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 Информационные системы и технологии

(в проектировании и производстве)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине «Объектно-ориентированное программирование»

на тему: «*WPF* приложение для учета продаж для мебельного магазина»

Исполнитель: студент гр. ИТП-22

Шух Н.С.

Руководитель: доцент

Курочка К.С.

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

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

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

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

Подписи членов комиссии

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

Гомель 2022

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc105089364)

[1 Обзор средств для разработки информационной системы 5](#_Toc105089365)

[1.1 Язык программирования *C*# как средство разработки приложения 5](#_Toc105089366)

[1.2 Существующие способы и модели хранения информации 7](#_Toc105089367)

[1.3 Платформа *Windows Presentation Foundation* 10](#_Toc105089368)

[1.4 Принципы *SOLID* 11](#_Toc105089369)

[1.5 Технология *LINQ* 11](#_Toc105089370)

[2 Архитектура приложения для учета продаж мебельного магазина 13](#_Toc105089371)

[2.2 Архитектура спроектированного программного обеспечения 13](#_Toc105089372)

[2.3 Cтруктура базы данных 14](#_Toc105089373)

[2.4 Структура программного комплекса 15](#_Toc105089374)

[3 Этапы проведения верификации и тестирования программного обеспечения 18](#_Toc105089375)

[3.1 Графический интерфейс пользователя 18](#_Toc105089376)

[3.2 Результаты проведения модульного тестирования 25](#_Toc105089377)

[Заключение 27](#_Toc105089378)

[Список использованных источников 28](#_Toc105089379)

[Приложение А Листинг програмы 29](#_Toc105089380)

[Приложение Б Иерархическая схема классов 94](#_Toc105089381)

[Приложение В Руководство системного программиста 95](#_Toc105089382)

[Приложение Г Руководство программиста 96](#_Toc105089383)

[Приложение Д Руководство пользователя 97](#_Toc105089384)

**ВВЕДЕНИЕ**

Информационные технологии активно внедряются во многие сферы деятельности людей, в том числе и в сферу предприятий. Во многих магазинах активно используются приложения, помогающие комфортнее работать.

Технологический процесс позволяет значительно ускорить современное производство и облегчить человеческий труд. Внедрение автоматизированных систем даёт возможность не только заметно сократить время на подготовку и заполнение документации, но и контролировать все проводимые операции в удаленном режиме через интернет.

Информационная система – это система, предназначенная для хранения, обработки, поиска, передачи, а также предоставления информации. В более простом описании, это совокупность программно-технического обеспечения, которое взаимодействует между собой. Автоматизированная система в состоянии осуществлять контроль чего-либо с минимальным участием человека во внутренних процессах.

В настоящее время ни одна компания или предприятие, будь это частная или государственная собственность, не могут обходится без использования баз данных. Они являются удобным средством хранения. А для удобной обработки и предоставления информации, хранящейся в базах данных необходимы автоматизированные системы.

Администраторы получают возможность быстрого поиска и обновления информации о сотрудниках и товарах.

Менеджерам, информационная система позволит исключить ошибки при заполнении отчётов.

Отчеты являются важным атрибутом приложения. С их помощью есть возможность быстрого просмотра результатов по необходимым критериям.

1. **ОБЗОР СРЕДСТВ ДЛЯ РАЗРАБОТКИ ИНФОРМАЦИОННО****Й**

**СИСТЕМЫ**

## 1.1 Язык программирования *C*# как средство разработки приложения

Для разработки приложения для учета использовался объектно-ориентированный язык программирования. Использование такого языка программирования подразумевает возможность построения структуры, состоящей из взаимодействующих объектов. Объектно-ориентированное программирование или ООП (*object-oriented programming*) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного типа, использующая механизм пересылки сообщений и классы, организованные в иерархию наследования.

Язык программирования *C#* будет использоваться для разработки приложения для учета продаж мебельного магазина. Этот язык программирования поддерживает платформу .*NET*. *C#* – это объектно-компонентно-ориентированный язык программирования. *C#* предоставляет языковые конструкции для поддержки этой концепции работы. Это делает *C#* подходящим для создания и использования программных компонентов. С момента своего создания *C#* был обогащен функциями для поддержки новых рабочих нагрузок и современных передовых методов разработки программного обеспечения.

Объектно-ориентированное программирование является в настоящее время основой всей индустрии прикладного программирования благодаря выигрышу в конкурентной борьбе с альтернативными технологиями программирования.

Центральным элементом ООП является абстракция, с помощью которой данные преобразуются в объекты, а последовательность обработки этих данных превращается в набор сообщений, передаваемых между этими объектами. Каждый из объектов имеет свое собственное уникальное поведение. С объектами можно обращаться как с конкретными сущностями, которые реагируют на сообщения, приказывающие им выполнить какие-то действия.

Основная цель абстракции в программировании заключается в отделении интерфейса от реализации.

Объектно-ориентированное программирование подразумевает некоторые принципы, а именно:

– класс – это специальный тип данных, включающий описание данных и описание процедур и функций (методов), которые могут быть выполнены над представителем класса – объектом;

– объект – это конкретный экземпляр класса. В классе задается функциональность (поведение объекта), тем самым все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия, классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования;

– память и поведение, связанное с экземплярами определенного класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве [1].

Необходимо использовать *Language Integrated Query* (*LINQ*) для работы с данными и доступа к ним. *LINQ* реализован на языке программирования *C#* и используется с объектами, реализующими интерфейс *IEnumerable*. *LINQ* обеспечивает доступ к коллекциям, массивам, данным в *MySQL Server*, *XML*-файлам и другим объектам, в которых хранятся данные.

*LINQ to SQL* – это компонент .*NET Framework*, который предоставляет инфраструктуру времени выполнения для управления реляционными данными как объектами. *LIINQ to SQL* сопоставляет модель данных реляционной базы данных с объектной моделью, выраженной на языке программирования. При использовании приложения *LINQ to SQL* переводит интегрированные в язык запросы объектной модели в *SQL* и отправляет их в базу данных. Когда база данных возвращает результаты запроса, *LINQ to SQL* преобразует их в объекты, позволяя работать с ними, используя язык программирования [1].

Язык *C#* с некоторыми особенностями поддерживает основные концепции ООП, такие как абстракция, инкапсуляция, полиморфизм и наследование.

Абстракция в ООП дает объекту характеристики, которые четко определяют его конкретные границы, выделяя этот объект среди всех остальных. Основная идея состоит в том, чтобы отделить дополнительные детали реализации от основных характеристик объектов. Реализация абстракции на языке программирования *C#* возможна за счет использования интерфейсов и абстрактных классов. Абстрактные классы не имеют или имеют только частичную реализацию. Экземпляры таких классов создавать нельзя.

Основные концепции ООП:

– инкапсуляция (пакетирование) механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования. Инкапсуляция позволяет создавать объект, являющийся логическим целым, включающим данные и код для работы с этими данными. Объект обеспечивает защиту против случайной или несанкционированной модификации частных (*private*) составляющих его членов;

– полиморфизм – принцип (подход), обеспечивающий возможность использования одного и того же кода для решения разных задач. Полиморфизм позволяет уменьшить сложность программы посредством использования одного и того же интерфейса для задания целого класса действий. Задача выбора специфического действия (метода) в зависимости от конкретной ситуации (количества и типа передаваемых аргументов) возлагается на компилятор.

– наследование представляет собой процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта. Объект, используя наследование, нуждается только в определении специфичных только для этого объекта свойств, отличающих его от других объектов этого класса.

## 1.2 Существующие способы и модели хранения информации

База данных (БД) – это совокупность данных, хранящихся в соответствии со схемой данных, управляемая в соответствии с правилами инструментов моделирования данных [2].

Модели данных – это своего рода абстракция, которая применяется к конкретным данным, после чего пользователи и разработчики могут использовать их как готовую информацию. База данных включает элементы данных и отношения между ними. Чтобы создать базу данных из базы данных, сначала нужно завершить процесс моделирования, то есть вам нужно создать диаграмму, которая будет отображать отношения между элементами данных. Такая схема называется моделью данных.

Базы данных классифицируются по модели данных. Базовые модели данных:

– иерархическая;

– объектная или объектно-ориентированная;

– объектно-реляционная;

– реляционная;

– сетевая;

– функциональная.

Реляционная база данных – это набор данных с предопределенными отношениями между ними. Данные организованы с помощью наборов таблиц. Таблицы состоят из столбцов и строк. В таблицах хранится вся информация об объектах, представленных в базе данных. Каждый столбец в таблице хранит определенный тип данных, а каждая ячейка хранит значение атрибута. Каждая строка в таблице представляет собой набор связанных значений, которые относятся к одной сущности или одному объекту. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, и не повторяться в других строках. Строки из нескольких таблиц можно связать с помощью внешних ключей. Доступ к этим данным можно получить разными способами.

Между таблицами в базе данных устанавливаются три типа отношений: «один ко многим», «один к одному» и «многие ко многим».

Тип отношения «один ко многим» является наиболее часто используемым. В этом случае одна или несколько строк в одной таблице относятся к одной из строк в другой таблице.

Установление связи между таблицами осуществляется с помощью ключей. Внешний ключ добавляется в дочернюю таблицу в виде столбца. Внешний ключ в дочерней таблице – это значение уникального идентификатора (первичного ключа) родительской таблицы.

Связь «один к одному» между таблицами означает, что каждая строка в одной таблице соответствует одной строке в другой таблице, и наоборот. Индивидуальные таблицы можно объединить в одну. Две таблицы вместо одной используются по соображениям конфиденциальности, для удобства (если в одной таблице слишком много столбцов), чтобы сэкономить место на диске.

Отношение «многие ко многим» – это отношение, в котором несколько записей из одной таблицы могут соответствовать нескольким записям из другой. Это соединение можно использовать для создания связей между учениками и учителями. В университетах каждый преподаватель обучает множество студентов, и каждого студента могут обучать несколько преподавателей.

В реляционные базы данных легко вносить изменения. Если изменить данные об объекте в одной таблице, то соответствующая информация автоматически перейдет в другие таблицы, связанные с измененной.

У реляционных баз данных есть ряд преимуществ:

– простая схема представления данных в виде таблиц;

– простые инструменты для поддержки реляционной модели;

– оптимизация доступа к базе данных;

– защита данных.

Недостатком реляционной модели является жесткость структуры данных. Например, невозможно указать строку таблицы произвольной длины.

Несмотря на недостатки реляционной модели, ее использование упрощает работу с данными, в отличие от иерархической модели.

В иерархических моделях основная структура представления данных имеет вид дерева. На самом высоком уровне иерархии есть только одна вершина, которая называется корнем дерева. Эта вершина имеет связи с вершинами второго уровня, вершины второго уровня имеют связи с вершинами третьего уровня и так далее. Между вершинами одного уровня нет связей. Следовательно, данные в иерархической структуре неравны и жестко подчинены другим. Доступ к информации возможен только по вертикальной схеме, начиная с корня, поскольку между вершинами одного уровня нет связей[3].

Иерархическая модель данных – одна из самых первых и простых. Для иерархически организованных данных определены операции для добавления новых записей, изменения значений данных ранее извлеченной записи, удаления некоторых записей и извлечения с возможностью установки условия выбора.

Сетевая модель данных была разработана после иерархической модели. В иерархической модели вершины соединяются только на разных уровнях, в то время как в сетевой модели любые элементы могут быть соединены друг с другом. Сеть образована записями, которые могут иметь ссылки на другие записи или наборы записей. В сетевых моделях данных могут быть произвольные ссылки.

Сетевая модель данных позволяет использовать больше операций с данными, чем иерархическая модель. Основные из них:

– добавление записей;

– получение записей;

– обновление записей;

– удаление записей;

– включение записей в групповое отношение;

– исключение записей из групповых отношений;

– переключение записей из одной групповой связи в другую.

Основное преимущество сетевой модели данных – эффективное использование памяти. Но из-за множества подключений схема базы данных очень сложная и жесткая. Множественные и произвольные отношения ослабляют контроль целостности.

Иерархическая модель данных имеет сложные логические отношения, что вызывает сложную обработку данных. Сетевая модель эффективно использует память, но имеет очень сложные схемы базы данных. По этой причине предпочтительно использовать реляционную модель данных [4].

Система управления базами данных (СУБД) – набор программных и лингвистических средств общего или специального назначения, обеспечивающий управление созданием и использованием баз данных. СУБД – это набор программ, которые позволяют создавать базу данных и манипулировать данными. Основные операции с данными – это вставка, обновление, удаление и выборка данных. Система обеспечивает безопасность, надежность хранения и целостность данных и предоставляет инструменты для администрирования баз данных.

По сути, СУБД – это способ манипулировать списками информации. В этом случае информация может поступать из разных источников.

Существует множество СУБД, каждая из которых имеет свои преимущества и недостатки. Одна из самых популярных СУБД – *MySQL*. Возможности этой СУБД позволяют выбирать движки для системы хранения информации, которые, в свою очередь, могут изменять функциональность инструментов и обрабатывать данные, хранящиеся в различных типах таблиц. *MySQL* очень гибкая, поддерживая большое количество типов таблиц. Эта СУБД бесплатна, хорошо документирована, поддерживает набор пользовательских интерфейсов и может работать с другими базами данных. *MySQL* квалифицируется как система управления реляционными базами данных.

Для работы с реляционными таблицами используется язык *SQL* (*Structured Query Language*). *SQL* – это язык программирования структурированных запросов, который используется для хранения данных, поиска, обновления, извлечения или удаления из базы данных.

*SQL* тесно связан с реляционной моделью, потому что результатом *SQL*-запроса является таблица, называемая набором результатов. Таким образом, в реляционной базе данных можно создать новую постоянную таблицу, просто сохранив набор результатов запроса. Точно так же запрос может использовать как постоянные таблицы, так и наборы результатов других запросов в качестве входных данных[5].

Большинство языков программирования имеют возможность описывать переменные и структуры данных, использовать условную логику, циклические конструкции и разделять код на небольшие повторно используемые части (объекты, функции, процедуры). Код передается компилятору, и полученный исполняемый код делает именно то, что запрограммировано. При работе с любым процедурным языком программирования программист полностью контролирует действия программы. В отличие от многих языков программирования, *SQL* не является процедурным языком. С *SQL* нужно отказаться от обычного контроля над выполнением, потому что выражения *SQL* определяют требуемые входные и выходные данные, а то, как выражение выполняется, зависит от компонента механизма СУБД, называемого оптимизатором. Задача оптимизатора – просмотреть инструкцию *SQL* и на основе конфигурации таблицы и доступных индексов выбрать наиболее эффективный способ выполнения запроса. Большинство СУБД позволяют программисту влиять на решения оптимизатора с помощью подсказок, таких как предложения по использованию определенного индекса. Но использование подсказок требует хорошего понимания работы СУБД.

Писать полные приложения при помощи *SQL* нельзя. Если нужно создать что-то более сложное, чем простой скрипт для работы с конкретными данными, необходимо интегрировать *SQL* с процедурным языком программирования.

**1.3 Платформа *Windows Presentation Foundation***

Причиной выбора именно языка *C#* является то, что на нем гораздо удобнее разрабатывать доступные приложения. Он имеет удобную технологию *WPF*, в отличие от *Java*. Консольные приложения, приложение *NT Service*, приложения *Windows Forms* и приложения *Windows Presentation Foundation* (*WPF*) являются саморазмещающимися (*self-hosted*) и снабжены управляемыми *EXE*-файлами. Инициализировав процесс при помощи такого файла, *Windows* загружает оболочку совместимости, которая исследует информацию в заголовке *CLR*, содержащуюся в сборке приложения (*EXE*-файле). Эта информация указывает версию *CLR*, которая использовалась при сборке и тестировании приложения. Именно с ее помощью оболочка совместимости определяет, какую версию *CLR* следует загрузить в процесс. Загрузив и инициализировав среду, оболочка совместимости снова исследует заголовок ее сборки, чтобы определить, какой метод является точкой входа приложения (*Main*). *CLR* вызывает этот метод, и приложение начинает работу.

Вместо реализации пользовательского интерфейса приложений в виде веб-страниц можно использовать более мощную и высокопроизводительную функциональность, предоставляемую технологиями *Windows Store*, *WPF* (*Windows* *Presentation* *Foundation*) и *Windows Forms.* Такие приложения *Windows* с расширенным графическим интерфейсом могут использовать события элементов управления, меню, сенсорного экрана, мыши, пера и клавиатуры, а также могут обмениваться информацией с операционной системой, выдавать запросы к базам данных и пользоваться веб-службами.

*Windows Presentation Foundation* (*WPF*) заменяет разрозненные презентационные технологии *Microsoft* унифицированной современной платформой для построения приложений [6].

**1.4 Принципы *SOLID***

Принципы *SOLID* – то, что необходимо знать и использовать каждому уважающему себя программисту для написания чистого, понятного, масштабируемого и оптимизированного кода.

*SOLID* – аббревиатура пяти основных принципов проектирования в объектно-ориентированном программировании.

Эти принципы позволяют строить на базе ООП масштабируемые и сопровождаемые программные продукты.

Принцип единственной обязанности / ответственности (*single responsibility principle*) обозначает, что каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности.

Принцип открытости / закрытости (*open-closed principle*) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.

Принцип подстановки Барбары Лисков (*Liskov substitution principle*) декларирует, что функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Принцип разделения интерфейса (*interface segregation principle*) говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты «маленьких» интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Таким образом, в результате использования всех вышеперечисленных принципов, возможно написать приложение, которое будет отвечать требованиям к масштабируемости, скорости работы, простоты, эстетической привлекательности кода.

**1.5 Технология *LINQ***

Технология *LINQ* (*Laпguage lпtegrated Query*, запрос, интегрированный в язык) позволяет писать безопасные в смысле типизации структурированные запросы к локальным коллекциям объектов и удаленным источникам данных. Это новая функциональная возможность языка *С*# 3.0 и платформы *Framework*. *LINQ* позволяет строить запросы к любой коллекции, реализующей интерфейс *IEnumerable*, будь то массив, список, коллекция *XML* *DOM* или удаленный источник данных, такой как таблицы на *SQL*-cepвepe. Технология *LINQ* предлагает сочетание достоинств проверки типов на этапе компиляции и динамического составления запросов

Базовыми единицами данных в *LINQ* являются последовательности и элементы. Последовательность - это любой объект, который реализует обобщенный интерфейс *IEnumerable*, а элемент - это просто элемент последовательности.

Существует несколько разновидностей *LINQ*:

* *LINQ to Objects*;
* *LINQ to Entities*;
* *LINQ to XML*;
* *LINQ to DataSet* и *SQL*;
* *Parallel LINQ*.

*LINQ to Objects* – название, данное *API*-интерфейсу *IEnumerable<T>* для стандартных операций запросов (*Standard Query Operators*). Именно *LINQ* *to Objects* позволяет выполнять запросы к массивам и находящимся в памяти коллекциям данных.

*LINQ to XML* – название, назначенное *API*-интерфейсу *LINQ*, который ориентирован на работу с *XML*. В *Microsoft* не только добавили необходимые библиотеки *XML* для работы с *LINQ*, но также восполнили недостатки стандартной модели *XML DOM*, существенно облегчив работу с *XML*. Прошли те времена, когда приходилось создавать *XmlDocument* только для работы с небольшим фрагментом *XML*-кода.

*LINQ to DataSet* – название, данное *API*-интерфейсу *LINQ*, который предназначен для работы с *DataSet*. У многих разработчиков есть масса кода, полагающегося на *DataSet*. Те, кто не хочет отставать от новых веяний, но и не готовы переписывать свой код, благодаря этому интерфейсу могут воспользоваться всей мощью *LINQ*.

*LINQ to SQL* – наименование, присвоенное *API*-интерфейсу *IQuerya-ble<T>*, который позволяет запросам *LINQ* работать с базой данных *Microsoft SQL Server*.

*LINQ to Entities* – альтернативный *API*-интерфейс *LINQ*, используемый для обращения к базе данных. Он отделяет сущностную объектную модель от физической базы данных, вводя логическое отображение между ними двумя. С таким отделением возрастает мощь и гибкость, но также растет и сложность. Если нужна более высокая гибкость, чем обеспечивается *LINQ to SQL*, имеет смысл рассмотреть эту альтернативу.

1. **АРХИТЕКТУРА ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА ПРОДАЖ**

**МЕБЕЛЬНОГО МАГАЗИНА**

**2.1 Постановка задачи**

Необходимо разработать приложение для учёта продаж мебельного магазина. О каждом товаре известно: наименование товара, категория, цена. О каждом сотруднике известно: ФИО, занимаемая должность.

Приложение должно соответствовать *SOLID* (*Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion*).

К приложению должен быть обеспечен многопользовательский доступ: администратор, менеджер, продавец.

Администратор может управлять пользователями: добавлять/удалять пользователей, сбрасывать их пароли. Менеджер может просматривать отчеты и видить информацию о товарах. продавец может вносить данные о продажах.

В качестве источника данных использовать СУБД *MySQL*. Для реализации использовать средства языка программирования *C*# *Windows Presentation Foundation*. Для доступа к данным использовать *LINQ (Language Integrated Query)*.

## 2.2 Архитектура спроектированного программного обеспечения

При разработке приложения были выделены три библиотеки классов, каждая из которых должна выполнять свою область задач:

– библиотека с интерфейсом приложения;

– библиотека с основными классами приложения;

– библиотека с классами, работающими с *MySQL*.

Первым ключевыми классом в приложении являются класс *User* – хранит информацию о пользователе. Для класса *User*, в приложении имеется многопользовательский доступ: администратор, продавец, менеджер. Администратор имеет полный доступ к приложению. Продавец может добавлять данные о продажах. Менеджер может просматривать товары и отчеты.

Архитектура программного обеспечения представляет собой совокупность решений об организации программной системы, которые должны соответствовать бизнес требованиям. Производительность системы должна соответствовать атрибутам качества.

Основными атрибутами являются:

– адаптивность. ПО имеет длинный процесс разработки, но предугадать все компоненты программы невозможно. Приложение должно быть разработано таким образом, чтобы можно было легко расширить его функционал и приспособить к определенным условиям;

– совместимость. ПО должно использоваться на различных ПК без его модификации, при этом не теряя своей функциональности;

– надежность. ПО должно быть безопасным в использовании, легким в техобслуживании и техподдержке;

– эффективность. ПО должно обеспечивать пользователю получение желаемого результата;

– гибкость. ПО должно уметь адаптироваться к внутренним и внешним изменениям, то есть легко расширять свой функционал.

## 2.3 Cтруктура базы данных

В качестве источника данных приложения выбрана база данных в СУБД *MySQL*. В связи с тем, что данная СУБД имеет ряд преимуществ по сравнению с аналогами:

* высокая скорость работы;
* хранение данных в одном файле;
* минимализм;
* кроссплатформенность.

Для реализации структуры будущей базы данных необходимо руководствоваться всеми правилами (условиями) нормализации для исключения потенциальной противоречивости хранимых данных. Так, для удобства последующего извлечения этих данных, таблицы объединены связями и ограничениями целостности, а большинство связей представлены связью «один ко многим».

Диаграмма базы данных, состоящей из таблиц их полей и связей между этими таблицами, представлена на рисунке 2.1.

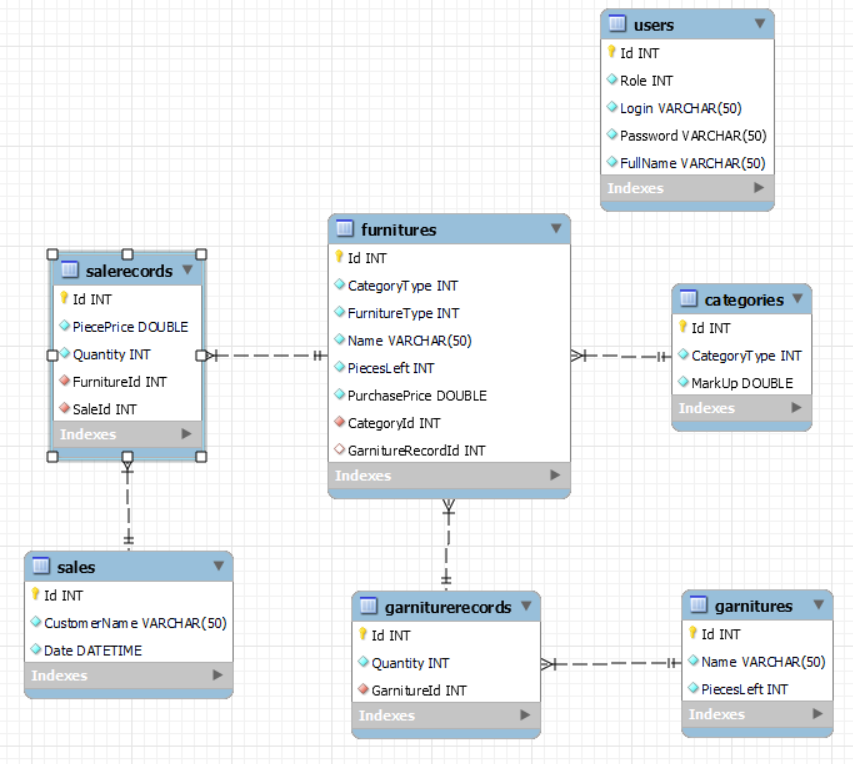


Рисунок 2.1 – Диаграмма базы данных

*Furnitures* – таблица, которая содержит в себе информацию о фарнитуре. Состоит из следующих столбцов: *Id* – идентификатор, *CategoryType* – тип категории, *Name* – название мебели, *PiecesLeft* – остаток мебели, *PurchasePrice* – цена, *CategoryId* – номер категории.

*Categories* – таблица, которая содержит в себе информацию о категориях. Состоит из следующих столбцов: *Id* – идентификатор, *CategoryType* – тип категории, *MarkUp* – наценка.

*Users* – таблица, которая содержит информацию о всех зарегистрированных пользователях приложением. Состоит из следующих столбцов: *Id* – идентификатор, *Login* – логин пользователя, *Password* – пароль пользователя, *FullName* – имя пользователя, *Role* –роли пользователя: 0 – администратор, 1 – менеджер, 2 – продавец.

*Garniturerecords* – таблица, которая содержит информацию о записи мебели. Состоит из следующих столбцов: *Id* – идентификатор, *Quantity* – количество мебели, *GarnitureId* – идентификатор гарнитуры.

*Garnitures* – таблица, которая содержит информацию о записи мебели. Состоит из следующих столбцов: *Id* – идентификатор, *Name* – название гарнитуры, *PiecesLeft* – остаток гарнитуры.

*Salerecords* – таблица, которая содержит в себе информацию о фурнитуре. Состоит из следующих столбцов: *Id* – идентификатор, *PiecePrice* – цена за штуку, *Quantity* – количество продажах, FurnitureId – идентификатор фурнитуры, *SaleId* – идентификатор распродаж.

*Sales* – таблица, которая содержит информацию о записи мебели. Состоит из следующих столбцов: *Id* – идентификатор, *СustomerName* – имя клиента, *Date* – дата.

Итак, возможно сделать вывод о том, что данная структура базы данных выполняет все основные требования к правилам нормализации, т.к. для хранения одинаковых записей созданы отдельные таблицы, которые, в свою очередь, и позволяют заменить связи «многие ко многим» связями «один ко многим».

## 2.4 Структура программного комплекса

После определения архитектуры приложения и структуры базы данных следует также отметить и основной функционал для разных уровней доступа пользователя соответственно, который позже необходимо учитывать в процессе разработки графической части приложения. Данный функционал подробно описан на диаграмме вариантов использования.

Диаграмма вариантов использования – это диаграмма, описывающая взаимоотношения и зависимости между группами вариантов использования и лицами, которые могут использовать приложение (пользователи). Диаграмма вариантов использования системы учета продажи билетов в цирке представлена на рисунке 2.2.

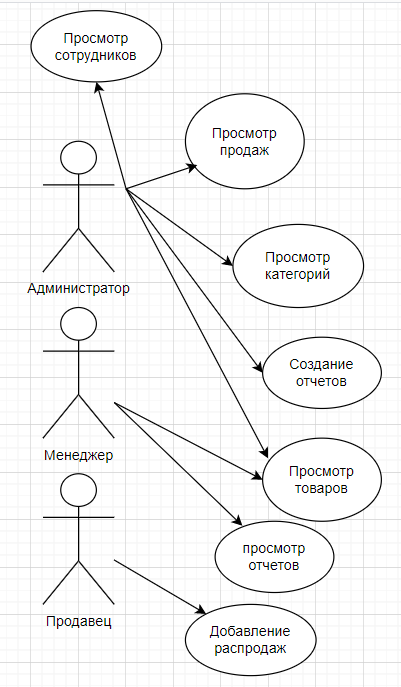


Рисунок 2.2 – Диаграмма прецедентов

К системе приложения имеют доступ три категории пользователей: администратор, продавец и менеджер. Таким образом, менеджер, при авторизации в систему, имеет возможность:

* просматривать информацию о товарах;
* просматривать отчеты.

Продавец имеет право только добавлять распродажи.

Администратор обладает полными правами доступа в систему и имеет возможность:

* управлять всеми данными приложения: удалять, добавлять и редактировать записи;
* формировать отчеты;
* просматривать существующие записи.

Описание классов:

Класс *User* – описывает сущьность пользователя. От него наследуются классы *Administrator*, *Manager*, *Seller*.

Класс *Seller* – описывает сущность продавца.

Класс *UserFactory* – Представляет метод создания выбранного пользователя.

Класс *IDAO* реализует доступ к данным (механизм получения или отправки данных);

Класс *GarnitureBuilder* – предоставляет методы для создания гарнитуры.

Класс *Manager* – описывает сущность менеджера.

Класс *Administrator* – описывает сущность администратора.

Класс *ReportService* – создаёт отчёты. Отчёты может создавать только администратор.

Класс *FurnitureCreator* – представляет базавый класс для создателей мебели.

Класс *GarnitureBuilder* – предоставляет методы для создания гарнитуры.

Класс *Category* – описывает категории мебели.

Класс *Furniture* – описывает типы фарнитуры.

Класс *GarnitureRecord* – представляет мебельную запись.

Класс *FurnitureFactory* – представляет базавый класс для создателей мебели.

Класс *IProduct* – представляет продукты.

Класс *ConnectionSingleton* – обеспечивает подключение к базе данных *MySql*.

Класс *MySqlDB* предоставляет доступ к базе данных *MySql.*

Листинг классов находится в приложении Б. Таким образом, из предметной области выделены сущности и по ним создана иерархия классов итаблицы *MySQL*, доступ к данным осуществлён с помощью *LINQ*. Для использования данного приложения конечными пользователями спроектирован и разработан графический интерфейс с помощью технологии *Windows Presentation Foundation.*

# 3 ЭТАПЫ ПРОВЕДЕНИЯ ВЕРИФИКАЦИИ И ТЕСТИРОВАНИЯ

**ПРОГРАММНОГО ОБЕСПЕЧАЕНИЯ**

## 3.1 Графический интерфейс пользователя

Для создания интерфейса использована технология *Windows Presentation Foundation* (*WPF*). Интерфейс позволяет пользователю взаимодействовать с программной.

В реализованной программе используются следующие элементы пользовательского интерфейса: окна, отображающие различную информацию, меню, где реализуется выбор всех заданных команд, кнопки, выполняющие заданые функции, таблицы, отображающие информацию из базы данных и поля ввода, необходимые для ввода необходимой информации пользователями.

При создании окон приложения для всех окон было отключена возможность масштабирования окна. Для диалоговых окон помимо возможности масштабирования была отключена возможность сворачивания.

Как ранее говорилось, приложение имеет многопользовательский доступ, где пользователи имеют строго заданные роли и, соответственно, заданные полномочия.

При запуске приложения в первую очередь открывается общее для всех типов пользователей окно, представленное на рисунке 3.1 изображен шаблон окна авторизации.

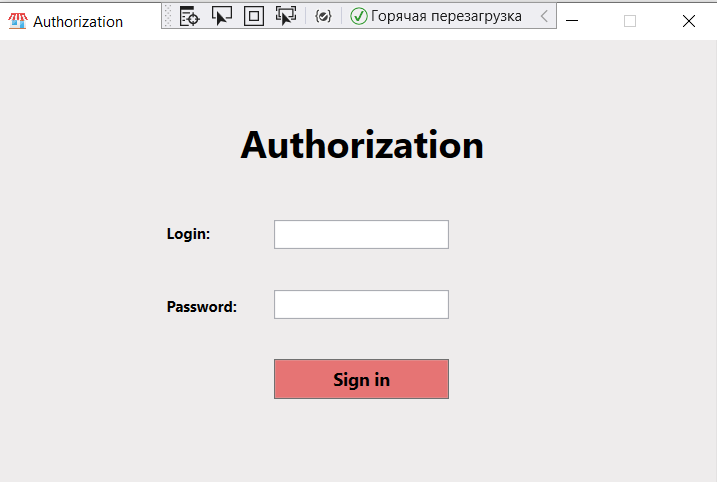


Рисунок 3.1 – Шаблон главного окна

В случае, если пользователь введёт несуществующий логин или пароль, то на экран будет выведено сообщения о некорректности введённых данных. Пример ошибочного ввода приведён на рисунке 3.2.

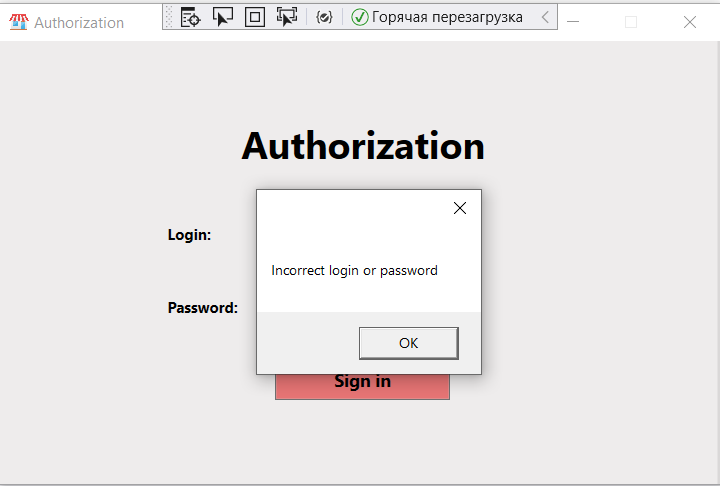


Рисунок 3.2 – Пример некорректного ввода данных

После авторизации открывается главное окно, где перечислены основные возможности для данного пользователя. В зависимости от типа аккаунта будет отличаться главное окно, некоторые функции будут недоступны для продавца и менеджера. Шаблон интерфейса главного окно относительно каждого пользователя изображён на рисунках 3.3 – 3.5.

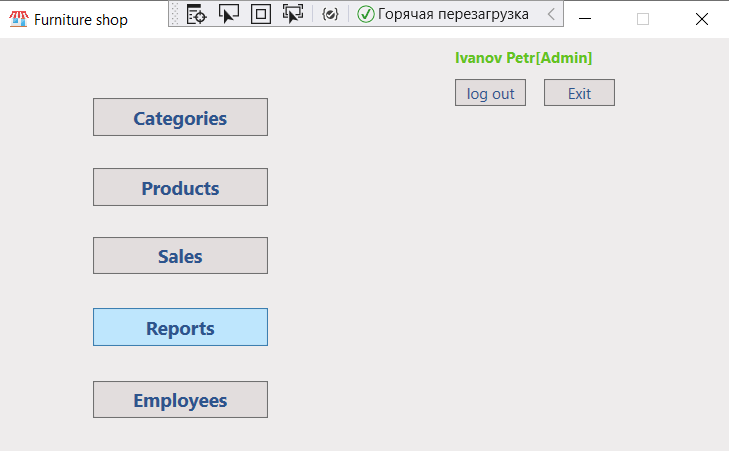


Рисунок 3.3 – Шаблон интерфейса главного меню для администратора

В окне для администратора доступны следующие кнопки управления:

– категории;

– товары;

– продажи;

– отчеты;

– сотрудники.

Окна администратора, менеджера и продавца значительно отличаются. Так как приложение является относительно закрытым и предназначено для частных организаций, добавлять, удалять и изменять функции программы имеет право только администратор в соответствующем окне.

Для менеджера в главном окне доступен просмотр категорий, просмотр товаров, продажи и кнопка отчеты. Шаблон интерфейса главного меню для менеджера представлен на рисунке 3.4.

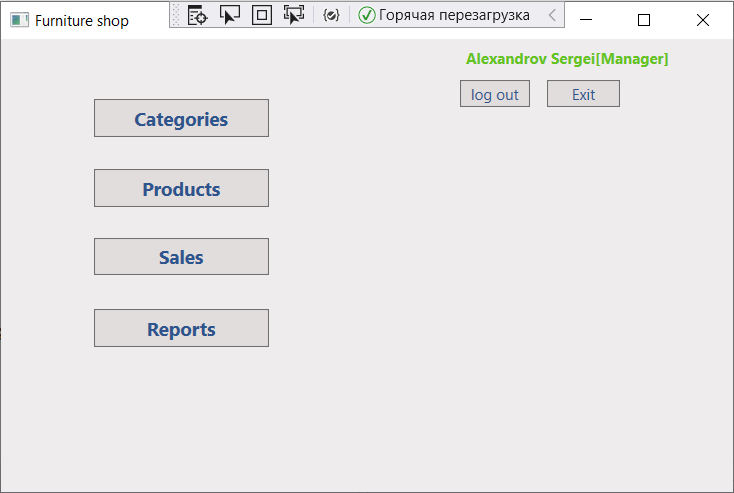


Рисунок 3.4 – Шаблон интерфейса главного меню для менеджера

Окно продавца состоит из списка категорий, товары и кнопки добавить распродажу. Шаблон интерфейса главного меню для продавца представлен на рисунке 3.5.

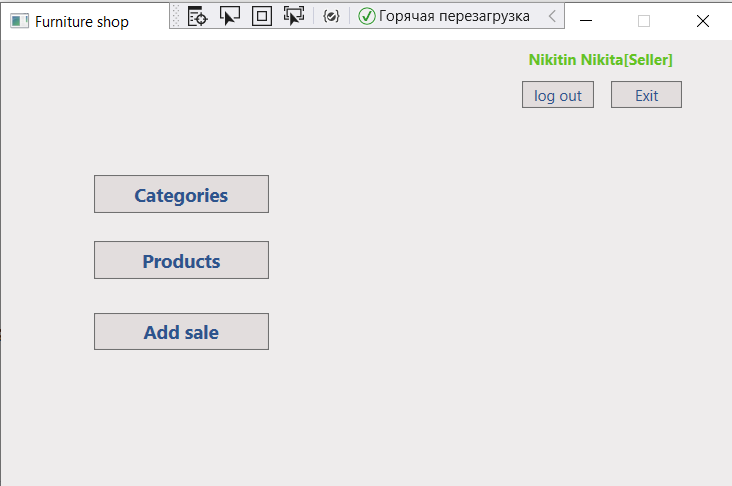


Рисунок 3.5 – Шаблон интерфейса главного меню для продавца

При нажатии на кнопку «категории» открывается окно, в котором администратор может просмотреть все ранее занесенные категории в базу данных. Окно с категориями приведено на рисунке 3.6.

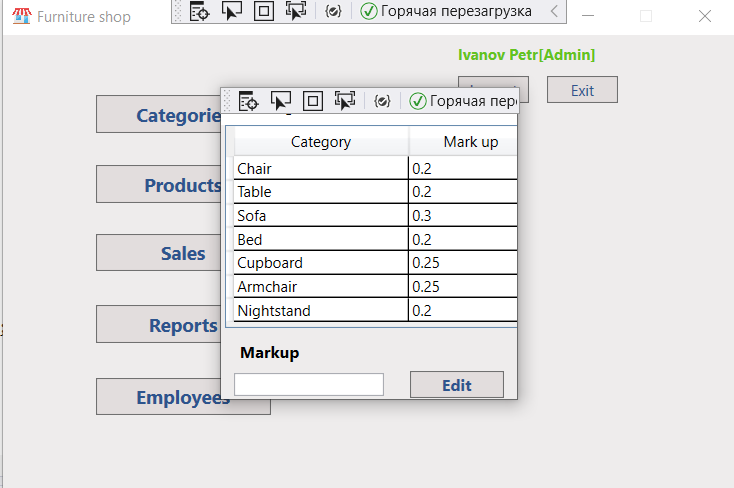


Рисунок 3.6 – Окно «категории» для администратора

При закрытии окна «категории» администратор возвращается в окно с главным меню. Помимо окна «категории», у администратора есть кнопка «товары». При нажатии на кнопку «товары» будет открыто окно, в данном окне находятся товары, также есть возможность добавит гарнитуру, добавить мебель, удалить запись и обновить. Окно с товарами приведено на рисунке 3.7.

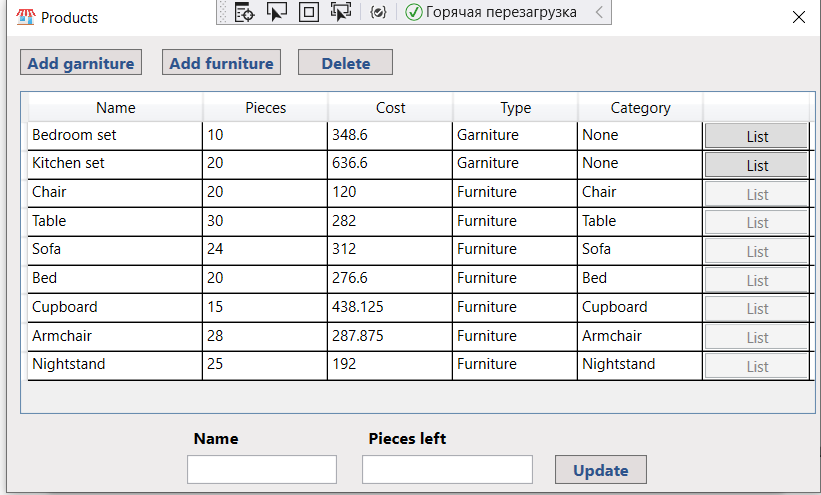


Рисунок 3.7 – Окно товары для администратора

При закрытии окна «товары» администратор возвращается в окно с главным меню. Помимо окна «товары», у администратора есть кнопка «отчеты». При нажатии на кнопку «отчеты» будет открыто окно, в данном окне находятся отчеты, также есть возможность отсортировать отчет по двум критериям продажа по категориям и прибыль по категориям после выбора необходимых данных при нажатии кнопки «сохранить отчет» нужно выбрать место куда сохранить. Окно с отчетом приведено на рисунке 3.8.

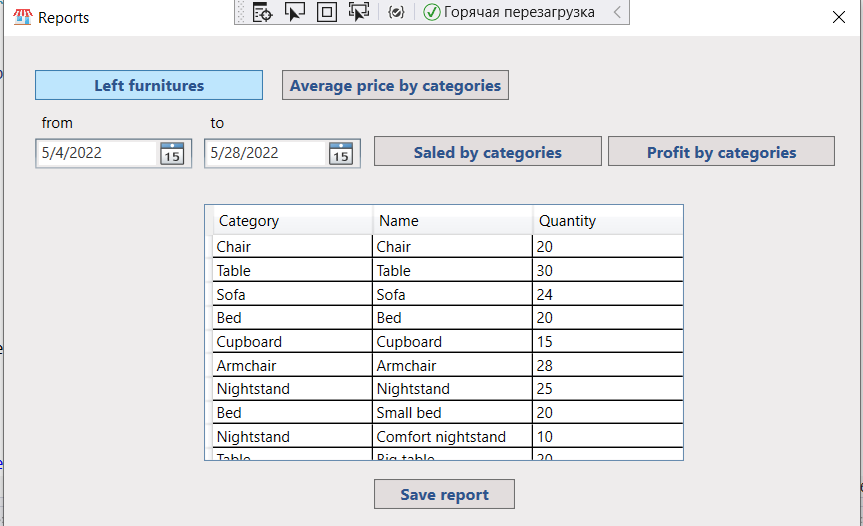


Рисунок 3.8 – Окно отчет для администратора

При закрытии окна «отчеты» администратор возвращается в окно с главным меню. Помимо окна «отчеты», у администратора есть кнопка «продажи». При нажатии на кнопку «продажи» будет открыто окно, в данном окне находится информация о продажах, также есть возможность добавления продаж и есть возможность удалить продажи. Окно с продажами приведено на рисунке 3.9.

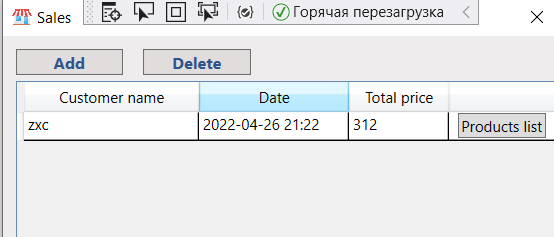


Рисунок 3.9 – Окно продажи для администратора

При закрытии окна «продажи» администратор возвращается в окно с главным меню. Помимо окна «продажи», у администратора есть кнопка «сотрудники». При нажатии на кнопку «сотрудники» будет открыто окно, в данном окне находится список сотрудников. У администратора также есть возможность управлять сотрудниками (добавление и удаление). Окно с сотрудниками приведено на рисунке 3.10.

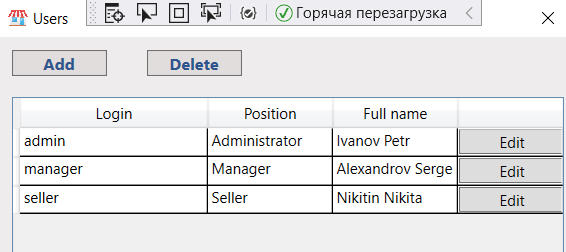


Рисунок 3.10 – Окно сотрудники для администратора

При нажатии на кнопку «категории» открывается окно, в котором администратор может просмотреть все ранее занесенные категории в базу данных и редактировать их. Для редактирования необходимо ввести новое значение оценки в соответствующее поле снизу и нажать кнопку «Редактировать». Результат редактирования категории приведен на рисунке 3.11.

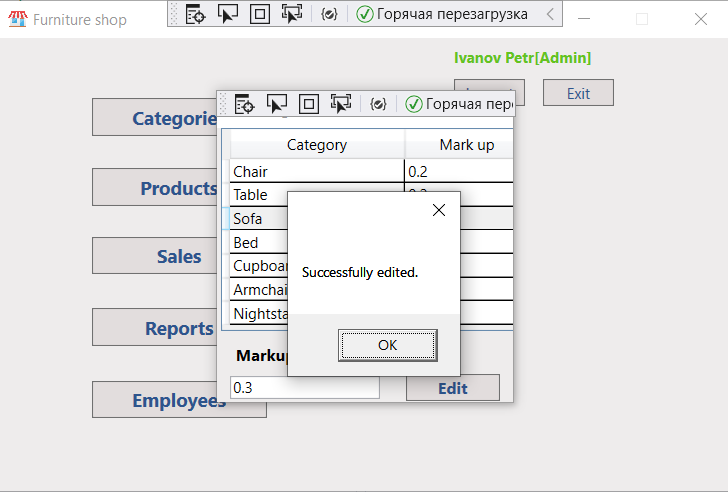


Рисунок 3.11 – Окно администратора для редактирования категории

Главное окно для менеджера, вошедшего в приложение изображено на рисунке 3.12. В меню в верхнем правом углу располагается информация о менеджере (его фамилия и имя), вошедшего в приложение. Также у менеджера присутствуют следующие возможности:

– просмотр и редактирование категорий;

– просмотр и редактирование товаров;

– формирование отчетов;

– добавление распродаж.

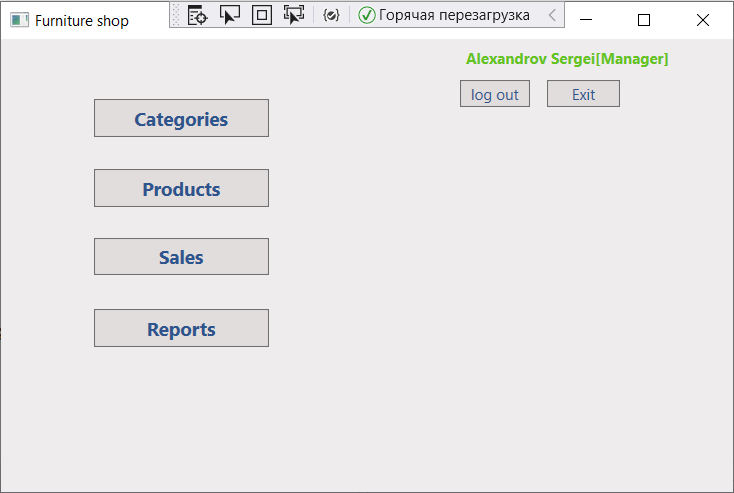


Рисунок 3.12 – Главное окно для менеджера

Главное окно продавца изображено на рисунке 3.13. У продавца присутствуют следующие возможности:

– просмотр списка категорий;

– просмотр списка товаров и продаж;

– добавление новой покупки.

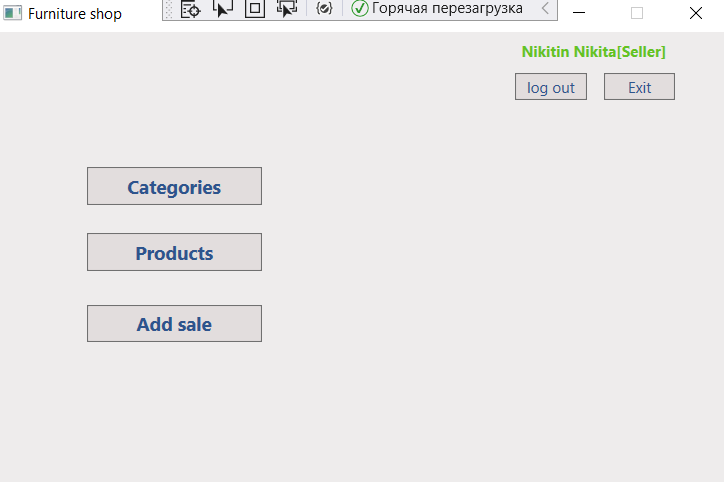


Рисунок 3.13 – Главное окно для продавца

При нажатии на кнопку «добавить распродажи» будет открыто окнос информацией о распродажах. В данном окне продавец может выполнять следующие действия:

– добавление распродажи;

Окно с товарами приведено на рисунке 3.14.

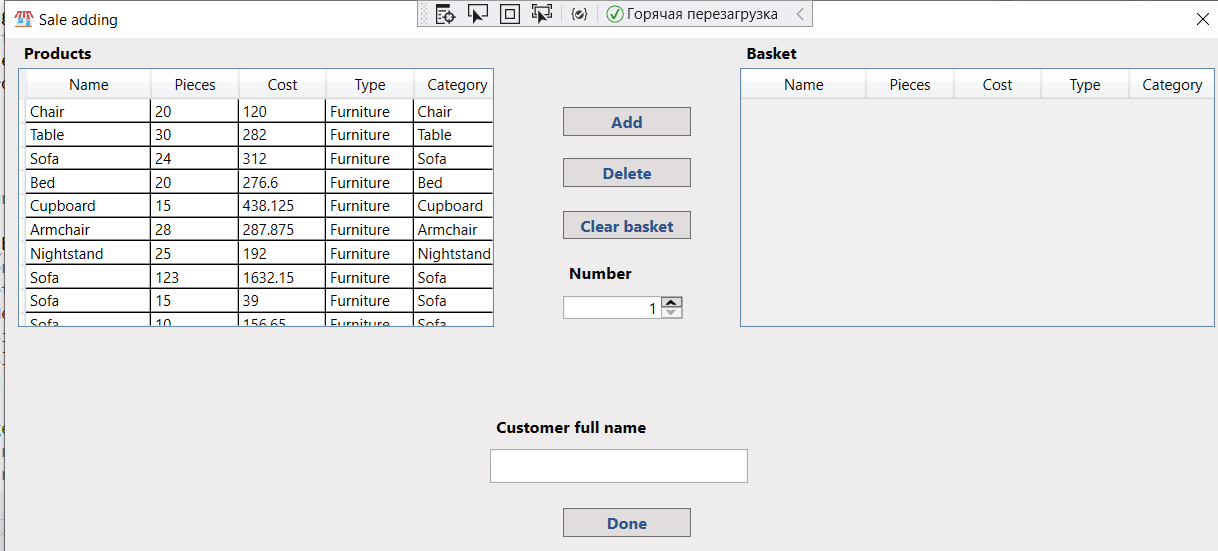


Рисунок 3.14 – Окно добавление распродаж для продавца

## 3.2 Результаты проведения модульного тестирования

Модульные тесты написаны на том же языке программирования, что и исходный код приложения. Они создаются непосредственно для проверки этого кода путем создания библиотеки классов *UnitTests*. То есть модульные тесты – это код, который проверяет корректность другого кода. Модульное тестирование максимально влияет на качество кода, когда оно является неотъемлемой частью рабочего процесса разработки ПО. Тестирование должно покрывать весь функционал приложения, однако протестировать все методы не всегда возможно. В данной информационной системе нужно протестировать работу получения данных.

Результаты проведения модульного тестирования представлены на рисунке 3.14.

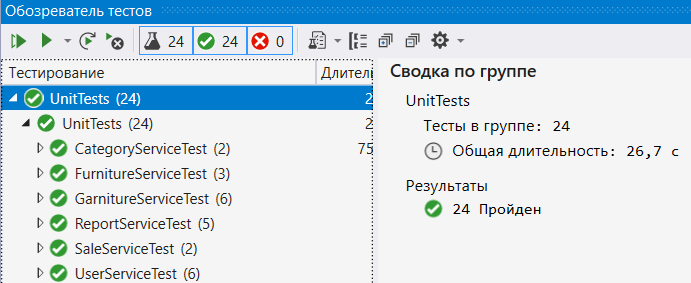


Рисунок 3.14 – Результат выполнения модульного тестирования приложения

На рисунке 3.15 представлен результат проведения тестирования классов и их методов группы классов *UnitTests*. Данная группа классов проверяет функционирование всех методов.

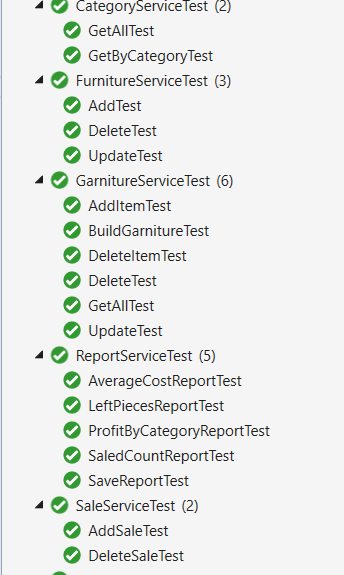


Рисунок 3.15 – Группа классов *UnitTests*

Так, класс *AddSaleTest* содержит в себе методы проверки функционала по добавлению распродаж.

Класс *DeleteSaleTest* содержит в себе методы проверки функционала по удалению распродаж.

Класс *SaveReportTest* содержит в себе методы проверки функционала сохранения отчетов.

Класс *SaledCountReportTest* содержит в себе методы проверки функционала отчета продаж.

По результатам модульных тестов возможно сделать вывод о том, что разработанное приложение работает корректно и выполняет все возложенные на него функции соответственно.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта решена поставленная задача. Разработано приложение для учета продаж мебельного магазина. Выполнены поставленные цели. Изучены технологии *LINQ* для доступа и управления реляционными данными *MySQL*.

Программный комплекс разработан в соответствие со всеми современными стандартами и возможностями объектно-ориентированного языка *C*#. Соблюдены все требования к удобочитаемости интерфейса, за счет этого дальнейшее использование данного программного комплекса осуществляется намного проще и эффективнее.

К приложению обеспечен многопользовательский доступ для различных типов пользователей. Каждый пользователь имеет свои возможности:

– продавец может только добавлять распродажи;

– администратор может удалять и добавлять пользователей, сбрасывать пароли, регистрировать пользователей, подавших заявку на регистрацию;

– менеджер может просматривать товары и отчеты.

Разработанный продукт, позволяющий продавать и непосредственно вести учет продаж мебельного магазина, имеет ряд преимуществ по сравнению с существующими аналогами:

– простая и масштабируемая архитектура приложения, предоставляющая независимость между уровнями;

– наличие системы, информирующей пользователя о его действиях;

– удобный и простой графический интерфейс;

– доступ в систему трех видов пользователей;

– низкие системные требования.

Проведенные модульные и функциональные тесты полностью доказывают работоспособность данной программы и функционирование всех требуемых ролей и методов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы объектно-ориентированного программирования [Электронный ресурс]. – Режим доступа: http://algolist.manual.ru/ds/walk.php.

2. Куликов, С. С. Реляционные базы данных в примерах: практическое пособие / С. С. Куликов. – Минск: Четыре четверти, 2020. – 424 с.

3. 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>.

4. Васильев, А. C#. Объектно-ориентированное программирование. Учебный курс / А. Васильев. – СПб.: Питер, 2012. – 320 c.

5. Основы тестирования программного обеспечения: Файловый архив для студентов. *Studfiles*. – Электрон. данные. – Режим доступа: https://studfile.net/preview/4494386/. – Дата доступа: 22.05.2022.

6. База данных [Электронный ресурс] – Режим доступа: [https://ru.wikipedia.org/wiki/База\_данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

# ПРИЛОЖЕНИЕ А

(обязательное)

**Листинг программы**

**ArmchairCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

public class ArmchairCreator : FurnitureCreator

{

/// <summary>

/// Creates a new instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

public override Furniture Create(int id, string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new ArmchairFurniture(id, name, quantity, purchasePrice, type, categoryId);

}

}

}

**BedCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

public class BedCreator:FurnitureCreator

{

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <returns>New instance</returns>

public override Furniture Create(int id, string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new BedFurniture(id, name, quantity, purchasePrice, type, categoryId);

}

}

}

**ChairCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

public class ChairCreator:FurnitureCreator

{

/// <summary>

/// Creates a new instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

public override Furniture Create(int id,string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new ChairFurniture(id,name, quantity, purchasePrice, type, categoryId);

}

}

}

**CupboardCreator.cs**

using System;

using BLL.Models.Products;

namespace FurnitureFactories

{

/// <summary>

/// Provides an implemention for a method to create a CupboardFurniture class instance

/// </summary>

public class CupboardCreator : FurnitureCreator

{

/// <summary>

/// Creates a new instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <returns>New instance</returns>

public override Furniture Create(int id, string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new CupboardFurniture(id, name, quantity, purchasePrice, type, categoryId);

}

}

}

**FurnitureCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

/// <summary>

/// Represents base class for furniture creators

/// </summary>

public abstract class FurnitureCreator

{

/// <summary>

/// Creates a new instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <returns>New instance</returns>

public abstract Furniture Create(int id,string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId);

}

}

**FurnitureFactory.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

public static class FurnitureFactory

{

/// <summary>

/// Creates furniture of selected category

/// </summary>

/// <param name="category">Category type</param>

/// <param name="name">Name</param>

/// <param name="quantity">Qunatity</param>

/// <param name="purchasePrice">Purchse price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <param name="id">Id of furniture</param>

/// <returns>New furniture instance</returns>

public static Furniture CreateFurniture(Categories category,string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId = 0,int id = 0)

{

Furniture furniture = null;

switch (category)

{

case Categories.Armchair:

furniture = new ArmchairCreator().Create(id, name, quantity, purchasePrice, type, categoryId);

break;

case Categories.Bed:

furniture = new BedCreator().Create(id, name, quantity, purchasePrice, type, categoryId);

break;

case Categories.Chair:

furniture = new ChairCreator().Create(id, name, quantity, purchasePrice, type, categoryId);

break;

case Categories.Cupboard:

furniture = new CupboardCreator().Create(id, name, quantity, purchasePrice, type, categoryId);

break;

case Categories.Nightstand:

furniture = new NightstandCreator().Create(id, name, quantity, purchasePrice, type, categoryId);

break;

case Categories.Sofa:

furniture = new SofaCreator().Create(id, name, quantity, purchasePrice, type, categoryId);

break;

case Categories.Table:

furniture = new TableCreator().Create(id,name, quantity, purchasePrice, type, categoryId);

break;

}

return furniture;

}

}

}

**NightstandCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

/// <summary>

/// Provides an implemention for a method to create a NightstandFurniture class instance

/// </summary>

public class NightstandCreator : FurnitureCreator

{

/// <summary>

/// Creates a new instance of furntire

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <returns>New instance</returns>

public override Furniture Create(int id, string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new NightstandFurniture(id, name, quantity, purchasePrice, type, categoryId);

}

}

}

**SofaCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

/// <summary>

/// Provides an implemention for a method to create a SofaFurniture class instance

/// </summary>

public class SofaCreator : FurnitureCreator

{

/// <summary>

/// Creates a new instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <returns>New instance</returns>

public override Furniture Create(int id, string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new SofaFurniture(id, name, quantity, purchasePrice, type, categoryId);

}

}

}

**TableCreator.cs**

using BLL.Models.Products;

namespace FurnitureFactories

{

/// <summary>

/// Provides an implemention for a method to create a TableFurniture class instance

/// </summary>

public class TableCreator : FurnitureCreator

{

/// <summary>

/// Creates a new instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quntity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="type">Furniture type</param>

/// <param name="categoryId">Id of category</param>

/// <returns>New instance</returns>

public override Furniture Create(int id,string name, int quantity, double purchasePrice, FurnitureTypes type,int categoryId)

{

return new TableFurniture(id,name, quantity, purchasePrice, type, categoryId);

}

}

}

**GarnitureBuilder.cs**

using System;

using FurnitureFactories;

using BLL.Models.Products;

namespace GarnitureFactories

{

/// <summary>

/// Provides methods for garniture creating

/// </summary>

public class GarnitureBuilder

{

/// <summary>

/// Builder instance

/// </summary>

private static readonly Lazy<GarnitureBuilder> builder = new Lazy<GarnitureBuilder>(() => new GarnitureBuilder());

/// <summary>

/// Creates an instance of GarnitureBuilder class

/// </summary>

private GarnitureBuilder(){}

/// <summary>

/// Gets builder instance

/// </summary>

/// <returns>Builder instance</returns>

public static GarnitureBuilder GetBuilder()

{

return builder.Value;

}

/// <summary>

/// Current gurniture

/// </summary>

private Garniture garniture;

/// <summary>

/// Starts garniture creating

/// </summary>

public void Begin()

{

garniture = new Garniture();

}

/// <summary>

/// Adds item to garniture

/// </summary>

/// <param name="categoryType">Category of item</param>

/// <param name="name">Name</param>

/// <param name="quantity">Qunatity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="category">Category instance</param>

/// <returns>Garniture record</returns>

public GarnitureRecord AddItem(Categories categoryType,string name,int quantity,double purchasePrice,Category category)

{

var furniture = FurnitureFactory.CreateFurniture(categoryType, name,0,purchasePrice,FurnitureTypes.Garniture, category.Id);

furniture.Category = category;

var record = new GarnitureRecord { Quantity = quantity,Furniture = furniture };

garniture.Records.Add(record);

return record;

}

/// <summary>

/// Removes item from garniture by garniture record

/// </summary>

/// <param name="record">Garntiure record</param>

/// <returns></returns>

public bool RemoveItem(GarnitureRecord record)

{

return garniture.Records.Remove(record);

}

/// <summary>

/// Ends garntire creating

/// </summary>

/// <param name="name">Garntire name</param>

/// <param name="piecesNumber">Number of garniture pieces</param>

/// <returns>Garntiure</returns>

public Garniture BuildGarniture(string name,int piecesNumber)

{

garniture.Name = name;

garniture.PiecesLeft = piecesNumber;

foreach(var record in garniture.Records)

{

record.Furniture.PiecesLeft = piecesNumber \* record.Quantity;

}

return garniture;

}

}

}

**AdministratorCreator.cs**

using BLL.Models.Users;

namespace UsersFactory

{

/// <summary>

/// Provides an implemention for a method to create an Administrator class instance

/// </summary>

public class AdministratorCreator : UserCreator

{

/// <summary>

/// Creates an user instance

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

/// <returns>Created user</returns>

public override User Create(int id, string login, string password, string fullname)

{

return new Administrator(id, login, password, fullname);

}

}

}

**ManagerCreator.cs**

using BLL.Models.Users;

namespace UsersFactory

{

/// <summary>

/// Provides an implemention for a method to create an Manager class instance

/// </summary>

public class ManagerCreator : UserCreator

{

/// <summary>

/// Creates an user instance

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

/// <returns>Created user</returns>

public override User Create(int id, string login, string password, string fullname)

{

return new Manager(id, login, password,fullname);

}

}

}

**SellerCreator.cs**

using BLL.Models.Users;

namespace UsersFactory

{

/// <summary>

/// Provides an implemention for a method to create an Seller class instance

/// </summary>

public class SellerCreator : UserCreator

{

/// <summary>

/// Creates an user instance

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

/// <returns>Created user</returns>

public override User Create(int id, string login, string password,string fullname)

{

return new Seller(id,login,password,fullname);

}

}

}

**UserCreator.cs**

using BLL.Models.Users;

namespace UsersFactory

{

/// <summary>

/// Represents base class for user creators

/// </summary>

public abstract class UserCreator

{

/// <summary>

/// Creates an user instance

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

/// <returns>Created user</returns>

public abstract User Create(int id,string login, string password,string fullname);

}

}

**UserFactory.cs**

using BLL.Models.Users;

namespace UsersFactory

{

/// <summary>

/// Represents method for creating selected user

/// </summary>

public static class UserFactory

{

/// <summary>

/// Creates user by role

/// </summary>

/// <param name="role">Role</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

/// <param name="id">User id</param>

/// <returns>Created user</returns>

public static User CreateByRole(Roles role,string login,string password,string fullname, int id=0)

{

User user = null;

switch (role)

{

case Roles.Administrator:

user = new AdministratorCreator().Create(id, login, password,fullname);

break;

case Roles.Manager:

user = new ManagerCreator().Create(id, login, password, fullname);

break;

case Roles.Seller:

user = new SellerCreator().Create(id, login, password, fullname);

break;

}

return user;

}

}

}

**Category.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Models.Products

{

/// <summary>

/// Furniture categories

/// </summary>

public enum Categories

{

None,

Chair,

Table,

Sofa,

Bed,

Cupboard,

Armchair,

Nightstand

}

/// <summary>

/// Represents category

/// </summary>

public class Category

{

/// <summary>

/// Creates an instance of Category class

/// </summary>

public Category(){}

// <summary>

/// Creates an instance of Category class

/// </summary>

/// <param name="id">Id</param>

/// <param name="categoryType">Category type</param>

/// <param name="markUp">Markup</param>

public Category(int id,Categories categoryType,double markUp)

{

Id = id;

CategoryType = categoryType;

MarkUp = markUp;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Category type

/// </summary>

public Categories CategoryType { get; set; }

/// <summary>

/// Mark up

/// </summary>

public double MarkUp { get; set; }

/// <summary>

/// Returns string representation of category

/// </summary>

/// <returns>String representation of category</returns>

public override string ToString()

{

return CategoryType.ToString();

}

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Category category = (Category)obj;

return Id == category.Id && CategoryType == category.CategoryType && MarkUp == category.MarkUp;

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**Furniture.cs**

using System;

namespace BLL.Models.Products

{

/// <summary>

/// Furntiru types

/// </summary>

public enum FurnitureTypes

{

Piece,

Garniture

}

/// <summary>

/// Represents an furniture

/// </summary>

public abstract class Furniture : IProduct

{

/// <summary>

/// Creates an instance of furniture

/// </summary>

public Furniture() { }

/// <summary>

/// Creates an instance of furniture

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quantity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="furnitureType">Furniture type</param>

/// <param name="categoryId">Category id</param>

/// <param name="garnitureRecordId">Garniture record id</param>

public Furniture(int id,string name, int quantity, double purchasePrice, FurnitureTypes furnitureType,int categotyId, int? garnitureRecordId)

{

Id = id;

Name = name;

PiecesLeft = quantity;

PurchasePrice = purchasePrice;

FurnitureType = furnitureType;

CategoryId = categotyId;

GarnitureRecordId = garnitureRecordId;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Category type

/// </summary>

public abstract Categories CategoryType { get; }

/// <summary>

/// Product type

/// </summary>

public ProductTypes ProductType { get =>ProductTypes.Furniture;}

/// <summary>

/// Furntiure type

/// </summary>

public FurnitureTypes FurnitureType { get; set; }

/// <summary>

/// Name

/// </summary>

public string Name { get; set; }

/// <summary>

/// Pieces left

/// </summary>

public int PiecesLeft { get; set; }

/// <summary>

/// Purchase price

/// </summary>

public double PurchasePrice { get; set; }

/// <summary>

/// Retail price

/// </summary>

public double RetailPrice { get=>PurchasePrice+PurchasePrice\*Category.MarkUp;}

/// <summary>

/// Category id

/// </summary>

public int CategoryId { get; set; }

/// <summary>

/// Garniture record id

/// </summary>

public int? GarnitureRecordId { get; set; }

public Category Category { get; set; }

/// <summary>

/// Decreases pieces quantity

/// </summary>

/// <param name="quantity">Quantity</param>

public void TakePieces(int quantity)

{

PiecesLeft -= quantity;

}

/// <summary>

/// Increases pieces quantity

/// </summary>

/// <param name="quantity">Quantity</param>

public void PutPieces(int quantity)

{

PiecesLeft += quantity;

}

/// <summary>

/// Gets a copy of instance

/// </summary>

/// <returns>The copy of instance</returns>

public abstract object Clone();

}

}

**Garniture.cs**

using System.Collections.Generic;

using System.Linq;

namespace BLL.Models.Products

{

/// <summary>

/// Represents a garniture

/// </summary>

public class Garniture : IProduct

{

/// <summary>

/// Creates an instance of Garniture class

/// </summary>

public Garniture() { }

/// <summary>

/// Creates an instance of Garniture class

/// </summary>

/// <param name="id">Id</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quantity</param>

public Garniture(int id, string name, int quantity)

{

Id = id;

Name = name;

PiecesLeft = quantity;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Product type

/// </summary>

public ProductTypes ProductType { get => ProductTypes.Garniture; }

/// <summary>

/// Name

/// </summary>

public string Name { get; set; }

/// <summary>

/// Pieces left

/// </summary>

public int PiecesLeft { get; set; }

/// <summary>

/// Retail price

/// </summary>

public double RetailPrice { get => GetPrice(); }

/// <summary>

/// Category type

/// </summary>

public Categories CategoryType { get => Categories.None; }

/// <summary>

/// Garniture records

/// </summary>

public List<GarnitureRecord> Records { get; set; }=new List<GarnitureRecord>();

/// <summary>

/// Gets price of the garniture

/// </summary>

/// <returns>Price of the garniture</returns>

public double GetPrice()

{

var sum = 0.0;

foreach (var record in Records)

{

sum += record.Price;

}

return sum;

}

/// <summary>

/// Decreases pieces quantity

/// </summary>

/// <param name="quantity">Quantity</param>

public void TakePieces(int quantity)

{

PiecesLeft -= quantity;

foreach(var record in Records)

{

record.Furniture.PiecesLeft -= quantity \* record.Quantity;

}

}

/// <summary>

/// Increases pieces quantity

/// </summary>

/// <param name="quantity">Quantity</param>

public void PutPieces(int quantity)

{

PiecesLeft += quantity;

foreach (var record in Records)

{

record.Furniture.PiecesLeft += quantity \* record.Quantity;

}

}

/// <summary>

/// Gets copie of garniture

/// </summary>

/// <returns>Copie of garniture</returns>

public object Clone()

{

return this.MemberwiseClone();

}

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Garniture garniture = (Garniture)obj;

return Id==garniture.Id && Name == garniture.Name &&

PiecesLeft == garniture.PiecesLeft && RetailPrice == garniture.RetailPrice &&

Records.SequenceEqual(garniture.Records);

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**GarnitureRecord*.cs***

using System.Collections.Generic;

namespace BLL.Models.Products

{

/// <summary>

/// Represents a garniture record

/// </summary>

public class GarnitureRecord

{

/// <summary>

/// Creates an instance of GanirnitureRecord class

/// </summary>

public GarnitureRecord() { }

/// <summary>

/// Creates an instance of GanirnitureRecord class

/// </summary>

/// <param name="id">Id</param>

/// <param name="quantity">Quantity</param>

/// <param name="garnitureId">Ganiture id</param>

public GarnitureRecord(int id,int quantity,int garnitureId)

{

Id = id;

Quantity = quantity;

GarnitureId = garnitureId;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Quantity

/// </summary>

public int Quantity { get; set; }

/// <summary>

/// Price of furnitures

/// </summary>

public double Price { get => Furniture.RetailPrice \* Quantity; }

/// <summary>

/// Garniture id

/// </summary>

public int? GarnitureId { get; set; }

/// <summary>

/// Furntiture of record

/// </summary>

public Furniture Furniture{ get; set; }

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

GarnitureRecord record = (GarnitureRecord)obj;

return Id == record.Id && Quantity == record.Quantity && Price == record.Price &&

GarnitureId == record.GarnitureId && Furniture.Equals(record.Furniture);

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**IProduct.cs**  
  
using System;

namespace BLL.Models.Products

{

public enum ProductTypes

{

Furniture,

Garniture

}

/// <summary>

/// Represents a product

/// </summary>

public interface IProduct : ICloneable

{

/// <summary>

/// Id

/// </summary>

int Id { get; set; }

/// <summary>

/// Product type

/// </summary>

ProductTypes ProductType { get;}

/// <summary>

/// Name

/// </summary>

string Name { get; set; }

///<summary>

/// Pieces left

/// </summary>

int PiecesLeft { get; set; }

/// <summary>

/// Retail price

/// </summary>

double RetailPrice { get;}

/// <summary>

/// Category of product

/// </summary>

Categories CategoryType { get; }

/// <summary>

/// Decreases pieces quantity

/// </summary>

/// <param name="quantity">Quantity</param>

void TakePieces(int quantity);

/// <summary>

/// Increases pieces quantity

/// </summary>

/// <param name="quantity">Quantity</param>

void PutPieces(int quantity);

}

}

**Sale.cs**using System;

using System.Collections.Generic;

using System.Linq;

namespace BLL.Models.Sales

{

/// <summary>

/// Represents a sale

/// </summary>

public class Sale

{

/// <summary>

/// Creates an instance of a sale

/// </summary>

public Sale() { }

/// <summary>

/// Creates an instance of a sale

/// </summary>

/// <param name="id">Id</param>

/// <param name="customerName">Customer name</param>

/// <param name="date">Date</param>

public Sale(int id,string customerName,DateTime date)

{

Id = id;

CustomerName = customerName;

Date = date;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Customer name

/// </summary>

public string CustomerName { get; set; }

/// <summary>

/// Sale date

/// </summary>

public DateTime Date { get; set; }

/// <summary>

/// Total price

/// </summary>

public double TotalPrice { get => GetTotalPrice();}

/// <summary>

/// Sale records

/// </summary>

public List<SaleRecord> Records { get; set; } = new List<SaleRecord>();

/// <summary>

/// Gets total price

/// </summary>

/// <returns>Total price of the sale</returns>

private double GetTotalPrice()

{

var totalPrice = 0.0;

foreach(var record in Records)

{

totalPrice += record.SumPrice;

}

return totalPrice;

}

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Sale sale = (Sale)obj;

return Id == sale.Id && CustomerName == sale.CustomerName && Date == sale.Date && Records.SequenceEqual(sale.Records);

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**SaleRecord.cs**using BLL.Models.Products;

namespace BLL.Models.Sales

{

/// <summary>

/// Represents a sale record

/// </summary>

public class SaleRecord

{

/// <summary>

/// Creates an instance of SaleRecord class

/// </summary>

public SaleRecord() { }

/// <summary>

/// Creates an instance of SaleRecord class

/// </summary>

/// <param name="id">Id</param>

/// <param name="piecePrice">Piece price</param>

/// <param name="quantity">Quantity</param>

/// <param name="saleId">Sale id</param>

/// <param name="furnitureId">Funriture id</param>

public SaleRecord(int id,double piecePrice,int quantity,int saleId,int furnitureId)

{

Id = id;

PiecePrice = piecePrice;

Quantity = quantity;

FurnitureId = furnitureId;

SaleId = saleId;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Piece price

/// </summary>

public double PiecePrice { get; set; }

/// <summary>

/// Quantity

/// </summary>

public int Quantity { get; set; }

/// <summary>

/// Sum price

/// </summary>

public double SumPrice { get => PiecePrice \* Quantity; }

/// <summary>

/// Furniture id

/// </summary>

public int FurnitureId { get; set; }

/// <summary>

/// Sale id

/// </summary>

public int? SaleId { get; set; }

/// <summary>

/// Furniture

/// </summary>

public Furniture Furniture { get; set; }

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

SaleRecord saleRecord = (SaleRecord)obj;

return Id == saleRecord.Id && PiecePrice == saleRecord.PiecePrice && Quantity == saleRecord.Quantity &&

FurnitureId == FurnitureId && SaleId == SaleId && Furniture.Equals(saleRecord.Furniture);

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**Administrator.cs**

using System;

using System.Collections.Generic;

namespace BLL.Models.Users

{

/// <summary>

/// Represents an administrator

/// </summary>

public class Administrator : User

{

/// <summary>

/// Creates an instance of Administrator class

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

public Administrator(int id, string login, string password,string fullname):

base(id,login,password,fullname) { }

/// <summary>

/// Private field of role

/// </summary>

private Roles role = Roles.Administrator;

/// <summary>

/// Role

/// </summary>

public override Roles Role { get => role; set => role=value; }

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Administrator user = (Administrator)obj;

return Role == user.Role && Login == user.Login && Password == user.Password && FullName == user.FullName;

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**Manager.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Models.Users

{

/// <summary>

/// Represents a manager

/// </summary>

public class Manager:User

{

/// <summary>

/// Creates an instance of Manager class

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

public Manager(int id, string login, string password,string fullname) :

base(id, login, password,fullname) { }

/// <summary>

/// Private field of role

/// </summary>

private Roles role = Roles.Manager;

/// <summary>

/// Role

/// </summary>

public override Roles Role { get => role; set => role = value; }

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Manager user = (Manager)obj;

return Role == user.Role && Login == user.Login && Password == user.Password && FullName == user.FullName;

}

public override int GetHashCode()

{

return Id;

}

}

}

**Seller.cs**using System;

using System.Collections.Generic;

namespace BLL.Models.Users

{

/// <summary>

/// Represents a seller

/// </summary>

public class Seller:User

{

/// <summary>

/// Creates an instance of Seller class

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

public Seller(int id, string login, string password,string fullname) :

base(id, login, password,fullname) { }

/// <summary>

/// Private field of role

/// </summary>

private Roles role = Roles.Seller;

/// <summary>

/// Role

/// </summary>

public override Roles Role { get => role; set => role = value; }

/// <summary>

/// Redefining the Equals method

/// </summary>

/// <param name="obj"></param>

/// <returns>True if the specified object is equal to the current object; otherwise, false</returns>

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Seller user = (Seller)obj;

return Role==user.Role && Login == user.Login && Password == user.Password && FullName == user.FullName;

}

/// <summary>

/// Gets hash code of object

/// </summary>

/// <returns>Hash code of object</returns>

public override int GetHashCode()

{

return Id;

}

}

}

**User.cs**

namespace BLL.Models.Users

{

/// <summary>

/// User roles

/// </summary>

public enum Roles

{

Administrator,

Manager,

Seller,

Guest

}

/// <summary>

/// Represents an user

/// </summary>

public abstract class User

{

/// <summary>

/// Creates an user instance

/// </summary>

/// <param name="id">Id</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullname">Fullname</param>

public User(int id,string login,string password,string fullname)

{

Id = id;

Login = login;

Password = password;

FullName = fullname;

}

/// <summary>

/// Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// Login

/// </summary>

public string Login { get; set; }

/// <summary>

/// Password

/// </summary>

public string Password { get; set; }

/// <summary>

/// Role

/// </summary>

public abstract Roles Role { get; set; }

/// <summary>

/// Fullname

/// </summary>

public string FullName { get; set; }

}

}

**ICategoryService.cs**

using System.Collections.Generic;

using BLL.Models.Products;

namespace BLL.ServiceInterfaces

{

/// <summary>

/// Represents an interface of category service

/// </summary>

public interface ICategoryService

{

/// <summary>

/// Gets all categories

/// </summary>

/// <returns>List of categories</returns>

List<Category> GetAll();

/// <summary>

/// Gets by given category type

/// </summary>

/// <param name="categoryType">Category type</param>

/// <returns>An instance of a Category class</returns>

Category GetByCategory(Categories categoryType);

/// <summary>

/// Updates a category

/// </summary>

/// <param name="category"></param>

/// <returns></returns>

bool Update(Category category);

}

}

**IFurnitureService.cs**

using BLL.Models.Products;

using System.Collections.Generic;

namespace BLL.ServiceInterfaces

{

/// <summary>

/// Represents an interface of furniture service

/// </summary>

public interface IFurnitureService

{

/// <summary>

/// Adds a furniture

/// </summary>

/// <param name="category">Category type</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quantity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="categoryId">Category id</param>

/// <returns>Id of added furniture</returns>

int Add(Categories category, string name, int quantity, double purchasePrice, int categoryId);

/// <summary>

/// Deletes a furniuture

/// </summary>

/// <param name="furniture"></param>

/// <returns>Result of operation</returns>

bool Delete(Furniture furniture);

/// <summary>

/// Gets all furniture

/// </summary>

/// <returns>List of furniture</returns>

List<Furniture> GetAll();

/// <summary>

/// Gets all furniture by given type

/// </summary>

/// <param name="type"></param>

/// <returns>List of furniture</returns>

List<Furniture> GetAllByType(FurnitureTypes type);

/// <summary>

/// Gets furniture by id

/// </summary>

/// <param name="id"></param>

/// <returns>List of furntiure</returns>

Furniture GetById(int id);

/// <summary>

/// Updates furniture

/// </summary>

/// <param name="furniture"></param>

/// <returns>Result of operation</returns>

bool Update(Furniture furniture);

}

}

**IGarnitureService.cs**using System.Collections.Generic;

using BLL.Models.Products;

namespace BLL.ServiceInterfaces

{

/// <summary>

/// Represents an interface of garniture service

/// </summary>

public interface IGarnitureService

{

/// <summary>

/// Adds item to garniture

/// </summary>

/// <param name="categoryType">Category type</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quantity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="category">Category</param>

/// <returns>Garniture record</returns>

GarnitureRecord AddItem(Categories categoryType, string name, int quantity, double purchasePrice, Category category);

/// <summary>

/// Adds a garniture to database

/// </summary>

/// <param name="name">Name</param>

/// <param name="quantity">Qunatity</param>

/// <returns>Id of added garniture</returns>

int BuildGarniture(string name, int quantity);

/// <summary>

/// Deletes a garniture

/// </summary>

/// <param name="garniture"></param>

/// <returns>Result of operation</returns>

bool Delete(Garniture garniture);

/// <summary>

/// Deletes item from garniture

/// </summary>

/// <param name="record">Garniture record</param>

void DeleteItem(GarnitureRecord record);

/// <summary>

/// Gets all garnitures

/// </summary>

/// <returns>List of garniture</returns>

List<Garniture> GetAll();

/// <summary>

/// Gets garniture by id

/// </summary>

/// <param name="id">Id</param>

/// <returns>Garniture</returns>

Garniture GetById(int id);

/// <summary>

/// Starts creating of new garniture

/// </summary>

void MakeGarniture();

/// <summary>

/// Updates a garniture

/// </summary>

/// <param name="garniture"></param>

/// <returns>Result of operation</returns>

bool Update(Garniture garniture);

}

}

**IReportService.cs**

using System;

using System.Data;

namespace BLL.ServiceInterfaces

{

/// <summary>

/// Represents an interface of report service

/// </summary>

public interface IReportService

{

/// <summary>

/// Creates an average cost report

/// </summary>

/// <returns>Report table</returns>

DataTable AverageCostReport();

/// <summary>

/// Creates a left pieces report

/// </summary>

/// <returns>Report table</returns>

DataTable LeftPiecesReport();

/// <summary>

/// Creates a profit by category report

/// </summary>

/// <param name="fromDate">From date</param>

/// <param name="toDate">To date</param>

/// <returns>Report table</returns>

DataTable ProfitByCategoryReport(DateTime fromDate, DateTime toDate);

/// <summary>

/// Creates a saled count report

/// </summary>

/// <param name="fromDate">From date</param>

/// <param name="toDate">To date</param>

/// <returns>Report table</returns>

DataTable SaledCountReport(DateTime fromDate, DateTime toDate);

/// <summary>

/// Saves data to file

/// </summary>

/// <param name="path">Path</param>

/// <param name="dataTable">Data</param>

void SaveReport(string path, DataTable dataTable);

}

}

**ISaleService.cs**

using BLL.Models.Products;

using BLL.Models.Sales;

using System;

using System.Collections.Generic;

namespace BLL.ServiceInterfaces

{

/// <summary>

/// Represents an interface of sale service

/// </summary>

public interface ISaleService

{

/// <summary>

/// Adds sale

/// </summary>

/// <param name="customerName">Customer name</param>

/// <param name="date">Sale date</param>

/// <param name="products">Products list</param>

/// <returns>Sale id</returns>

int AddSale(string customerName, DateTime date, List<IProduct> products);

/// <summary>

/// Deletes sale

/// </summary>

/// <param name="sale">Sale</param>

/// <returns>Result of operation</returns>

bool Delete(Sale sale);

/// <summary>

/// Gets all sales

/// </summary>

/// <returns>Sales list</returns>

List<Sale> GetAll();

/// <summary>

/// Gets sale by id

/// </summary>

/// <param name="id"></param>

/// <returns>Sale</returns>

Sale GetById(int id);

}

}

**IUserService.cs**

using System.Collections.Generic;

using BLL.Models.Users;

namespace BLL.ServiceInterfaces

{

/// <summary>

/// Represents an interface of user service

/// </summary>

public interface IUserService

{

/// <summary>

/// Authorizes user

/// </summary>

/// <param name="password">Passowrd</param>

/// <param name="login">Login</param>

/// <param name="user">User</param>

/// <returns>Result of autorization</returns>

bool Authorize(string password, string login, out User user);

/// <summary>

/// Creates new account

/// </summary>

/// <param name="role">Role</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullName">Fullname</param>

/// <returns>Id</returns>

int CreateAccount(Roles role, string login, string password,string fullName);

/// <summary>

/// Edits account

/// </summary>

/// <param name="user">User</param>

/// <returns>Result of operation</returns>

bool EditAccount(User user);

/// <summary>

/// Deletes account

/// </summary>

/// <param name="user">User</param>

/// <returns>Result of operation</returns>

bool DeleteAccount(User user);

/// <summary>

/// Checks user existing by login

/// </summary>

/// <param name="login">Login</param>

/// <returns>Result of checking user existing</returns>

bool IsExistUser(string login);

/// <summary>

/// Gets user by id

/// </summary>

/// <param name="id">Id</param>

/// <returns>User</returns>

User GetById(int id);

/// <summary>

/// Gets all users

/// </summary>

/// <returns>List of users</returns>

List<User> GetUsers()

}

}

**CategoryService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DAL.ORM;

using BLL.Models.Products;

using BLL.ServiceInterfaces;

namespace BLL.Services

{

/// <summary>

/// Represents an category service

/// </summary>

public class CategoryService : ICategoryService

{

private IDataService<Category> categoryDataService;

/// <summary>

/// Creates category service instance

/// </summary>

/// <param name="categoryDataService">Category data service</param>

public CategoryService(IDataService<Category> categoryDataService)

{

this.categoryDataService = categoryDataService;

}

/// <summary>

/// Updates a category

/// </summary>

/// <param name="category"></param>

/// <returns></returns>

public bool Update(Category category)

{

return categoryDataService.Update(category);

}

/// <summary>

/// Gets by given category type

/// </summary>

/// <param name="categoryType">Category type</param>

/// <returns>An instance of a Category class</returns>

public Category GetByCategory(Categories categoryType)

{

return categoryDataService.GetAll().Find(category => category.CategoryType == categoryType);

}

/// <summary>

/// Gets all categories

/// </summary>

/// <returns>List of categories</returns>

public List<Category> GetAll()

{

return categoryDataService.GetAll();

}

}

}

**FurnitureService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DAL.ORM;

using FurnitureFactories;

using BLL.Models.Products;

using BLL.ServiceInterfaces;

namespace BLL.Services

{

/// <summary>

/// Represents a furniture service

/// </summary>

public class FurnitureService : IFurnitureService

{

private IDataService<Furniture> furnitureDataService;

/// <summary>

/// Creates an instance of furniture service

/// </summary>

/// <param name="furnitureDataService">Funiture data service</param>

public FurnitureService(IDataService<Furniture> furnitureDataService)

{

this.furnitureDataService = furnitureDataService;

}

/// <summary>

/// Adds a furniture

/// </summary>

/// <param name="category">Category type</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quantity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="categoryId">Category id</param>

/// <returns>Id of added furniture</returns>

public int Add(Categories category, string name, int quantity, double purchasePrice, int categoryId)

{

var furniture = FurnitureFactory.CreateFurniture(category, name, quantity, purchasePrice, FurnitureTypes.Piece, categoryId);

return furnitureDataService.Insert(furniture);

}

/// <summary>

/// Updates furniture

/// </summary>

/// <param name="furniture"></param>

/// <returns>Result of operation</returns>

public bool Update(Furniture furniture)

{

return furnitureDataService.Update(furniture);

}

/// <summary>

/// Deletes a furniuture

/// </summary>

/// <param name="furniture"></param>

/// <returns>Result of operation</returns>

public bool Delete(Furniture furniture)

{

return furnitureDataService.Delete(furniture);

}

/// <summary>

/// Gets all furniture

/// </summary>

/// <returns>List of furniture</returns>

public List<Furniture> GetAll()

{

return furnitureDataService.GetAll();

}

/// <summary>

/// Gets all furniture by given type

/// </summary>

/// <param name="type"></param>

/// <returns>List of furniture</returns>

public List<Furniture> GetAllByType(FurnitureTypes type)

{

return furnitureDataService.GetAll().Where(furniture => furniture.FurnitureType == type).ToList();

}

/// <summary>

/// Gets furniture by id

/// </summary>

/// <param name="id"></param>

/// <returns>List of furntiure</returns>

public Furniture GetById(int id)

{

return furnitureDataService.GetById(id);

}

}

}

**GarnitureService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DAL.ORM;

using GarnitureFactories;

using BLL.Models.Products;

using BLL.ServiceInterfaces;

namespace BLL.Services

{

/// <summary>

/// Represents a garniture service

/// </summary>

public class GarnitureService : IGarnitureService

{

private IDataService<Garniture> garnitureDataService;

private IDataService<GarnitureRecord> garnitureRecordDataService;

private IDataService<Furniture> furnitureDataService;

public GarnitureService(IDataService<Garniture> garnitureDataService, IDataService<GarnitureRecord> garnitureRecordDataService, IDataService<Furniture> furnitureDataService)

{

this.garnitureDataService = garnitureDataService;

this.garnitureRecordDataService = garnitureRecordDataService;

this.furnitureDataService = furnitureDataService;

}

/// <summary>

/// Starts creating of new garniture

/// </summary>

public void MakeGarniture()

{

var builder = GarnitureBuilder.GetBuilder();

builder.Begin();

}

/// <summary>

/// Adds item to garniture

/// </summary>

/// <param name="categoryType">Category type</param>

/// <param name="name">Name</param>

/// <param name="quantity">Quantity</param>

/// <param name="purchasePrice">Purchase price</param>

/// <param name="category">Category</param>

/// <returns>Garniture record</returns>

public GarnitureRecord AddItem(Categories categoryType, string name, int quantity, double purchasePrice, Category category)

{

var builder = GarnitureBuilder.GetBuilder();

return builder.AddItem(categoryType, name, quantity, purchasePrice, category);

}

/// <summary>

/// Deletes item from garniture

/// </summary>

/// <param name="record">Garniture record</param>

public void DeleteItem(GarnitureRecord record)

{

var builder = GarnitureBuilder.GetBuilder();

builder.RemoveItem(record);

}

/// <summary>

/// Adds a garniture to database

/// </summary>

/// <param name="name">Name</param>

/// <param name="quantity">Qunatity</param>

/// <returns>Id of added garniture</returns>

public int BuildGarniture(string name, int quantity)

{

var builder = GarnitureBuilder.GetBuilder();

var garniture = builder.BuildGarniture(name, quantity);

var id = garnitureDataService.Insert(garniture);

foreach (var record in garniture.Records)

{

record.GarnitureId = id;

var recordId = garnitureRecordDataService.Insert(record);

record.Furniture.GarnitureRecordId = recordId;

furnitureDataService.Insert(record.Furniture);

}

return id;

}

/// <summary>

/// Updates a garniture

/// </summary>

/// <param name="garniture"></param>

/// <returns>Result of operation</returns>

public bool Update(Garniture garniture)

{

var operationResult = garnitureDataService.Update(garniture);

foreach (var record in garniture.Records)

{

if (!garnitureRecordDataService.Update(record))

operationResult = false;

if (!furnitureDataService.Update(record.Furniture))

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Deletes a garniture

/// </summary>

/// <param name="garniture"></param>

/// <returns>Result of operation</returns>

public bool Delete(Garniture garniture)

{

return garnitureDataService.Delete(garniture);

}

/// <summary>

/// Gets all garnitures

/// </summary>

/// <returns>List of garniture</returns>

public List<Garniture> GetAll()

{

return garnitureDataService.GetAll();

}

/// <summary>

/// Gets garniture by id

/// </summary>

/// <param name="id">Id</param>

/// <returns>Garniture</returns>

public Garniture GetById(int id)

{

return garnitureDataService.GetById(id);

}

}

}

**ReportService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DAL.ORM;

using BLL.Models.Products;

using System.Xml.Linq;

using BLL.Models.Sales;

using BLL.ServiceInterfaces;

using System.Data;

namespace BLL.Services

{

/// <summary>

/// Represents a report service

/// </summary>

public class ReportService : IReportService

{

private IDataService<Sale> saleDataService;

private IDataService<Furniture> furnitureDataService;

/// <summary>

/// Creates report service instance

/// </summary>

/// <param name="saleDataService">Sale data service</param>

/// <param name="furnitureDataService">Furniture data service</param>

public ReportService(IDataService<Sale> saleDataService, IDataService<Furniture> furnitureDataService)

{

this.saleDataService = saleDataService;

this.furnitureDataService = furnitureDataService;

}

/// <summary>

/// Creates an average cost report

/// </summary>

/// <returns>Report table</returns>

public DataTable AverageCostReport()

{

var gropedFurniture = furnitureDataService.GetAll().GroupBy(element => element.CategoryType);

XDocument document = new XDocument();

XElement categories = new XElement("Categories");

foreach (var group in gropedFurniture)

{

XElement category = new XElement("Category");

XElement categoryName = new XElement("Name", group.Key.ToString());

XElement categoryAverage = new XElement("Average", Math.Round(group.Average(furniture => furniture.PurchasePrice), 2));

XElement categoryMarkUp = new XElement("Markup", group.First().Category.MarkUp);

category.Add(categoryName);

category.Add(categoryAverage);

category.Add(categoryMarkUp);

categories.Add(category);

}

document.Add(categories);

DataSet dataset = new DataSet();

dataset.ReadXml(document.CreateReader());

DataTable table = null;

if (dataset.Tables.Count != 0)

table = dataset.Tables[0];

return table;

}

/// <summary>

/// Creates a left pieces report

/// </summary>

/// <returns>Report table</returns>

public DataTable LeftPiecesReport()

{

var furnitures = furnitureDataService.GetAll().Where(furniture => furniture.PiecesLeft != 0);

XDocument document = new XDocument();

XElement furnitureElements = new XElement("Furnitures");

foreach (var furniture in furnitures)

{

XElement furnitureElement = new XElement("Furniture");

XElement categoryName = new XElement("Category", furniture.CategoryType);

XElement furnitureName = new XElement("Name", furniture.Name);

XElement quantity = new XElement("Quantity", furniture.PiecesLeft);

furnitureElement.Add(categoryName);

furnitureElement.Add(furnitureName);

furnitureElement.Add(quantity);

furnitureElements.Add(furnitureElement);

}

document.Add(furnitureElements);

DataSet dataset = new DataSet();

dataset.ReadXml(document.CreateReader());

DataTable table = null;

if (dataset.Tables.Count != 0)

table = dataset.Tables[0];

return table;

}

/// <summary>

/// Creates a profit by category report

/// </summary>

/// <param name="fromDate">From date</param>

/// <param name="toDate">To date</param>

/// <returns>Report table</returns>

public DataTable ProfitByCategoryReport(DateTime fromDate, DateTime toDate)

{

var records = saleDataService.GetAll().Where(sale => sale.Date.Date >= fromDate.Date && sale.Date.Date <= toDate.Date).SelectMany(sale => sale.Records);

var gropedRecords = records.GroupBy(record => record.Furniture.CategoryType);

XDocument document = new XDocument();

XElement categories = new XElement("Categories");

foreach (var group in gropedRecords)

{

XElement category = new XElement("Category");

XElement categoryName = new XElement("Name", group.Key.ToString());

XElement categoryProfit = new XElement("Profit", Math.Round(group.Sum(record => record.Quantity \* (record.PiecePrice - record.Furniture.PurchasePrice)),1));

category.Add(categoryName);

category.Add(categoryProfit);

categories.Add(category);

}

document.Add(categories);

DataSet dataset = new DataSet();

dataset.ReadXml(document.CreateReader());

DataTable table = null;

if (dataset.Tables.Count != 0)

table = dataset.Tables[0];

return table;

}

/// <summary>

/// Creates a saled count report

/// </summary>

/// <param name="fromDate">From date</param>

/// <param name="toDate">To date</param>

/// <returns>Report table</returns>

public DataTable SaledCountReport(DateTime fromDate, DateTime toDate)

{

var records = saleDataService.GetAll().Where(sale => sale.Date.Date >= fromDate.Date && sale.Date.Date <= toDate.Date).SelectMany(sale => sale.Records);

var gropedRecords = records.GroupBy(record => record.Furniture.CategoryType);

XDocument document = new XDocument();

XElement categories = new XElement("Categories");

foreach (var group in gropedRecords)

{

XElement category = new XElement("Category");

XElement categoryName = new XElement("Name", group.Key.ToString());

XElement categoryQuantity = new XElement("Quantity", group.Sum(record => record.Quantity));

category.Add(categoryName);

category.Add(categoryQuantity);

categories.Add(category);

}

document.Add(categories);

DataSet dataset = new DataSet();

dataset.ReadXml(document.CreateReader());

DataTable table = null;

if (dataset.Tables.Count != 0)

table = dataset.Tables[0];

return table;

}

/// <summary>

/// Saves data to file

/// </summary>

/// <param name="path">Path</param>

/// <param name="dataTable">Data</param>

public void SaveReport(string path,DataTable dataTable)

{

dataTable.WriteXml(path);

}

}

}

**SaleService.sc**

using System;

using System.Collections.Generic;

using System.Linq;

using DAL.ORM;

using BLL.Models.Products;

using BLL.Models.Sales;

using BLL.ServiceInterfaces;

namespace BLL.Services

{

/// <summary>

/// Represents sale service

/// </summary>

public class SaleService : ISaleService

{

private IDataService<Sale> saleDataService;

private IDataService<SaleRecord> saleRecordDataService;

private IDataService<Garniture> garnitureDataService;

private IDataService<GarnitureRecord> garnitureRecordDataService;

/// <param name="saleDataService">Sale data service</param>

/// <param name="saleRecordDataService">Sale record data service</param>

/// <param name="garnitureDataService">Garniture data service</param>

/// <param name="garnitureRecordDataService">Garniture record data service</param>

/// <param name="furnitureDataService">Furniture data service</param>

public SaleService(IDataService<Sale> saleDataService, IDataService<SaleRecord> saleRecordDataService, IDataService<Garniture> garnitureDataService, IDataService<GarnitureRecord> garnitureRecordDataService, IDataService<Furniture> furnitureDataService)

{

this.saleDataService = saleDataService;

this.saleRecordDataService = saleRecordDataService;

this.garnitureDataService = garnitureDataService;

this.garnitureRecordDataService = garnitureRecordDataService;

this.furnitureDataService = furnitureDataService;

}

/// <summary>

/// Adds sale

/// </summary>

/// <param name="customerName">Customer name</param>

/// <param name="date">Sale date</param>

/// <param name="products">Products list</param>

/// <returns>Sale id</returns>

public int AddSale(string customerName, DateTime date, List<IProduct> products)

{

var sale = new Sale { CustomerName = customerName, Date = date };

foreach (var product in products)

{

if (product.ProductType == ProductTypes.Garniture)

{

var garniture = garnitureDataService.GetById(((Garniture)product).Id);

garniture.TakePieces(product.PiecesLeft);

garnitureDataService.Update(garniture);

foreach (var garnitureRecord in garniture.Records)

{

furnitureDataService.Update(garnitureRecord.Furniture);

sale.Records.Add(new SaleRecord

{

PiecePrice = garnitureRecord.Furniture.RetailPrice,

Quantity = product.PiecesLeft \* garnitureRecord.Quantity,

FurnitureId = garnitureRecord.Furniture.Id,

Furniture = garnitureRecord.Furniture,

});

}

}

else

{

var furniture = furnitureDataService.GetById(((Furniture)product).Id);

furniture.TakePieces(product.PiecesLeft);

furnitureDataService.Update(furniture);

sale.Records.Add(new SaleRecord

{

PiecePrice = furniture.RetailPrice,

Quantity = product.PiecesLeft,

FurnitureId = furniture.Id,

Furniture = furniture

});

}

}

var id = saleDataService.Insert(sale);

foreach (var saleRecord in sale.Records)

{

saleRecord.SaleId = id;

saleRecordDataService.Insert(saleRecord);

}

return id;

}

/// <summary>

/// Deletes sale

/// </summary>

/// <param name="sale">Sale</param>

/// <returns>Result of operation</returns>

public bool Delete(Sale sale)

{

var garnitures = new List<Garniture>();

foreach (var record in sale.Records)

{

if (record.Furniture.GarnitureRecordId != null)

{

var garnitureRecord = garnitureRecordDataService.GetById((int)record.Furniture.GarnitureRecordId);

if (garnitures.All(t => t.Id != garnitureRecord.GarnitureId))

{

var garniture = garnitureDataService.GetById((int)garnitureRecord.GarnitureId);

garniture.PutPieces(record.Quantity / garnitureRecord.Quantity);

garnitureDataService.Update(garniture);

garnitures.Add(garniture);

}

}

else

{

record.Furniture.PutPieces(record.Quantity);

furnitureDataService.Update(record.Furniture);

}

}

return saleDataService.Delete(sale);

}

public List<Sale> GetAll()

{

return saleDataService.GetAll();

}

/// <summary>

/// Gets sale by id

/// </summary>

/// <param name="id"></param>

/// <returns>Sale</returns>

public Sale GetById(int id)

{

return saleDataService.GetById(id);

}

}

}

**UserService.cs**

using System.Collections.Generic;

using System.Linq;

using UsersFactory;

using BLL.ServiceInterfaces;

using DAL.ORM;

using BLL.Models.Users;

namespace BLL.Services

{

/// <summary>

/// Represents an user service

/// </summary>

public class UserService : IUserService

{

private IDataService<User> userDataService;

/// <summary>

/// Creates an instance of user service

/// </summary>

/// <param name="userDataService"></param>

public UserService(IDataService<User> userDataService)

{

this.userDataService = userDataService;

}

/// <summary>

/// Authorizes user

/// </summary>

/// <param name="password">Passowrd</param>

/// <param name="login">Login</param>

/// <param name="user">User</param>

/// <returns>Result of autorization</returns>

public bool Authorize(string login, string password, out User user)

{

var users = userDataService.GetAll();

user = users.Find(t => t.Password == password && t.Login == login );

return user != null;

}

/// <summary>

/// Creates new account

/// </summary>

/// <param name="role">Role</param>

/// <param name="login">Login</param>

/// <param name="password">Password</param>

/// <param name="fullName">Fullname</param>

/// <returns>Id</returns>

public int CreateAccount(Roles role, string login, string password, string fullName)

{

var id = 0;

var user = UserFactory.CreateByRole(role, login, password, fullName);

if (user != null)

{

id = userDataService.Insert(user);

}

return id;

}

/// <summary>

/// Edits account

/// </summary>

/// <param name="user">User</param>

/// <returns>Result of operation</returns>

public bool EditAccount(User user)

{

return userDataService.Update(user);

}

/// <summary>

/// Deletes account

/// </summary>

/// <param name="user">User</param>

/// <returns>Result of operation</returns>

public bool DeleteAccount(User user)

{

return userDataService.Delete(user);

}

/// <summary>

/// Checks user existing by login

/// </summary>

/// <param name="login">Login</param>

/// <returns>Result of checking user existing</returns>

public bool IsExistUser(string login)

{

return userDataService.GetAll().Any(user => user.Login == login);

}

/// <summary>

/// Gets user by id

/// </summary>

/// <param name="id">Id</param>

/// <returns>User</returns>

public User GetById(int id)

{

return userDataService.GetById(id);

}

/// <summary>

/// Gets all users

/// </summary>

/// <returns>List of users</returns>

public List<User> GetUsers()

{

return userDataService.GetAll();

}

}

}

**IDAO.cs**

using System.Collections.Generic;

namespace DAL.DAO

{

/// <summary>

/// Represents an interface for access to database table

/// </summary>

/// <typeparam name="T">Type</typeparam>

public interface IDAO<T>

{

/// <summary>

/// Inserts element to database table

/// </summary>

/// <param name="element">Element</param>

/// <returns>Inserted element id</returns>

int Insert(T element);

/// <summary>

/// Updates element in database table

/// </summary>

/// <param name="element"></param>

/// <returns>Result of operation</returns>

bool Update(T element);

/// <summary>

/// Deletes element from database table

/// </summary>

/// <param name="element">Element</param>

/// <returns>Result of operation</returns>

bool Delete(T element);

/// <summary>

/// Gets element by id

/// </summary>

/// <param name="id"></param>

/// <returns>Element</returns>

T GetById(int id);

/// <summary>

/// Gets all elements from database table

/// </summary>

/// <returns>Elements list</returns>

List<T> GetAll();

}

}

**CategoryDataService.cs**

using System.Collections.Generic;

using BLL.Models.Products;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for categories

/// </summary>

public class CategoryDataService : IDataService<Category>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of CategoryDataService class

/// </summary>

/// <param name="database">Access to database</param>

public CategoryDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts category to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Category element)

{

return database.Categories.Insert(element);

}

/// <summary>

/// Updates category in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Category element)

{

return database.Categories.Update(element);

}

/// <summary>

/// Deletes category from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Category element)

{

return database.Categories.Delete(element);

}

/// <summary>

/// Gets category by id

/// </summary>

/// <param name="id"></param>

/// <returns>Category</returns>

public Category GetById(int id)

{

return database.Categories.GetById(id);

}

/// <summary>

/// Gets categories list from database table

/// </summary>

/// <returns>Categories list</returns>

public List<Category> GetAll()

{

return database.Categories.GetAll();

}

}

}

**FurnitureDataService.cs**

using System.Collections.Generic;

using BLL.Models.Products;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for furnitures

/// </summary>

public class FurnitureDataService :IDataService<Furniture>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of FurnitureDataService class

/// </summary>

/// <param name="database">Access to database</param>

public FurnitureDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts furniture to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Furniture element)

{

return database.Furnitures.Insert(element);

}

/// <summary>

/// Updates furniture in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Furniture element)

{

return database.Furnitures.Update(element);

}

/// <summary>

/// Deletes furniture from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Furniture element)

{

return database.Furnitures.Delete(element);

}

/// <summary>

/// Gets furniture by id

/// </summary>

/// <param name="id"></param>

/// <returns>Furniture</returns>

public Furniture GetById(int id)

{

var furniture = database.Furnitures.GetById(id);

if(furniture != null)

furniture.Category = database.Categories.GetById(furniture.CategoryId);

return furniture;

}

/// <summary>

/// Gets furnitures list from database table

/// </summary>

/// <returns>Furnitures list</returns>

public List<Furniture> GetAll()

{

var furnitures = database.Furnitures.GetAll();

foreach (var furniture in furnitures)

{

if(furniture != null)

furniture.Category = database.Categories.GetById(furniture.CategoryId);

}

return furnitures;

}

}

}

**GarnitureDataService.cs**

using System.Collections.Generic;

using System.Linq;

using BLL.Models.Products;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for garnitures

/// </summary>

public class GarnitureDataService :IDataService<Garniture>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of GarnitureDataService class

/// </summary>

/// <param name="database">Access to database</param>

public GarnitureDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts garniture to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Garniture element)

{

return database.Garnitures.Insert(element);

}

/// <summary>

/// Updates garniture in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Garniture element)

{

return database.Garnitures.Update(element);

}

/// <summary>

/// Deletes garniture from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Garniture element)

{

return database.Garnitures.Delete(element);

}

/// <summary>

/// Gets garniture by id

/// </summary>

/// <param name="id"></param>

/// <returns>Garniture</returns>

public Garniture GetById(int id)

{

var garniture = database.Garnitures.GetById(id);

if (garniture != null)

{

garniture.Records.AddRange(database.GarnitureRecords.GetAll().Where(record => record.GarnitureId == garniture.Id).ToList());

var furnitures = database.Furnitures.GetAll();

var categories = new List<Category>();

foreach (var record in garniture.Records)

{

record.Furniture = furnitures.Where(furniture => furniture.GarnitureRecordId == record.Id).FirstOrDefault();

var category = categories.Find(t => t.Id == record.Furniture.CategoryId);

if (category == null)

{

category = database.Categories.GetById(record.Furniture.CategoryId);

categories.Add(category);

}

record.Furniture.Category = category;

}

}

return garniture;

}

/// <summary>

/// Gets garniture list from database table

/// </summary>

/// <returns>Garnitures list</returns>

public List<Garniture> GetAll()

{

var garnitures = database.Garnitures.GetAll();

var garnitureRecords = database.GarnitureRecords.GetAll();

var furnitures = database.Furnitures.GetAll();

var categories = new List<Category>();

foreach (var garniture in garnitures)

{

garniture.Records.AddRange(garnitureRecords.Where(record => record.GarnitureId == garniture.Id).ToList());

foreach (var record in garniture.Records)

{

record.Furniture = furnitures.Where(furniture => furniture.GarnitureRecordId == record.Id).FirstOrDefault();

var category = categories.Find(t => t.Id == record.Furniture.CategoryId);

if (category == null)

{

category = database.Categories.GetById(record.Furniture.CategoryId);

}

record.Furniture.Category = category;

}

}

return garnitures;

}

}

}

**GarnitureRecordDataService.cs**

using System.Collections.Generic;

using BLL.Models.Products;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for garniture records

/// </summary>

public class GarnitureRecordDataService : IDataService<GarnitureRecord>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of GarnitureRecordDataService class

/// </summary>

/// <param name="database">Access to database</param>

public GarnitureRecordDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts garniture record to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(GarnitureRecord element)

{

return database.GarnitureRecords.Insert(element);

}

/// <summary>

/// Updates garniture record in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(GarnitureRecord element)

{

return database.GarnitureRecords.Update(element);

}

/// <summary>

/// Deletes garniture record from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(GarnitureRecord element)

{

return database.GarnitureRecords.Delete(element);

}

/// <summary>

/// Gets garniture record by id

/// </summary>

/// <param name="id"></param>

/// <returns>Garniture record</returns>

public GarnitureRecord GetById(int id)

{

var record = database.GarnitureRecords.GetById(id);

record.Furniture = database.Furnitures.GetAll().Find(furniture => furniture.GarnitureRecordId == record.Id);

record.Furniture.Category = database.Categories.GetById(record.Furniture.CategoryId);

return record;

}

/// <summary>

/// Gets garniture records list from database table

/// </summary>

/// <returns>Garniture records list</returns>

public List<GarnitureRecord> GetAll()

{

var records = database.GarnitureRecords.GetAll();

var furnitures = database.Furnitures.GetAll();

var categories = new List<Category>();

foreach (var record in records)

{

record.Furniture = furnitures.Find(furniture => furniture.GarnitureRecordId == record.Id);

var category = categories.Find(t => t.Id == record.Furniture.CategoryId);

if (category == null)

{

category = database.Categories.GetById(record.Furniture.CategoryId);

categories.Add(category);

}

record.Furniture.Category = category;

}

return records;

}

}

}

**IAccessDB.cs**

using DAL.DAO;

using BLL.Models.Sales;

using BLL.Models.Products;

using BLL.Models.Users;

namespace DAL.ORM

{

/// <summary>

/// Represents an inteface for access to datavase tables

/// </summary>

public interface IAccessDB

{

/// <summary>

/// User dao

/// </summary>

IDAO<User> Users { get; set; }

/// <summary>

/// Sale dao

/// </summary>

IDAO<Sale> Sales { get; set; }

/// <summary>

/// Sale record dao

/// </summary>

IDAO<SaleRecord> SaleRecords { get; set; }

/// <summary>

/// Furniture dao

/// </summary>

IDAO<Furniture> Furnitures { get; set; }

/// <summary>

/// Garniture dao

/// </summary>

IDAO<Garniture> Garnitures { get; set; }

/// <summary>

/// Garniture record dao

/// </summary>

IDAO<GarnitureRecord> GarnitureRecords { get; set; }

/// <summary>

/// Category dao

/// </summary>

IDAO<Category> Categories { get; set; }

}

}

**IDataService.cs**

using System.Collections.Generic;

namespace DAL.ORM

{

/// <summary>

/// Represents an mapping interface

/// </summary>

/// <typeparam name="T">Type</typeparam>

public interface IDataService<T>

{

/// <summary>

/// Inserts element to database table

/// </summary>

/// <param name="element">Element</param>

/// <returns>Inserted element id</returns>

int Insert(T element);

/// <summary>

/// Updates element in database table

/// </summary>

/// <param name="element"></param>

/// <returns>Result of operation</returns>

bool Update(T element);

/// <summary>

/// Deletes element from database table

/// </summary>

/// <param name="element">Element</param>

/// <returns>Result of operation</returns>

bool Delete(T element);

/// <summary>

/// Gets element by id

/// </summary>

/// <param name="id"></param>

/// <returns>Element</returns>

T GetById(int id);

/// <summary>

/// Gets all elements from database table

/// </summary>

/// <returns>Elements list</returns>

List<T> GetAll();

}

}

**SaleDataService.cs**

using System.Collections.Generic;

using System.Linq;

using BLL.Models.Sales;

using BLL.Models.Products;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for sales

/// </summary>

public class SaleDataService :IDataService<Sale>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of SaleDataService class

/// </summary>

/// <param name="database">Access to database</param>

public SaleDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts sale to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Sale element)

{

return database.Sales.Insert(element);

}

/// <summary>

/// Updates sale in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Sale element)

{

return database.Sales.Update(element);

}

/// <summary>

/// Deletes sale from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Sale element)

{

return database.Sales.Delete(element);

}

/// <summary>

/// Gets sale by id

/// </summary>

/// <param name="id"></param>

/// <returns>Sale</returns>

public Sale GetById(int id)

{

var sale = database.Sales.GetById(id);

if (sale != null)

{

sale.Records.AddRange(database.SaleRecords.GetAll().Where(record => record.SaleId == sale.Id));

var furnitures = new List<Furniture>();

var categories = new List<Category>();

foreach (var record in sale.Records)

{

var furniture = furnitures.Find(t => t.Id == record.FurnitureId);

if (furniture == null)

{

furniture = database.Furnitures.GetById(record.FurnitureId);

furnitures.Add(furniture);

}

record.Furniture = furniture;

var category = categories.Find(t => t.Id == record.Furniture.CategoryId);

if (category == null)

{

category = database.Categories.GetById(record.Furniture.CategoryId);

}

record.Furniture.Category = category;

}

}

return sale;

}

/// <summary>

/// Gets sales list from database table

/// </summary>

/// <returns>Sales list</returns>

public List<Sale> GetAll()

{

var sales = database.Sales.GetAll();

var saleRecords = database.SaleRecords.GetAll();

var furnitures = new List<Furniture>();

var categories = new List<Category>();

foreach (var sale in sales)

{

sale.Records.AddRange(saleRecords.Where(record => record.SaleId == sale.Id));

foreach (var record in sale.Records)

{

var furniture = furnitures.Find(t => t.Id == record.FurnitureId);

if (furniture == null)

{

furniture = database.Furnitures.GetById(record.FurnitureId);

furnitures.Add(furniture);

}

record.Furniture = furniture;

var category = categories.Find(t => t.Id == record.Furniture.CategoryId);

if (category == null)

{

category = database.Categories.GetById(record.Furniture.CategoryId);

}

record.Furniture.Category = category;

}

}

return sales;

}

}

}

**SaleRecordDataService.cs**

using System.Collections.Generic;

using BLL.Models.Sales;

using BLL.Models.Products;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for sale records

/// </summary>

public class SaleRecordDataService : IDataService<SaleRecord>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of SaleRecordDataService class

/// </summary>

/// <param name="database">Access to database</param>

public SaleRecordDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts sale record to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(SaleRecord element)

{

return database.SaleRecords.Insert(element);

}

/// <summary>

/// Updates sale record in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(SaleRecord element)

{

return database.SaleRecords.Update(element);

}

/// <summary>

/// Deletes sale record from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(SaleRecord element)

{

return database.SaleRecords.Delete(element);

}

/// <summary>

/// Gets sale record by id

/// </summary>

/// <param name="id"></param>

/// <returns>Sale record</returns>

public SaleRecord GetById(int id)

{

var record = database.SaleRecords.GetById(id);

record.Furniture = database.Furnitures.GetById(record.FurnitureId);

record.Furniture.Category = database.Categories.GetById(record.Furniture.CategoryId);

return record;

}

/// <summary>

/// Gets sale records list from database table

/// </summary>

/// <returns>Sale records list</returns>

public List<SaleRecord> GetAll()

{

var records = database.SaleRecords.GetAll();

var categories = new List<Category>();

var furnitures = new List<Furniture>();

foreach (var record in records)

{

var furniture = furnitures.Find(t => t.Id == record.FurnitureId);

if (furniture == null)

{

furniture = database.Furnitures.GetById(record.FurnitureId);

furnitures.Add(record.Furniture);

}

record.Furniture = furniture;

var category = categories.Find(t => t.Id == record.Furniture.CategoryId);

if (category == null)

{

category = database.Categories.GetById(record.Furniture.CategoryId);

categories.Add(record.Furniture.Category);

}

record.Furniture.Category = category;

}

return records;

}

}

}

**UserDataService.cs**

using System.Collections.Generic;

using BLL.Models.Users;

namespace DAL.ORM

{

/// <summary>

/// Provides mapping for users

/// </summary>

public class UserDataService :IDataService<User>

{

private IAccessDB database;

/// <summary>

/// Creates an instance of UserDataService class

/// </summary>

/// <param name="database">Access to database</param>

public UserDataService(IAccessDB database)

{

this.database = database;

}

/// <summary>

/// Inserts user to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(User element)

{

return database.Users.Insert(element);

}

/// <summary>

/// Updates user in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(User element)

{

return database.Users.Update(element);

}

/// <summary>

/// Deletes user from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(User element)

{

return database.Users.Delete(element);

}

/// <summary>

/// Gets user by id

/// </summary>

/// <param name="id"></param>

/// <returns>User</returns>

public User GetById(int id)

{

return database.Users.GetById(id);

}

/// <summary>

/// Gets users list from database table

/// </summary>

/// <returns>Users list</returns>

public List<User> GetAll()

{

return database.Users.GetAll();

}

}

}

**CategoryDAO.cs**

using System.Collections.Generic;

using MySql.Data.MySqlClient;

using DAL.DAO;

using BLL.Models.Products;

using BLL.Models.Users;

namespace DAL.ProcessingMySQL

{

/// <summary>

/// Provides methods for access to categories table in database

/// </summary>

public class CategoryDAO : IDAO<Category>

{

private string idColumn = "Id";

private string categoryColumn = "CategoryType";

private string markUpColumn = "MarkUp";

private Roles role;

/// <summary>

/// Creates dao instance

/// </summary>

/// <param name="role">User role</param>

public CategoryDAO(Roles role)

{

this.role = role;

}

/// <summary>

/// Inserts category to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Category element)

{

var id = 0;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"INSERT Categories ({categoryColumn},{markUpColumn}) VALUES (@{categoryColumn},@{markUpColumn})";

query.Parameters.AddWithValue($"@{categoryColumn}", element.CategoryType);

query.Parameters.AddWithValue($"@{markUpColumn}", element.MarkUp);

query.ExecuteNonQuery();

id = (int)query.LastInsertedId;

}

catch

{

}

return id;

}

/// <summary>

/// Updates category in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Category element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"UPDATE Categories SET {categoryColumn}=@{categoryColumn},{markUpColumn}=@{markUpColumn} WHERE Id='{element.Id}'";

query.Parameters.AddWithValue($"@{categoryColumn}", element.CategoryType);

query.Parameters.AddWithValue($"@{markUpColumn}", element.MarkUp);

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Deletes category from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Category element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"DELETE FROM Categories WHERE Id='{element.Id}'";

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Gets category by id

/// </summary>

/// <param name="id"></param>

/// <returns>Category</returns>

public Category GetById(int id)

{

Category category = null;

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {idColumn},{categoryColumn},{markUpColumn} FROM Categories WHERE Id='{id}'";

reader = query.ExecuteReader();

if (reader.HasRows)

{

var values = new List<object>();

reader.Read();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

category = new Category((int)values[0], (Categories)values[1], (double)values[2]);

}

}

catch

{

}

finally

{

reader?.Close();

}

return category;

}

/// <summary>

/// Gets categories list from database table

/// </summary>

/// <returns>Categories list</returns>

public List<Category> GetAll()

{

var categories = new List<Category>();

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {idColumn},{categoryColumn},{markUpColumn} FROM Categories";

reader = query.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var values = new List<object>();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

var category = new Category((int)values[0], (Categories)values[1], (double)values[2]);

if (category != null)

{

categories.Add(category);

}

}

}

}

catch

{

}

finally

{

reader?.Close();

}

return categories;

}

}

}

**ConnectionSingleton.cs**

using System;

using BLL.Models.Users;

using MySql.Data.MySqlClient;

using System.Configuration;

namespace DAL.ProcessingMySQL

{

/// <summary>

/// Provides connection to MySql database

/// </summary>

public sealed class ConnectionSingleton :IDisposable

{

private static readonly Lazy<ConnectionSingleton> instance = new Lazy<ConnectionSingleton>(() => new ConnectionSingleton());

private readonly MySqlConnection adminConnection = new MySqlConnection(ConfigurationManager.ConnectionStrings["administrator"].ConnectionString);

private readonly MySqlConnection managerConnection = new MySqlConnection(ConfigurationManager.ConnectionStrings["manager"].ConnectionString);

private readonly MySqlConnection sellerConnection = new MySqlConnection(ConfigurationManager.ConnectionStrings["seller"].ConnectionString);

private readonly MySqlConnection guestConnection = new MySqlConnection(ConfigurationManager.ConnectionStrings["guest"].ConnectionString);

/// <summary>

/// Finalizes connection singleton

/// </summary>

~ConnectionSingleton()

{

Dispose();

}

private ConnectionSingleton()

{

}

/// <summary>

/// Gets instance of ConnectionSingleton class

/// </summary>

public static ConnectionSingleton Instance

{

get

{

return instance.Value;

}

}

/// <summary>

/// Gets connection by user role

/// </summary>

/// <param name="role">User role</param>

/// <returns>connection to database</returns>

public MySqlConnection GetDBConnection(Roles role)

{

switch (role)

{

case Roles.Administrator:

if(adminConnection.State == System.Data.ConnectionState.Closed)

adminConnection.Open();

return adminConnection;

case Roles.Manager:

if (managerConnection.State == System.Data.ConnectionState.Closed)

managerConnection.Open();

return managerConnection;

case Roles.Seller:

if (sellerConnection.State == System.Data.ConnectionState.Closed)

sellerConnection.Open();

return sellerConnection;

default:

if (guestConnection.State == System.Data.ConnectionState.Closed)

guestConnection.Open();

return guestConnection;

}

}

/// <summary>

/// Closes all connections

/// </summary>

public void Dispose()

{

adminConnection.Close();

managerConnection.Close();

sellerConnection.Close();

guestConnection.Close();

}

}

}

**FurnitureDAO.cs**

using System;

using System.Collections.Generic;

using MySql.Data.MySqlClient;

using DAL.DAO;

using BLL.Models.Users;

using BLL.Models.Products;

using FurnitureFactories;

namespace DAL.ProcessingMySQL

{

/// <summary>

/// Provides methods for access to furnitures table in database

/// </summary>

public class FurnitureDAO : IDAO<Furniture>

{

private string idColumn = "Id";

private string categoryTypeColumn = "CategoryType";

private string furnitureTypeColumn = "FurnitureType";

private string nameColumn = "Name";

private string leftColumn = "PiecesLeft";

private string priceColumn = "PurchasePrice";

private string categoryIdColumn = "CategoryId";

private string garnitureRecordIdColumn = "GarnitureRecordId";

private Roles role;

/// <summary>

/// Creates dao instance

/// </summary>

/// <param name="role">User role</param>

public FurnitureDAO(Roles role)

{

this.role = role;

}

/// <summary>

/// Inserts furniture to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Furniture element)

{

var id = 0;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"INSERT Furnitures ({categoryTypeColumn},{furnitureTypeColumn},{nameColumn},{leftColumn}," +

$"{priceColumn},{categoryIdColumn},{garnitureRecordIdColumn}) VALUES (@{categoryTypeColumn},@{furnitureTypeColumn}," +

$"@{nameColumn},@{leftColumn},@{priceColumn},@{categoryIdColumn},@{garnitureRecordIdColumn})";

query.Parameters.AddWithValue($"@{categoryTypeColumn}", element.CategoryType);

query.Parameters.AddWithValue($"@{furnitureTypeColumn}", element.FurnitureType);

query.Parameters.AddWithValue($"@{nameColumn}", element.Name);

query.Parameters.AddWithValue($"@{leftColumn}", element.PiecesLeft);

query.Parameters.AddWithValue($"@{priceColumn}", element.PurchasePrice);

query.Parameters.AddWithValue($"@{categoryIdColumn}", element.CategoryId);

query.Parameters.AddWithValue($"@{garnitureRecordIdColumn}", element.GarnitureRecordId);

query.ExecuteNonQuery();

id = (int)query.LastInsertedId;

}

catch

{

}

return id;

}

/// <summary>

/// Updates furniture in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Furniture element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"UPDATE Furnitures SET {categoryTypeColumn}=@{categoryTypeColumn},{furnitureTypeColumn}=@{furnitureTypeColumn}," +

$"{nameColumn}=@{nameColumn},{leftColumn}=@{leftColumn},{priceColumn}=@{priceColumn}," +

$"{categoryIdColumn}=@{categoryIdColumn},{garnitureRecordIdColumn}=@{garnitureRecordIdColumn} WHERE Id='{element.Id}'";

query.Parameters.AddWithValue($"@{categoryTypeColumn}", element.CategoryType);

query.Parameters.AddWithValue($"@{furnitureTypeColumn}", element.FurnitureType);

query.Parameters.AddWithValue($"@{nameColumn}", element.Name);

query.Parameters.AddWithValue($"@{leftColumn}", element.PiecesLeft);

query.Parameters.AddWithValue($"@{priceColumn}", element.PurchasePrice);

query.Parameters.AddWithValue($"@{categoryIdColumn}", element.CategoryId);

query.Parameters.AddWithValue($"@{garnitureRecordIdColumn}", element.GarnitureRecordId);

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Deletes furniture from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Furniture element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"DELETE FROM Furnitures WHERE Id='{element.Id}'";

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Gets furniture by id

/// </summary>

/// <param name="id"></param>

/// <returns>Furniture</returns>

public Furniture GetById(int id)

{

Furniture furniture = null;

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {categoryTypeColumn},{nameColumn},{leftColumn},{priceColumn},{furnitureTypeColumn}," +

$"{categoryIdColumn},{idColumn},{garnitureRecordIdColumn} FROM Furnitures WHERE Id='{id}'";

reader = query.ExecuteReader();

if (reader.HasRows)

{

var values = new List<object>();

reader.Read();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

var furnitureType = (FurnitureTypes)values[4];

furniture = FurnitureFactory.CreateFurniture((Categories)values[0],

(string)values[1], (int)values[2], (double)values[3], furnitureType,(int)values[5],(int)values[6]);

if (furniture != null)

{

if (furnitureType == FurnitureTypes.Garniture)

furniture.GarnitureRecordId = (int)values[7];

}

}

}

finally

{

reader?.Close();

}

return furniture;

}

/// <summary>

/// Gets furnitures list from database table

/// </summary>

/// <returns>Furnitures list</returns>

public List<Furniture> GetAll()

{

var furnitureList = new List<Furniture>();

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {categoryTypeColumn},{nameColumn},{leftColumn},{priceColumn},{furnitureTypeColumn}," +

$"{categoryIdColumn},{idColumn},{garnitureRecordIdColumn} FROM Furnitures";

reader = query.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var values = new List<object>();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

var furnitureType = (FurnitureTypes)values[4];

var furniture = FurnitureFactory.CreateFurniture((Categories)values[0],

(string)values[1], (int)values[2], (double)values[3], furnitureType, (int)values[5], (int)values[6]);

if (furniture != null)

{

if (furnitureType == FurnitureTypes.Garniture)

furniture.GarnitureRecordId = (int)values[7];

furnitureList.Add(furniture);

}

}

}

}

finally

{

reader?.Close();

}

return furnitureList;

}

}

}

**GarnitureDAO.cs**

using System.Collections.Generic;

using MySql.Data.MySqlClient;

using BLL.Models.Products;

using BLL.Models.Users;

using DAL.DAO;

namespace DAL.ProcessingMySQL

{

/// <summary>

/// Provides methods for access to garnitures table in database

/// </summary>

public class GarnitureDAO : IDAO<Garniture>

{

private string idColumn = "Id";

private string nameColumn = "Name";

private string leftColumn = "PiecesLeft";

private Roles role;

/// <summary>

/// Creates dao instance

/// </summary>

/// <param name="role">User role</param>

public GarnitureDAO(Roles role)

{

this.role = role;

}

/// <summary>

/// Inserts garniture to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(Garniture element)

{

var id = 0;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"INSERT Garnitures ({nameColumn},{leftColumn}) VALUES " +

$"(@{nameColumn},@{leftColumn})";

query.Parameters.AddWithValue($"@{nameColumn}", element.Name);

query.Parameters.AddWithValue($"@{leftColumn}", element.PiecesLeft);

query.ExecuteNonQuery();

id = (int)query.LastInsertedId;

}

catch

{

}

return id;

}

/// <summary>

/// Updates garniture in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(Garniture element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"UPDATE Garnitures SET {nameColumn}=@{nameColumn},{leftColumn}=@{leftColumn} WHERE Id='{element.Id}'";

query.Parameters.AddWithValue($"@{nameColumn}", element.Name);

query.Parameters.AddWithValue($"@{leftColumn}", element.PiecesLeft);

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Deletes garniture from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(Garniture element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"DELETE FROM Garnitures WHERE Id='{element.Id}'";

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Gets garniture by id

/// </summary>

/// <param name="id"></param>

/// <returns>Garniture</returns>

public Garniture GetById(int id)

{

Garniture garniture = null;

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {idColumn},{nameColumn},{leftColumn} FROM Garnitures WHERE Id='{id}'";

reader = query.ExecuteReader();

if (reader.HasRows)

{

var values = new List<object>();

reader.Read();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

garniture = new Garniture((int)values[0], (string)values[1], (int)values[2]);

}

}

finally

{

reader.Close();

}

return garniture;

}

/// <summary>

/// Gets garniture list from database table

/// </summary>

/// <returns>Garnitures list</returns>

public List<Garniture> GetAll()

{

var garnitures = new List<Garniture>();

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {idColumn},{nameColumn},{leftColumn} FROM Garnitures";

reader = query.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var values = new List<object>();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

var garniture = new Garniture((int)values[0], (string)values[1], (int)values[2]);

if (garniture != null)

{

garnitures.Add(garniture);

}

}

}

}

finally

{

reader.Close();

}

return garnitures;

}

}

}

**GarnitureRecordDAO.cs**

using System.Collections.Generic;

using MySql.Data.MySqlClient;

using DAL.DAO;

using BLL.Models.Users;

using BLL.Models.Products;

namespace DAL.ProcessingMySQL

{

/// <summary>

/// Provides methods for access to garniture records table in database

/// </summary>

public class GarnitureRecordDAO : IDAO<GarnitureRecord>

{

private string idColumn = "Id";

private string quantityColumn = "Quantity";

private string garnitureIdColumn = "GarnitureId";

private Roles role

/// <summary>

/// Creates dao instance

/// </summary>

/// <param name="role">User role</param>

public GarnitureRecordDAO(Roles role)

{

this.role = role;

}

/// <summary>

/// Inserts garniture record to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(GarnitureRecord element)

{

var id = 0;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"INSERT GarnitureRecords ({quantityColumn},{garnitureIdColumn}) VALUES " +

$"(@{quantityColumn},@{garnitureIdColumn})";

query.Parameters.AddWithValue($"@{quantityColumn}", element.Quantity);

query.Parameters.AddWithValue($"@{garnitureIdColumn}", element.GarnitureId);

query.ExecuteNonQuery();

id = (int)query.LastInsertedId;

}

catch

{

}

return id;

}

/// <summary>

/// Updates garniture record in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(GarnitureRecord element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"UPDATE GarnitureRecords SET {quantityColumn}=@{quantityColumn},{garnitureIdColumn}=@{garnitureIdColumn} WHERE Id='{element.Id}'";

query.Parameters.AddWithValue($"@{quantityColumn}", element.Quantity);

query.Parameters.AddWithValue($"@{garnitureIdColumn}", element.GarnitureId);

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Deletes garniture record from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(GarnitureRecord element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"DELETE FROM GarnitureRecords WHERE Id='{element.Id}'";

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Gets garniture record by id

/// </summary>

/// <param name="id"></param>

/// <returns>Garniture record</returns>

public GarnitureRecord GetById(int id)

{

GarnitureRecord record = null;

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {idColumn},{quantityColumn},{garnitureIdColumn} FROM GarnitureRecords WHERE Id='{id}'";

reader = query.ExecuteReader();

if (reader.HasRows)

{

var values = new List<object>();

reader.Read();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

record = new GarnitureRecord((int)values[0], (int)values[1], (int)values[2]);

}

}

finally

{

reader.Close();

}

return record;

}

/// <summary>

/// Gets garniture records list from database table

/// </summary>

/// <returns>Garniture records list</returns>

public List<GarnitureRecord> GetAll()

{

var records = new List<GarnitureRecord>();

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {idColumn},{quantityColumn},{garnitureIdColumn} FROM GarnitureRecords";

reader = query.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var values = new List<object>();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

var record = new GarnitureRecord((int)values[0], (int)values[1], (int)values[2]);

if (record != null)

{

records.Add(record);

}

}

}

}

finally

{

reader.Close();

}

return records;

}

}

}

**UserDAO.cs**

using System.Collections.Generic;

using MySql.Data.MySqlClient;

using DAL.DAO;

using BLL.Models.Users;

using UsersFactory;

namespace DAL.ProcessingMySQL

{

/// <summary>

/// Provides methods for access to users table in database

/// </summary>

public class UserDAO : IDAO<User>

{

private string idColumn = "Id";

private string roleColumn = "Role";

private string loginColumn = "Login";

private string passwordColumn = "Password";

private string fullnameColumn = "FullName";

private Roles role;

/// <summary>

/// Creates dao instance

/// </summary>

/// <param name="role">User role</param>

public UserDAO(Roles role)

{

this.role = role;

}

/// <summary>

/// Inserts user to database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Inserted category id</returns>

public int Insert(User element)

{

var id = 0;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"INSERT Users ({roleColumn},{loginColumn},{passwordColumn},{fullnameColumn}) VALUES " +

$"(@{roleColumn},@{loginColumn},@{passwordColumn},@{fullnameColumn});";

query.Parameters.AddWithValue($"@{roleColumn}", element.Role);

query.Parameters.AddWithValue($"@{loginColumn}", element.Login);

query.Parameters.AddWithValue($"@{passwordColumn}", element.Password);

query.Parameters.AddWithValue($"@{fullnameColumn}", element.FullName);

query.ExecuteNonQuery();

id = (int)query.LastInsertedId;

}

catch

{

}

return id;

}

/// <summary>

/// Updates user in database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Update(User element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"UPDATE Users SET {roleColumn}=@{roleColumn},{loginColumn}=@{loginColumn}," +

$"{passwordColumn}=@{passwordColumn},{fullnameColumn} =@{fullnameColumn} WHERE Id='{element.Id}'";

query.Parameters.AddWithValue($"@{roleColumn}", element.Role);

query.Parameters.AddWithValue($"@{loginColumn}", element.Login);

query.Parameters.AddWithValue($"@{passwordColumn}", element.Password);

query.Parameters.AddWithValue($"@{fullnameColumn}", element.FullName);

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Deletes user from database table

/// </summary>

/// <param name="element">Category</param>

/// <returns>Result of operation</returns>

public bool Delete(User element)

{

var operationResult = true;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"DELETE FROM Users WHERE Id='{element.Id}'";

query.ExecuteNonQuery();

}

catch

{

operationResult = false;

}

return operationResult;

}

/// <summary>

/// Gets user by id

/// </summary>

/// <param name="id"></param>

/// <returns>User</returns>

public User GetById(int id)

{

User user = null;

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {roleColumn},{loginColumn},{passwordColumn},{fullnameColumn},{idColumn} FROM Users WHERE Id='{id}'";

reader = query.ExecuteReader();

if (reader.HasRows)

{

var values = new List<object>();

reader.Read();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

user = UserFactory.CreateByRole((Roles)values[0], (string)values[1], (string)values[2],(string)values[3],(int)values[4]);

}

}

finally

{

reader.Close();

}

return user;

}

/// <summary>

/// Gets users list from database table

/// </summary>

/// <returns>Users list</returns>

public List<User> GetAll()

{

var users = new List<User>();

MySqlDataReader reader = null;

try

{

var connection = ConnectionSingleton.Instance.GetDBConnection(role);

MySqlCommand query = connection.CreateCommand();

query.CommandText = $"SELECT {roleColumn},{loginColumn},{passwordColumn},{fullnameColumn},{idColumn} FROM Users";

reader = query.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

var values = new List<object>();

for (var i = 0; i < reader.FieldCount; i++)

{

values.Add(reader.GetValue(i));

}

var user = UserFactory.CreateByRole((Roles)values[0], (string)values[1], (string)values[2],(string)values[3],(int)values[4]);

if (user != null)

{

users.Add(user);

}

}

}

}

finally

{

reader.Close();

}

return users;

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Иерархическая схема классов**

# ПРИЛОЖЕНИЕ В

(обязательное)

**Руководство системного программиста**

Разработанное приложение предназначено для автоматизации учета продаж для мебельного магазина с возможность вывода информации о всех существующих товарах. Следует отметит тот факт, что приложение разработано для использования тремя видами пользователей: администратором, продавец и менеджер, а данные считываются с СУБД *MySQL*.

Данное приложение позволяет продавцу добавлять данные о продажах, менеджер просматривать информацию о товарах и просматривать отчеты, а пользователю с правами администратора управлять данными: удалять, добавлять, редактировать соответственно и формировать отчеты.

Минимальные системные требования:

* операционная система семейства *Windows*: *Windows* 10;
* не менее 256 мегабайт свободного места на жестком диске;
* не менее 512 мегабайт оперативной памяти;

Для полноценного функционирования программы необходимо соответствие минимальным системным требованиям и наличие всех файлов приложения, расположенных в одном любом каталоге.

Для стандартного запуска программы необходимо запустить исполняемый файл *CourseProject*. После запуска исполняемого файла и ожидания в течении нескольких секунд появится графическое окно программы. В том случае, если по некоторым причинам окно не отобразилось, то следует убедиться в целостности данных и наличии всех подключенных библиотек, а после и вовсе перезапустить приложение. Если ошибки не исчезли, то следует непосредственно обратиться к разработчику программы или поддерживающего на данный момент ее программисту.

В том случае, если запуск приложения прошел удачно, то работоспособность приложения можно проверить непосредственно при авторизации на модульных тестах приложения. При проверке авторизации, а позже и всего функционала приложения, следует использовать данные пользователя с правами администратора, которые уже изначально прописаны в системе: логин – «admin» и пароль – «admin».

# ПРИЛОЖЕНИЕ Г

(обязательное)

**Руководство программиста**

Разработанное приложение обеспечивает доступ к данным, которые хранятся в таблицах СУБД *MySQL*. При использовании программы, в зависимости от уровня доступа, пользователю доступны функции просмотра товара, добавление распродаж, создание отчетов, управление данными: создание нового товара, удаления и редактирования уже существующих. Данное приложение используется для автоматизации учета продажи товаров в мебельном магазине.

Существует многопользовательский доступ к системе: администратор, продавец и менеджер. Администратор имеет полные права доступа к всему функционалу системы, продавец может добавлять распродажи, а менеджер может просматривать информацию о товарах и просматривать отчеты.

Контроль каждого действия пользователя осуществляется с помощью разработанной системы информирования. Так, в случае неверного ввода данных, система оповестит пользователя об этом.

Следует отметить, что входные и выходные данные приложения – данные, полученные путем считывания из СУБД *MySQL*.

Доступ к данным осуществляется при помощи технологии *ADO.NET* и *LINQ*.

Ограничения пользовательского функционала при вводе данных с обобщенными полями приводит к необходимости использования выпадающих списков значений для тех или иных полей и графических элементов, что снижает или полностью исключает риск некорректного ввода.

# ПРИЛОЖЕНИЕ Д

(обязательное)

**Руководство пользователя**

Для использования созданного приложения необходимы некоторые требования к пользователю:

* минимальные навыки работы с компьютером;
* полноценное понимание предметной области приложения;
* ознакомление с минимальными системными требования к приложению;
* наличие всего пакета документов приложения, расположенного в одном каталоге.

Программа не требует установки дополнительных программных продуктов. Для работы пользователю необходимо запустить исполняемый файл разработанного приложения. Если все инструкции соблюдены, и приложение не выдает никаких сообщений об ошибках, следовательно, программа работает исправно.

Контроль каждого действия пользователя осуществляется с помощью разработанной системы информирования. Так, в случае неверного ввода данных, система оповестит пользователя об этом.

Чтобы просмотреть все товары, редактировать записи, необходимо авторизоваться в систему как администратор. Если пользователь является администратором, то ему предоставлен следующий функционал:

* управление записями: добавление, редактирование и удаление;
* формирование отчетов.

В том случае, если пользователь является продовцом, то ему предоставлен следующий функционал:

* вносить данных о продажах.