEventArgs e)

{

try

{

Furniture furniture;

Manufacturer curManufacturer = \_manufacturers.Find(m => m.ManufacturerName == ManufacturerComboBox.Items[ManufacturerComboBox.SelectedIndex].ToString());

FurnitureType curFurnitureType = \_furnitureTypes.Find(t => t.TypeName == FurnitureTypesComboBox.Items[FurnitureTypesComboBox.SelectedIndex].ToString());

switch (FurnitureVarietyComboBox.Items[FurnitureVarietyComboBox.SelectedIndex].ToString().ToLower())

{

case "стул":

switch (FurnitureTypesComboBox.Items[FurnitureTypesComboBox.SelectedIndex].ToString().ToLower())

{

case "кухонная":

furniture = new KitchenChair(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

case "офисная":

furniture = new OfficeChair(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

}

break;

case "стол":

switch (FurnitureTypesComboBox.Items[FurnitureTypesComboBox.SelectedIndex].ToString().ToLower())

{

case "кухонная":

furniture = new KitchenTable(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

case "офисная":

furniture = new OfficeTable(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

}

break;

case "шкаф":

switch (FurnitureTypesComboBox.Items[FurnitureTypesComboBox.SelectedIndex].ToString().ToLower())

{

case "спальная":

furniture = new BedroomCloset(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

case "офисная":

furniture = new OfficeCloset(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

}

break;

case "диван":

switch (FurnitureTypesComboBox.Items[FurnitureTypesComboBox.SelectedIndex].ToString().ToLower())

{

case "гостинная":

furniture = new LoungeSofa(FurnitureNameTextBox.Text, Convert.ToDecimal(FurniturePriceTextBox.Text), Convert.ToInt32(FurnitureQuantityTextBox.Text));

\_furnitureController.Create(furniture);

break;

}

break;

default:

throw new Exception("Магазин не продает данный тип мебели");

}

}

catch (Exception)

{

MessageBox.Show("Ошибка ввода параметров товара");

}

}

private void UpdateFurnitureButton\_Click(object sender, RoutedEventArgs e)

{

try

{

\_currentFurniture = (Furniture)FurnitureListBox.Items[FurnitureListBox.SelectedIndex];

\_currentFurniture.FurnitureQuantity = Convert.ToInt32(FurnitureQuantityTextBox.Text);

\_furnitureController.Update(\_currentFurniture);

}

catch (Exception)

{

MessageBox.Show("Ошибка указания товара!");

}

}

private void DeleteFurnitureButton\_Click(object sender, RoutedEventArgs e)

{

try

{

\_currentFurniture = (Furniture)FurnitureListBox.Items[FurnitureListBox.SelectedIndex];

\_furnitureController.Delete(\_currentFurniture);

}

catch (Exception)

{

MessageBox.Show("Ошибка указания товара!");

}

}

private void RefreshFurnitureLBButton\_Click(object sender, RoutedEventArgs e)

{

\_furnitures = \_furnitureController.Read();

FurnitureListBox.ItemsSource = \_furnitures;

}

private void CreateReportButton\_Click(object sender, RoutedEventArgs e)

{

\_sales = \_saleController.Read();

ReportListBox.ItemsSource = \_sales;

}

private void ReportByManufacturer\_Click(object sender, RoutedEventArgs e)

{

if (ManufacturerName.SelectedIndex != -1)

{

\_sales = \_saleController.Read();

string manufacturerName = ManufacturerName.Items[ManufacturerName.SelectedIndex].ToString();

var salesByManufacturer = \_sales.FindAll(sale => sale.ManufacturerName == manufacturerName);

ReportListBox.ItemsSource = salesByManufacturer;

}

else

MessageBox.Show("Необходимо выбрать производителя");

}

private void TabControl\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (e.AddedItems.Contains(SalesTabItem))

{

\_furnitures = \_furnitureController.Read();

\_manufacturers = \_manufacturerController.Read();

\_furnitureTypes = \_furnitureTypeController.Read();

ManufacturerComboBox.ItemsSource = \_manufacturers;

FurnitureTypesComboBox.ItemsSource = \_furnitureTypes;

FurnitureVarietyComboBox.ItemsSource = \_furnitureController.GetVarieties(\_furnitures).Distinct();

if (FurnitureListBox.SelectedIndex != -1)

{

\_currentFurniture = (Furniture)FurnitureListBox.Items[FurnitureListBox.SelectedIndex];

FurnitureNameTextBox.Text = \_currentFurniture.FurnitureName;

FurniturePriceTextBox.Text = \_currentFurniture.FurniturePrice.ToString();

FurnitureQuantityTextBox.Text = \_currentFurniture.FurnitureQuantity.ToString();

}

}

if (e.AddedItems.Contains(ReportsTabItem))

{

ManufacturerName.ItemsSource = \_manufacturerController.Read();

}

}

private void XMLReportButton\_Click(object sender, RoutedEventArgs e)

{

\_sales = \_saleController.Read();

\_saleController.GenerateXMLReport(\_sales);

MessageBox.Show("Отчет успешно сформирован в XML формате");

}

private void CartButton\_Click(object sender, RoutedEventArgs e)

{

CartWindow cartWindow = new CartWindow(\_furnitureList);

cartWindow.Show();

}

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

if (!isReLogined)

DBConnection.GetInstance.CloseConnection();

}

}

}

**Код класса *User***

namespace FurnitureDBLibrary.UserModels

{

public abstract class User

{

private string \_userName;

private string \_password;

public User(string userName, string password)

{

\_userName = userName;

\_password = password;

}

public string UserName { get { return \_userName; } }

public string Password { get { return \_password; } }

public abstract string RoleName { get; }

}

}

**Код класса  *Salesman.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.UserModels

{

public class Salesman : User

{

public Salesman(string userName, string password) : base(userName, password) { }

public override string RoleName { get { return "Salesman"; } }

}

}

**Код класса  *Manager***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.UserModels

{

public class Manager : User

{

public Manager(string userName, string password) : base(userName, password) { }

public override string RoleName { get { return "Manager"; } }

}

}

**Код класса  *Admin***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.UserModels

{

public class Admin : User

{

public Admin(string userName, string password) : base(userName, password) { }

public override string RoleName { get { return "Admin"; } }

}

}

**Код класса  *Sale***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models

{

public class Sale

{

private string \_furnitureName;

private decimal \_furnitureRetailPrice;

private string \_manufacturerName;

private string \_typeName;

private int \_furnitureSaledQuantity;

private DateTime \_saleDate;

private string \_setName;

public Sale(string furnitureName, decimal furnitureRetailPrice,string manufacturerName,string typeName, int furnitureSaledQuantity, DateTime saleDate)

{

\_furnitureName = furnitureName;

\_furnitureRetailPrice = furnitureRetailPrice;

\_manufacturerName = manufacturerName;

\_typeName = typeName;

\_furnitureSaledQuantity = furnitureSaledQuantity;

\_saleDate = saleDate;

}

public string FurnitureName { get { return \_furnitureName; } }

public decimal FurnitureRetailPrice { get { return \_furnitureRetailPrice; } }

public string ManufacturerName { get { return \_manufacturerName; } }

public string TypeName { get { return \_typeName; } }

public int FurnitureSaledQuantity { get { return \_furnitureSaledQuantity; } set { \_furnitureSaledQuantity = value; } }

public DateTime SaleDate { get { return \_saleDate; } }

public string SetName { get { return \_setName; } }

public decimal TotalCost

{

get

{

return FurnitureRetailPrice \* \_furnitureSaledQuantity;

}

}

public override string ToString()

{

return $"{FurnitureName} | {FurnitureRetailPrice:#.00} | {ManufacturerName} | {TypeName} | {FurnitureSaledQuantity} | {TotalCost} | {SaleDate:d}";

}

}

}

**Код класса  *FurnitureSet***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models

{

public class FurnitureSet

{

private string \_furnitureSetName;

public FurnitureSet(string furnitureSetName)

{

\_furnitureSetName = furnitureSetName;

}

public string FurnitureSetName { get { return \_furnitureSetName; } }

}

}

**Код класса  *DBConnection***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Configuration;

using Npgsql;

namespace FurnitureDBLibrary

{

public class DBConnection

{

private static readonly DBConnection \_instance = new DBConnection();

private readonly string \_connectionString = ConfigurationManager.ConnectionStrings["NpgsqlConnectionString"].ConnectionString;

private readonly NpgsqlConnection \_connection;

private DBConnection()

{

\_connection = new NpgsqlConnection(\_connectionString);

\_connection.Open();

}

public static DBConnection GetInstance { get { return \_instance; } }

public NpgsqlConnection Connection { get { return \_connection; } }

public void CloseConnection()

{

\_connection.Close();

\_connection.Dispose();

}

}

}

**Код класса Furniture**

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models

{

public abstract class Furniture : IType, IManufacturer

{

private string \_furnitureName;

private decimal \_furniturePrice;

private int \_furnitureQuantity;

public Furniture(string furnitureName, decimal furniturePrice, int furnitureQuantity)

{

\_furnitureName = furnitureName;

\_furniturePrice = furniturePrice;

\_furnitureQuantity = furnitureQuantity;

}

public string FurnitureName { get { return \_furnitureName; } }

public decimal FurniturePrice { get { return \_furniturePrice; } }

public int FurnitureQuantity { get { return \_furnitureQuantity; } set { \_furnitureQuantity = value; } }

public override string ToString()

{

return $"{FurnitureName}\t{GetRetailPrice():#.00}";

}

public abstract string TypeName { get; }

public abstract decimal TypeMarkup { get; }

public abstract string ManufacturerName { get; }

public abstract decimal ManufacturerMarkup { get; }

public abstract string FurnitureVariety { get; }

public abstract string FurnitureImage { get; }

public abstract decimal GetRetailPrice();

}

}

**Код класса  *BedroomCloset***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.CurrentFurnitures

{

public class BedroomCloset : Furniture, IBedroom, IMozirles

{

public BedroomCloset(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Шкаф"; } }

public override string TypeName { get { return "Спальная"; } }

public override decimal TypeMarkup { get { return (decimal)0.15; } }

public override string ManufacturerName { get { return "ЗАО «Мозырьлес»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.08; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\OakCloset.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *KitchenChair***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.CurrentFurnitures

{

public class KitchenChair : Furniture, IKitchen, IDelcom

{

public KitchenChair(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Стул"; } }

public override string TypeName { get { return "Кухонная"; } }

public override decimal TypeMarkup { get { return (decimal)0.15; } }

public override string ManufacturerName { get { return "ООО «ДЕЛКОМ40»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.05; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\KitchenChair.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *KitchenTable***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Furnitures

{

public class KitchenTable : Furniture, IKitchen, IDelcom

{

public KitchenTable(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Стол"; } }

public override string TypeName { get { return "Кухонная"; } }

public override decimal TypeMarkup { get { return (decimal)0.15; } }

public override string ManufacturerName { get { return "ООО «ДЕЛКОМ40»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.05; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\KitchenTable.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *LoungeSofa***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.CurrentFurnitures

{

public class LoungeSofa : Furniture, ILounge, IRechicadrev

{

public LoungeSofa(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Диван"; } }

public override string TypeName { get { return "Гостиная"; } }

public override decimal TypeMarkup { get { return (decimal)0.15; } }

public override string ManufacturerName { get { return "ОАО «Речицадрев»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.05; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\LoungeSofa.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *OfficeChair***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Furnitures

{

public class OfficeChair : Furniture, IOffice, IRechicadrev

{

public OfficeChair(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Стул"; } }

public override string TypeName { get { return "Офисная"; } }

public override decimal TypeMarkup { get { return (decimal)0.15; } }

public override string ManufacturerName { get { return "ОАО «Речицадрев»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.05; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\OfficeChair.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *OfficeCloset***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Furnitures

{

public class OfficeCloset : Furniture, IOffice, IMozirles

{

public OfficeCloset(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Шкаф"; } }

public override string TypeName { get { return "Офисная"; } }

public override decimal TypeMarkup { get { return (decimal)0.15; } }

public override string ManufacturerName { get { return "ЗАО «Мозырьлес»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.08; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\OfficeCloset.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *OfficeTable***

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.CurrentFurnitures

{

public class OfficeTable : Furniture, IOffice, IRechicadrev

{

public OfficeTable(string furnitureName, decimal furniturePrice, int furnitureQuantity)

: base(furnitureName, furniturePrice, furnitureQuantity) { }

public override string FurnitureVariety { get { return "Стол"; } }

public override string TypeName { get { return "Офисная"; } }

public override decimal TypeMarkup { get { return (decimal)0.10; } }

public override string ManufacturerName { get { return "ОАО «Речицадрев»"; } }

public override decimal ManufacturerMarkup { get { return (decimal)0.06; } }

public override string FurnitureImage { get { return @"D:\КурсоваяРабота\Images\OfficeTable.png"; } }

public override decimal GetRetailPrice()

{

return FurniturePrice + FurniturePrice \* TypeMarkup + FurniturePrice \* ManufacturerMarkup;

}

}

}

**Код класса  *SetItems***

using FurnitureDBLibrary.Models.FurnitureSetItems;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models

{

public abstract class SetItems : ISet

{

private readonly List<Furniture> \_furnitures = new List<Furniture>();

public SetItems(List<Furniture> furnitures)

{

\_furnitures = furnitures;

}

public List<Furniture> FurnitureList { get { return \_furnitures; } }

public abstract string SetName { get; }

public override string ToString()

{

string info = "";

foreach (var item in \_furnitures)

{

info += $"{item.FurnitureName}\t{item.GetRetailPrice():#.##}\n";

}

return info;

}

}

}

**Код класса  *BedroomSetItems***

using FurnitureDBLibrary.Models.CurrentFurnitures;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSetItems

{

public class BedroomSetItems : SetItems

{

public BedroomSetItems(List<Furniture> furnitures)

: base(furnitures) { }

public override string SetName { get { return "Спальный"; } }

}

}

**Код класса  *KitchenSetItems***

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models

{

public class KitchenSetItems : SetItems

{

public KitchenSetItems(List<Furniture> furnitures)

: base(furnitures) { }

public override string SetName { get { return "Кухонный"; } }

}

}

**Код класса  *LoungeSetItems***

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSetItems

{

public class LoungeSetItems : SetItems

{

public LoungeSetItems(List<Furniture> furnitures)

: base(furnitures) { }

public override string SetName { get { return "Гостиный"; } }

}

}

**Код класса**

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSetItems

{

public class OfficeSetItems : SetItems

{

public OfficeSetItems(List<Furniture> furnitures)

: base(furnitures) { }

public override string SetName { get { return "Офисный"; } }

}

}

**Код класса  *SpecificSetItems***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSetItems

{

public class SpecificSetItems : SetItems

{

public SpecificSetItems(List<Furniture> furnitures)

: base(furnitures) { }

public override string SetName { get { return "Особый"; } }

}

}

**Код класса  *ISet***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSetItems

{

public interface ISet

{

string SetName { get; }

}

}

**Код класса  *FurnitureSetCreator***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSetItems

{

public abstract class FurnitureSetCreator

{

public abstract List<Furniture> Creator(List<Furniture> furnitures);

}

}

**Код класса  *BedroomSetCreator***

using FurnitureDBLibrary.Models.FurnitureSetItems;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSets

{

public class BedroomSetCreator : FurnitureSetCreator

{

public override List<Furniture> Creator(List<Furniture> furnitures)

{

var loungeFurnitures = furnitures.FindAll(f => f.TypeName == "Спальная");

return loungeFurnitures;

}

}

}

**Код класса  *KitchenSetCreator***

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.FurnitureSetItems;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Furnitures

{

public class KitchenSetCreator : FurnitureSetCreator

{

public override List<Furniture> Creator(List<Furniture> furnitures)

{

var kitchenFurnitures = furnitures.FindAll(f => f.TypeName == "Кухонная");

return kitchenFurnitures;

}

}

}

**Код класса  *LoungeSetCreator***

using FurnitureDBLibrary.Models.FurnitureSetItems;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSets

{

public class LoungeSetCreator : FurnitureSetCreator

{

public override List<Furniture> Creator(List<Furniture> furnitures)

{

var loungeFurnitures = furnitures.FindAll(f => f.TypeName == "Гостиная");

return loungeFurnitures;

}

}

}

**Код класса  *OfficeSetCreator***

using FurnitureDBLibrary.Models.FurnitureSetItems;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureSets

{

public class OfficeSetCreator : FurnitureSetCreator

{

public override List<Furniture> Creator(List<Furniture> furnitures)

{

var loungeFurnitures = furnitures.FindAll(f => f.TypeName == "Офисная");

return loungeFurnitures;

}

}

}

**Код класса  *FurnitureType***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public abstract class FurnitureType

{

private string \_typeName;

private decimal \_typeMarkup;

public FurnitureType(string typeName, decimal typeMarkup)

{

\_typeName = typeName;

\_typeMarkup = typeMarkup;

}

public string TypeName { get { return \_typeName; } }

public decimal TypeMarkup { get { return \_typeMarkup; } set { \_typeMarkup = value; } }

public override string ToString() { return TypeName; }

}

}

**Код класса  *IBedroom***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public interface IBedroom

{

string TypeName { get; }

decimal TypeMarkup { get; }

}

}

**Код класса  *IKitchen***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public interface IKitchen

{

string TypeName { get; }

decimal TypeMarkup { get; }

}

}

**Код класса  *ILounge***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public interface ILounge

{

string TypeName { get; }

decimal TypeMarkup { get; }

}

}

**Код класса  *IOffice***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public interface IOffice

{

string TypeName { get; }

decimal TypeMarkup { get; }

}

}

**Код класса  *IType***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public interface IType

{

string TypeName { get; }

decimal TypeMarkup { get; }

}

}

**Код класса  *BedroomType***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public class BedroomType : FurnitureType

{

public BedroomType(string typeName, decimal typeMarkup) : base(typeName,typeMarkup){ }

}

}

**Код класса  *KitchenType***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public class KitchenType : FurnitureType

{

public KitchenType(string typeName, decimal typeMarkup) : base(typeName, typeMarkup) { }

}

}

**Код класса  *LoungeType***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public class LoungeType : FurnitureType

{

public LoungeType(string typeName, decimal typeMarkup) : base(typeName, typeMarkup) { }

}

}

**Код класса  *OfficeType***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public class OfficeType : FurnitureType

{

public OfficeType(string typeName, decimal typeMarkup) : base(typeName, typeMarkup) { }

}

}

**Код класса  *Specific***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.FurnitureTypes

{

public class Specific : FurnitureType

{

public Specific(string typeName, decimal typeMarkup) : base(typeName, typeMarkup) { }

}

}

**Код класса  *Manufacturer***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models

{

public abstract class Manufacturer

{

private string \_manufacturerName;

private decimal \_manufacturerMarkup;

public Manufacturer(string manufacturerName, decimal manufacturerMarkup)

{

\_manufacturerName = manufacturerName;

\_manufacturerMarkup = manufacturerMarkup;

}

public string ManufacturerName { get { return \_manufacturerName; } }

public decimal ManufacturerMarkup { get { return \_manufacturerMarkup; } set { \_manufacturerMarkup = value; } }

public override string ToString()

{

return $"{ManufacturerName}";

}

}

}

**Код класса  *IDelcom***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public interface IDelcom

{

string ManufacturerName { get; }

decimal ManufacturerMarkup { get; }

}

}

**Код класса  *IManufacturer***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public interface IManufacturer

{

string ManufacturerName { get; }

decimal ManufacturerMarkup { get; }

}

}

**Код класса  *IMozirles***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

internal interface IMozirles

{

string ManufacturerName { get; }

decimal ManufacturerMarkup { get; }

}

}

**Код класса  *IRechicadrev***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public interface IRechicadrev

{

string ManufacturerName { get; }

decimal ManufacturerMarkup { get; }

}

}

**Код класса  *Delcom***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public class Delcom : Manufacturer

{

public Delcom(string manufacturerName, decimal manufacturerMarkup) : base(manufacturerName,manufacturerMarkup) { }

}

}

**Код класса  *Mozirles***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public class Mozirles : Manufacturer

{

public Mozirles(string manufacturerName, decimal manufacturerMarkup) : base(manufacturerName, manufacturerMarkup) { }

}

}

**Код класса  *Rechicadrev***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public class Rechicadrev : Manufacturer

{

public Rechicadrev(string manufacturerName, decimal manufacturerMarkup) : base(manufacturerName, manufacturerMarkup) { }

}

}

**Код класса  *SpecificManufacturer***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.Models.Manufacturers

{

public class SpecificManufacturer : Manufacturer

{

public SpecificManufacturer(string manufacturerName, decimal manufacturerMarkup) : base(manufacturerName, manufacturerMarkup) { }

}

}

**Код класса  *DBObjectController***

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public abstract class DBObjectController<T>

{

private readonly NpgsqlConnection \_connection;

protected NpgsqlCommand \_command;

public DBObjectController()

{

\_connection = DBConnection.GetInstance.Connection;

\_command = new NpgsqlCommand();

\_command.Connection = \_connection;

\_command.CommandType = System.Data.CommandType.Text;

}

public abstract List<T> Read();

public abstract void Create(T model);

public abstract void Delete(T model);

public abstract void Update(T model);

}

}

**Код класса  *FurnitureController***

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using FurnitureDBLibrary.Models.FurnitureTypes;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public class FurnitureController : DBObjectController<Furniture>

{

public FurnitureController() : base() { }

public override List<Furniture> Read()

{

ManufacturerController manufacturerController = new ManufacturerController();

List<Manufacturer> manufacturers = manufacturerController.Read();

FurnitureTypeController furnitureTypeController = new FurnitureTypeController();

List<FurnitureType> furnitureTypes = furnitureTypeController.Read();

List<Furniture> furnitures = new List<Furniture>();

string command = "select furniturename,furnitureprice,furniturequantity,furnituretypename,manufacturername,varietyname from furnitures " +

"join furnituretypes on furnitures.furnituretypeid = furnituretypes.furnituretypeid " +

"join manufacturers on manufacturers.manufacturerid = furnitures.manufacturerid " +

"join furniturevarieties on furniturevarieties.varietyid = furnitures.varietyid order by @param;";

NpgsqlParameter idParam = new NpgsqlParameter("@param", "furnitureid");

\_command.CommandText = command;

\_command.Parameters.Add(idParam);

\_command.ExecuteNonQuery();

NpgsqlDataReader reader = \_command.ExecuteReader();

if (reader.HasRows)

{

while(reader.Read())

{

var type = furnitureTypes.Find(t => t.TypeName == reader.GetString(3));

var manufacturer = manufacturers.Find(m => m.ManufacturerName == reader.GetString(4));

switch (reader.GetString(5).ToLower())

{

case "стул":

switch (reader.GetString(3).ToLower())

{

case "кухонная":

furnitures.Add(new KitchenChair(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

case "офисная":

furnitures.Add(new OfficeChair(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

}

break;

case "стол":

switch (reader.GetString(3).ToLower())

{

case "кухонная":

furnitures.Add(new KitchenTable(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

case "офисная":

furnitures.Add(new OfficeTable(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

}

break;

case "шкаф":

switch (reader.GetString(3).ToLower())

{

case "спальная":

furnitures.Add(new BedroomCloset(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

case "офисная":

furnitures.Add(new OfficeCloset(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

}

break;

case "диван":

switch (reader.GetString(3).ToLower())

{

case "гостиная":

furnitures.Add(new LoungeSofa(reader.GetString(0), reader.GetDecimal(1), reader.GetInt32(2)));

break;

}

break;

default:

throw new Exception("Магазин не продает данной мебели");

}

}

}

reader.Close();

return furnitures;

}

public override void Create(Furniture model)

{

short typeId, manufactId;

int varietyId;

string command = $"select GetTypeId('{model.TypeName}')";

\_command.CommandText = command;

typeId = Convert.ToInt16(\_command.ExecuteScalar());

command = $"select GetManufacturerId('{model.ManufacturerName}')";

\_command.CommandText = command;

manufactId = Convert.ToInt16(\_command.ExecuteScalar());

command = $"select GetVarietyId('{model.FurnitureVariety.ToLower()}')";

\_command.CommandText = command;

varietyId = Convert.ToInt32(\_command.ExecuteScalar());

command = $"insert into furnitures(furniturename,furnitureprice,furniturequantity,furnituretypeid,manufacturerid,varietyid) values " +

$"('{model.FurnitureName}',{model.FurniturePrice.ToString().Replace(',','.')},{model.FurnitureQuantity},{typeId},{manufactId},{varietyId});";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Delete(Furniture model)

{

string command = $"delete from furnitures where furniturename = '{model.FurnitureName}';";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Update(Furniture model)

{

short typeId, manufactId;

string command = $"select GetTypeId('{model.TypeName}')";

\_command.CommandText = command;

typeId = Convert.ToInt16(\_command.ExecuteScalar());

command = $"select GetManufacturerId('{model.ManufacturerName}')";

\_command.CommandText = command;

manufactId = Convert.ToInt16(\_command.ExecuteScalar());

command = $"update furnitures set furniturename = '{model.FurnitureName}',furnitureprice={model.FurniturePrice.ToString("0.00",System.Globalization.CultureInfo.InvariantCulture)},furniturequantity={model.FurnitureQuantity},furnituretypeid={typeId},manufacturerid={manufactId} where furnitureName = '{model.FurnitureName}';";

\_command.CommandText= command;

\_command.ExecuteNonQuery();

}

public List<string> GetVarieties(List<Furniture> furnitures)

{

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

foreach (Furniture furniture in furnitures)

{

list.Add(furniture.FurnitureVariety.ToString());

}

return list;

}

}

}

**Код класса  *FurnitureSetController***

using FurnitureDBLibrary.Models;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public class FurnitureSetController : DBObjectController<FurnitureSet>

{

public FurnitureSetController() : base() { }

public override List<FurnitureSet> Read()

{

List<FurnitureSet> furnitureSets = new List<FurnitureSet>();

string command = "select furnituresetname from furnituresets;";

\_command.CommandText = command;

NpgsqlDataReader reader = \_command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

furnitureSets.Add(new FurnitureSet(reader.GetString(0)));

}

}

reader.Close();

return furnitureSets;

}

public override void Create(FurnitureSet model)

{

string command = $"insert into furnituresets values ({model.FurnitureSetName});";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Delete(FurnitureSet model)

{

string command = $"delete from furnituresets where furnituresetid = {model.FurnitureSetName}";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Update(FurnitureSet model)

{

string command = $"update furnituresets set furnituresetname = '{model.FurnitureSetName}' where furnituresetname = {model.FurnitureSetName};";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

}

}

**Код класса  *FurnitureSetItemController***

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using FurnitureDBLibrary.Models.FurnitureSetItems;

using FurnitureDBLibrary.Models.FurnitureSets;

using FurnitureDBLibrary.Models.FurnitureTypes;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public class FurnitureSetItemController : DBObjectController<SetItems>

{

public FurnitureSetItemController() : base() { }

private FurnitureSetCreator creator;

public override List<SetItems> Read()

{

FurnitureController furnitureController = new FurnitureController();

List<Furniture> furnitures = furnitureController.Read();

ManufacturerController manufacturerController = new ManufacturerController();

List<Manufacturer> manufacturers = manufacturerController.Read();

FurnitureTypeController furnitureTypeController = new FurnitureTypeController();

List<FurnitureType> furnitureTypes = furnitureTypeController.Read();

List<SetItems> furnitureSetItems = new List<SetItems>();

string command = $"select furniturename,furnitureprice,furniturequantity,furnituretypename,manufacturername,furnituresetname " +

$"from furnituresetitems " +

$"join furnitures on furnitures.furnitureid = furnituresetitems.furnitureid " +

$"join furnituresets on furnituresets.furnituresetid = furnituresetitems.furnituresetid " +

$"join furnituretypes on furnitures.furnituretypeid = furnituretypes.furnituretypeid " +

$"join manufacturers on manufacturers.manufacturerid = furnitures.manufacturerid " +

$"order by @param;";

NpgsqlParameter idParam = new NpgsqlParameter("@param", "furnituresetid");

\_command.CommandText = command;

\_command.Parameters.Add(idParam);

NpgsqlDataReader reader = \_command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

List<Furniture> furnitureList = new List<Furniture>();

var type = furnitureTypes.Find(t => t.TypeName == reader.GetString(3));

var manufacturer = manufacturers.Find(m => m.ManufacturerName == reader.GetString(4));

switch (reader.GetString(5).ToLower())

{

case "кухонный":

creator = new KitchenSetCreator();

furnitureSetItems.Add(new KitchenSetItems(creator.Creator(furnitures)));

break;

case "спальный":

creator = new BedroomSetCreator();

furnitureSetItems.Add(new BedroomSetItems(creator.Creator(furnitures)));

break;

case "офисный":

creator = new OfficeSetCreator();

furnitureSetItems.Add(new OfficeSetItems(creator.Creator(furnitures)));

break;

case "гостиный":

creator = new LoungeSetCreator();

furnitureSetItems.Add(new LoungeSetItems(creator.Creator(furnitures)));

break;

case "особый":

creator = new SpecificSetCreator();

furnitureSetItems.Add(new SpecificSetItems(creator.Creator(furnitures)));

break;

default:

throw new Exception("Магазин не реализует данный тип гарнитура");

}

}

}

reader.Close();

return furnitureSetItems;

}

public override void Create(SetItems model)

{

int setId, furnitureId;

try

{

string command = $"select GetSetId('{model.SetName}')";

\_command.CommandText = command;

setId = Convert.ToInt32(\_command.ExecuteScalar());

foreach (var item in model.FurnitureList)

{

command = $"select GetFurnitureId('{item.FurnitureName}')";

\_command.CommandText = command;

furnitureId = Convert.ToInt32(\_command.ExecuteScalar());

command = $"insert into furnituresetitems values ({setId},{furnitureId});";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public override void Delete(SetItems model)

{

int setId, furnitureId;

try

{

string command = $"select GetSetId('{model.SetName}')";

\_command.CommandText = command;

setId = Convert.ToInt32(\_command.ExecuteScalar());

foreach (var item in model.FurnitureList)

{

command = $"select GetFurnitureId('{item.FurnitureName}')";

\_command.CommandText = command;

furnitureId = Convert.ToInt32(\_command.ExecuteScalar());

command = $"delete from furnituresetitems where furnituresetitemid = {setId} and where furnitureid = {furnitureId};";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public override void Update(SetItems model)

{

int setId, furnitureId;

try

{

string command = $"select GetSetId('{model.SetName}')";

\_command.CommandText = command;

setId = Convert.ToInt32(\_command.ExecuteScalar());

foreach (var item in model.FurnitureList)

{

command = $"select GetFurnitureId('{item.FurnitureName}')";

\_command.CommandText = command;

furnitureId = Convert.ToInt32(\_command.ExecuteScalar());

command = $"update furnituresetitems set furnituresetid = {setId}, furnitureid = {furnitureId} where furnituresetitemid = {setId} and where furnitureid = {furnitureId};";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

}

}

**Код класса  *FurnitureTypeController***

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.FurnitureTypes;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public class FurnitureTypeController : DBObjectController<FurnitureType>

{

public FurnitureTypeController() : base() { }

public override List<FurnitureType> Read()

{

List<FurnitureType> furnitureTypes = new List<FurnitureType>();

string command = "select furnituretypename,furnituretypemarkup from furnituretypes;";

\_command.CommandText = command;

NpgsqlDataReader reader = \_command.ExecuteReader();

if(reader.HasRows)

{

while(reader.Read())

{

switch (reader.GetString(0).ToLower())

{

case "кухонная":

furnitureTypes.Add(new KitchenType(reader.GetString(0), reader.GetDecimal(1)));

break;

case "офисная":

furnitureTypes.Add(new OfficeType(reader.GetString(0), reader.GetDecimal(1)));

break;

case "спальная":

furnitureTypes.Add(new BedroomType(reader.GetString(0), reader.GetDecimal(1)));

break;

case "гостиная":

furnitureTypes.Add(new LoungeType(reader.GetString(0), reader.GetDecimal(1)));

break;

case "особая":

furnitureTypes.Add(new Specific(reader.GetString(0), reader.GetDecimal(1)));

break;

default:

throw new Exception("Магазин не реализует данный тип мебели");

}

}

}

reader.Close();

return furnitureTypes;

}

public override void Create(FurnitureType model)

{

string command = $"insert into furnituretypes(furnituretypename,furnituretypemarkup) values ('{model.TypeName}',{model.TypeMarkup.ToString().Replace(',','.')});";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Delete(FurnitureType model)

{

string command = $"delete from furnituretypes where furnituretypename = '{model.TypeName}';";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Update(FurnitureType model)

{

string command = $"update furnituretypes set furnituretypemarkup={model.TypeMarkup.ToString().Replace(',','.')} where furnituretypename = '{model.TypeName}';";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

}

}

**Код класса  *ManufacturerController***

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.FurnitureTypes;

using FurnitureDBLibrary.Models.Manufacturers;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public class ManufacturerController : DBObjectController<Manufacturer>

{

public ManufacturerController() : base() { }

public override List<Manufacturer> Read()

{

List<Manufacturer> manufacturers = new List<Manufacturer>();

string command = "select manufacturername,manufacturermarkup from manufacturers;";

\_command.CommandText = command;

NpgsqlDataReader reader = \_command.ExecuteReader();

if(reader.HasRows)

{

while(reader.Read())

{

switch (reader.GetString(0))

{

case "ООО «ДЕЛКОМ40»":

manufacturers.Add(new Delcom(reader.GetString(0), reader.GetDecimal(1)));

break;

case "ОАО «Речицадрев»":

manufacturers.Add(new Rechicadrev(reader.GetString(0), reader.GetDecimal(1)));

break;

case "ЗАО «Мозырьлес»":

manufacturers.Add(new Mozirles(reader.GetString(0), reader.GetDecimal(1)));

break;

case "Особый":

manufacturers.Add(new SpecificManufacturer(reader.GetString(0), reader.GetDecimal(1)));

break;

default:

throw new Exception("Магазин не сотрудничает с данным производителем");

}

}

}

reader.Close();

return manufacturers;

}

public override void Create(Manufacturer model)

{

string command = $"insert into manufacturers(manufacturername,manufacturermarkup) values ('{model.ManufacturerName}',{model.ManufacturerMarkup.ToString().Replace(',','.')});";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Delete(Manufacturer model)

{

string command = $"delete from manufacturers where manufacturername = '{model.ManufacturerName}';";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Update(Manufacturer model)

{

string command = $"update manufacturers set manufacturermarkup = {model.ManufacturerMarkup.ToString().Replace(',', '.')} where manufacturerName = '{model.ManufacturerName}';";

\_command.CommandText= command;

\_command.ExecuteNonQuery();

}

}

}

**Код класса  *SaleController***

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.FurnitureTypes;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml;

namespace FurnitureDBLibrary.DataAccess

{

public class SaleController : DBObjectController<Sale>

{

public SaleController() : base() { }

public override List<Sale> Read()

{

ManufacturerController manufacturerController = new ManufacturerController();

List<Manufacturer> manufacturers = manufacturerController.Read();

FurnitureTypeController furnitureTypeController = new FurnitureTypeController();

List<FurnitureType> furnitureTypes = furnitureTypeController.Read();

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

string command = "select furniturename,furnitureprice,furnituretypename,manufacturername,furnituresaledquantity,saledate " +

"from sales join furnitures on furnitures.furnitureid = sales.furnitureid join furnituretypes on furnitures.furnituretypeid = furnituretypes.furnituretypeid join manufacturers on manufacturers.manufacturerid = furnitures.manufacturerid";

\_command.CommandText = command;

NpgsqlDataReader reader = \_command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var type = furnitureTypes.Find(t => t.TypeName == reader.GetString(2));

var manufacturer = manufacturers.Find(m => m.ManufacturerName == reader.GetString(3));

sales.Add(new Sale(reader.GetString(0), reader.GetDecimal(1), manufacturer.ManufacturerName, type.TypeName, reader.GetInt32(4), reader.GetDateTime(5)));

}

}

reader.Close();

return sales;

}

public override void Create(Sale model)

{

int furnitureId;

string command = $"select GetFurnitureId('{model.FurnitureName}')";

\_command.CommandText = command;

furnitureId = Convert.ToInt32(\_command.ExecuteScalar());

command = $"insert into sales(furnitureid,furnituresaledquantity,saledate) values ({furnitureId},{model.FurnitureSaledQuantity},'{model.SaleDate}');";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Update(Sale model)

{

int furnitureId;

string command = $"select GetFurnitureId('{model.FurnitureName}')";

\_command.CommandText = command;

furnitureId = Convert.ToInt32(\_command.ExecuteScalar());

command = $"update sales set furnitureid = {furnitureId}, furnituresaledquantity = {model.FurnitureSaledQuantity}, saledate = '{model.SaleDate}' where furnitureid = '{furnitureId}' and saledate = '{model.SaleDate}' ;";

\_command.CommandText = command;

\_command.ExecuteNonQuery();

}

public override void Delete(Sale model)

{

throw new Exception("Нельзя удалять отчеты!");

}

public void GenerateXMLReport(List<Sale> sales)

{

XmlDocument xmlDocument = new XmlDocument();

xmlDocument.Load("D:\\КурсоваяРабота\\CourseProgram\\FurnitureDBLibrary\\FurnitureShopReport.xml");

XmlElement xRoot = xmlDocument.DocumentElement;

if (xRoot != null)

xRoot.RemoveAll();

foreach (Sale s in sales)

{

XmlElement sale = xmlDocument.CreateElement("sale");

XmlElement furniture = xmlDocument.CreateElement("furniture");

XmlElement retailPrice = xmlDocument.CreateElement("retailPrice");

XmlElement manufacturer = xmlDocument.CreateElement("manufacturer");

XmlElement type = xmlDocument.CreateElement("type");

XmlElement quantity = xmlDocument.CreateElement("quantity");

XmlElement totalCost = xmlDocument.CreateElement("totalCost");

XmlElement date = xmlDocument.CreateElement("date");

XmlText furnitureText = xmlDocument.CreateTextNode($"{s.FurnitureName}");

XmlText retailPriceText = xmlDocument.CreateTextNode($"{s.FurnitureRetailPrice}");

XmlText manufacturerText = xmlDocument.CreateTextNode($"{s.ManufacturerName}");

XmlText typeText = xmlDocument.CreateTextNode($"{s.TypeName}");

XmlText quantityText = xmlDocument.CreateTextNode($"{s.FurnitureSaledQuantity}");

XmlText totalCostText = xmlDocument.CreateTextNode($"{s.TotalCost}");

XmlText dateText = xmlDocument.CreateTextNode($"{s.SaleDate:d}");

furniture.AppendChild(furnitureText);

retailPrice.AppendChild(retailPriceText);

manufacturer.AppendChild(manufacturerText);

type.AppendChild(typeText);

quantity.AppendChild(quantityText);

totalCost.AppendChild(totalCostText);

date.AppendChild(dateText);

sale.AppendChild(furniture);

sale.AppendChild(retailPrice);

sale.AppendChild(manufacturer);

sale.AppendChild(type);

sale.AppendChild(quantity);

sale.AppendChild(totalCost);

sale.AppendChild(date);

xRoot.AppendChild(sale);

xmlDocument.Save("D:\\КурсоваяРабота\\CourseProgram\\FurnitureDBLibrary\\FurnitureShopReport.xml");

}

}

}

}

**Код класса  *UserController***

using FurnitureDBLibrary.UserModels;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FurnitureDBLibrary.DataAccess

{

public class UserController : DBObjectController<User>

{

public UserController() : base() { }

public override List<User> Read()

{

\_command.Parameters.Clear();

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

string command = "select username,password,rolename from users join roles on roles.roleid = users.roleid order by @param;";

NpgsqlParameter roleParam = new NpgsqlParameter("@param","roleid");

\_command.CommandText = command;

\_command.Parameters.Add(roleParam);

NpgsqlDataReader reader = \_command.ExecuteReader();

if(reader.HasRows)

{

while(reader.Read())

{

switch (reader.GetString(2).ToLower())

{

case "admin":

users.Add(new Admin(reader.GetString(0),reader.GetString(1)));

break;

case "salesman":

users.Add(new Salesman(reader.GetString(0), reader.GetString(1)));

break;

case "manager":

users.Add(new Manager(reader.GetString(0), reader.GetString(1)));

break;

default:

throw new Exception("Роль не существует!");

}

}

}

reader.Close();

return users;

}

public override void Create(User model)

{

throw new Exception("Нельзя создавать новые роли");

}

public override void Update(User model)

{

throw new Exception("Нельзя изменять роли");

}

public override void Delete(User model)

{

throw new Exception("Нельзя удалять роли");

}

}

}

ПРИЛОЖЕНИЕ В

(обязательное)

**ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ**

**Код класса *FurnitureSetTests***

using FurnitureDBLibrary.DataAccess;

using FurnitureDBLibrary.Models.Furnitures;

using FurnitureDBLibrary.Models;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using FurnitureDBLibrary.Models.FurnitureSetItems;

using System.Collections.Generic;

using FurnitureDBLibrary.Models.FurnitureSets;

namespace FurnitureShopUnitTests

{

[TestClass]

public class FurnitureSetTests

{

readonly FurnitureSetItemController furnitureSetItemController = new FurnitureSetItemController();

readonly FurnitureController furnitureController = new FurnitureController();

SetItems newSetItems;

[TestMethod]

public void CreateTests()

{

List<Furniture> furnitures = furnitureController.Read().FindAll(item => item.TypeName == "Офисная");

SetItems newSetItems = new OfficeSetItems(furnitures);

if (furnitureSetItemController.Read().Find(setitem => setitem.SetName == "Офисный") != null)

furnitureSetItemController.Delete(newSetItems);

furnitureSetItemController.Create(newSetItems);

}

[TestMethod]

public void UpdateTest()

{

List<Furniture> furnitures = furnitureController.Read().FindAll(item => item.TypeName == "Офисная");

SetItems newSetItems = new OfficeSetItems(furnitures);

furnitureSetItemController.Create(newSetItems);

furnitureSetItemController.Update(newSetItems);

}

[TestMethod]

public void ReadTest()

{

List<Furniture> furnitures = furnitureController.Read().FindAll(item => item.TypeName == "Офисная");

SetItems newSetItems = new OfficeSetItems(furnitures);

if (furnitureSetItemController.Read().Find(setitem => setitem.SetName == "Офисный") != null)

furnitureSetItemController.Delete(newSetItems);

furnitureSetItemController.Create(newSetItems);

furnitureSetItemController.Read();

}

[TestMethod]

public void DeleteTest()

{

List<Furniture> furnitures = furnitureController.Read().FindAll(item => item.TypeName == "Офисная");

SetItems newSetItems = new OfficeSetItems(furnitures);

if (furnitureSetItemController.Read().Find(setitem => setitem.SetName == "Офисный") == null)

{

furnitureSetItemController.Create(newSetItems);

}

furnitureSetItemController.Delete(newSetItems);

}

}

}

**Код класса *FurnitureTests***

using FurnitureDBLibrary.DataAccess;

using FurnitureDBLibrary.Models;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using FurnitureDBLibrary;

using System;

using Npgsql;

using FurnitureDBLibrary.Models.CurrentFurnitures;

using FurnitureDBLibrary.Models.Furnitures;

using System.Collections.Generic;

namespace FurnitureShopUnitTests

{

[TestClass]

public class FurnitureTests

{

readonly FurnitureController furnitureController = new FurnitureController();

Furniture newFurniture = new KitchenTable("Тестовый стол", 2100, 22);

[TestMethod]

public void FurnitureCreateTests()

{

if (furnitureController.Read().Find(f => f.FurnitureName == "Тестовый стол") != null)

furnitureController.Delete(newFurniture);

furnitureController.Create(newFurniture);

}

[TestMethod]

public void FurnitureUpdateTest()

{

if (furnitureController.Read().Find(f => f.FurnitureName == "Тестовый стол") != null)

furnitureController.Delete(newFurniture);

furnitureController.Create(newFurniture);

int expected = 33;

newFurniture.FurnitureQuantity = 33;

furnitureController.Update(newFurniture);

Assert.AreEqual(expected, furnitureController.Read().Find(type => type.FurnitureName == "Тестовый стол").FurnitureQuantity);

furnitureController.Delete(newFurniture);

}

[TestMethod]

public void FurnitureReadTest()

{

if (furnitureController.Read().Find(f => f.FurnitureName == "Тестовый стол") != null)

furnitureController.Delete(newFurniture);

furnitureController.Create(newFurniture);

int expected = 22;

Furniture furniture = furnitureController.Read().Find(f => f.FurnitureName == "Тестовый стол");

Assert.AreEqual(expected, furniture.FurnitureQuantity);

furnitureController.Delete(newFurniture);

}

[TestMethod]

public void FurnitureDeleteTest()

{

if (furnitureController.Read().Find(f => f.FurnitureName == "Тестовый стол") == null)

furnitureController.Create(newFurniture);

furnitureController.Delete(newFurniture);

}

}

}

**Код класса *FurnitureTypeTests***

using FurnitureDBLibrary.DataAccess;

using FurnitureDBLibrary.Models;

using FurnitureDBLibrary.Models.FurnitureTypes;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Npgsql.Internal;

using System;

namespace FurnitureShopUnitTests

{

[TestClass]

public class FurnitureTypeTests

{

readonly FurnitureTypeController furnitureTypeController = new FurnitureTypeController();

FurnitureType newFurnitureType = new Specific("Особая", (decimal)0.99);

[TestMethod]

public void ReadTest()

{

furnitureTypeController.Create(newFurnitureType);

FurnitureType furnitureType = furnitureTypeController.Read().Find(type => type.TypeName == "Особая");

decimal expectedMarkup = (decimal)0.99;

Assert.AreEqual(expectedMarkup, furnitureType.TypeMarkup);

furnitureTypeController.Delete(newFurnitureType);

}

[TestMethod]

public void Createtest()

{

furnitureTypeController.Create(newFurnitureType);

furnitureTypeController.Delete(newFurnitureType);

}

[TestMethod]

public void Updatetest()

{

furnitureTypeController.Create(newFurnitureType);

decimal expectedMarkup = (decimal)1;

newFurnitureType.TypeMarkup = (decimal)1;

furnitureTypeController.Update(newFurnitureType);

FurnitureType furnitureType = furnitureTypeController.Read().Find(type => type.TypeName == "Особая");

Assert.AreEqual(expectedMarkup, furnitureType.TypeMarkup);

furnitureTypeController.Delete(furnitureType);

}

[TestMethod]

public void Deletetest()

{

furnitureTypeController.Create(newFurnitureType);

furnitureTypeController.Delete(newFurnitureType);

}

}

}

**Код класса *ManufacturerTests***

using FurnitureDBLibrary.DataAccess;

using FurnitureDBLibrary.Models;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using FurnitureDBLibrary.Models.Manufacturers;

using FurnitureDBLibrary.Models.FurnitureTypes;

namespace FurnitureShopUnitTests

{

[TestClass]

public class ManufacturerTests

{

readonly ManufacturerController manufacturerController = new ManufacturerController();

Manufacturer newManufacturer = new SpecificManufacturer("Особый", (decimal)0.20);

[TestMethod]

public void ReadTest()

{

manufacturerController.Create(newManufacturer);

Manufacturer manufacturer = manufacturerController.Read().Find(manufact => manufact.ManufacturerName == "Особый");

decimal expectedMarkup = (decimal)0.20;

Assert.AreEqual(expectedMarkup, manufacturer.ManufacturerMarkup);

manufacturerController.Delete(newManufacturer);

}

[TestMethod]

public void Createtest()

{

manufacturerController.Create(newManufacturer);

manufacturerController.Delete(newManufacturer);

}

[TestMethod]

public void Updatetest()

{

manufacturerController.Create(newManufacturer);

decimal expectedMarkup = (decimal)1;

newManufacturer.ManufacturerMarkup = (decimal)1;

manufacturerController.Update(newManufacturer);

Manufacturer manufacturer = manufacturerController.Read().Find(manufact => manufact.ManufacturerName == "Особый");

Assert.AreEqual(expectedMarkup, manufacturer.ManufacturerMarkup);

manufacturerController.Delete(manufacturer);

}

[TestMethod]

public void Deletetest()

{

manufacturerController.Create(newManufacturer);

manufacturerController.Delete(newManufacturer);

}

}

}

ПРИЛОЖЕНИЕ Г

(обязательное)

**Руководство пользователя**

Руководство пользователя содержит материал необходимый для ознакомления тем, кто пользуется разработанной программой. В руководстве описывается уровень допустимых знаний пользователя, документы, описывающие особенности работы и функционирования системы.

Пользователь системы должен обладать следующей квалификацией:

– пользовательские навыки работы с ЭВМ (электронная вычислительная машина);

– понимание предметной области.

Список документов, предоставляемых к обязательному ознакомлению перед началом работы:

– настоящее руководство пользователя;

– указания по технике безопасности при работе с ЭВМ.

Для работы пользователю необходимо запустить исполняемый файл разработанного приложения.

Программа не требует установки дополнительных программных продуктов. Для работы пользователю необходимо запустить исполняемый файл разработанного приложения.

При запуске программы осуществляется вход в систему и открывается окно авторизации, в котором пользователю необходимо ввести логин и пароль, а затем нажать кнопку *«Enter»*. После этого пользователь перейдет на главное окно, и для дальнейшей работы необходимо нажать клавишу «*Войти*». Внешний вид начального окна программы показан на рисунке Г.1.

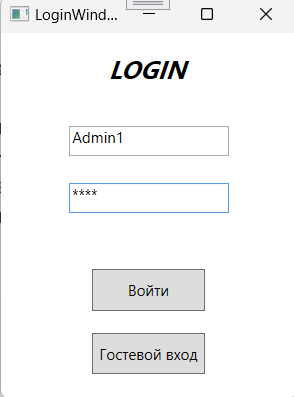


Рисунок Г.1 – Начальное окно программы (окно авторизации)

В программе первостепенным является пользователь с ролью администратора. При входе администратора в систему открывается окно с вкладками главного окна, окна отчетов и окна продаж, в которых администратор имеет возможность просмотра, создания, редактирования и удаления данных, а также формирования отчетов за выбранный период. Внешний вид окна после авторизации приведен на рисунке Г.3.

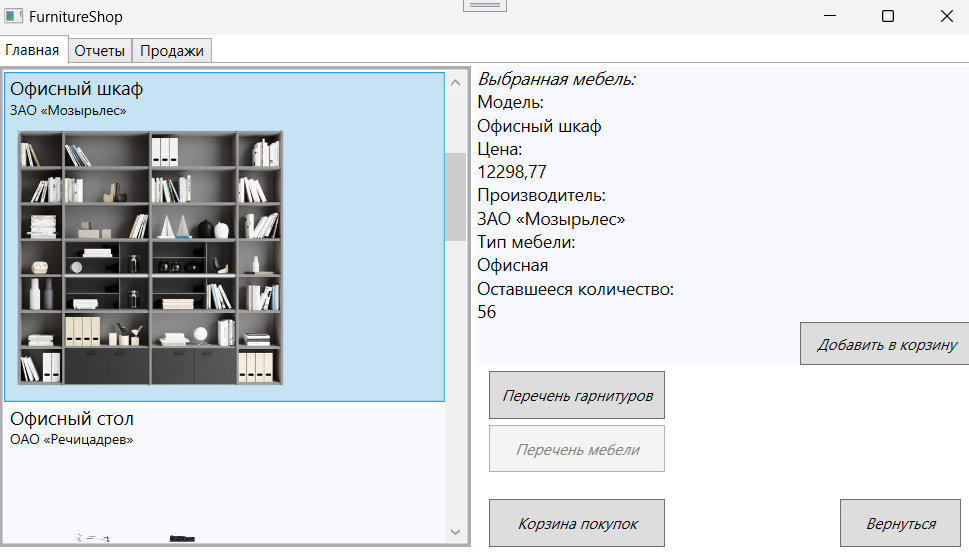


Рисунок Г.3 – Главное окно пользователя

Администратор может вносить новые единицы мебели. Для этого нужно перейти на вкладку *«*Продажи*»* и нажать кнопку *«*Добавить*»*. Для добавления новой модели мебели необходимо учитывать, реализует ли магазин данную модель от заданного производителя и заданной категории. Окно администратора редактирования товаров мебели представлено на рисунке Г.5 соответственно.

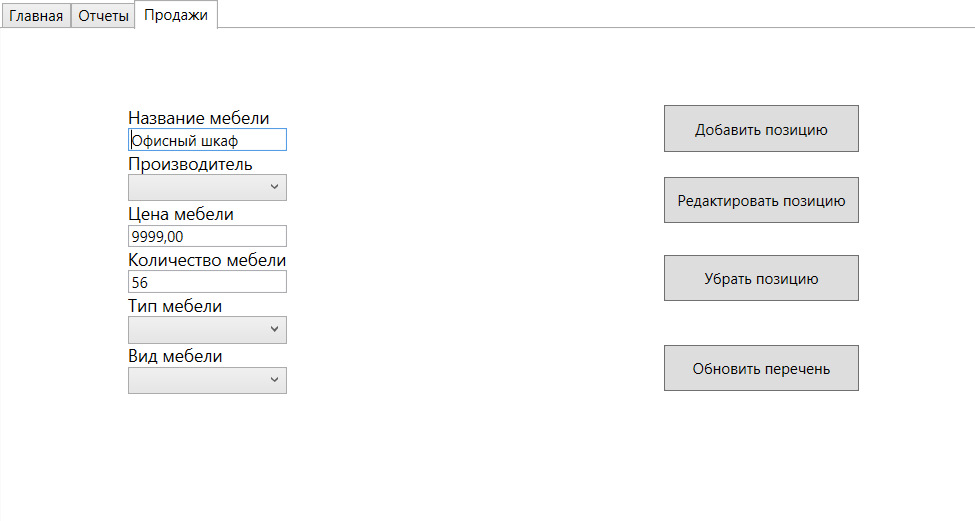


Рисунок Г.5 – Окно администратора, в котором указаны марки автомобилей

Администратору доступна возможность просмотра и формирования отчетов как непосредственно в самом приложении, так и в электронных таблицах *Excel.* На экране появится главное окно вывода информации о совершенных сделках и 3 кнопки «Отчет», «Отчет по производителю» и «Сформировать *XML*». Кнопка «Отчет» выводит на главное окно все когда-либо совершенные продажи, кнопка «Отчет по производителю» выводит все продажи по заданному производителю, кнопка «Сформировать *XML*» формирует отчет в *XML* формате для последующего открытия в *Excel.* Окно отчетов представлено на рисунке Г.6 соответственно.

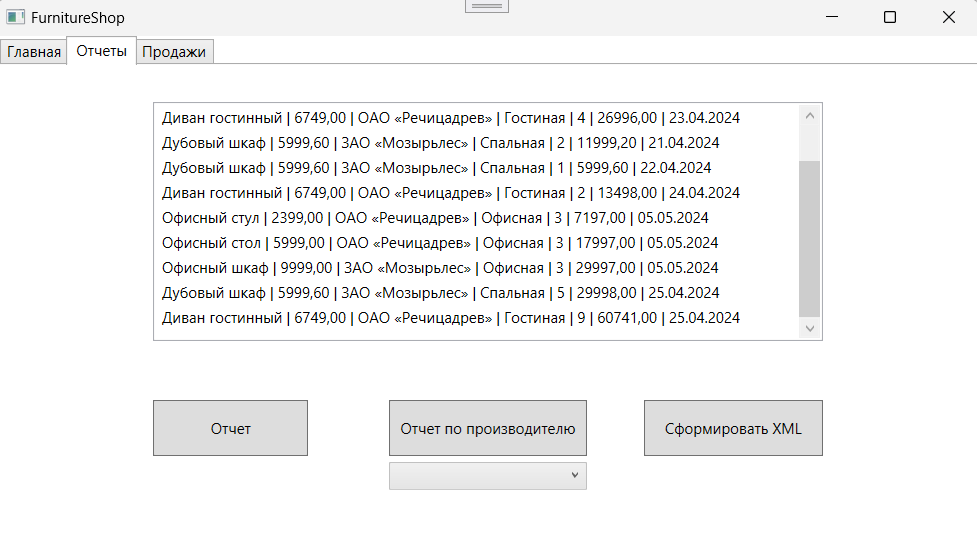


Рисунок Г.6 – Окно отчетов по продажам

ПРИЛОЖЕНИЕ Д

(обязательное)

**РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА**

Программа предназначена для создания информационной системы учета продаж мебели. Приложение предоставляет возможность авторизации в системе, взаимодействия с пользователями, реализуемыми товарами и составления отчетов.

Данное приложение выполняет следующие функции:

− управление данными: добавление, удаление, просмотр, редактирование данных;

− выборка данных для предоставления отчётов;

− получение данных из базы данных.

Для функционирования программно-аппаратного комплекса на стороне клиента необходимо наличие *.exe* файла с приложением.

Для работы программы требуется ЭВМ с установленной операционной системой *Windows*, которая содержитпрограммную платформу .*NET Framework*. Характеристики ЭВМ должны быть достаточными для работы с операционными системами семейства *Windows*: *Windows* 7 / 8 / 8.1 / 10/ 11.

Для функционирования программы не требуется другое специализированное программное обеспечение. Программе не нужна установка, так как она является переносимой и работает в любом каталоге, из которого была запущена.

Для запуска программы нужно запустить файл FurnitureDBLibrary.sln. После запуска исполняемого файла и ожидания в течение нескольких секунд появляется графическое окно программы.

Проверить работоспособность по авторизации пользователя можно с помощью входа от имени пользователя с правами администратора, который изначально по умолчанию имеет логин «*Admin*» и пароль «1234».

В случае непредвиденных ошибок требуется перезапустить программу, если ошибки не исчезли рекомендуется проверить корректность введённых данных. Если ошибки так и не удалось устранить, требуется обратится к разработчику программы либо поддерживающему на данный момент её программисту.

ПРИЛОЖЕНИЕ Е

(обязательное)

**РУКОВОДСТВО ПРОГРАММИСТА**

Разработанное приложение позволяет осуществлять доступ к данным, содержащимся в *LINQ*. При работе с программой пользователю предоставляется доступ к функциям просмотра, добавления, изменения, фильтрации данных. Программа разработана для системы учета продаж мебели.

Для функционирования программно-аппаратного комплекса на стороне клиента необходимо наличие *.exe* файла с приложением.

Никаких ограничений и требований для работы и запуска программы не налагается. Для контроля правильности выполнения программы предусмотрены сообщения пользователю о некорректно введённых данных.

Входными данными для приложения являются данные из базы данных, а также данные, вводимые пользователем в программе, где это необходимо.

К выходным данным можно отнести результаты выполнения различных действий в программе, а также различные сообщения о некорректном вводе.

При регистрации пользователю необходимо ввести логин и пароль. В программе используются механизмы валидации вводимых пользователем данных. При неправильном или некорректном вводе появляются соответствующие сообщения об ошибках.

Если ввести логин несуществующего пользователя или неверный пароль для существующего пользователя, то появится сообщение об ошибке неправильного логина или пароля. Для устранения ошибки нужно ввести логин пользователя, который сохранён в системе.