**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**

Найденышевой Алисы Сергеевны

(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 начальница Найденышева М.С. \_\_\_\_\_\_\_\_\_\_

*должность, ФИО*

Люберцы 2023

**ХАРАКТЕРИСТИКА ПРАКТИКАНТА**

Найденышевой Алисы Сергеевны

(Ф.И.О. практиканта)

Работала 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 показала себя с положительной стороны. В течение всего периода Найденышева Алиса Сергеевна внимательно и ответственно относилась к выполняемой работе.

В отношениях с коллегами Найденышева Алиса Сергеевна проявила 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 Найденышевой Алисы Сергеевны положительно.

Руководитель практической подготовки от организации

Начальница АХО Найденышева М. С.

*(должность, ФИО)*

М.П.

**Описание организации**

1. Название организации: ООО «Магнум»
2. Адрес: 140187, Московская Область, г Жуковский, ул Мясищева, д. 1
3. Адрес для писем: 140187, Московская Область, г Жуковский, ул Мясищева, д. 1
4. Телефон: +7 495-995-56-32
5. Факсы -
6. E-mail: info@magnumsport.com
7. Отрасль: Розничная торговля.
8. Год основания: 2002г.
9. Форма собственности: Общество с ограниченной ответственностью
10. История:

Компания «Магнум» присутствует на рынке с 1995 года, профессионально занимаясь комплексной реализацией объектов спортивного назначения от эскизных проработок до ввода в эксплуатацию готовых, оснащенных всем необходимым оборудованием, спортивных сооружений. За это время было реализовано более тысячи спортивных объектов — от открытых спортивных площадок до спортивных комплексов и стадионов, принимающих соревнования мирового уровня. Среди наших конкурентных преимуществ можно отметить: накопленный опыт проектирования и строительства, сильную команду спортивных технологов, эксклюзивные дилерские соглашения с ведущими зарубежными и российскими производителями спортивно-технологического оборудования, индивидуальный подход к каждому заказчику.

**Описание подразделения, в котором была пройдена практика**

1. Название подразделения: ООО Магнум
2. Руководитель подразделения: Начальница Найденышева М. С
3. Куратор практики: Найденышева М. С
4. Структура и функции подразделения (в краткой форме должностные обязанности): Спортивно-строительная компания ООО «Магнум» занимается такими делами как:

- ПРОЕКТИРОВАНИЕ СПОРТИВНЫХ СООРУЖЕНИЙ

- СТРОИТЕЛЬСТВО СПОРТИВНЫХ ОБЪЕКТОВ ПОД КЛЮЧ

- ОСНАЩЕНИЕ

- ПРОДАЖА СТРОИТЕЛЬНЫХ МАТЕРИАЛОВ

5. Резюме руководителя подразделения

− Высшее экономическое образование.

− Стаж работы в данной сфере 8 лет

**Введение**

[Описание предметной области 6](#_Toc132108554)

[Постановка задачи 7](#_Toc132108555)

[Создание ER-диаграммы 9](#_Toc132108556)

[User Story 11](#_Toc132108557)

[Use case 12](#_Toc132108558)

[Разработка базы данных 13](#_Toc132108559)

[Разработка WPF приложения 16](#_Toc132108560)

[Код приложения 22](#_Toc132108561)

[Страница авторизации 22](#_Toc132108562)

[Вывод данных в DataGrid 33](#_Toc132108563)

[ListView 40](#_Toc132108564)

[Корзина 47](#_Toc132108565)

[Заключение 50](#_Toc132108566)

[Список литературы 52](#_Toc132108567)

# **Описание предметной области**

Разработка WPF (Windows Presentation Foundation) приложения для работы с информационной сетью предприятия по продаже строительных материалов - это процесс создания программного обеспечения, которое будет использоваться для управления бизнес-процессами в компании, занимающейся продажей строительных материалов.

Данная предметная область включает в себя несколько ключевых аспектов, которые необходимо учитывать при разработке WPF приложения. Во-первых, это работа с базами данных, которые хранят информацию о клиентах, товарах, заказах и т.д. Для эффективной работы с базами данных необходимо использовать соответствующие технологии и инструменты, такие как Entity Framework, LINQ и SQL Server.

Во-вторых, важно учитывать особенности бизнес-процессов компании. Например, приложение должно предоставлять возможность быстро и удобно осуществлять поиск и фильтрацию товаров, обрабатывать заказы, формировать отчетность и т.д.

В-третьих, приложение должно быть удобным и интуитивно понятным для пользователей. Это может быть достигнуто путем использования соответствующих элементов управления, графического дизайна, а также различных функций и опций, которые облегчают работу с приложением.

Наконец, необходимо учитывать вопросы безопасности и защиты данных. Приложение должно быть защищено от несанкционированного доступа и взлома, а также должно соответствовать соответствующим стандартам и требованиям по защите данных.

Разработка WPF приложения для работы с информационной сетью предприятия по продаже одежды и обуви - это сложный процесс, который требует учета множества различных факторов и аспектов. Однако, правильно спроектированное и реализованное приложение может значительно улучшить бизнес-процессы компании и повысить эффективность работы персонала.

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

**Microsoft SQL Server Management Studio (SSMS)** - это среда для управления базами данных Microsoft SQL Server. Она предоставляет графический интерфейс для управления экземплярами SQL Server, а также для создания, изменения и удаления баз данных, таблиц, представлений, процедур и других объектов баз данных.

**Visual Studio 2022** - это среда разработки приложений и веб-сайтов, созданная компанией Microsoft для операционных систем Windows и мобильных устройств. Среда поддерживает множество языков программирования, включая C#, C++, Visual Basic, Python, JavaScript и другие. Visual Studio предоставляет разработчикам множество инструментов для создания, отладки, тестирования и развертывания приложений, ускоряя процесс разработки и упрощая взаимодействие между членами команды.

# 

# **Постановка задачи**

База данных будет использоваться для учета продаж и поставок строительных материалов в магазине "Магнум" 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 магазина "Магнум" упрощает процедуру поиска необходимой информации о заказах, деталях, и ценах на материалы. С ее помощью сотрудник может легко узнать информацию о проданных материалах, клиент может просматривать существующий в наличии ряд предметов для строительства, а руководитель отслеживать деятельность сотрудников и динамику продаж.

На рисунке 1 отобразим набор задач, который необходимо решить на производственной практической подготовке:

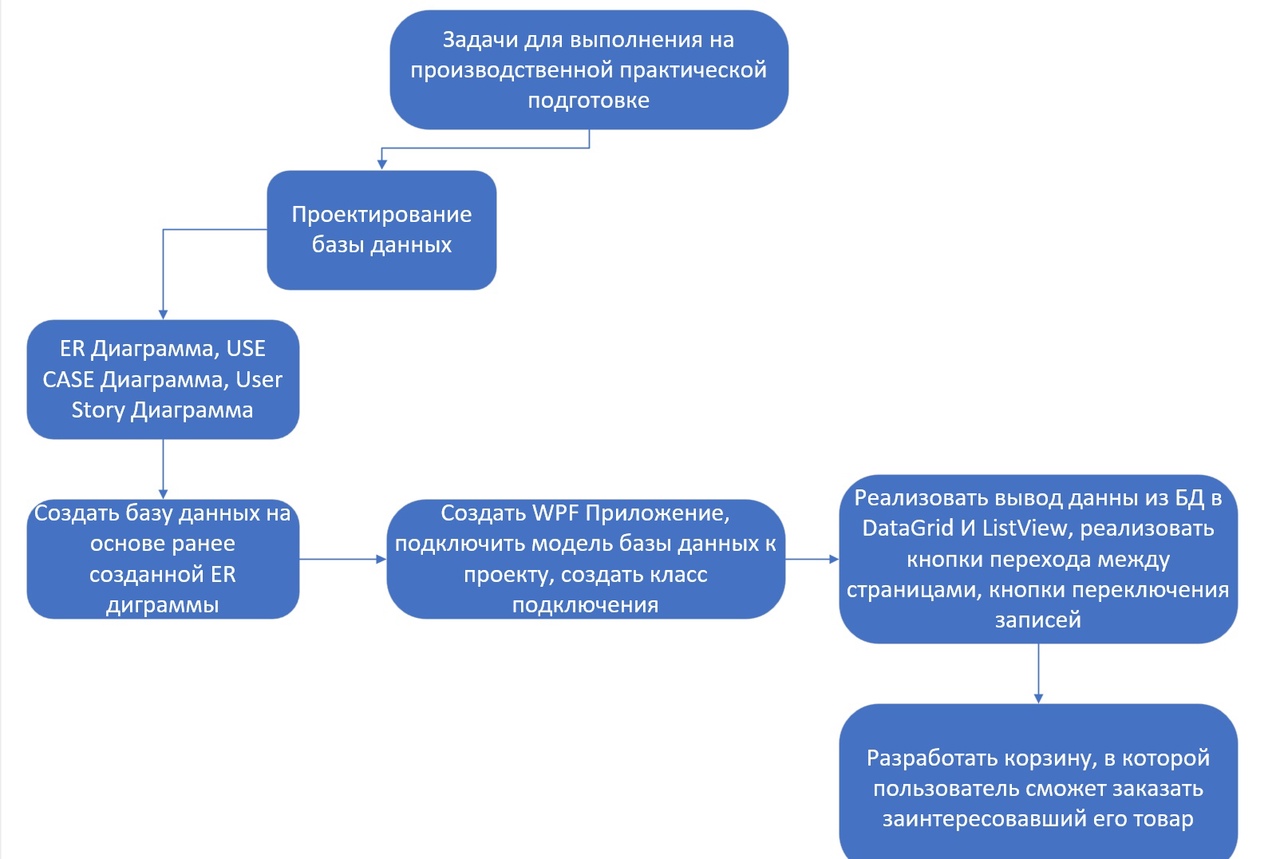


Рисунок 1. Набор задач

# **Создание ER-диаграммы**

Visio - это программное обеспечение, которое используется для создания диаграмм, схем и других визуальных представлений. Оно разработано компанией Microsoft и является частью пакета Microsoft Office.

С помощью Visio можно создавать различные типы диаграмм, такие как блок-схемы, сетевые диаграммы, организационные диаграммы, планы этажей и другие. Visio предоставляет широкий набор инструментов для создания и редактирования диаграмм, а также возможность интеграции с другими программами Microsoft Office, такими как Excel и PowerPoint.

В Visio необходимо создать представление информационной системы автосалона, для этого нужно запустить программу и выбрать нужный шаблон (Рисунок 2).

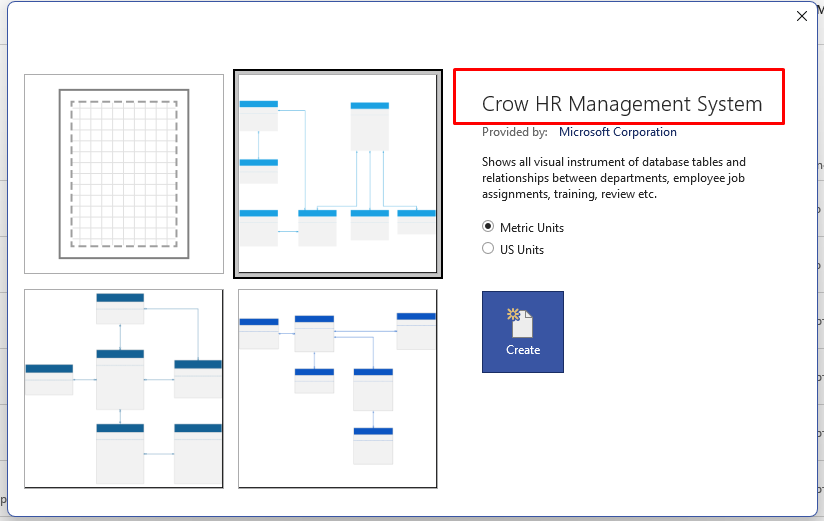


Рисунок 2. Выбор шаблона для разработки диаграммы

С помощью представленных наборов фигур составим таблицы, необходимые для работы с информационной системой отдела продаж магазина строительных материалов (Рисунок 3).

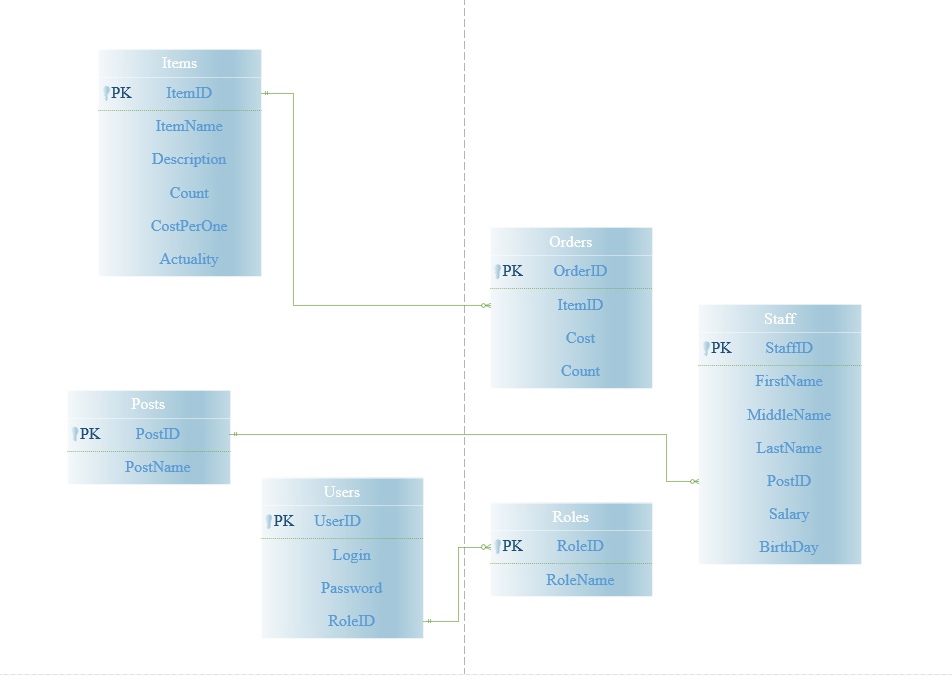


Рисунок 3. Готовая ERD диаграмма

На диаграмме отображены основные таблицы для работы с ИС отдела продаж магазина строительных материалов, а именно:

* Товары
* Заказы
* Должности
* Пользователи
* Роли
* Работники

# **User Story**

User Story (история пользователя) - это краткое и простое описание требований к функциональности продукта или сервиса, написанное в виде истории от лица пользователя. Эта форма позволяет команде разработчиков понять задачу, которую необходимо выполнить, а также для какой аудитории это будет полезно.

При создании User Stories нужно использовать методологию разработки программного обеспечения BDD (Behavior-driven development), которая является 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 создания User Story следует использовать следующий шаблон:

История: Поиск нужного товара

Как Менеджер

Я хочу иметь возможность найти нужный мне товар

Чтобы я мог увидеть интересующий меня товар, его описание, наличие и стоимость

Сценарий: Нужный элемент найден, получена полная информация о нём

Дано: менеджер открыл программу и ввёл название нужного элемента

Когда страница с товарами открыта

Тогда менеджер получит полную информацию о товаре

Сценарий: Нужный элемент не найден, не получена информация о товаре

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) представляет собой описание того, как конкретный пользователь может использовать приложение, систему или продукт для достижения своей 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 актеры 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 режиссера

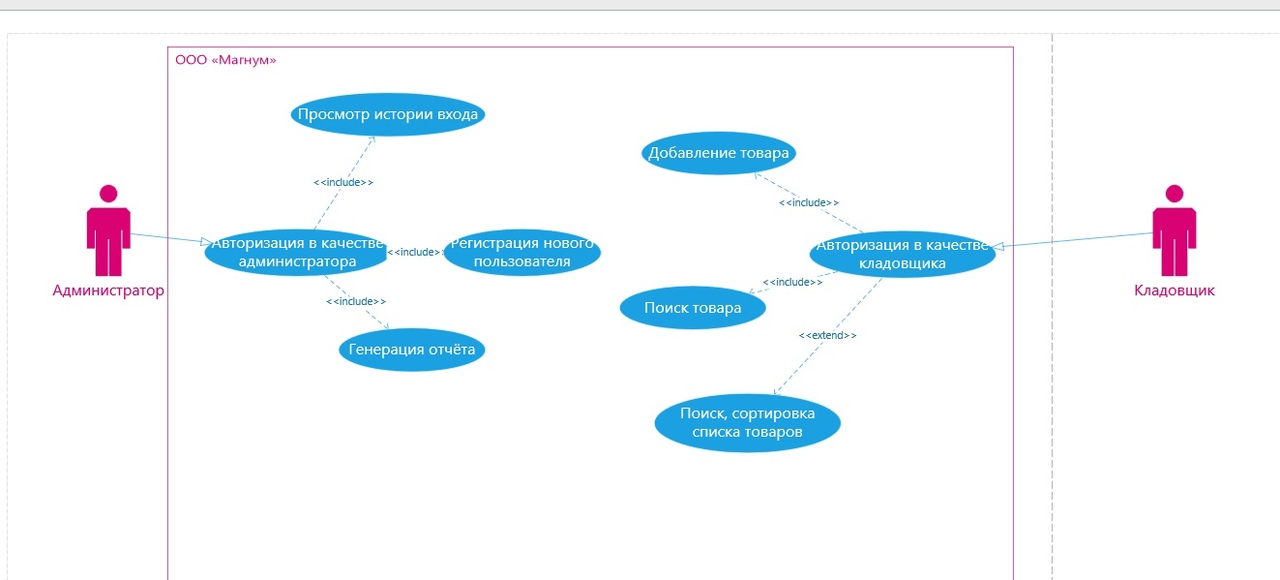


Рисунок 4. Use case отдела продаж

# **Разработка базы данных**

После подключения к ЯДРУ БД нажимаем правой кнопкой мыши по папке «DataBases» и выбираем «New Database» (Рисунок 5).

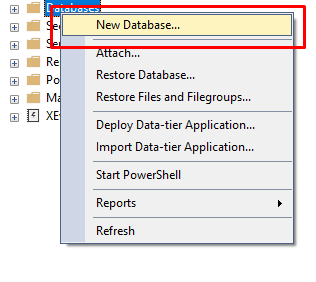


Рисунок 5. Создание новой базы данных.

Вводим название БД и переходим к созданию таблиц (Рисунок 6).

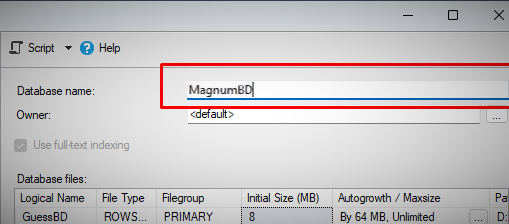


Рисунок 6. Выбор имени БД.

После успешного создания БД нажимаем правой кнопкой мыши по папке «Tables» и выбираем пункт «New Table» (Рисунок 7).

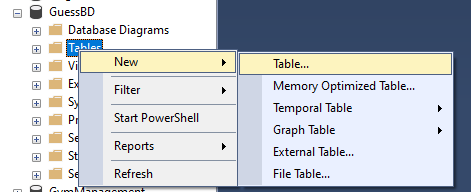


Рисунок 7. Создание новой таблицы.

Перед нами открывается окно с созданием таблица, в которой нам необходимо выбрать название столбца и его тип данных (Рисунок 8).

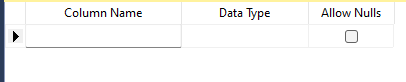


Рисунок 8. Создание таблицы.

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 9).

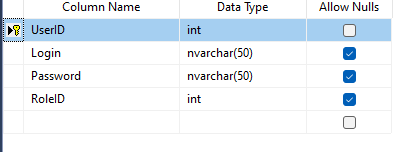


Рисунок 9. Таблица пользователей.

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 10).

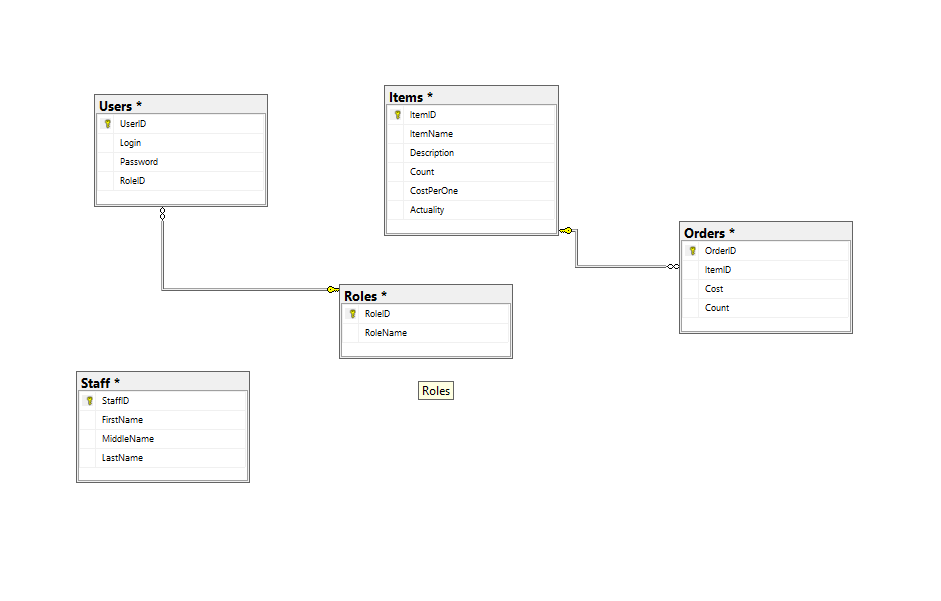


Рисунок 10. Диаграмма базы данных созданная в SQL Server Management Studio.

# **Разработка WPF приложения**

WPF (Windows Presentation Foundation) - это технология разработки графического интерфейса пользователя (GUI) для приложений под Windows, которая позволяет создавать многофункциональные и интерактивные приложения.

Для создания приложения переходим в заранее установленное приложение Visual Studio 2022 и создаем новый проект с использованием шаблона WPF (см. Рисунок 11).

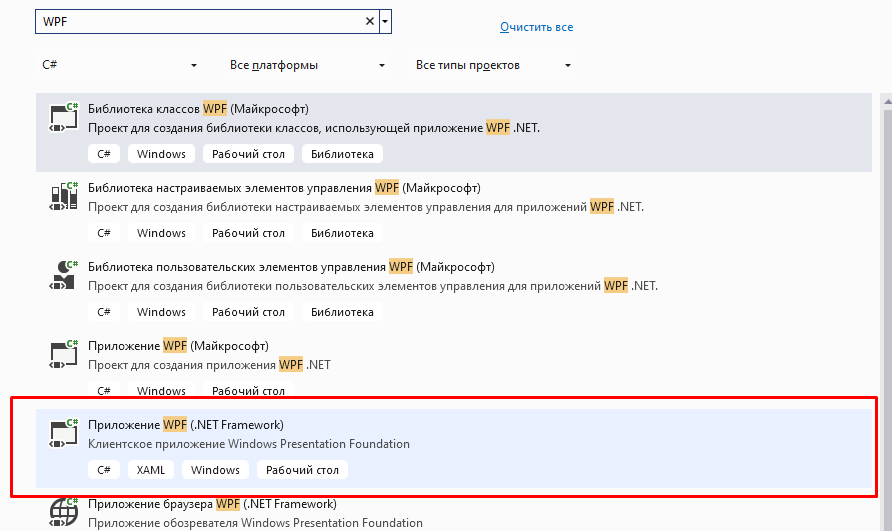


Рисунок 11. Создание нового WPF проекта

Вводим название проекта и жмём кнопку «Создать» (Рисунок 12).

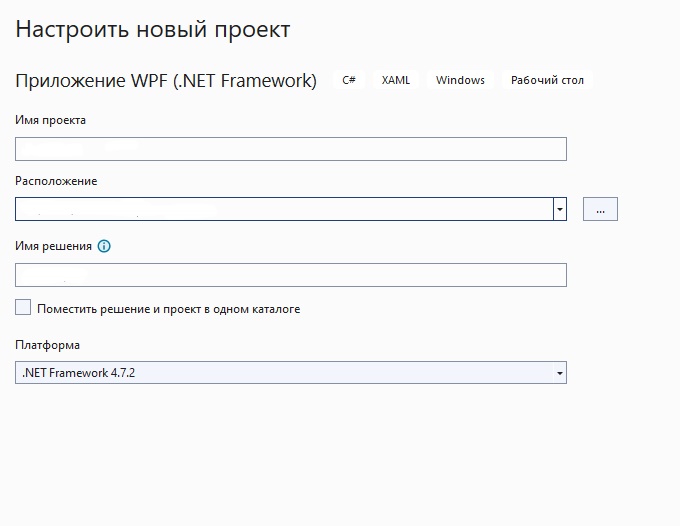


Рисунок 12. Название проекта

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 13).

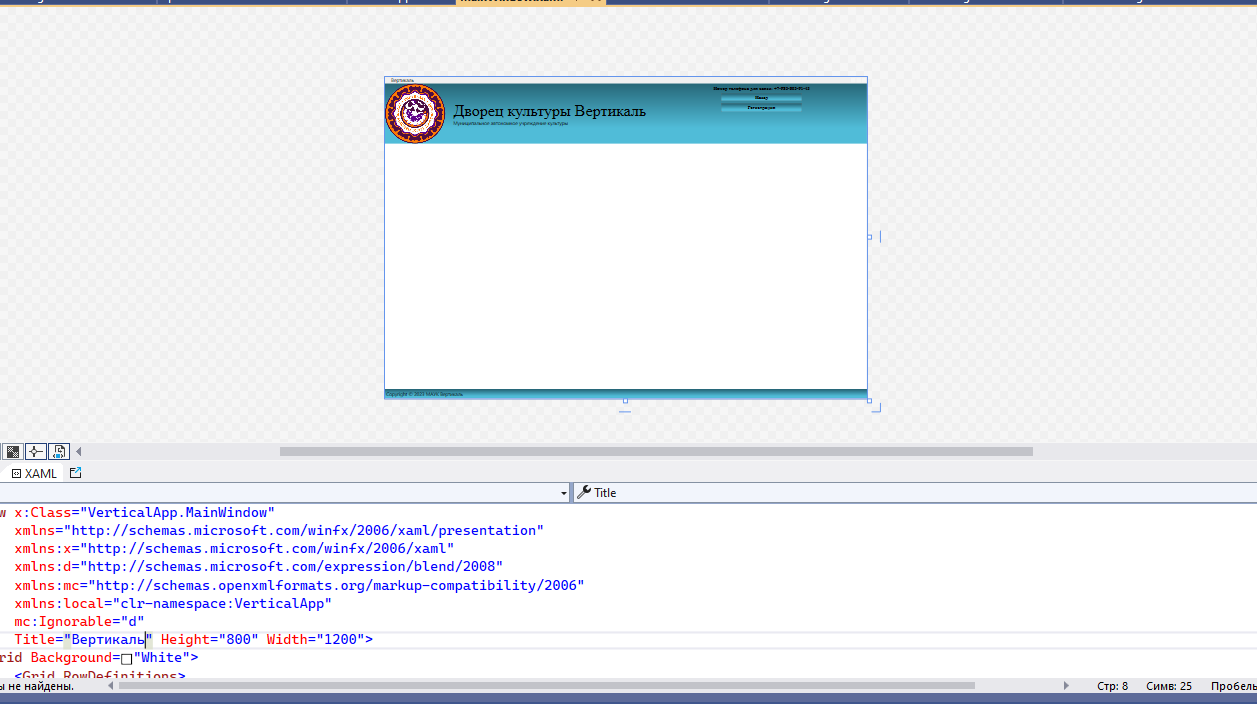


Рисунок 13. Главное окно MainWindow

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

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="150"></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Grid Background="Red">

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<StackPanel Grid.Column="0" Orientation="Horizontal">

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

<Label Content="cтроительные&#xD;&#xA; материалы" VerticalAlignment="Center" FontFamily="Impact" FontSize="18"></Label>

</StackPanel>

<StackPanel Grid.Column="2">

<Button Content="Назад" Width="100" Background="DarkRed" Margin="5" Click="Button\_Click"></Button>

</StackPanel>

</Grid>

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

</Frame>

</Grid>

Мы используем XAML разметку, чтобы задать размеры главного окна "MainFrame", "хэдэра" и "футэра". "Шапку" нашего окна мы разбиваем на три части: в первой размещаем логотип предприятия, во второй - название предприятия, а в третьей - кнопка "Назад" .

Для создания страницы авторизации нам нужно нажать правую кнопку мыши на нашем решении и выбрать опцию "Добавить" -> "Создать страницу WPF" (рис.14)

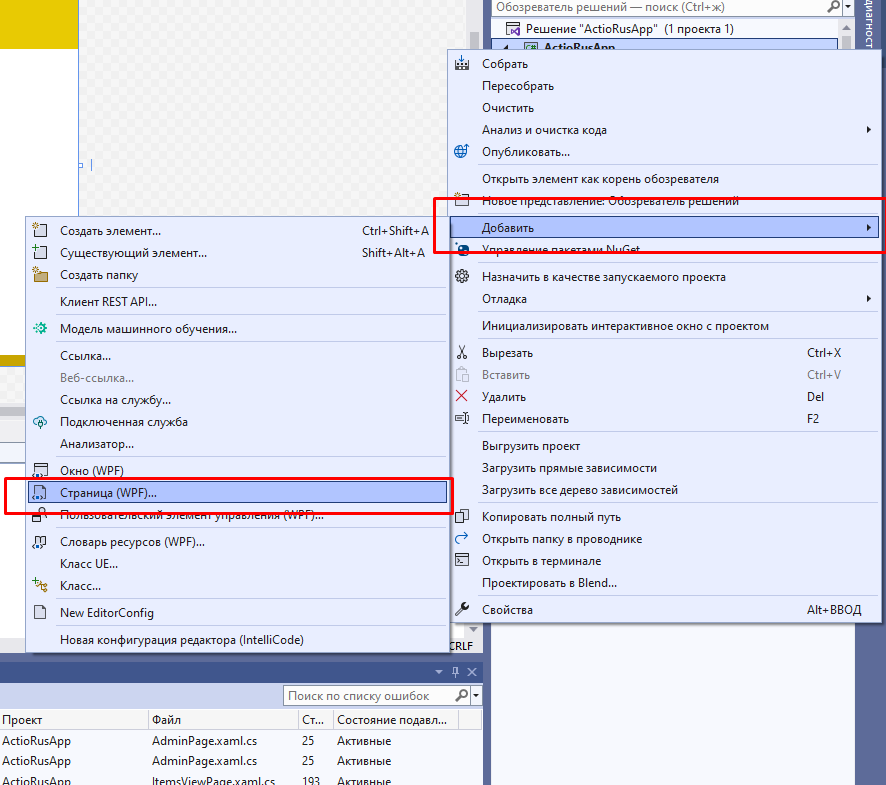


Рисунок 14. Создание страницы авторизации

Аналогичным образом создаём страницу авторизации, добавляя поля для ввода логина, пароля и кнопку «Войти» (Рисунок 15).

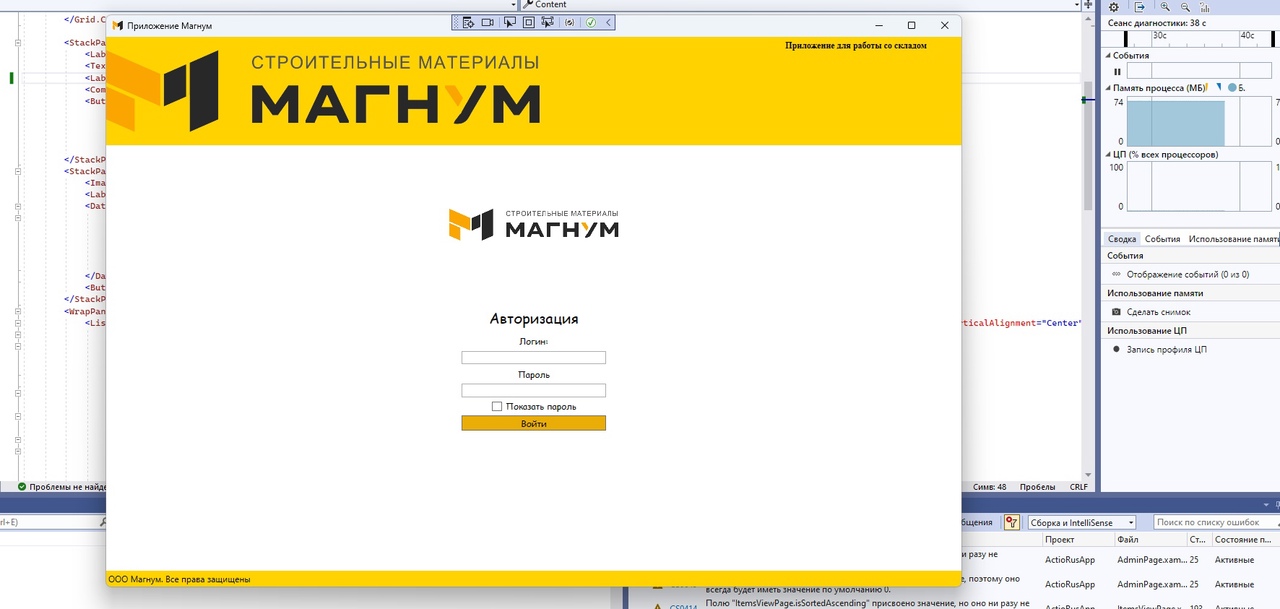


Рисунок 15. Страница авторизации

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 16).

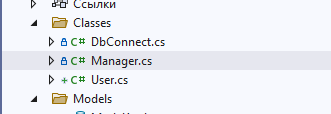


Рисунок 16. Папка Classes и класс Manager

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 17).

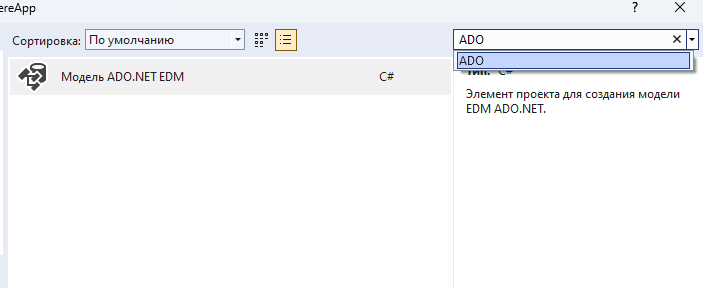


Рисунок 17. Добавление модели БД

Следуя всем инструкциям, вводим название нашей БД и добавляем в папку Models (Рисунок 18).

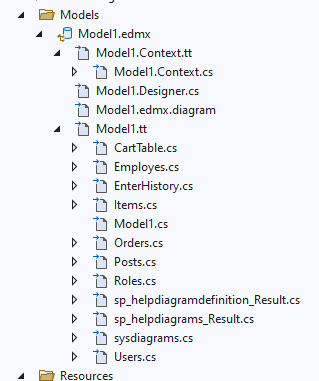


Рисунок 18. Подключенная модель

Теперь, написанный нами ранее код подключения, а именно класс DbConnect будет работать как положено.

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

public partial class MagnumBDEntities : DbContext

{

private static MagnumBDEntities \_context;

public MagnumBDEntities ()

: base("name= MagnumBDEntities ")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

public static MagnumBDEntities 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;

Мы добавили строчки запроса «контекста» из БД и определение текущего пользователя равному нулю.

# **Код приложения**

## **Страница авторизации**

Графический интерфейс страницы авторизации в WPF приложении позволяет пользователю ввести свои учетные данные (логин и пароль) для доступа к приложению. Он содержит текстовые поля для ввода имени пользователя и пароля, кнопку "Войти", которая отправляет данные авторизации на сервер, а также чекбокс "Показать пароль".

Для создания функционала страницы нужно выполнить следующие задачи: подключение базы данных, проверка введенного пароля, реализация функции "Показать пароль", написание логики кнопки "Войти", а также добавление капчи для защиты от неудачных попыток входа.

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 для демонстрации результата:

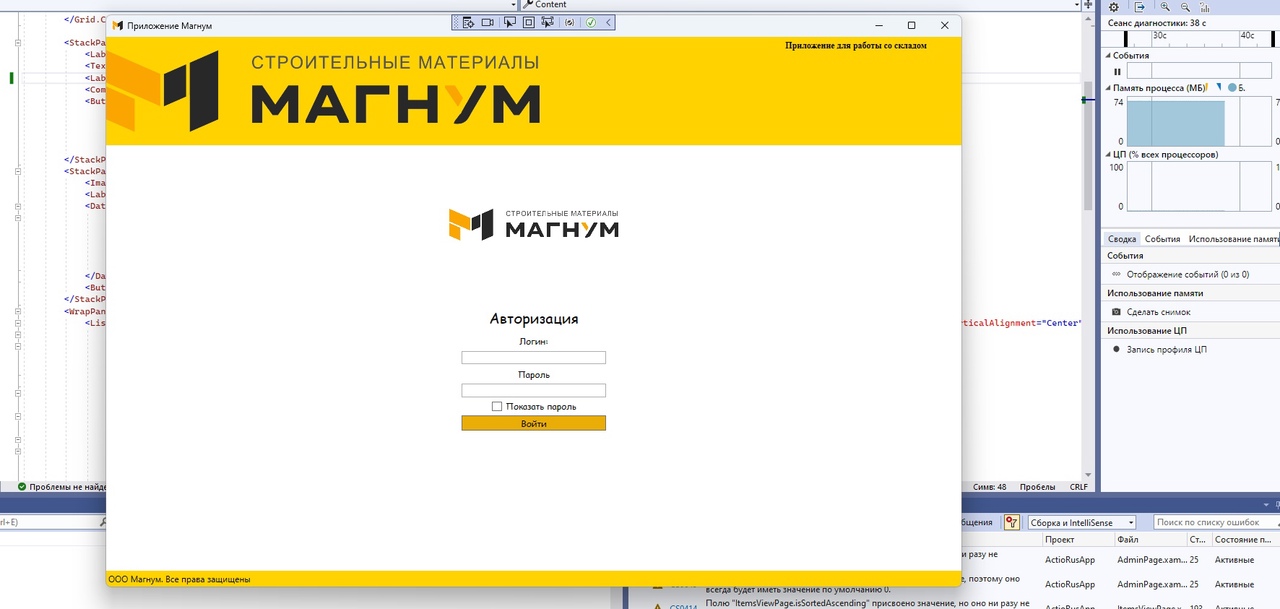


Рисунок 19. Показ пароля недоступен, пока не будет введён

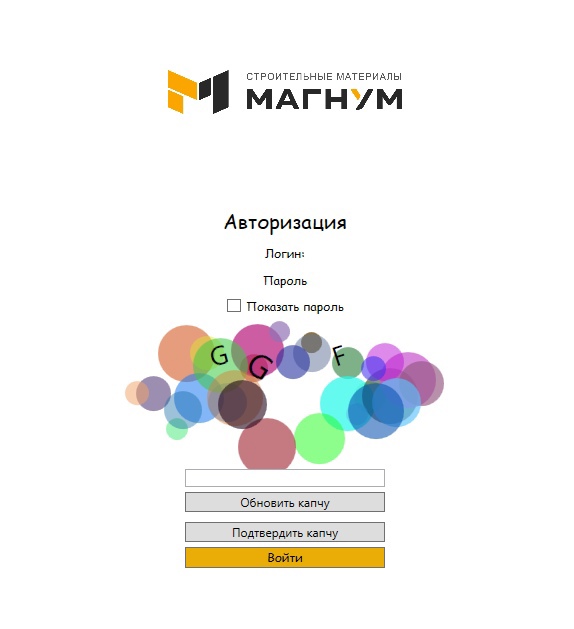


Рисунок 20.  Не успешная попытка войти

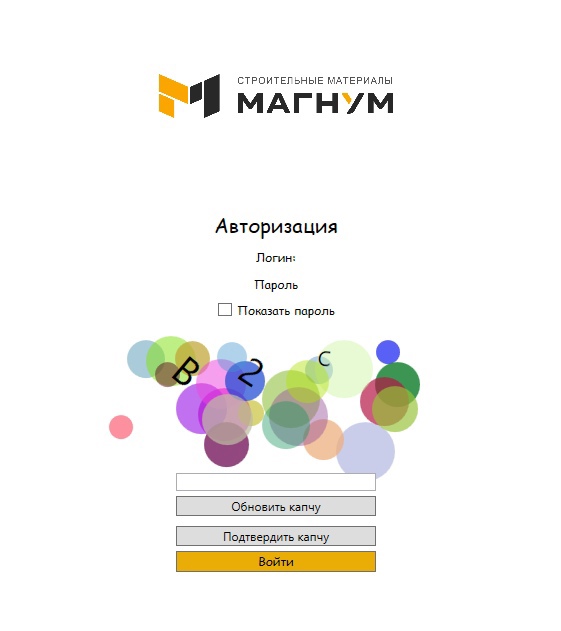


Рисунок 21.  Вторая ошибка авторизации

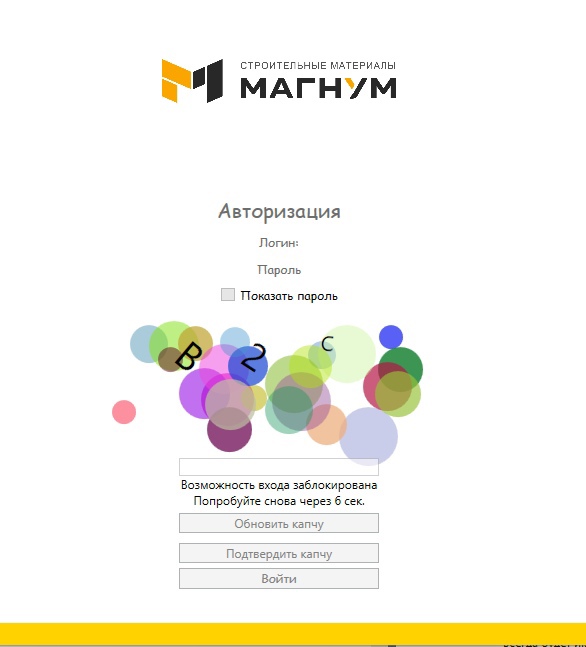


Рисунок 22.  Блокировка входа

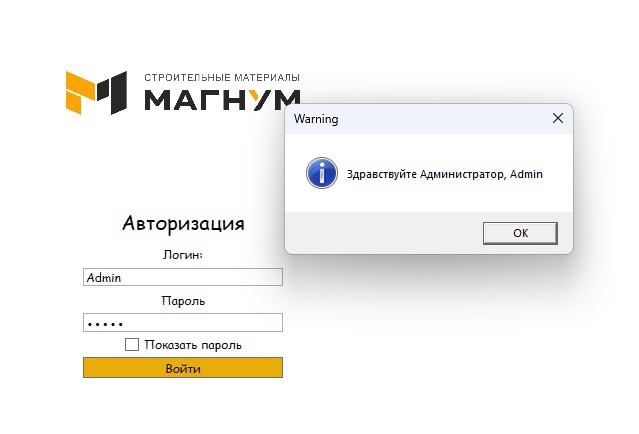


Рисунок 23. Успешный вход

## **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 24).

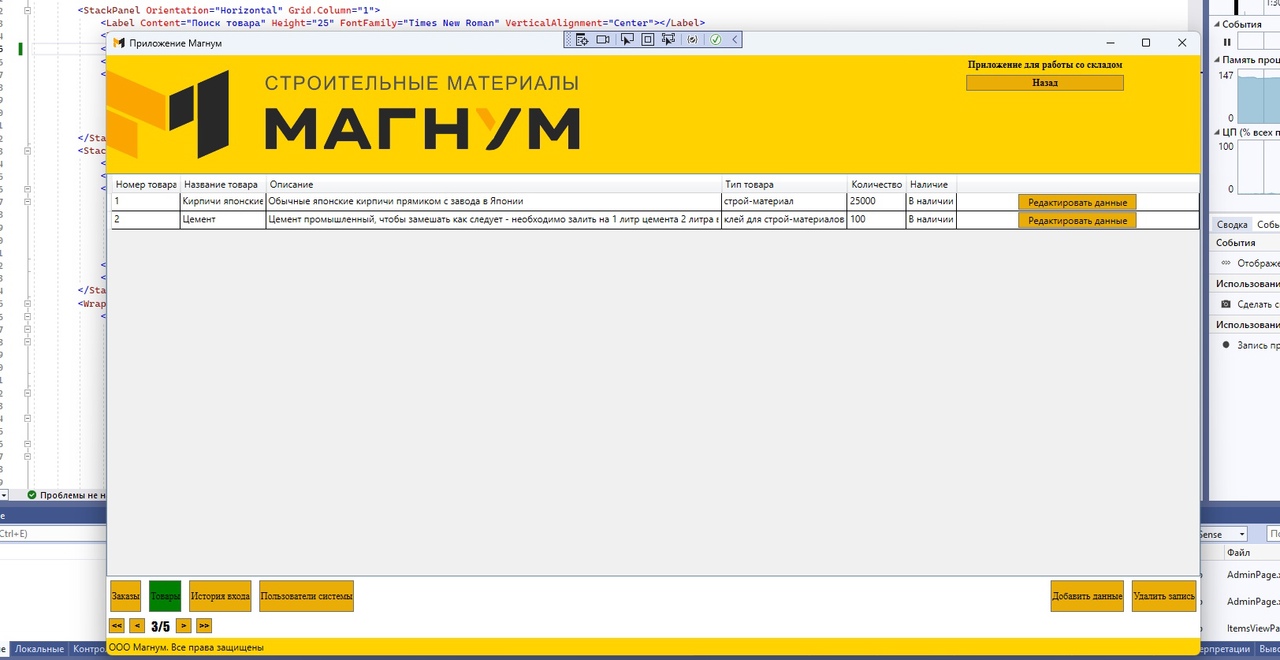


Рисунок 24. Страница с DataGrid товаров

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 зелёным цветом, чтобы обозначить на какой именно странице мы находимся. Проверим редактирование данных товаров (Рисунок 25).

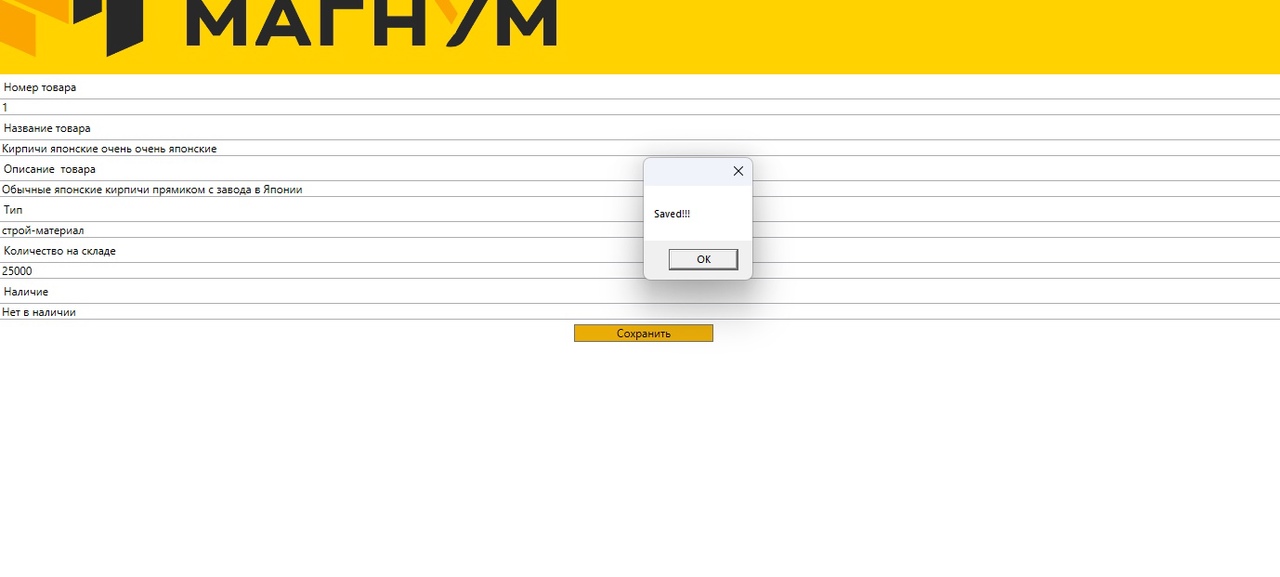


Рисунок 25. Редактирование

Уделим внимание удалению данных. Для удаления данных из выбранной строки необходимо выбрать строку(ки) и нажать кнопку «Удалить данные» (Рисунки 26,27).

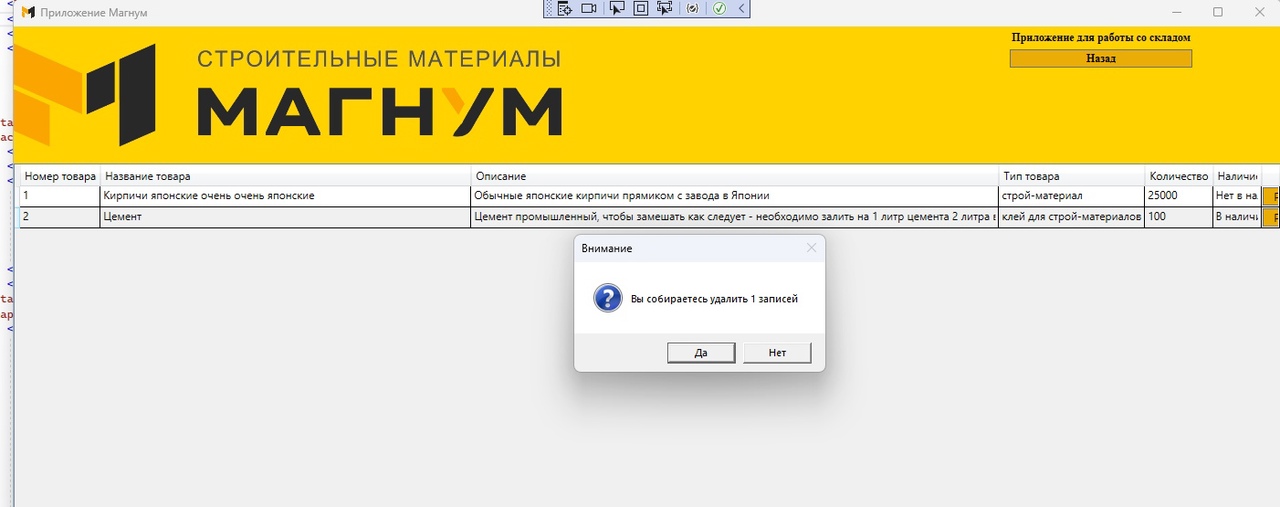


Рисунок 26. Подтверждение удаления

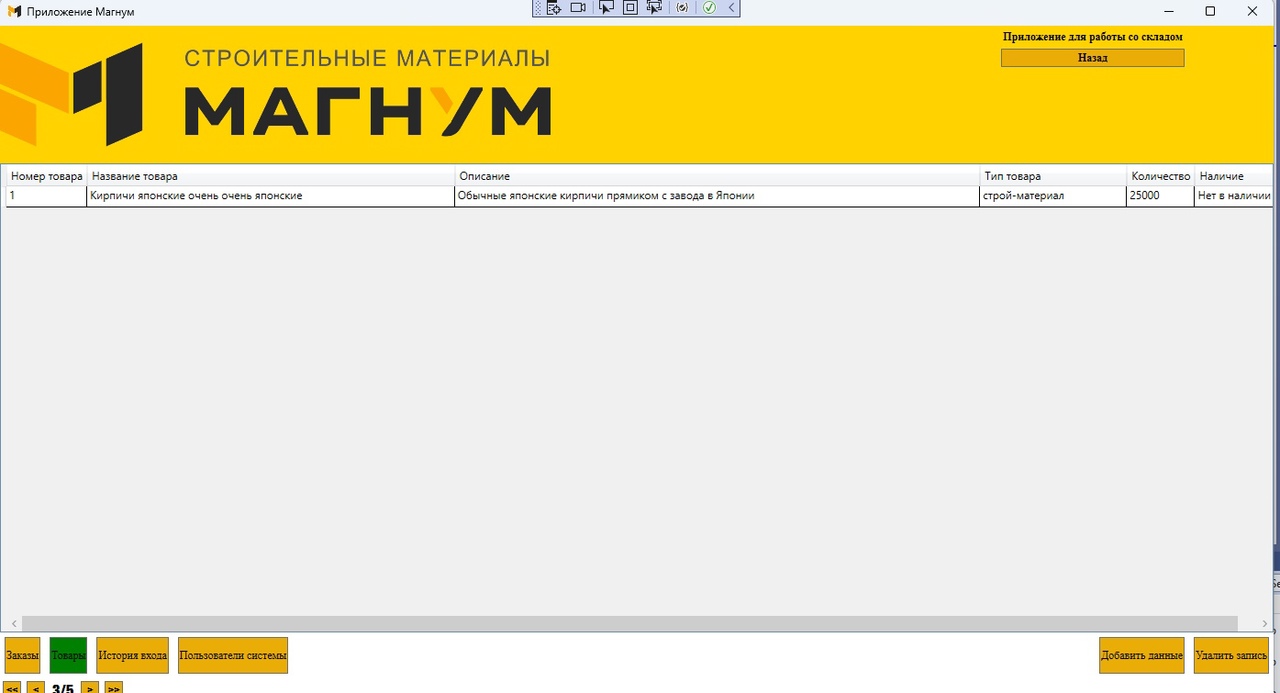


Рисунок 27. Удалённые данные

Для добавления новых и редактирования старых данных нам потребуется 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 28).

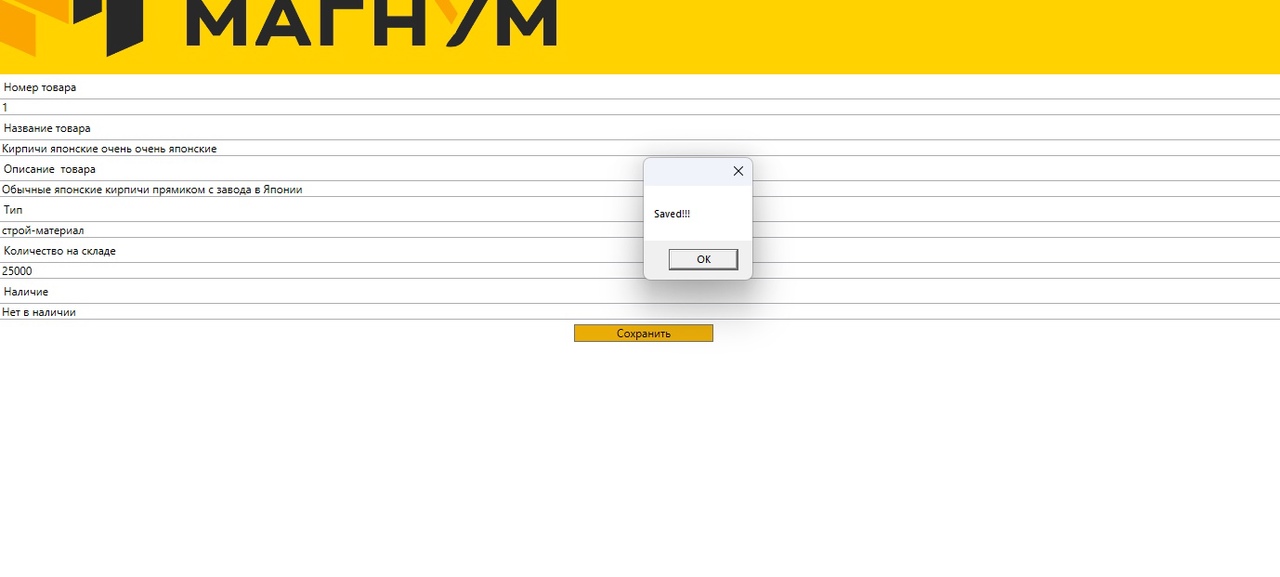


Рисунок 28. Страница редактирования данных

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:

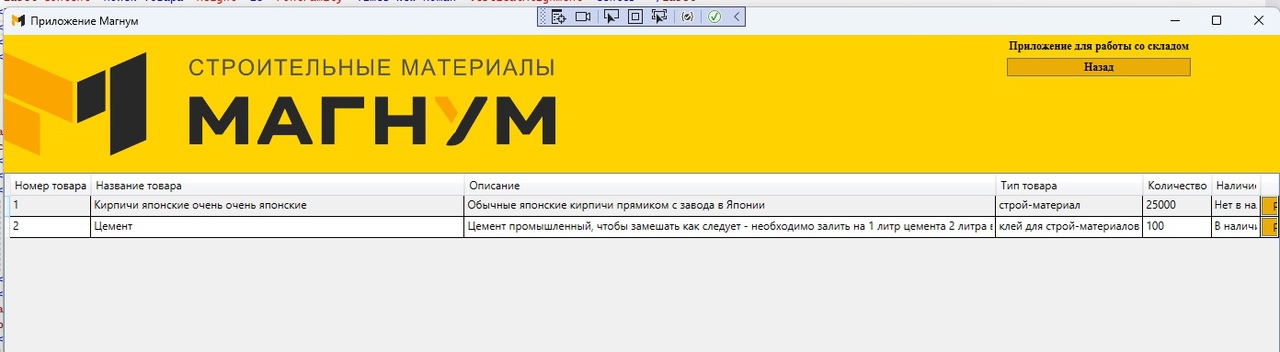


Рисунок 29. Данные после редактирования

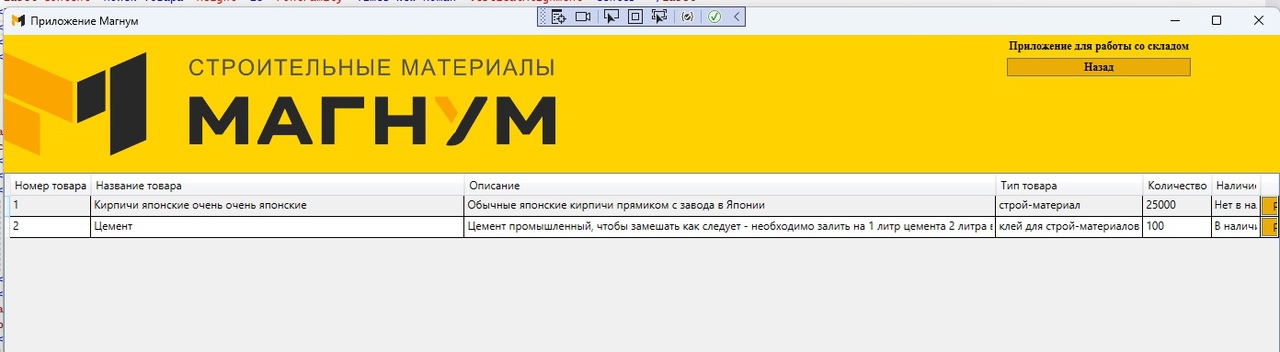


Рисунок 30. Данные после редактирования. Количество увеличено до 25000 единиц

## **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 функциональность отображения каталога товаров в WPF-приложении. Страница ItemsViewPage содержит элементы управления, которые позволяют пользователю просматривать и фильтровать список товаров.

В конструкторе ItemsViewPage происходит инициализация списка всех товаров, который затем используется для фильтрации. Также настраивается комбобокс JenresList, который отображает категории товаров, и заполняется список материалов по умолчанию с помощью метода UpdateCarsPage().

Метод UpdateCarsPage() фильтрует список товаров в соответствии с поисковым запросом и выбранной категорией, и затем обновляет список товаров на странице. После каждого изменения вызывается метод UpdateCarsPage() для фильтрации списка товаров и его обновления 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 окне MessageBox.Запускаем приложение и смотрим результат:

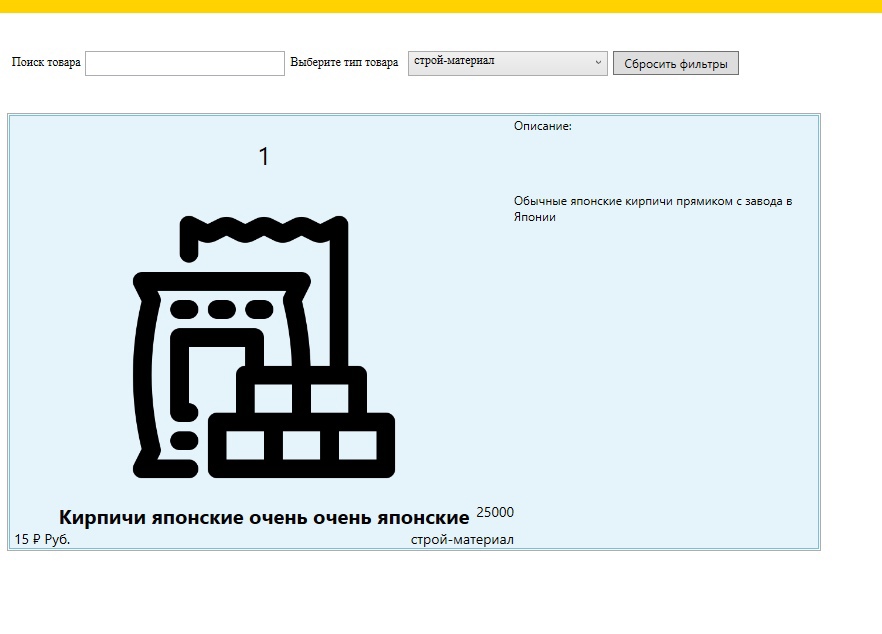


Рисунок 31. Стройматериалы

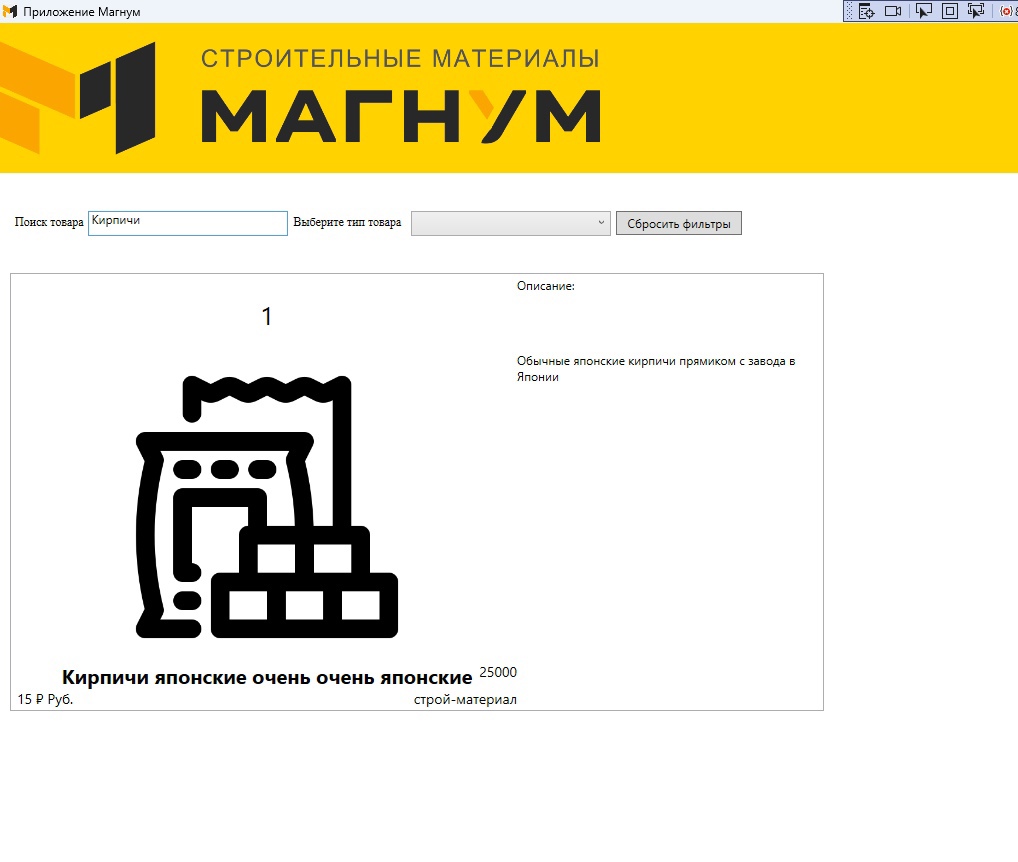


Рисунок 32. Поиск «кирпичи»

## **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 33).

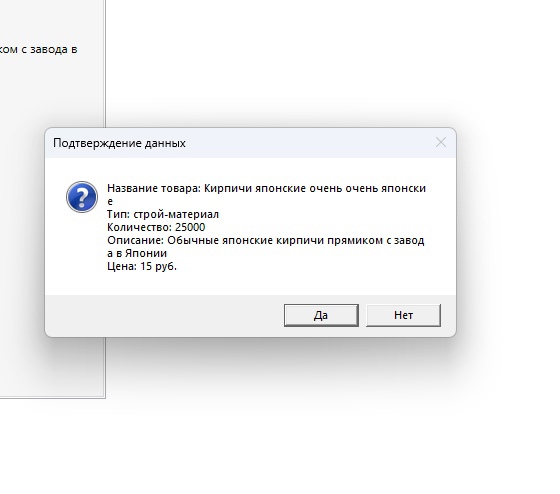


Рисунок 33. Подтверждение данных

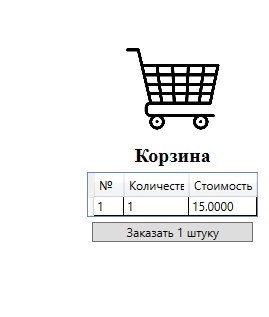


Рисунок 34. Корзина клиента

# **Заключение**

В ходе прохождения производственной (преддипломной) практической подготовки в ООО «Магнум» 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. "ASP.NET Core 3.1 в действии" Андрус Адамчик, Антон Тарасов - издание: 2020.
2. "Pro C# 8 with .NET Core 3.0" Эндрю Троелсен - издание: 2020.
3. "WPF .NET Core 3.0 Cookbook: Building robust and maintainable Windows presentation foundation applications" Джорджано Марияцци - издание: 2020.
4. "Microsoft SQL Server 2019. Руководство для начинающих" Джошуа Джонсон - издание: 2020.
5. "Классические алгоритмы на C#: Реализация и анализ" Джейсон Браун - издание: 2020.
6. "Entity Framework Core в действии" Джон Смит - издание: 2021.
7. "Основы программирования на Python 3 для начинающих" Джейсон Кэннон - издание: 2021.
8. "Самоучитель Python 3" Эрик Мэтиз - издание: 2021.
9. "Java. Полное руководство" Герберт Шилдт - издание: 2022.
10. "Oracle Database 19c. Полное руководство" Боб Брайант - издание: 2022.