**МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ**

**Государственное бюджетное профессиональное образовательное учреждение Московской области «Люберецкий техникум имени Героя Советского Союза, летчика-космонавта Ю.А.Гагарина»**

**ОТЧЕТ**

**ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ**

Демченкова Вадима Сергеевича

(Фамилия, имя, отчество студента)

по профессиональному модулю

**ПМ. 01 Разработка модулей программного обеспечения для компьютерных систем**

**ПМ. 02 Осуществление интеграции программных модулей**

Специальность

09.02.07 "Информационные системы и программирование"

Код, название

Курс 4 Группа № 195ИС

Период практической подготовки

с «16» февраля 2023 г. по «12» апреля 2023г.

Руководитель практической подготовки

от техникума преподаватель Жирнова Ю. В. \_\_\_\_\_\_\_\_\_\_

*должность, ФИО*

Руководитель практической подготовки

от организации директор по продажам Феоктистов А.А \_\_\_\_\_\_\_\_\_\_

*должность, ФИО*

Люберцы 2023

**ХАРАКТЕРИСТИКА ПРАКТИКАНТА**

Демченкова Вадима Сергеевича

(Ф.И.О. практиканта)

Работал в отделе продаж, разработчик WPF приложений

(подразделение , должность , сроки работы)

с «16» февраля 2023 г. по «12» апреля 2023 г.

Количество выходов на работу 39 дней.

Пропущено дней 0, из них по неуважительной причине 0

Прошел производственную (преддипломную) практическую подготовку по специальности

09.02.07 "Информационные системы и программирование"

Качество выполнения работы

За время прохождения производственной практической подготовки показал себя с положительной стороны. В течение всего периода Демченков Вадим Сергеевич внимательно и ответственно относился к выполняемой работе.

В отношениях с коллегами Демченков Вадим Сергеевич проявил себя с лучшей стороны: внимательность, умение выслушать и понять, стремление избежать конфликта, устойчивость к стрессам.

Особенно хочется отметить умение грамотно планировать свою деятельность в соответствии со стратегией развития коллектива и выполнение работы с максимальной эффективностью. Руководство организации оценивает работу Демченкова Вадима Сергеевича положительно.

Руководитель практической подготовки от организации

Директор по продажам Феоктистов Антон Андреевич

*(должность, ФИО)*

М.П.

**Описание организации**

* 1. ООО «Актио Рус»
* 2.Адрес: Московская область Дзержинское шоссе, дом 4/2
* 3.Адрес для писем: Московская область, Дзержинское шоссе, дом 4/2
* 4.8 (494) 545 00 54
* 5. Основан в 1998 году
* 6. Оптовая торговля прочими машинами и оборудованием
* 7. ООО "АКТИО РУС" образована в 1998 году для вида деятельности
* Торговля оптовая прочими машинами и оборудованием. Компания
* работает 18 лет на данном рынке. Руководителем является: Магнат Лев
* Викторович
* ОКПО 73506925, зарегистрировано 18.06.2004 находится по адресу
* 140053, Московская обл, г. Котельники, ш. Дзержинское, д. 4/2. Статус:
* Действующее с 18.06.1998.
* Компания работает 24 года и 6 месяцев, с 18 июня 1998 по настоящее
* время. Основной вид деятельности ООО "АКТИО РУС" - Торговля
* оптовая прочими машинами и оборудованием.
* 8. Компания осуществляет поставки запасных частей, дополнительного
* оборудования и расходных материалов от крупнейших мировых
* производителей: Case, Liebherr, Putzmeister, Grove, Bobcat, Bauer, Komatsu,
* Kato, Caterpillar, JCB.
* 9. продажа строительной техники Case: минипогрузчики, экскаваторы-
* погрузчики, гусеничные и колесные экскаваторы
* продажа запасных частей и комплектующих для иностранной дорожно-
* строительной техники
* продажа запасных частей и комплектующих для сельскохозяйственной
* техники Case IH, New Holland, Jone Deere, FENDT, CLAAS, KRONE и
* других производителей
* сервисное обслуживание и ремонт техники Case
* сервисное обслуживание и ремонт землеройных машин Liebherr

**Описание подразделения,**

**в котором была пройдена практика**

* 8-905-344-31-23 аctiorus@mail.ru
* Руководство персоналом;
* Контроль за выполнением задач и планов производства;
* Выпуск изделий высокого качества;
* Контроль состояния производственного оборудования;
* Разработка мероприятий по повышению производительности работ;
* Внедрение современного технологичного оборудования.
* сервисное обслуживание и ремонт техники Case
* сервисное обслуживание и ремонт землеройных машин Liebherr
* Резюме руководителя подразделения
* Высшее образование, опыт работы больше 15 лет

**Введение**

[Описание предметной области 5](#_Toc131853406)

[Постановка задачи 9](#_Toc131853407)

[Создание ER-диаграммы 10](#_Toc131853408)

[User Story 13](#_Toc131853409)

[Use case 14](#_Toc131853410)

[Разработка базы данных 16](#_Toc131853411)

[Разработка WPF приложения 20](#_Toc131853412)

[Код приложения 30](#_Toc131853413)

[Страница авторизации 30](#_Toc131853414)

[Вывод данных в DataGrid 39](#_Toc131853415)

[ListView 46](#_Toc131853416)

[Корзина 55](#_Toc131853417)

[Заключение 57](#_Toc131853418)

[Список литературы 59](#_Toc131853419)

# **Описание предметной области**

Предприятие, занимающееся продажей погрузчиков, гусеничных и колесных экскаваторов, а также строительной техники, выполняет следующие основные действия:

1. Выбор поставщиков и закупка техники. Предприятие ищет надежных поставщиков, с которыми заключает договоры на поставку погрузчиков, экскаваторов и другой строительной техники. Закупка может происходить как на территории страны, так и за рубежом.
2. Хранение и обработка техники. При получении техники на складе предприятия производится ее обработка, инспекция и тестирование, чтобы убедиться в соответствии со стандартами и спецификациями. Также на складе осуществляется хранение и учет всей имеющейся техники.
3. Продажа техники. Предприятие продает технику клиентам, которые могут быть как физическими лицами, так и компаниями. Продажа может осуществляться как в розницу, так и оптом, с привлечением финансовых инструментов, таких как лизинг и кредит.
4. Постпродажное обслуживание и ремонт техники. Предприятие оказывает своим клиентам услуги по постпродажному обслуживанию и ремонту техники. Это может включать в себя гарантийное и постгарантийное обслуживание, установку дополнительного оборудования и запчастей, а также профилактическое техническое обслуживание и ремонт в случае поломки.
5. Обучение и консультации. Предприятие может оказывать услуги по обучению и консультированию своих клиентов, чтобы помочь им правильно использовать и ухаживать за приобретенной техникой. Это может включать в себя обучение владельцев техники и их операторов, а также консультации по выбору и эксплуатации техники.

WPF приложение для предприятия по продаже погрузчиков и деталей для них должно включать в себя следующие компоненты:

1. Главное окно приложения. Это окно должно содержать элементы управления для навигации по различным функциям приложения, таким как поиск деталей, отображение информации о заказах и клиентах, и т.д.
2. Форма поиска деталей. Эта форма должна позволять пользователям искать детали по различным параметрам, таким как наименование детали, производитель, модель погрузчика, и т.д. После того как детали найдены, должна отображаться информация о них, такая как цена, наличие на складе и т.д.
3. Форма оформления заказа. Пользователи должны иметь возможность оформлять заказы на детали, используя данную форму. Форма должна позволять пользователю выбирать детали, указывать количество, указывать данные клиента и выбирать способ оплаты. После оформления заказа должна отображаться информация о заказе, такая как номер заказа и сумма заказа.
4. Форма управления заказами. Эта форма должна позволять пользователям просматривать информацию о заказах, изменять статусы заказов (например, отменить заказ) и генерировать отчеты о заказах.
5. Форма управления клиентами. Данная форма должна позволять пользователям просматривать информацию о клиентах, редактировать их данные и добавлять новых клиентов.
6. Форма управления складом. Данная форма должна позволять пользователям просматривать информацию о наличии деталей на складе, изменять количество деталей и добавлять новые детали на склад.
7. Форма отчетов. Данная форма должна позволять пользователям генерировать различные отчеты о продажах, заказах, клиентах и т.д.
8. Меню и панель инструментов. Данные элементы должны содержать функции навигации и быстрый доступ к основным функциям приложения.
9. База данных. Приложение должно использовать базу данных для хранения информации о деталях, заказах, клиентах и т.д.

В целом, WPF приложение для предприятия по продаже погрузчиков и деталей для них должно быть удобным, простым в использовании и обеспечивать эффективное управление заказами и деталями.

Теперь разберём программное обеспечение, которое будет использоваться во время нашей разработки

**Програ́ммное обеспе́чение** — программа или множество программ, используемых для управления компьютером.

Microsoft разработала Visual Studio 2022, интегрированную среду разработки (IDE), которая предназначена для создания приложений для операционных систем Windows, мобильных приложений и веб-разработки. В IDE поддерживаются множество языков программирования, включая C#, C++, Visual Basic, Python, JavaScript и многие другие.

С помощью Visual Studio разработчики получают доступ к широкому набору инструментов для создания, тестирования, отладки и развертывания приложений, что позволяет ускорить процесс разработки и облегчить взаимодействие между членами команды. В IDE также доступны множество плагинов и расширений, которые облегчают задачи разработчиков и расширяют функциональность IDE.

**Visual Studio** **2022** - это интегрированная среда разработки (IDE), которую разработала компания Microsoft для создания приложений для операционных систем Windows, а также для веб-разработки и разработки мобильных приложений. Среда поддерживает множество языков программирования, включая C#, C++, Visual Basic, Python, JavaScript и многие другие.

**Visual Studio** обеспечивает разработчикам широкий набор инструментов для создания, отладки, тестирования и развертывания приложений. Это позволяет ускорить процесс разработки и упростить взаимодействие между разработчиками в команде. В Visual Studio также есть множество плагинов и расширений, которые позволяют дополнительно расширить функциональность IDE для удобства разработки и облегчения задач.

**Microsoft SQL Server Management Studio (SSMS)** – это интегрированная среда для управления базами данных Microsoft SQL Server. Она предоставляет графический интерфейс для управления и настройки экземпляров SQL Server, а также позволяет создавать, изменять и удалять базы данных, таблицы, представления, процедуры и другие объекты баз данных.

# **Постановка задачи**

База данных будет использоваться для учета продаж и поставок запчастей и погрузчиков и будет относиться к классу баз данных управления предприятием. Созданная база данных поможет упростить процедуру поиска информации о товарах и ценах на них. Созданная база данных также включает информацию о поставщиках, сотрудниках, заказах.

Приложение для работы с информационной системой отдела продаж упрощает процедуру поиска необходимой информации о заказах, деталях, погрузчиков и ценах на них. С ее помощью сотрудник может легко узнать информацию о поставленном или проданной детали, клиент может просматривать существующий в наличии ряд запчастей, руководитель отслеживать деятельность сотрудников и динамику продаж.

Как у администратора, так и у клиента есть возможность найти интересующую их деталь, выставить нужные фильтры, отсортировать список деталей по наличию, по возрастанию цены и по типу деталей.

Определим задачи которые должны быть решены на производственной практической подготовке:

1. Создать БД 🡪 Разработать ERD диаграмму 🡪 Подготовить данные для импорта 🡪 Разработать БД с нужными таблицами 🡪 Импортировать данные.
2. Создание диаграмм 🡪 UseCase 🡪 UserStory
3. Разработать WPF приложение 🡪 Подключить БД к проекту 🡪 Реализовать возможность авторизации 🡪 Попытки входа и капча 🡪 Вывод данных, DataGrid, ListView 🡪 Создание корзины с товарами 🡪 Поиск, сортировка, фильтрация.

# **Создание ER-диаграммы**

Visio - это программа для создания диаграмм и графических изображений, разработанная компанией Microsoft. Она позволяет создавать различные типы диаграмм, включая блок-схемы, графики, диаграммы потоков данных, организационные диаграммы, диаграммы связей и многое другое.

Visio используется в бизнес-среде для создания диаграмм процессов, схемы сетей, планы проектов и других визуальных материалов, которые помогают улучшить понимание и коммуникацию между коллегами и стейкхолдерами.

Нам необходимо создать базовое представление БД для работы с отделом продаж предприятия занимающегося продажей погрузчиков и деталей для них. Откроем специализированную программу Visio и выберем шаблон (Рисунок 1).

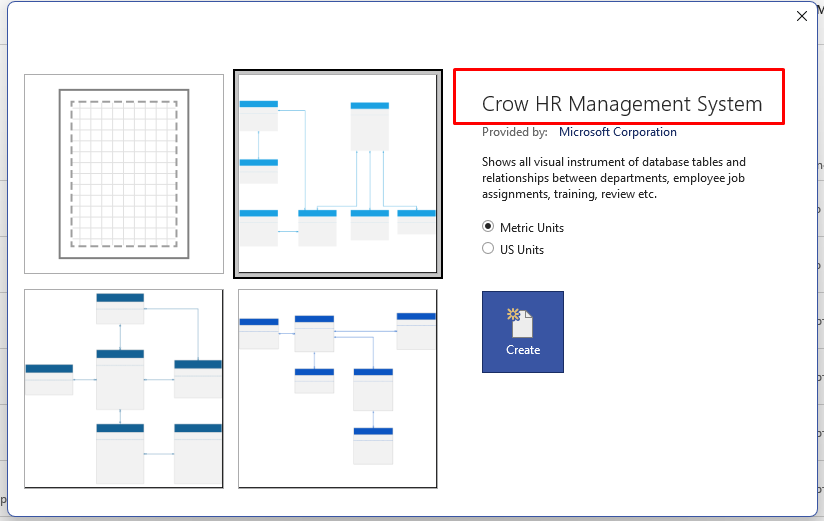


Рисунок 1. Выбор шаблона для разработки диаграммы

С помощью представленных наборов фигур составим таблицы, необходимые для работы с информационной системой отдела продаж (Рисунок 2).

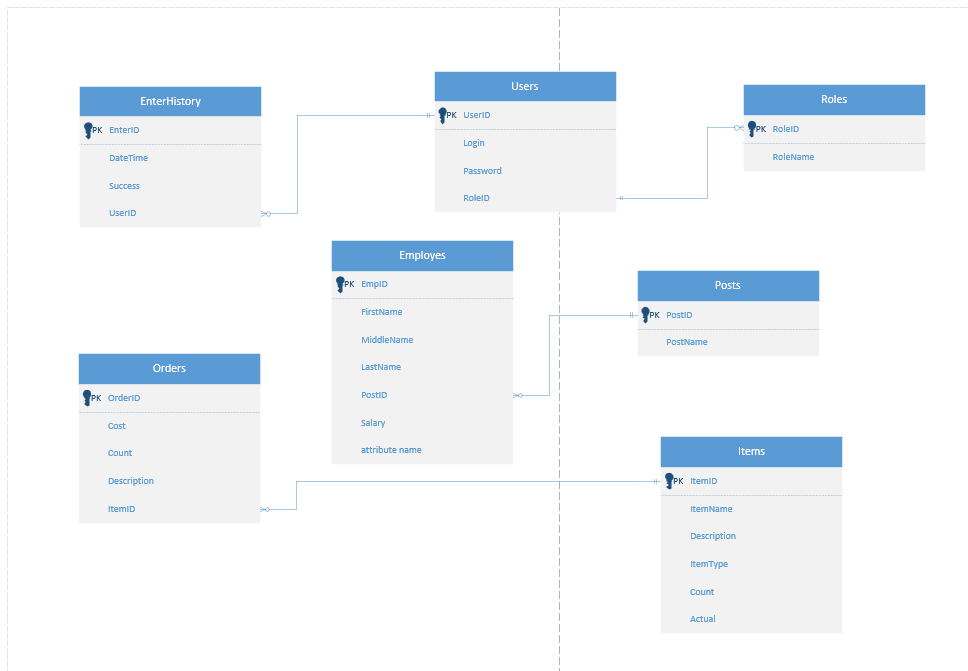


Рисунок 2. Готовая ERD диаграмма отдела продаж.

Таблица истории входа содержит информацию о каждом входе пользователя в систему, включая идентификатор входа (EnterID), дату и время входа (DateTime), статус успешности входа (Success) и идентификатор пользователя (UserID).

Таблица Users содержит информацию о каждом пользователе, включая идентификатор пользователя (UserID), логин (Login), пароль (Password) и идентификатор роли (RoleID).

Таблица работников содержит информацию о каждом сотруднике, включая идентификатор работника (EmpID), имя (FirstName), отчество (MiddleName), фамилию (LastName), идентификатор должности (PostID) и зарплату (Salary).

Таблица ролей содержит информацию о каждой роли в системе, включая идентификатор роли (RoleID) и название роли (RoleName).

Таблица заказов содержит информацию о каждом заказе, включая идентификатор заказа (OrderID), стоимость (Cost), количество (Count), описание (Description) и идентификатор товара (ItemID).

Таблица товара содержит информацию о каждом товаре, включая идентификатор товара (ItemID), название товара (ItemName), описание (Description), тип товара (ItemType), количество (Count) и фактическую наличность (Actual).

Таблица должностей содержит информацию о каждой должности в компании, включая идентификатор должности (PostID) и название должности (PostName).

Эти таблицы созданы для хранения различных типов данных, связанных с пользователем, сотрудником, заказами, товарами и должностями. Они могут использоваться в различных бизнес-приложениях, таких как учетная система или система управления персоналом, для обеспечения эффективной и безопасной работы.

# **User Story**

**User Story** (продуктовая история) - это короткая и простая форма описания требований к функциональности продукта или сервиса, написанная в виде истории из глаз пользователя. User Story помогает команде разработчиков продукта понять, какую задачу нужно выполнить и для какой целевой аудитории.

При разработке User Stories вам необходимо использовать технологию разработки программного обеспечения BDD (сокр. от англ. Behavior-driven development, дословно «разработка через поведение») — это методология разработки программного обеспечения, являющаяся ответвлением от методологии разработки через тестирование (TDD)).

User story вкратце описывает:

* человека, использующего систему (заказчик);
* то, что должно содержаться в данной системе (примечание);
* то, для чего она нужна пользователю (цель).

Для разработки используйте следующий шаблон:

История: Поиск нужной детали

Как Менеджер

Я хочу иметь возможность найти нужную мне деталь и и получить информацию о её наличии

Чтобы я мог увидеть нужную деталь, её описание, наличие и стоимость

Сценарий: Нужная деталь найдена, получена полная информация о ней

Дано: менеджер открыл программу и ввёл название нужной детали

Когда страница с деталями открыта

Тогда менеджер получит полную информацию о товаре

Сценарий: Нужная деталь не найдена, не получена информация

Дано: клиент открыл программу и ввёл параметры поиска

Когда страница с деталями не отобразила никаких данных

Тогда не удалось вывести информацию по параметрам поиска

# **Use case**

Use Case (случай использования) - это сценарий или описание того, как конкретный пользователь может использовать определенное приложение, систему или продукт для достижения определенной цели или решения определенной проблемы.

В рамках разработки программного обеспечения, Use Case является частью методологии разработки, которая позволяет описывать, как система будет использоваться в реальной жизни. Use Case может содержать информацию о том, кто будет использовать систему, как они будут ее использовать, какую информацию они вводят и получают из системы, и какие результаты они ожидают.

Use Case также может быть использован для описания бизнес-процессов, чтобы определить, какие роли и акторы будут участвовать в процессе, какие шаги они будут предпринимать и какие решения они будут принимать. Это позволяет бизнес-аналитикам и разработчикам создать систему, которая эффективно поддерживает бизнес-процессы и требования пользователей.

Ниже приведён пример Use Case диаграммы, на которой показаны возможности администратора, менеджера и клиента

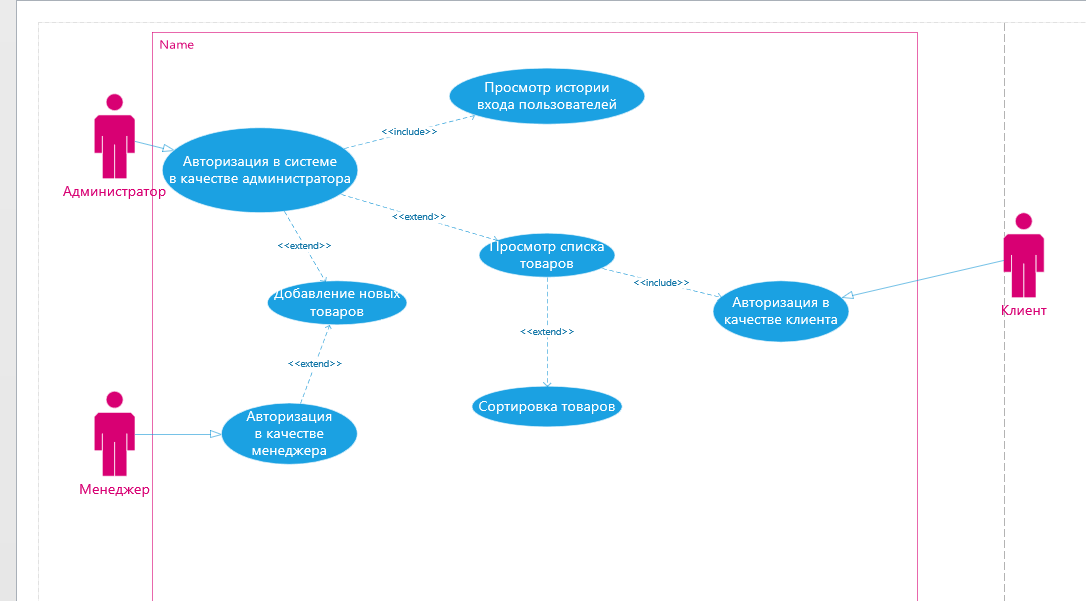


Рисунок 3. Use case отдела продаж

# **Разработка базы данных**

На основе ранее созданной нами ER диаграммы мы создадим таблицы, поставим нужные ключи и типы данных, для этого запустим Microsoft SQL Server Management Studio.

После подключения нажимаем правой кнопкой мыши по папке «DataBases» и выбираем «New Database» (Рисунок 4).

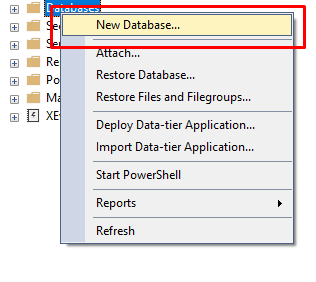


Рисунок 4. Создание новой базы данных.

Вводим название БД и переходим к созданию таблиц (Рисунок 5).

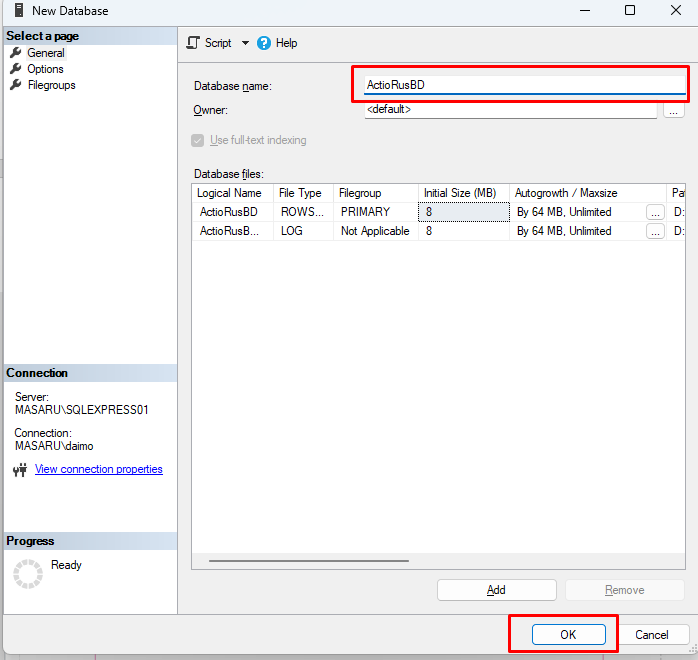


Рисунок 5. Выбор имени БД.

После успешного создания БД нажимаем правой кнопкой мыши по папке «Tables» и выбираем пункт «New Table» (Рисунок 6).

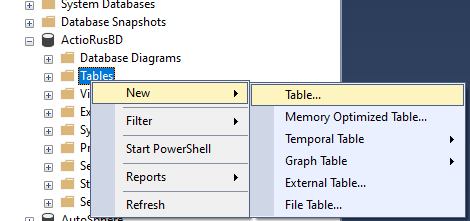


Рисунок 6. Создание новой таблицы.

Перед нами открывается окно с созданием таблица, в которой нам необходимо выбрать название столбца и его тип данных (Рисунок 7).

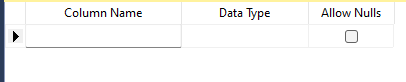


Рисунок 7. Создание таблицы.

Создадим таблицу пользователей, для этого определим названия столбцов и их типы данных. Первый столбец в основном используется для нумерации объекта и этому столбцу присваивается первичный ключ (Рисунок 8).

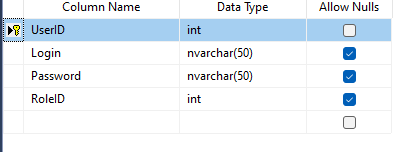


Рисунок 8. Таблица пользователей.

Мы определили такие столбцы как:

* Номер пользователя
* Логин
* Пароль
* Номер роли

В созданной нами таблице будет храниться информации о пользователях, его номер, логин, пароль и номер роли.

Аналогичным образом были созданы оставшиеся таблицы в БД, которые были представлены на ER диаграмме (Рисунок 9).

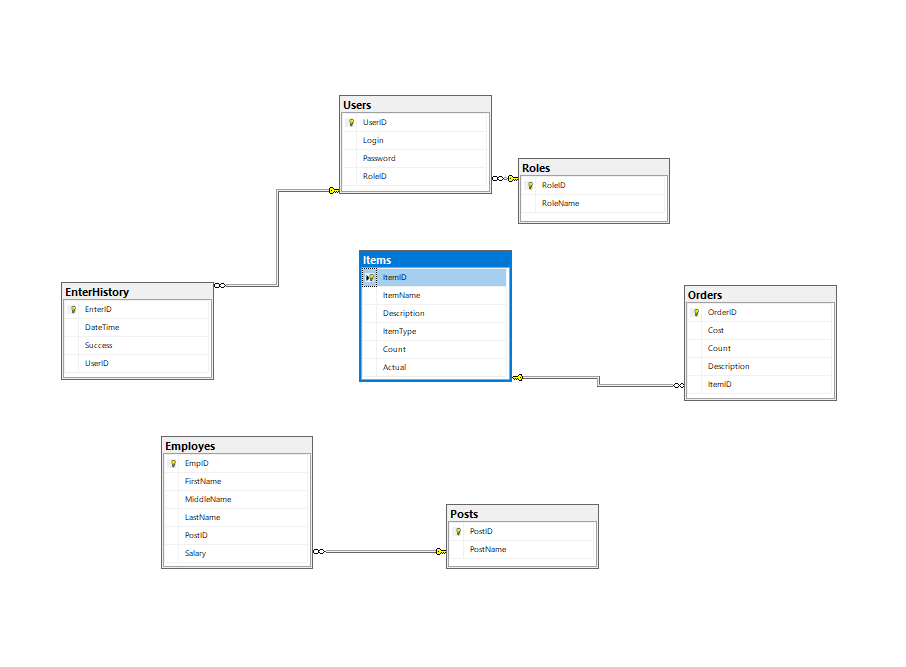


Рисунок 9. Диаграмма базы данных созданная в SQL Server Management Studio.

# **Разработка WPF приложения**

**WPF (Windows Presentation Foundation)** - это технология разработки графического интерфейса пользователя (GUI) для Windows приложений, которая предоставляет широкие возможности для создания интерактивных и многофункциональных приложений.

Перейдём в ранее установленное приложение Visual Studio 2022 для создания приложения, создадим новый проект с шаблоном WPF (Рисунок 10).

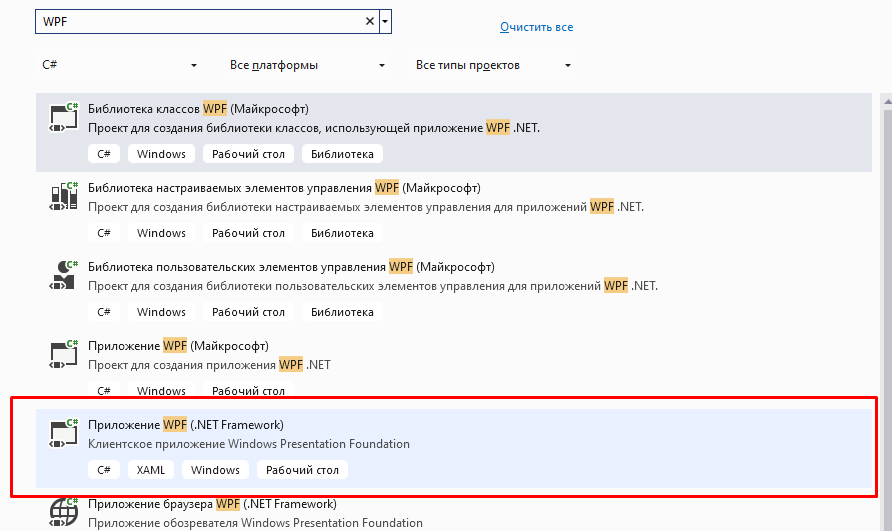


Рисунок 10. Создание нового WPF проекта

Вводим название проекта и жмём кнопку «Создать» (Рисунок 11).

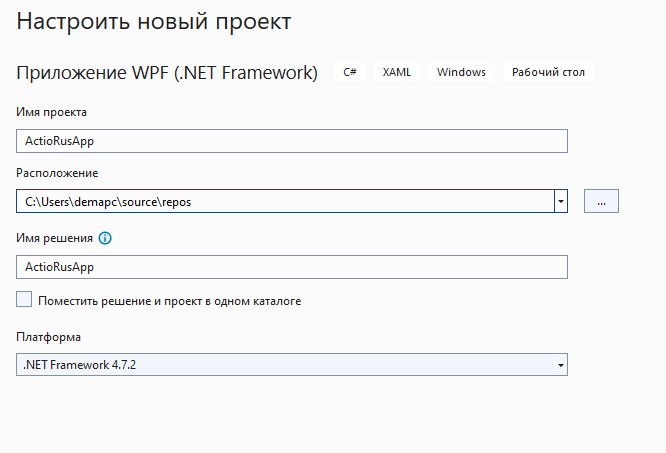


Рисунок 11. Название проекта

После загрузки нужных компонентов перед нами откроется главное окно приложения для разработки пользовательского интерфейса. С помощью XAML разметки создадим главное окно приложения, «шапку» и «футер», как на сайтах (Рисунок 12).

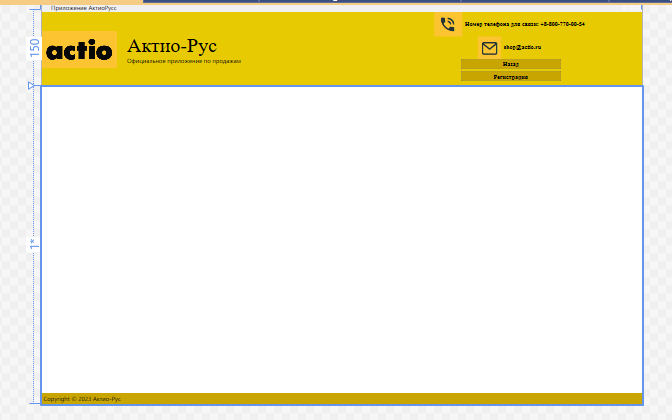


Рисунок 12. Главное окно MainWindow

С помощью XAML разметки создаём Grid и Frame, нужные кнопки, добавляем логотип кампании. Ниже приведён полный код страницы:

<Window x:Class="ActioRusApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ActioRusApp"

mc:Ignorable="d"

Title=" Приложение АктиоРусс" Height="800" Width="1200">

<Grid Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="150" x:Name="Head"></RowDefinition>

<RowDefinition Height="\*" ></RowDefinition>

</Grid.RowDefinitions>

<Frame ContentRendered="MainFrame\_ContentRendered" NavigationUIVisibility="Hidden" Name="MainFrame" Grid.Row="1" Navigated="MainFrame\_Navigated">

</Frame>

<Grid Background="#FFE8CA03">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="150"></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image Grid.Column="0" HorizontalAlignment="Left" Source="\Resources\logoActio.png" Width="150" Height="150"/>

<StackPanel Height="auto" VerticalAlignment="Center" HorizontalAlignment="Right" Grid.Column="1">

<TextBlock HorizontalAlignment="Left" Grid.Row="1" TextWrapping="Wrap" Text="Актио-Рус" FontSize="40" Width="505" FontFamily="Times New Roman"/>

<TextBlock Text="Официальное приложение по продажам"></TextBlock>

</StackPanel>

<StackPanel Grid.Column="3">

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">

<Image Source="\Resources\telephone.png"></Image>

<Label Grid.Column="3" VerticalAlignment="Center" Width="NaN" HorizontalAlignment="Center" Content="Номер телефона для связи: +8-800-770-00-54" FontFamily="Times New Roman" Height="25" FontWeight="Bold"/>

</StackPanel>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">

<Image Source="\Resources\email.png"></Image>

<Label Grid.Column="3" VerticalAlignment="Center" Width="NaN" HorizontalAlignment="Center" Content="shop@actio.ru" FontFamily="Times New Roman" Height="25" FontWeight="Bold"/>

</StackPanel>

<Button FontWeight="Bold" FontFamily="Times New Roman" Name="BackButton" Width="200" Content="Назад" Height="20" Click="BackButton\_Click" Background="#FFC8A500" />

<Button FontWeight="Bold" FontFamily="Times New Roman" Name ="RegistrationButton" Width="200" Content="Регистрация" Height="20" Click="RegistrationButton\_Click" Margin="5" Background="#FFC8A500" />

</StackPanel>

</Grid>

<DockPanel Grid.Row="1" VerticalAlignment="Bottom" Background="AntiqueWhite">

<StatusBar DockPanel.Dock="Bottom" Background="#FFC8A500">

<TextBlock Text="Copyright © 2023 Актио-Рус" />

</StatusBar>

<Grid>

</Grid>

</DockPanel>

</Grid>

</Window>

C помощью XAML разметки определяем размер «хэдэра», «футэра» и основного окна «MainFrame». Делим «шапку» нашего окна на три части, в первой части устанавливаем логотип предприятия, во втором пишем название предприятия, в третьем столбце добавляем кнопки «Назад» и «Регистрация», а так же номер телефона для связи и жлектронную почту.

Теперь нам необходимо создать страницу авторизации, для этого нажимаем правой кнопкой мыши по нашему решению и выбираем строку «Добавить» 🡪 «Создать страницу WPF» (Рисунок 13)

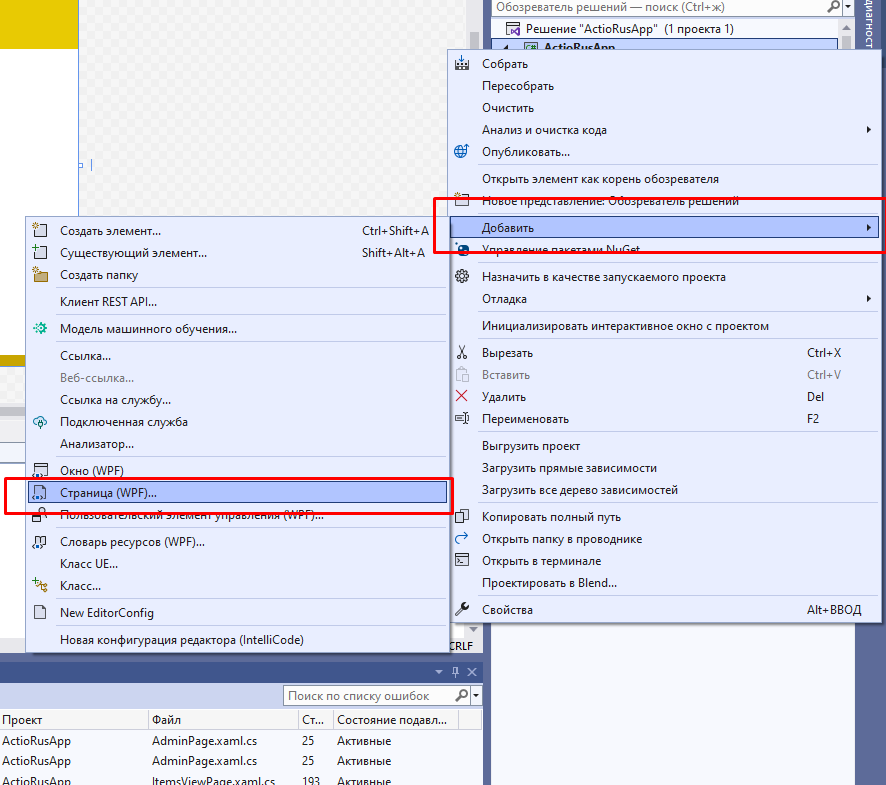


Рисунок 13. Создание страницы авторизации

Аналогичным образом создаём страницу авторизации, добавляя поля для ввода логина, пароля и кнопку «Войти» (Рисунок 14).

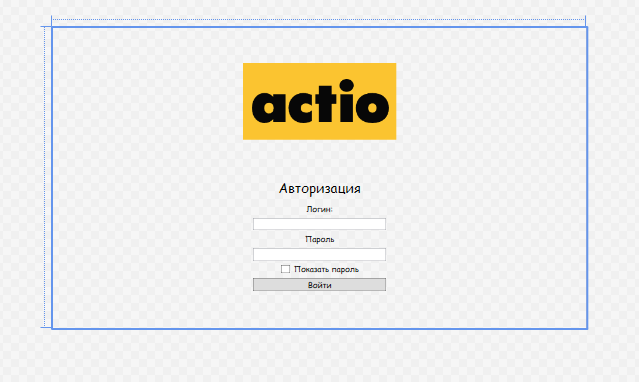


Рисунок 14. Окно авторизации

Далее, нам необходимо прописать логику авторизации, по которой программа будет проверять введённый логин и пароль в поля для ввода и сравнивать их с данными в БД, но для начала создадим два класса для навигации между страницами и класс подключения БД.

Для создания класса выберем правой кнопкой мыши наш проект и создадим новую папку «Classes» и уже в неё добавим новый класс «Manager» (Рисунок 15).

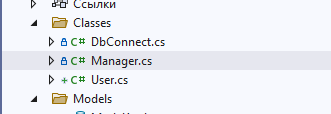


Рисунок 15. Папка Classes и класс Manager

Напишем следующий код в класс Manager:

class Manager

{

public static Frame MainFrame { get; set; }

}

Этот код определяет класс Manager, в котором объявляется статическое свойство MainFrame типа Frame. Это свойство позволяет получить доступ к Frame в различных частях приложения, таких как окна и страницы, и использовать его для навигации между различными страницами в приложении. Благодаря использованию статического свойства, MainFrame может быть доступен для всех экземпляров класса Manager без необходимости создания новых экземпляров.

Теперь нам необходимо создать подключение к БД и класс подключения. Добавляем класс Dbconnect в папку Classes и создаём публичный метод:

public static Models.AutoSphereEntities modeldb;

Эта строчка отвечает за подключение добавленной модели к нашему проекту. Теперь нам необходимо создать папку Models и добавить туда модель нашей БД. Для этого нажимаем правой кнопкой мыши по папке Models 🡪 Добавить

В поиске вводим «ADO» и нажимаем «Модель ADO.NET EDM» (Рисунок 16).

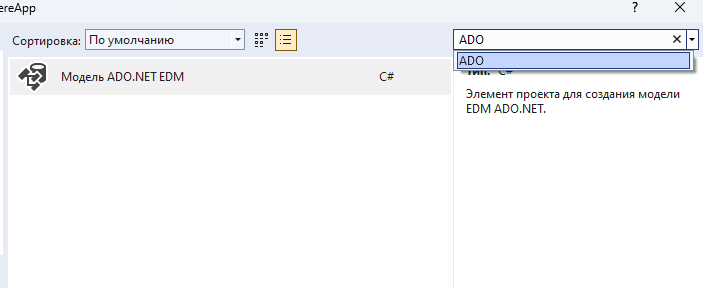


Рисунок 16. Добавление модели БД.

Следуя всем инструкциям, вводим название нашей БД и добавляем в папку Models (Рисунок 17).

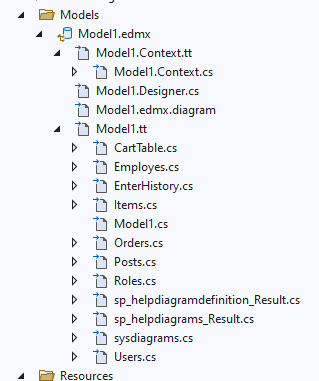


Рисунок 17. Подключенная модель

Теперь, написаный нами ранее код подключения, а именно класс DbConnect будет работать исправно и на страницах далее, нам нужно будет обращаться именно к этому классу для подключения к БД.

Следующим шагом в подключении БД является редактирование файла Model1.Context.cs. Переходим к странице кода и дописываем следующий код:

public partial class ActioRusBDEntities : DbContext

{

private static ActioRusBDEntities \_context;

public ActioRusBDEntities()

: base("name=ActioRusBDEntities")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

public static ActioRusBDEntities GetContext()

{

if(\_context == null)

{

\_context = new ActioRusBDEntities();

}

return \_context;

}

public virtual DbSet<Employes> Employes { get; set; }

public virtual DbSet<EnterHistory> EnterHistory { get; set; }

public virtual DbSet<Items> Items { get; set; }

public virtual DbSet<Orders> Orders { get; set; }

public virtual DbSet<Posts> Posts { get; set; }

public virtual DbSet<Roles> Roles { get; set; }

public virtual DbSet<sysdiagrams> sysdiagrams { get; set; }

public virtual DbSet<Users> Users { get; set; }

public virtual DbSet<CartTable> CartTable { get; set; }

public static Users currentuser = null;

public virtual int sp\_alterdiagram(string diagramname, Nullable<int> owner\_id, Nullable<int> version, byte[] definition)

{

var diagramnameParameter = diagramname != null ?

new ObjectParameter("diagramname", diagramname) :

new ObjectParameter("diagramname", typeof(string));

var owner\_idParameter = owner\_id.HasValue ?

new ObjectParameter("owner\_id", owner\_id) :

new ObjectParameter("owner\_id", typeof(int));

var versionParameter = version.HasValue ?

new ObjectParameter("version", version) :

new ObjectParameter("version", typeof(int));

var definitionParameter = definition != null ?

new ObjectParameter("definition", definition) :

new ObjectParameter("definition", typeof(byte[]));

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp\_alterdiagram", diagramnameParameter, owner\_idParameter, versionParameter, definitionParameter);

}

public virtual int sp\_creatediagram(string diagramname, Nullable<int> owner\_id, Nullable<int> version, byte[] definition)

{

var diagramnameParameter = diagramname != null ?

new ObjectParameter("diagramname", diagramname) :

new ObjectParameter("diagramname", typeof(string));

var owner\_idParameter = owner\_id.HasValue ?

new ObjectParameter("owner\_id", owner\_id) :

new ObjectParameter("owner\_id", typeof(int));

var versionParameter = version.HasValue ?

new ObjectParameter("version", version) :

new ObjectParameter("version", typeof(int));

var definitionParameter = definition != null ?

new ObjectParameter("definition", definition) :

new ObjectParameter("definition", typeof(byte[]));

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp\_creatediagram", diagramnameParameter, owner\_idParameter, versionParameter, definitionParameter);

}

public virtual int sp\_dropdiagram(string diagramname, Nullable<int> owner\_id)

{

var diagramnameParameter = diagramname != null ?

new ObjectParameter("diagramname", diagramname) :

new ObjectParameter("diagramname", typeof(string));

var owner\_idParameter = owner\_id.HasValue ?

new ObjectParameter("owner\_id", owner\_id) :

new ObjectParameter("owner\_id", typeof(int));

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp\_dropdiagram", diagramnameParameter, owner\_idParameter);

}

public virtual ObjectResult<sp\_helpdiagramdefinition\_Result> sp\_helpdiagramdefinition(string diagramname, Nullable<int> owner\_id)

{

var diagramnameParameter = diagramname != null ?

new ObjectParameter("diagramname", diagramname) :

new ObjectParameter("diagramname", typeof(string));

var owner\_idParameter = owner\_id.HasValue ?

new ObjectParameter("owner\_id", owner\_id) :

new ObjectParameter("owner\_id", typeof(int));

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<sp\_helpdiagramdefinition\_Result>("sp\_helpdiagramdefinition", diagramnameParameter, owner\_idParameter);

}

public virtual ObjectResult<sp\_helpdiagrams\_Result> sp\_helpdiagrams(string diagramname, Nullable<int> owner\_id)

{

var diagramnameParameter = diagramname != null ?

new ObjectParameter("diagramname", diagramname) :

new ObjectParameter("diagramname", typeof(string));

var owner\_idParameter = owner\_id.HasValue ?

new ObjectParameter("owner\_id", owner\_id) :

new ObjectParameter("owner\_id", typeof(int));

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<sp\_helpdiagrams\_Result>("sp\_helpdiagrams", diagramnameParameter, owner\_idParameter);

}

public virtual int sp\_renamediagram(string diagramname, Nullable<int> owner\_id, string new\_diagramname)

{

var diagramnameParameter = diagramname != null ?

new ObjectParameter("diagramname", diagramname) :

new ObjectParameter("diagramname", typeof(string));

var owner\_idParameter = owner\_id.HasValue ?

new ObjectParameter("owner\_id", owner\_id) :

new ObjectParameter("owner\_id", typeof(int));

var new\_diagramnameParameter = new\_diagramname != null ?

new ObjectParameter("new\_diagramname", new\_diagramname) :

new ObjectParameter("new\_diagramname", typeof(string));

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp\_renamediagram", diagramnameParameter, owner\_idParameter, new\_diagramnameParameter);

}

public virtual int sp\_upgraddiagrams()

{

return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp\_upgraddiagrams");

}

}

Этот код определяет класс ActioRusEnteties, который наследуется от класса DbContext в Entity Framework. Этот класс представляет контекст базы данных и содержит свойства DbSet для каждой таблицы в базе данных, а также статический метод GetContext(), который возвращает экземпляр контекста базы данных. Конструктор класса инициализирует базовый класс DbContext и задает имя подключения к базе данных. Метод OnModelCreating используется для настройки модели базы данных. Строка throw new UnintentionalCodeFirstException(); генерирует исключение, если модель базы данных была создана неправильно.

# **Код приложения**

## **Страница авторизации**

Страница авторизации в WPF приложении представляет собой графический интерфейс, который позволяет пользователю ввести свои учетные данные (например, логин и пароль) для входа в приложение. Она обычно содержит следующие элементы:

1. Поля ввода: текстовые поля для ввода имени пользователя (логина) и пароля.
2. Кнопка "Войти": пользователь может нажать на эту кнопку, чтобы отправить данные авторизации на сервер и войти в приложение.
3. Чек бокс «Показать пароль»

Перейдём к коду работы страницы. Нам нужно подключить БД, сверить введённый пароль, прописать логику «показать пароль», логику кнопки «Войти», а так же попытки входа, после которых будет выскакивать просьба пройти капчу.

Код страницы:

public partial class LogInPage : Page

{

private int attemps = 0;

Random \_random = new Random();

private TextBlock myTextBlock;

public LogInPage()

{

InitializeComponent();

Classes.Dbconnect.dbconnect = new Models.ActioRusBDEntities();

}

private void CheckBox\_Checked(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(Password.Password))

{

MessageBox.Show("Для начала, введите пароль!", "Уведомление");

CheckBoxPass.IsChecked = false;

}

else

{

TxbPassword.Visibility = Visibility.Visible;

Password.Visibility = Visibility.Collapsed;

TxbPassword.Text = Password.Password;

}

}

private void CheckBox\_UnChecked(object sender, RoutedEventArgs e)

{

TxbPassword.Visibility = Visibility.Collapsed;

Password.Visibility = Visibility.Visible;

TxbPassword.Text = Password.Password;

}

private void LogIN()

{

try

{

var userObj = Classes.Dbconnect.dbconnect.Users.FirstOrDefault(x => x.Login == LoginTB.Text && x.Password == Password.Password);

if (userObj != null)

{

Models.ActioRusBDEntities.currentuser = userObj;

MessageBox.Show("Здравствуйте " + userObj.Roles.RoleName + ", " + userObj.Login, "Warning", MessageBoxButton.OK, MessageBoxImage.Information);

switch (userObj.RoleID)

{

case 1:

Manager.MainFrame.Navigate(new AdminPage());

break;

case 2:

Manager.MainFrame.Navigate(new ManagerPage());

break;

case 3:

Manager.MainFrame.Navigate(new ClientPage());

break;

default:

MessageBox.Show("Данные не обнаружены!", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Warning);

break;

}

}

else

{

MessageBox.Show("Не верный логин или пароль", "Уведовление");

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка: " + ex.Message.ToString(), "Критическая работа приложения",

MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

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

{

var userObj = Classes.Dbconnect.dbconnect.Users.FirstOrDefault(x => x.Login == LoginTB.Text && x.Password == Password.Password);

if (userObj == null)

{

MessageBox.Show("Неверный логин или пароль", "Ошибка при авторизации",

MessageBoxButton.OK, MessageBoxImage.Error);

attemps++;

CheckAttemps();

}

else

{

LogIN();

}

}

private void CheckAttemps()

{

if (attemps == 2)

{

MessageBox.Show("Слишком много неудачных попыток! Подтвердите, что вы человек", "Не удается войти!", MessageBoxButton.OK, MessageBoxImage.Warning);

Noises.Visibility = Visibility.Visible;

SymbolsGen.Visibility = Visibility.Visible;

GetCapcha.Visibility = Visibility.Visible;

UpdateCapcha.Visibility = Visibility.Visible;

ConfirmCapcha.Visibility = Visibility.Visible;

LoginTB.Visibility = Visibility.Collapsed;

Password.Visibility = Visibility.Collapsed;

GenerateNoisesForCapcha(30);

GenerateSymbols(3);

if (GetCapcha.Text != Symbols)

{

}

}

else

{

if (attemps == 3)

{

MessageBox.Show("Возможность входа заблокирована", "Слишком много неудачных попыток", MessageBoxButton.OK, MessageBoxImage.Warning);

Blocked.Visibility = Visibility.Visible;

// Блокируем элементы интерфейса

foreach (UIElement element in UIInt.Children)

{

if (element is Control control && control.Name != "ExitButton" && control.Name != "SupportButton")

{

control.IsEnabled = false;

}

}

// Запускаем таймер на 10 секунд

double seconds = 10;

DispatcherTimer timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromSeconds(1);

timer.Tick += (sender, args) =>

{

seconds--;

TimerTextBlock.Text = $"Попробуйте снова через {seconds} сек.";

if (seconds == 0)

{

timer.Stop();

Blocked.Visibility = Visibility.Collapsed;

TimerTextBlock.Visibility = Visibility.Collapsed;

// Разблокируем элементы интерфейса

foreach (UIElement element in UIInt.Children)

{

if (element is Control control && control.Name != "ExitButton" && control.Name != "SupportButton")

{

control.IsEnabled = true;

attemps = 0;

}

}

}

};

timer.Start();

TimerTextBlock.Visibility = Visibility.Visible;

}

}

}

private void GenerateNoisesForCapcha(int volumeNoise)

{

Noises.Visibility = Visibility.Visible;

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

{

Ellipse ellipse = new Ellipse

{

Fill = new SolidColorBrush(Color.FromArgb((byte)\_random.Next(100, 200),

(byte)\_random.Next(0, 256),

(byte)\_random.Next(0, 256),

(byte)\_random.Next(0, 256)))

};

ellipse.Height = ellipse.Width = \_random.Next(20, 60);

Canvas.SetLeft(ellipse, \_random.Next(0, 290));

Canvas.SetTop(ellipse, \_random.Next(0, 100));

Noises.Children.Add(ellipse);

}

}

public string Symbols = "";

private void GenerateSymbols(int count)

{

string alphabet = "ABCDEFGHJKLMN123456789";

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

{

string symbol = alphabet.ElementAt(\_random.Next(0, alphabet.Length)).ToString();

TextBlock lbl = new TextBlock();

lbl.Text = symbol;

lbl.FontSize = \_random.Next(20, 40);

lbl.RenderTransform = new RotateTransform(\_random.Next(-45, 45));

lbl.Margin = new Thickness(20, 20, 20, 20);

Noises.Visibility = Visibility.Visible;

SymbolsGen.Children.Add(lbl);

Symbols = Symbols + symbol;

myTextBlock = lbl;

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

Symbols = "";

SymbolsGen.Children.Clear();

Noises.Children.Clear();

GenerateSymbols(3);

GenerateNoisesForCapcha(25);

}

}

Запускаем приложение и смотрим на результат:

Рисунок 18. Показ пароля

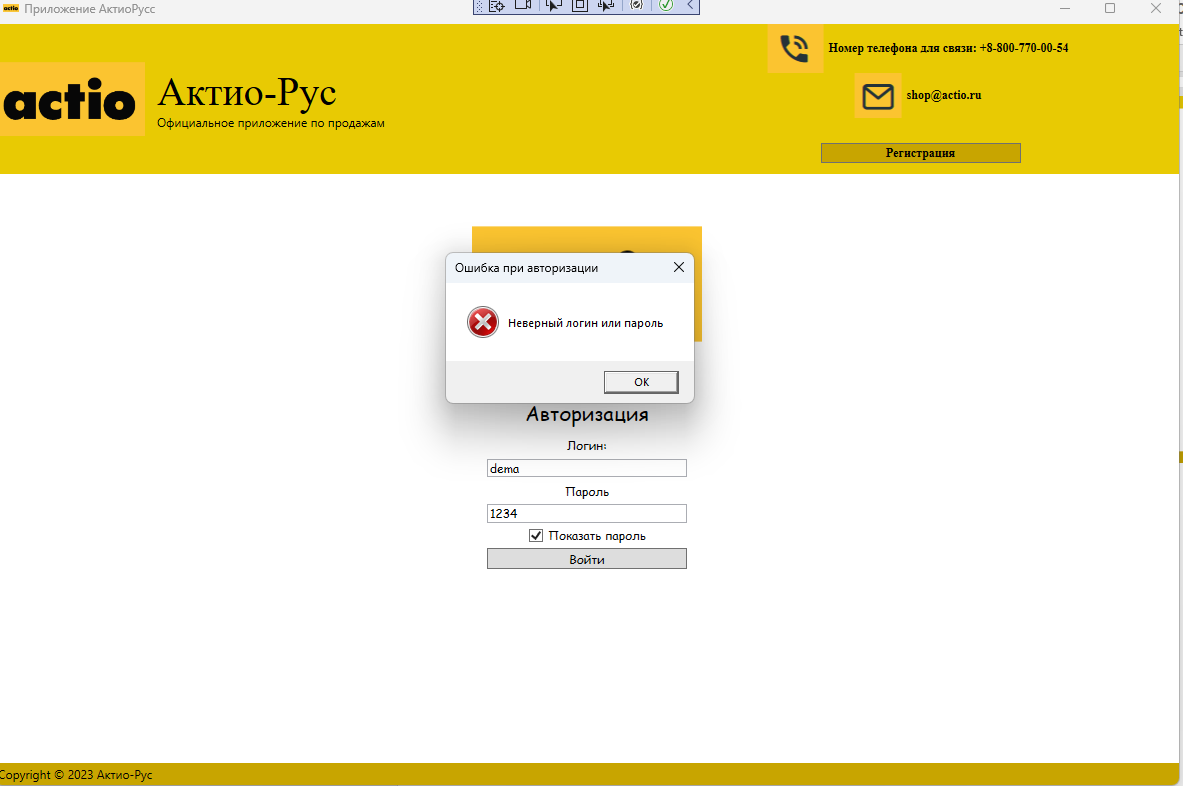


Рисунок 19. Неверный логин или пароль

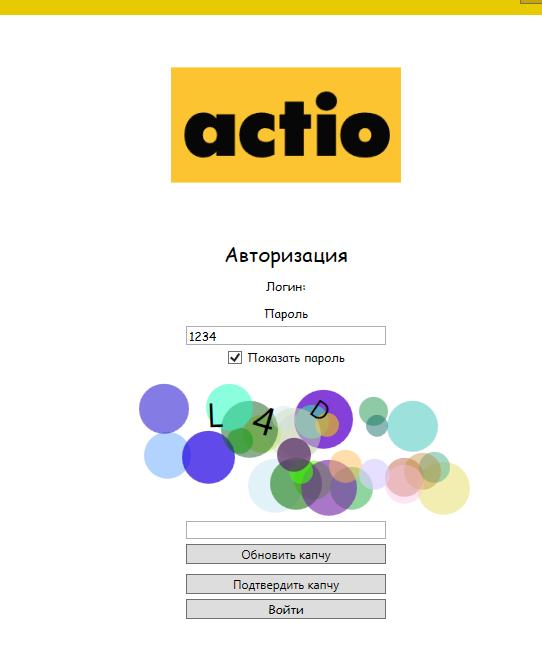


Рисунок 20. Просьба ввести капчу

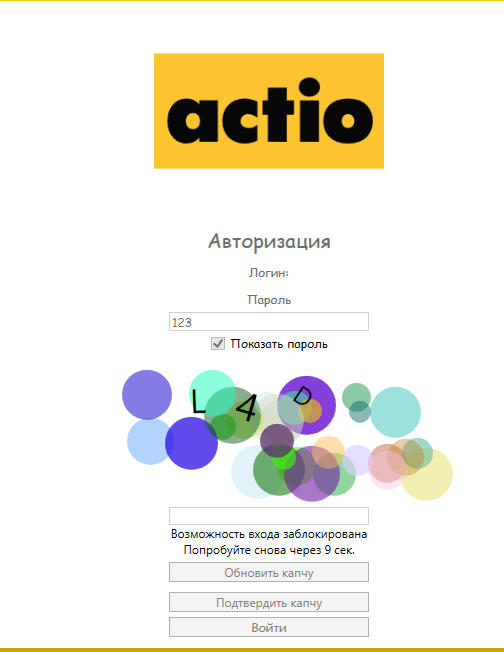


Рисунок 21. Блокировка входа

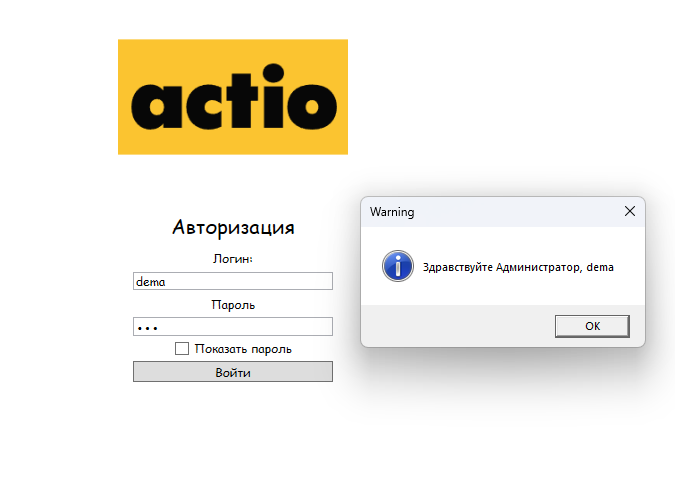


Рисунок 22. Успешный вход в качестве администратора

## **Вывод данных в DataGrid**

DataGrid в WPF поддерживает связывание данных, множественный выбор, сортировку, фильтрацию, редактирование и многие другие функции. Он также предоставляет множество событий, которые можно использовать для обработки действий пользователя.

XAML Код страницы с выводом данных в DataGrid представлен ниже:

<Grid>

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

<RowDefinition Height="50"></RowDefinition>

</Grid.RowDefinitions>

<DataGrid Name="ItemsView" AutoGenerateColumns="False" IsReadOnly="True">

<DataGrid.Columns>

<DataGridTextColumn Header="Номер товара" Binding="{Binding ItemID}"></DataGridTextColumn>

<DataGridTextColumn Header="Название товара" Binding="{Binding ItemName}"></DataGridTextColumn>

<DataGridTextColumn Header="Описание" Binding="{Binding Description}"></DataGridTextColumn>

<DataGridTextColumn Header="Тип товара" Binding="{Binding ItemType}"></DataGridTextColumn>

<DataGridTextColumn Header="Количество" Binding="{Binding Count}"></DataGridTextColumn>

<DataGridTextColumn Header="Наличие" Binding="{Binding Actual}"></DataGridTextColumn>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<Button Content="Редактировать данные" Click="BtnRedData\_Click"></Button>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

<StackPanel Orientation="Horizontal" Grid.Row="1">

<Button Background="#FFC8A500" Content="Заказы" FontFamily="Times New Roman" Margin="5" Click="Button\_Click" />

<Button Content="Товары" FontFamily="Times New Roman" Margin="5" Click="Button\_Click\_3" Background="#FFFF4848"/>

<Button Content="Пользователи" FontFamily="Times New Roman" Margin="5" Click="Button\_Click\_3" Background="#FFC8A500"/>

<Button Content="Роли" FontFamily="Times New Roman" Margin="5" Click="Button\_Click\_3" Background="#FFC8A500"/>

</StackPanel>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Right" Width="auto" Grid.Row="1">

<Button Name="BtnEdit" Content="Добавить данные" Click="BtnRedData\_Click" FontFamily="Times New Roman" HorizontalAlignment="Right" Margin="5" Background="#FFC8A500" />

<Button Name="BtnDel" Content="Удалить запись" Click="BtnDel\_Click" FontFamily="Times New Roman" HorizontalAlignment="Right" Margin="5" Background="#FFC8A500" />

</StackPanel>

<StackPanel Grid.Row="2" Orientation="Horizontal" Margin="0 0 0 25" Grid.RowSpan="2">

<Button Margin="3" Width="20" Name="BtnFirstPage" Click="BtnFirstPage\_Click" Content="&lt;&lt;" FontFamily="Impact" Background="#FFC8A500" />

<Button Margin="3" Width="20" Name="BtnPreviousPage" Click="BtnPreviousPage\_Click" Content="&lt;" FontFamily="Impact" Background="#FFC8A500"/>

<Label Name="LblPages" VerticalAlignment="Center" FontSize="16" FontFamily="Impact" Height="28" >3/5</Label>

<Button Margin="3" Width="20" Name="BtnNextPage" Click="BtnNextPage\_Click" Content="&gt;" FontFamily="Impact" Background="#FFC8A500"/>

<Button Margin="3" Width="20" Name="BtnLastPage" Click="BtnLastPage\_Click" Content="&gt;&gt;" FontFamily="Impact" Background="#FFC8A500"/>

</StackPanel>

</Grid>

Создаём DataGrid, определяем столбцы и присваиваем каждому из них соответствующий с БД поле. Далее, определяем RowDefinition, всего 3 штуки. Добавляем StackPanel, в который добавляем кнопки переходов между страницами, а так же кнопки смены страницы с записями (Рисунок 23).

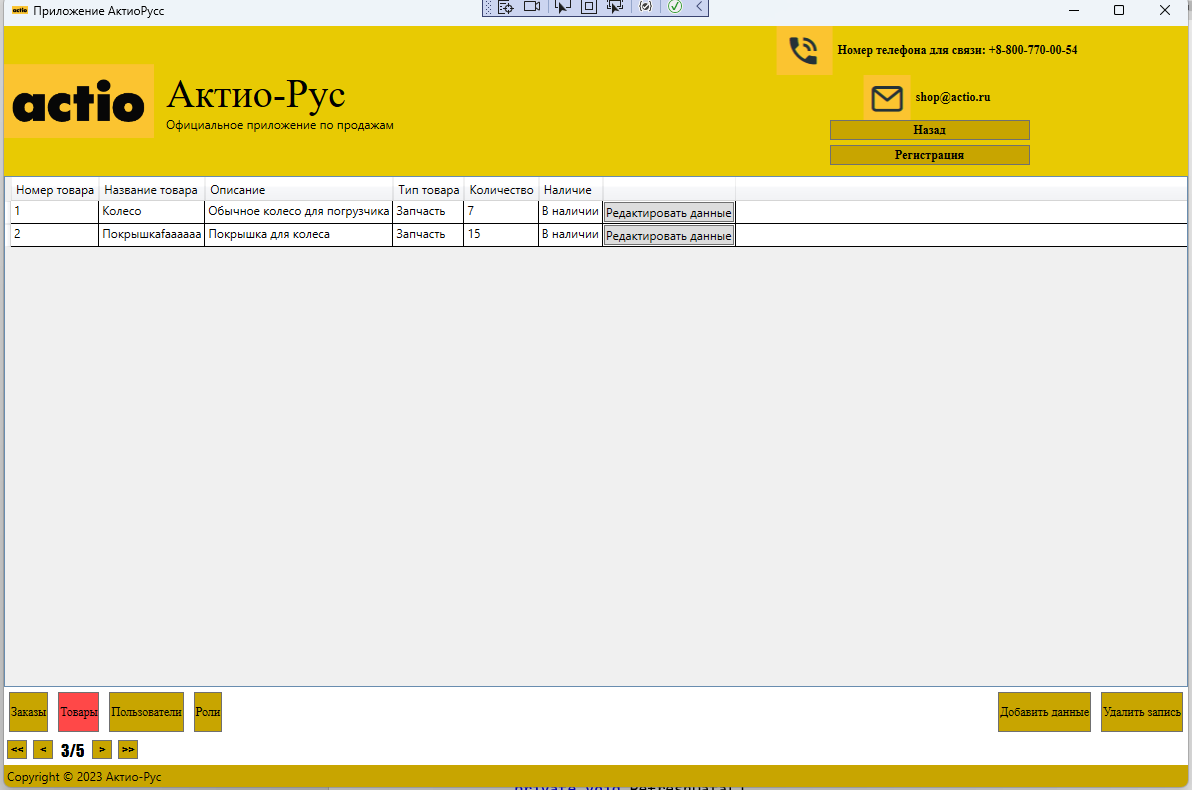


Рисунок 23. Страница с DataGrid товаров

Теперь перейдём к коду страницы:

public partial class AdminPage : Page

{

int \_currentPage = 1, \_countInPage = 10, \_maxPages;

public AdminPage()

{

InitializeComponent();

ItemsView.ItemsSource = ActioRusBDEntities.GetContext().Items.ToList();

}

private void Button\_RedClick(object sender, RoutedEventArgs e)

{

}

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

{

\_currentPage = 1;

RefreshData();

}

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

{

\_currentPage--;

RefreshData();

}

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

{

\_currentPage++;

RefreshData();

}

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

{

\_currentPage = \_maxPages;

RefreshData();

}

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

{

Manager.MainFrame.Navigate(new AddOrEditPageItems((sender as Button).DataContext as Items));

}

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

{

}

private void Button\_Click\_3(object sender, RoutedEventArgs e)

{

}

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

{

var ItemForRemoving = ItemsView.SelectedItems.Cast<Items>().ToList();

if (MessageBox.Show($"Вы собираетесь удалить {ItemForRemoving.Count()} записей", "Внимание", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

try

{

ActioRusBDEntities.GetContext().Items.RemoveRange(ItemForRemoving);

ActioRusBDEntities.GetContext().SaveChanges();

MessageBox.Show("Данные удалены");

RefreshData();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

private void RefreshData()

{

var data = ActioRusBDEntities.GetContext().Items.ToList();

ItemsView.ItemsSource= data;

}

}

Как только пользователь попадает на страницу – в методе «public AdminPage» происходит получение данных из БД в наш DataGrid. Далее, прописываем логику переключения страниц, после чего описываем метод «RefreshData», в котором мы снова присваиваем нашему дата гриду уже актуальные данные и таблицы.

Поскольку мы находимся на странице товаров, кнопка «Товары» горит красным цветом, чтобы обозначить на какой именно странице мы находимся. Проверим сортировку товаров по количеству от большего к меньшему (Рисунок 24).

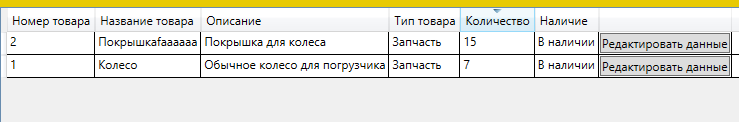


Рисунок 24. Сортировка по убыванию

Уделим внимание удалению данных. Для удаления данных из выбранной строки необходимо выбрать строку(ки) и нажать кнопку «Удалить данные» (Рисунки 25,26).

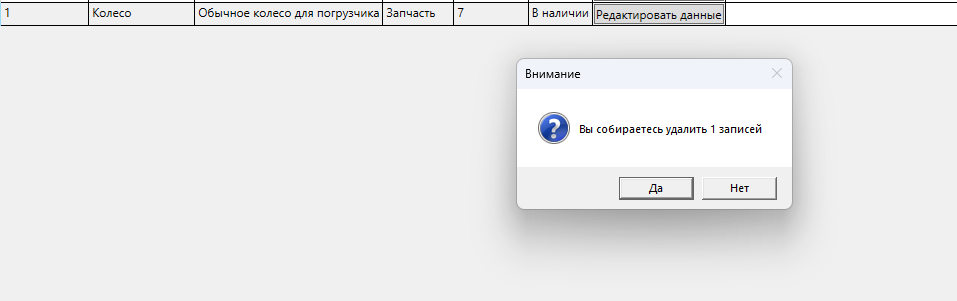


Рисунок 25. Подтверждение удаления

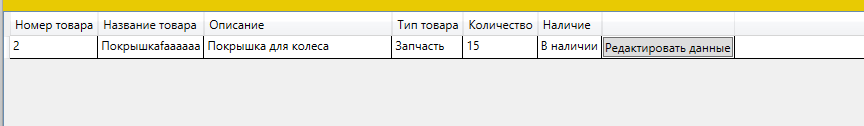


Рисунок 26. Удалённые данные

Для добавления новых и редактирования старых данных нам потреюуется создать новую страницу, специально предназначенную для этого. Создаём страницу и пишем XAML разметку для страницы редактирования и удаления данных:

<Grid>

<StackPanel>

<Label Content="Номер товара"></Label>

<TextBox Text="{Binding ItemID }" ></TextBox>

<Label Content="Название товара"></Label>

<TextBox Text="{Binding ItemName }" ></TextBox>

<Label Content="Описание товара"></Label>

<TextBox Text="{Binding Description }" ></TextBox>

<Label Content="Тип"></Label>

<TextBox Text="{Binding ItemType }" ></TextBox>

<Label Content="Количество на складе"></Label>

<TextBox Text="{Binding Count }" ></TextBox>

<Label Content="Наличие"></Label>

<TextBox Text="{Binding Actual }" ></TextBox>

<Button Content="Сохранить" Background="#FFC8A500" Click="Button\_Click"></Button>

</StackPanel>

</Grid>

Создаём StackPanel, в который пропишем Label и TextBox, каждому TextBox с помощью свойства «Binding» задаём столбец из таблицы Items. На выходе получаем следующую страницу (Рисунок 27).

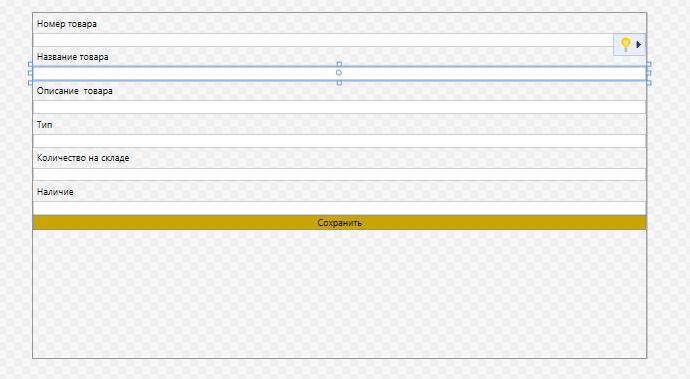


Рисунок 27. Страница редактирования данных

Пропишем логику работы страницы:

public partial class AddOrEditPageItems : Page

{

private bool \_isNew = false;

private Items \_currentitem= null;

public AddOrEditPageItems(Items currentitem)

{

InitializeComponent();

if(currentitem != null)

{

\_currentitem = currentitem;

DataContext= \_currentitem;

}

else

{

\_isNew = true;

\_currentitem = new Items();

DataContext= \_currentitem;

}

}

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

{

if(\_currentitem.ItemID == 0 && !\_isNew)

{

MessageBox.Show("Select row to change");

return;

}

try

{

ActioRusBDEntities context = ActioRusBDEntities.GetContext();

if (\_isNew)

{

context.Items.Add(\_currentitem);

}

context.SaveChanges();

MessageBox.Show("Saved!!!");

Manager.MainFrame.GoBack();

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

}

Здесь всё просто, мы определяем, есть ли данные с предыдущей страницы с предметами и если есть, то мы редактируем данные и сохраняем изменения и переходим на прошлую страницу. В случае, если данных для редактирования нет – добавляем новую запись в таблицу предметов. Запускаем программу и проверяем работу редактирования и добавления:

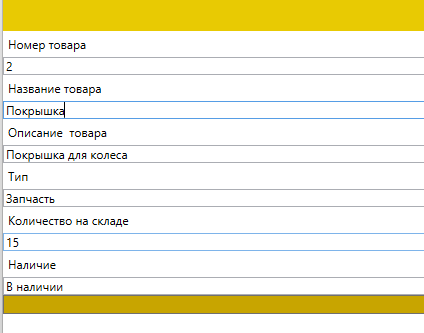


Рисунок 28. Данные ДО редактирования



Рисунок 29. Данные после редактирования. Количество увеличено до 1000 единиц



Рисунок 30. Добавили новый товар

## **ListView**

В нашем WPF приложении мы будем использовать ListView для отображения списка товаров, их описания, количества на складе, и цен, а также добавим функции поиска, сортировки и фильтрации.

Порядок действий схож с предыдущими, создаём страницу, создаём разметку, пишем код страницы. Начнём с кода XAML:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="100"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="10"></ColumnDefinition>

<ColumnDefinition Width="1700"></ColumnDefinition>

<ColumnDefinition Width="200" ></ColumnDefinition>

</Grid.ColumnDefinitions>

<StackPanel Orientation="Horizontal" Grid.Column="1">

<Label Content="Поиск детали" Height="25" FontFamily="Times New Roman" VerticalAlignment="Center"></Label>

<TextBox Name="SearchList" Width="200" Height="25" TextChanged="SearchList\_TextChanged"></TextBox>

<Label Content="Выберите тип детали" Height="25" FontFamily="Times New Roman"></Label>

<ComboBox Name="JenresList" Width="200" Height="25" FontFamily="Times New Roman" Cursor="Hand" Margin="5" SelectionChanged="MarksList\_SelectionChanged"></ComboBox>

<Button Name="ResetFilters" Height="24" Width="126" Content="Сбросить фильтры" Click="ResetFilters\_Click" Cursor="Hand" ></Button>

</StackPanel>

<StackPanel HorizontalAlignment="Left" Grid.Row="1" Grid.Column="2" VerticalAlignment="Top" Orientation="Vertical">

<Image Width="100" Source="\Resources\zaglushka.png"></Image>

<Label Content="Корзина" HorizontalAlignment="Center" FontFamily="Times New Roman" FontWeight="Bold" FontSize="20"></Label>

<DataGrid Name="Cart" AutoGenerateColumns="False" IsReadOnly="True">

<DataGrid.Columns>

<DataGridTextColumn Width="30" Header="№" Binding="{Binding ItemID}"></DataGridTextColumn>

<DataGridTextColumn Width="65" Header="Количество" Binding="{Binding Count}"></DataGridTextColumn>

<DataGridTextColumn Header="Стоимость" Binding="{Binding Cost}"></DataGridTextColumn>

</DataGrid.Columns>

</DataGrid>

<Button Margin="5" Content="Заказать 1 штуку" Click="Button\_Click"></Button>

</StackPanel>

<WrapPanel Grid.Row="1" Grid.Column="1" Width="auto" Visibility="Visible">

<ListView SelectionChanged="CarsViewPanel\_SelectionChanged" Name="CarsViewPanel" Grid.Column="1" Grid.Row="1" ScrollViewer.HorizontalScrollBarVisibility="Disabled" VerticalAlignment="Center" Visibility="Visible">

<ListView.ItemsPanel>

<ItemsPanelTemplate>

<WrapPanel Orientation="Horizontal" Cursor="" ></WrapPanel>

</ItemsPanelTemplate>

</ListView.ItemsPanel>

<ListView.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height = "75" ></RowDefinition >

<RowDefinition Height = "310" ></RowDefinition >

<RowDefinition Height = "auto" ></RowDefinition >

<RowDefinition Height = "auto" ></RowDefinition >

<RowDefinition Height = "auto" ></RowDefinition >

</Grid.RowDefinitions >

<Grid.ColumnDefinitions>

<ColumnDefinition Width="500"></ColumnDefinition>

<ColumnDefinition Width="auto"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image Width = "400" Grid.Row = "1" HorizontalAlignment = "Center" Margin = "5" >

<Image.Source>

<ImageSource >\Resources\zaglushka.png</ImageSource >

</Image.Source>

</Image >

<TextBlock Text = "{Binding ItemID}" VerticalAlignment = "Center" TextAlignment = "Center" Width = "450"

TextWrapping = "Wrap" HorizontalAlignment = "Center" FontSize = "26" Grid.Row = "0" ></TextBlock >

<TextBlock Text = "{Binding ItemName}" Grid.Row = "2" HorizontalAlignment = "Center" FontSize = "20" FontWeight = "Bold" ></TextBlock >

<TextBlock Text = "{Binding ItemType}" Grid.Row = "3" FontSize = "14" HorizontalAlignment = "Right" ></TextBlock >

<TextBlock Text = "{Binding Cost,StringFormat='{}{0:N0} ₽ Руб.'}" Grid.Row = "3" FontSize = "14" HorizontalAlignment = "Left" ></TextBlock >

<TextBlock Text = "{Binding Count}" Grid.Row = "2" FontSize = "14" HorizontalAlignment = "Right" ></TextBlock >

<TextBlock Text="Описание:" Grid.Column="2" Grid.Row="0" VerticalAlignment="Top"></TextBlock>

<TextBlock Text="{Binding Description}" Grid.Row="1" Width="300" Height="1000" Grid.Column="2" TextWrapping="Wrap"></TextBlock>

</Grid >

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<StackPanel Name="ClientDataPAnel" Visibility="Collapsed" Orientation="Vertical" Width="auto">

</StackPanel>

</WrapPanel>

</Grid>

Этот код создает пользовательский интерфейс с использованием элементов управления Grid, StackPanel, Label, TextBox, ComboBox, Button, Image, DataGrid, ListView и TextBlock.

Grid.RowDefinitions и Grid.ColumnDefinitions определяют количество строк и столбцов в сетке, а RowDefinition и ColumnDefinition задают высоту и ширину соответствующих строк и столбцов.

StackPanel располагает элементы управления в одну строку или столбец, а также вложенные панели StackPanel.

Label, TextBox, ComboBox, Button и DataGrid отображаются в StackPanel. ListView с содержимым определенным в DataTemplate отображается в WrapPanel.

TextBlock используется для отображения текстовых данных, в том числе описания деталей.

События, такие как TextChanged, SelectionChanged и Click, связываются с соответствующими обработчиками в коде приложения, чтобы реагировать на действия пользователя.

Теперь приступим к рабочему коду, который будет отвечать за вывод информации в ListView:

public partial class ItemsViewPage : Page

{

private List<Items> allItems;

public ItemsViewPage()

{

InitializeComponent();

var allMarks = ActioRusBDEntities.GetContext().Items.ToList();

JenresList.ItemsSource = allMarks;

JenresList.DisplayMemberPath = "ItemType";

JenresList.SelectedValue = "ItemType";

allItems = ActioRusBDEntities.GetContext().Items.ToList();

UpdateCarsPage();

}

private void UpdateCarsPage()

{

var currentCars = allItems.ToList();

string searchText = SearchList.Text.ToLower();

currentCars = currentCars.Where(p => p.ItemName.ToLower().Contains(searchText) || p.Description.ToLower().Contains(searchText)).ToList();

var selectedItemType = JenresList.SelectedItem as Items;

if (selectedItemType!= null)

{

var selectedTypeItem = selectedItems.ItemType;

currentItem = currentItem.Where(p => p.ItemType).ToList();

}

ItemsViewPanel.ItemsSource = currentCars;

}

private void ActualCars\_Unchecked(object sender, RoutedEventArgs e)

{

UpdateItemsPage ();

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

private void SearchList\_TextChanged(object sender, TextChangedEventArgs e)

{

UpdateItemsPage ();

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

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

{

UpdateItemsPage ();

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

private void ActualCars\_Checked(object sender, RoutedEventArgs e)

{

UpdateItemsPage ();

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

private void PodZakaz\_Unchecked(object sender, RoutedEventArgs e)

{

UpdateItemsPage ();

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

private void PodZakaz\_Checked(object sender, RoutedEventArgs e)

{

UpdateItemsPage();

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

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

{

SearchList.Text = "";

JenresList.SelectedIndex = -1;

// Сбрасываем сортировку

CollectionViewSource.GetDefaultView(CarsViewPanel.ItemsSource).SortDescriptions.Clear();

// Сбрасываем фильтры

CollectionViewSource.GetDefaultView(CarsViewPanel.ItemsSource).Filter = null;

ClientDataPAnel.Visibility = Visibility.Collapsed;

// Обновляем список товаров на странице

UpdateCarsPage();

}

private void Page\_Loaded(object sender, RoutedEventArgs e)

{

}

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

{

var selected = ItemsViewPanel.SelectedItem as Items;

if (selected != null)

{

var view = CollectionViewSource.GetDefaultView(CarsViewPanel.ItemsSource);

view.Filter = item => item == selected;

ClientDataPAnel.Visibility = Visibility.Visible;

}

}

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

{

var selected = ItemsViewPanel.SelectedItem as Items;

decimal discount = selected.Cost.Value \* (decimal)1;

decimal discountedPrice = selected.Cost.Value - discount;

// Выводим данные в MessageBox

string message = "Название товара: " + selected.ItemName + "\n" +

"Тип: " + selected.ItemType + "\n" +

"Количество: " + selected.Count + "\n" +

"Описание: " + selected.Description + "\n" +

"Цена: " + discount.ToString("#.##") + " руб. ";

MessageBoxResult result = MessageBox.Show(message, "Подтверждение данных", MessageBoxButton.YesNo, MessageBoxImage.Question);

var newItemBuy = new CartTable

{

ItemID = 1,

Cost = discountedPrice,

Count = 1,

ClientID = 1,

};

ActioRusBDEntities.GetContext().CartTable.Add(newTestDrive);

ActioRusBDEntities.GetContext().SaveChanges();

Cart.ItemsSource = ActioRusBDEntities.GetContext().CartTable.ToList();

if (result == MessageBoxResult.No)

{

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

}

private bool isSortedAscending = false;

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

}

}

}

Этот код реализует логику отображения списка товаров в WPF-приложении. Класс ItemsViewPage является страницей, на которой размещены элементы управления, позволяющие отображать и фильтровать список товаров.

В конструкторе ItemsViewPage инициализируется список всех товаров, которые затем используются для фильтрации. Также происходит настройка комбобокса JenresList, в котором отображаются типы товаров, и заполнение списка автомобилей по умолчанию с помощью метода UpdateCarsPage().

Метод UpdateCarsPage() фильтрует список товаров по поисковому запросу и выбранной марке, а затем обновляет список товаров на странице.

После каждого изменения вызывается метод UpdateCarsPage(), который фильтрует список товаров и обновляет его на странице.

Кнопка ResetFilters\_Click сбрасывает все фильтры и сортировки и вызывает метод UpdateCarsPage() для обновления списка товаров на странице.

Метод ItemsViewPanel\_SelectionChanged обрабатывает событие выбора товара из списка и отображает подробную информацию о выбранном товаре в элементе управления ClientDataPAnel. Кнопка Button\_Click добавляет выбранный товар в корзину и выводит детальную информацию о товаре в MessageBox.

Запускаем приложение и смотрим результат:

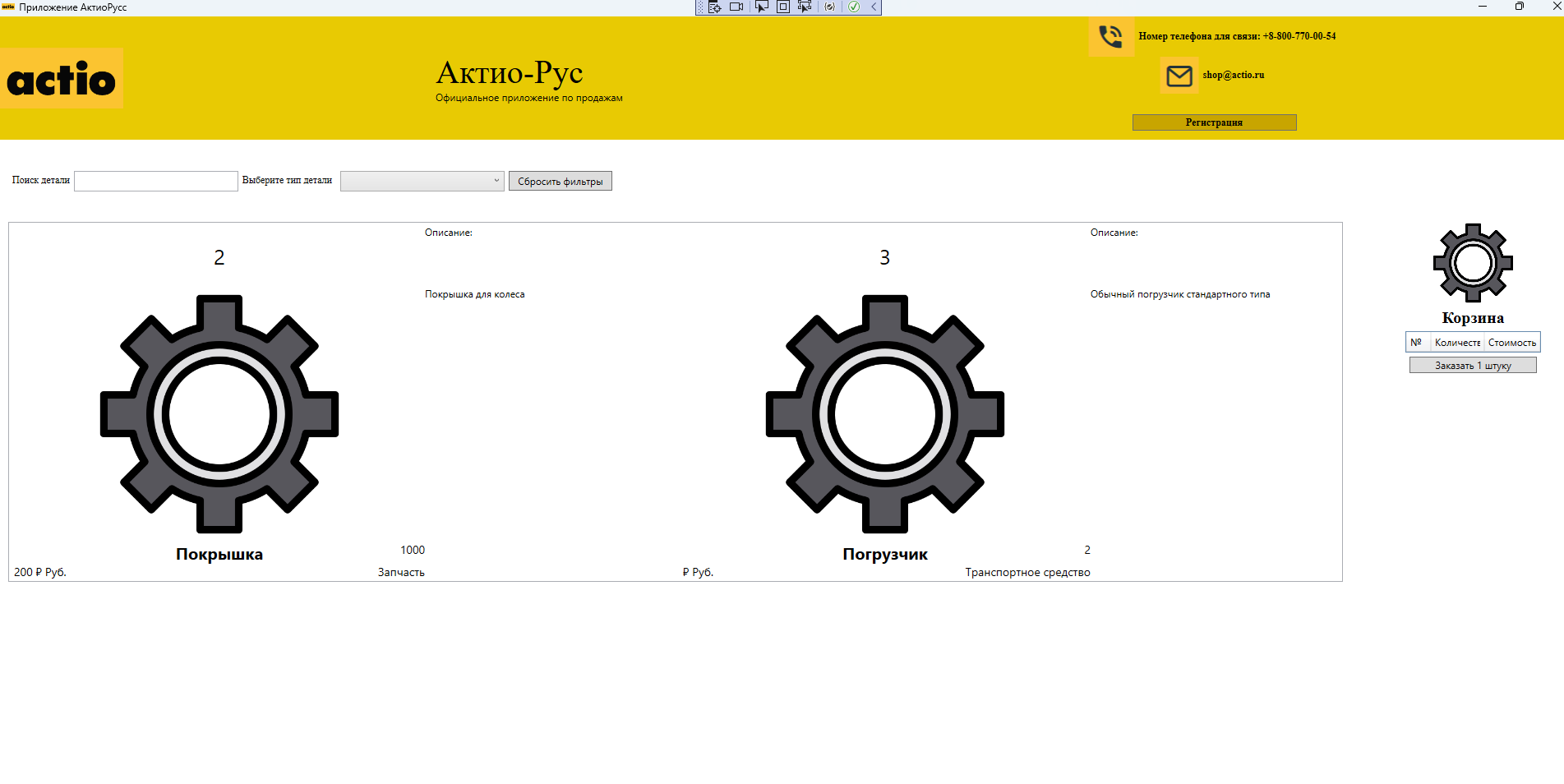


Рисунок 31. ListView всех товаров.

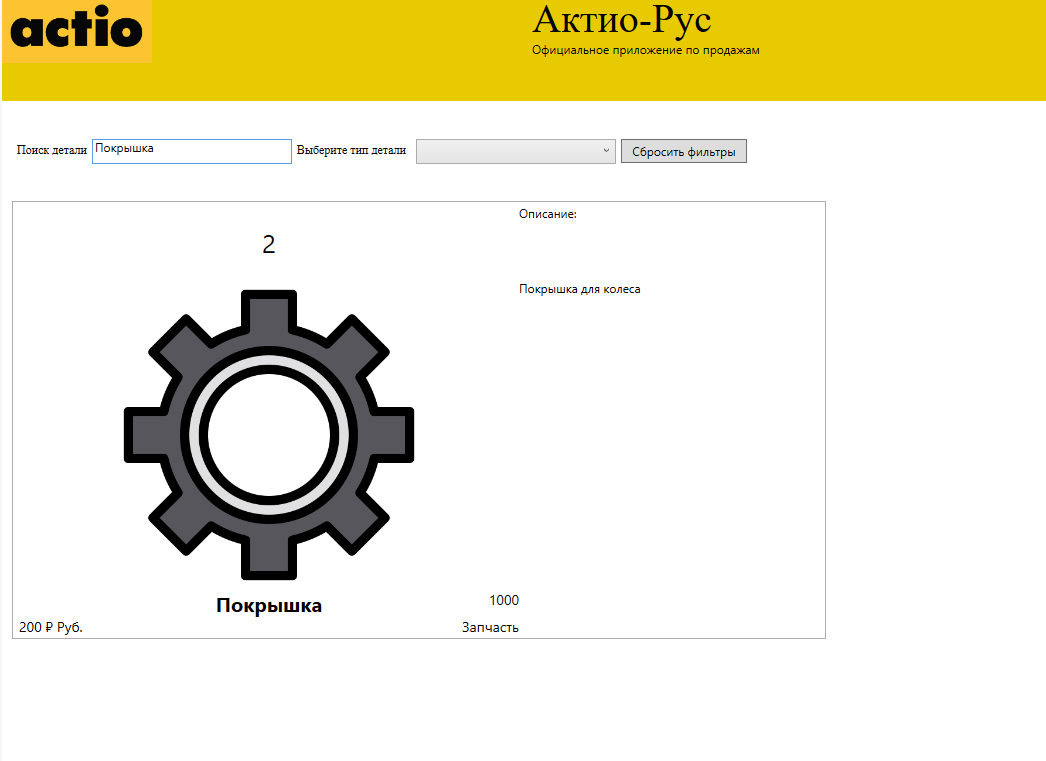


Рисунок 32. Поиск «Покрышка»

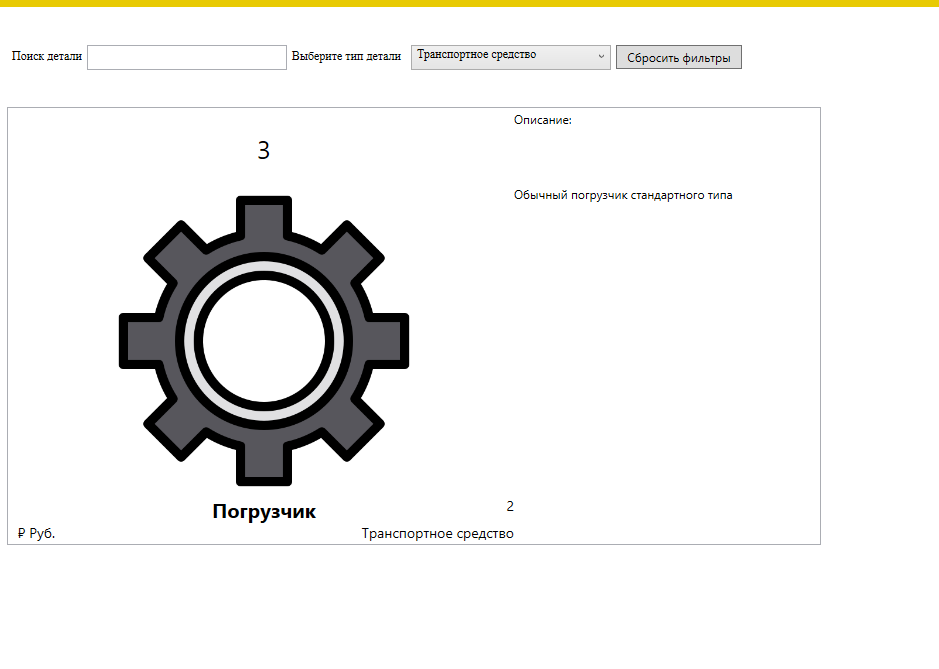


Рисунок 32. Фильтрация только транспортных средств

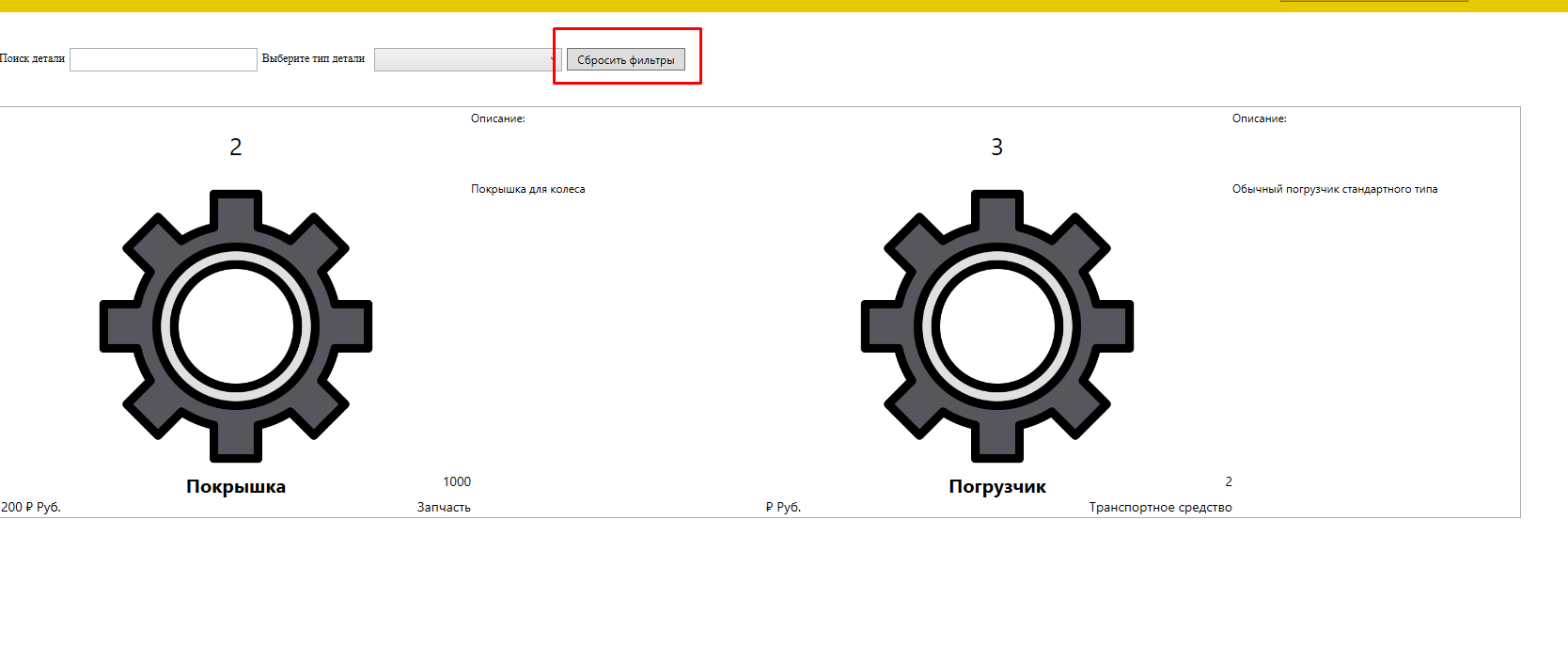


Рисунок 33. Все фильтры и сортировки сброшены

Страница ListView рассмотрена

## **Корзина**

Корзина будет представлять из себя DataGrid, в который будут выводиться заказанные тест драйвы конкретного пользователя. Клиент заказывает тест драйв и в базу данных заносится номер товара, номер клиента, количество и стоимость.

По нажатию кнопки «Заказать 1 штуку» будет происходить выполнение следующего кода:

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

{

var selected = CarsViewPanel.SelectedItem as Items;

decimal discount = selected.Cost.Value \* (decimal)1;

decimal discountedPrice = selected.Cost.Value - discount;

// Выводим данные в MessageBox с учетом скидки

string message = "Название товара: " + selected.ItemName + "\n" +

"Тип: " + selected.ItemType + "\n" +

"Количество: " + selected.Count + "\n" +

"Описание: " + selected.Description + "\n" +

"Цена: " + discount.ToString("#.##") + " руб. ";

MessageBoxResult result = MessageBox.Show(message, "Подтверждение данных", MessageBoxButton.YesNo, MessageBoxImage.Question);

var newTestDrive = new CartTable

{

ItemID = 1,

Cost = discountedPrice,

Count = 1,

ClientID = 1,

};

ActioRusBDEntities.GetContext().CartTable.Add(newTestDrive);

ActioRusBDEntities.GetContext().SaveChanges();

Cart.ItemsSource = ActioRusBDEntities.GetContext().CartTable.ToList();

if (result == MessageBoxResult.No)

{

// Если данные неверны, можно снова скрыть ClientDataPAnel

ClientDataPAnel.Visibility = Visibility.Collapsed;

}

Выбранный товар из ListView пекреносится в переменную «selected», далее переменная decimal высчитывает стоимость одной единицы товара, затем в переменной «message» Передаются данные столбцов из таблицы товаров и пользователя спрашивают, всё ли верно? После чего в таблицу вносятся полученные данные и происходит присвоение содержимого таблицы в датагрид( Рисунок 34).

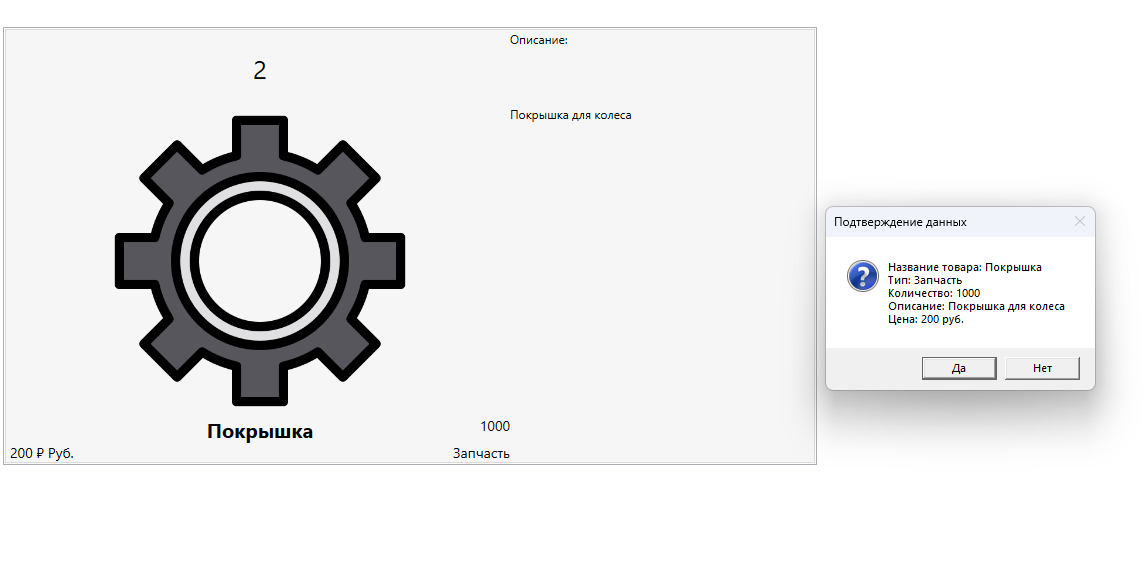


Рисунок 34. Подтверждение данных

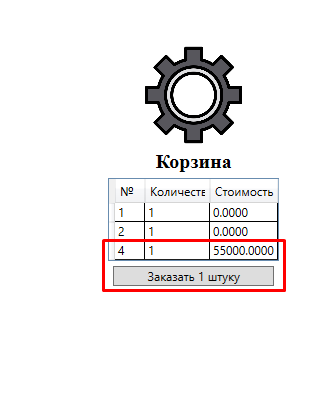


Рисунок 35. Корзина клиента

# **Заключение**

В ходе прохождения производственной (преддипломной) практической подготовки в ООО «Актио Рус» мною было выполнено индивидуальное задание по производственной практике и достигнута цель по приобретению знаний, умений и навыков по ПМ. 01 Разработка модулей программного обеспечения для компьютерных систем, ПМ. 02 Осуществление интеграции программных модулей:

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

ПК 1.6. Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент.

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК 2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования

Мною освоены следующие общие компетенции:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей.

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 08. Использовать средства физической культуры для сохраления и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 09. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языке.

ОК 11. Планировать предпринимательскую деятельность в профессиональной сфере.

# **Список литературы**

1. "Программирование на C# 7 для профессионалов" А. Троелсен, Ф. Джепикс. - издание: 2018.
2. "WPF 4.5 с примерами на C# 5.0" А. Петзольд. - издание: 2014.
3. "WPF в действии. Пер. с англ." Джон Госсман, Адам Нейро, Джон Стевенс. - издание: 2011.
4. "SQL Server 2017. Руководство для начинающих" Уильям Р. Станек. - издание: 2018.
5. "Программирование баз данных SQL Server 2017. Библиотека программиста" Джейсон Прайс. - издание: 2018.
6. "Microsoft SQL Server 2016. Полное руководство" Брайан Ларсон. - издание: 2016.
7. "Самоучитель SQL Server 2012 Express. Полное руководство для начинающих" Эндрю Брэйнд. - издание: 2013.
8. "Самоучитель Microsoft SQL Server 2014" Кевин Ксир, Питер Лернер, Скотт Шоуп. - издание: 2015.
9. "SQL Server 2017. Практическое руководство для начинающих" Конрад Кинг, Дэниел Киркнесс. - издание: 2018.
10. "Учимся работать с Microsoft SQL Server 2012" Брайан Ларсон. - издание: 2013.