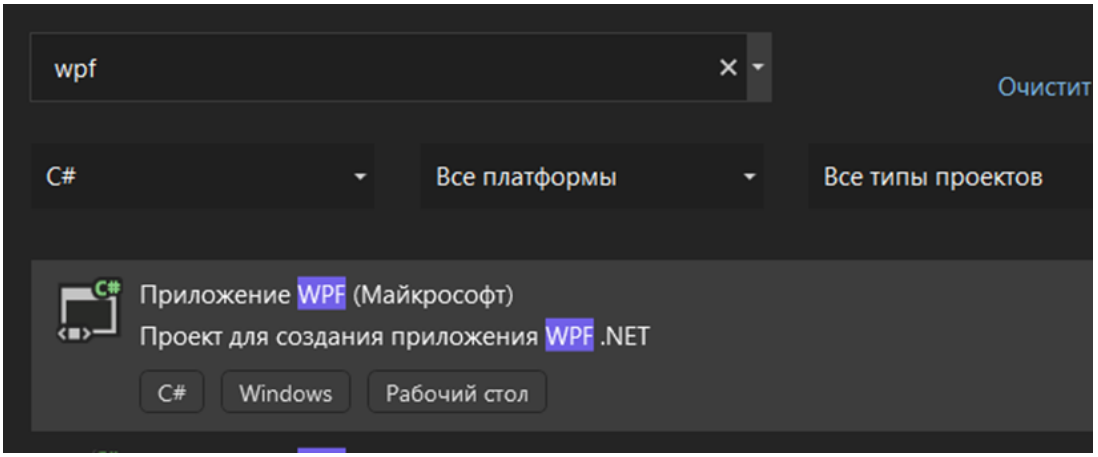
Методическое пособие по разработке АИС на платформе .NET

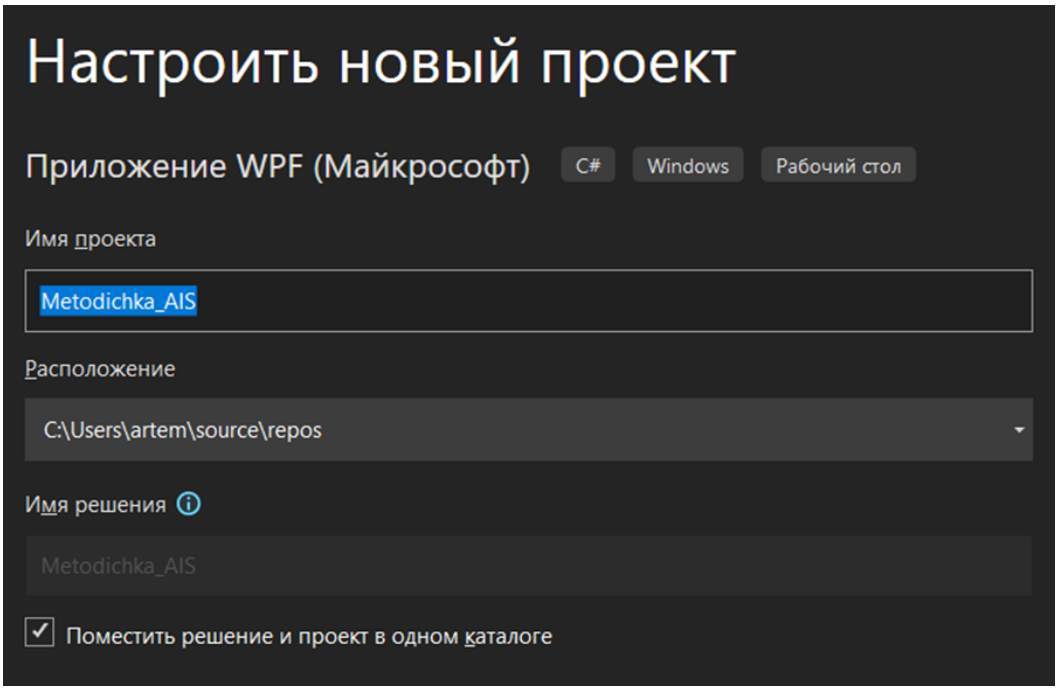
1. Создание проекта

При создании проекта в поиске пишем «WPF» и выбираем «Приложение WPF (Майкрософт)» или «WPF App», если англ. версия.

ПОЖАЛУЙСТА, НЕ ВЫБИРАЙТЕ .NET FRAMEWORK, ВНИМАТЕЛЬНЕЕ!!!

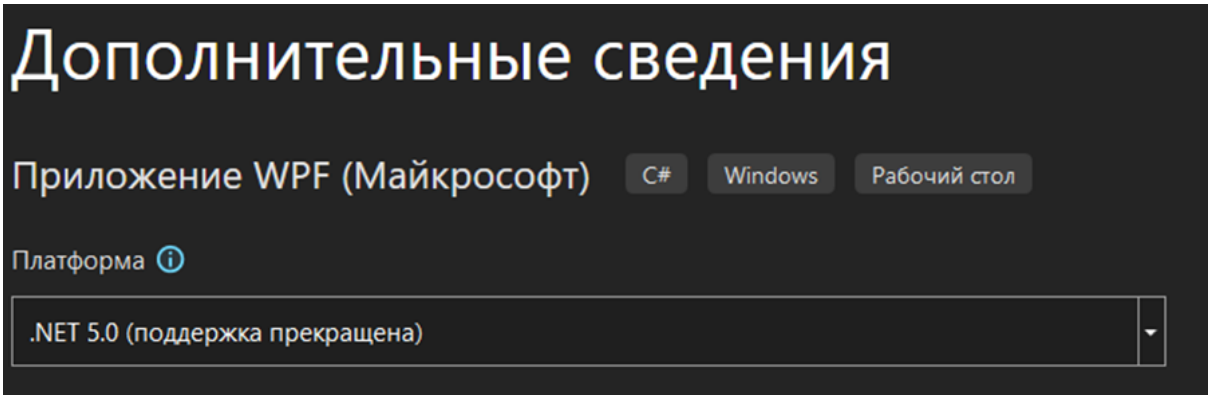


Называем его по предметной области



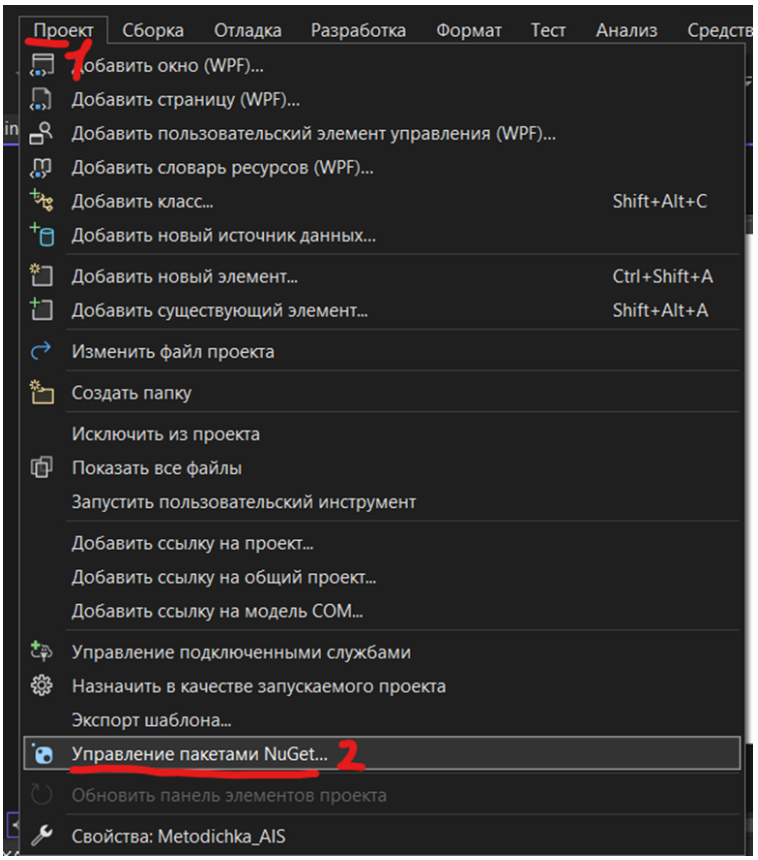
В дополнительных сведениях выбираем платформу «.NET 5.0» (Или .NET 6.0, сильных различий быть не должно, но показывать буду на 5-ом)

Если у вас нет этого выбора, нужно обновить VS



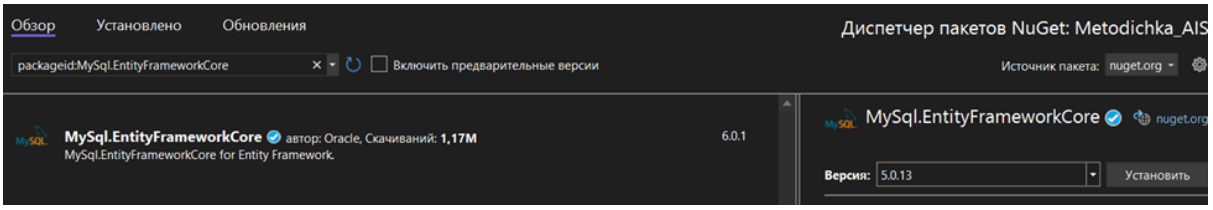
2. Подключение пакетов

Открываем вкладку «Проект» и выбираем «Управление пакетами NuGet»

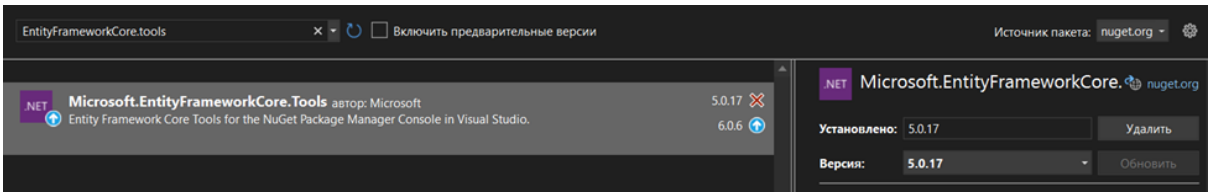


Далее ищем пакет “MySql.EntityFrameworkCore

Версию выбираем в зависимости от версии выбранной платформы, если .NET 5, выбираем версию 5.0.13, если .NET 6, то, соответственно, с цифрой 6.



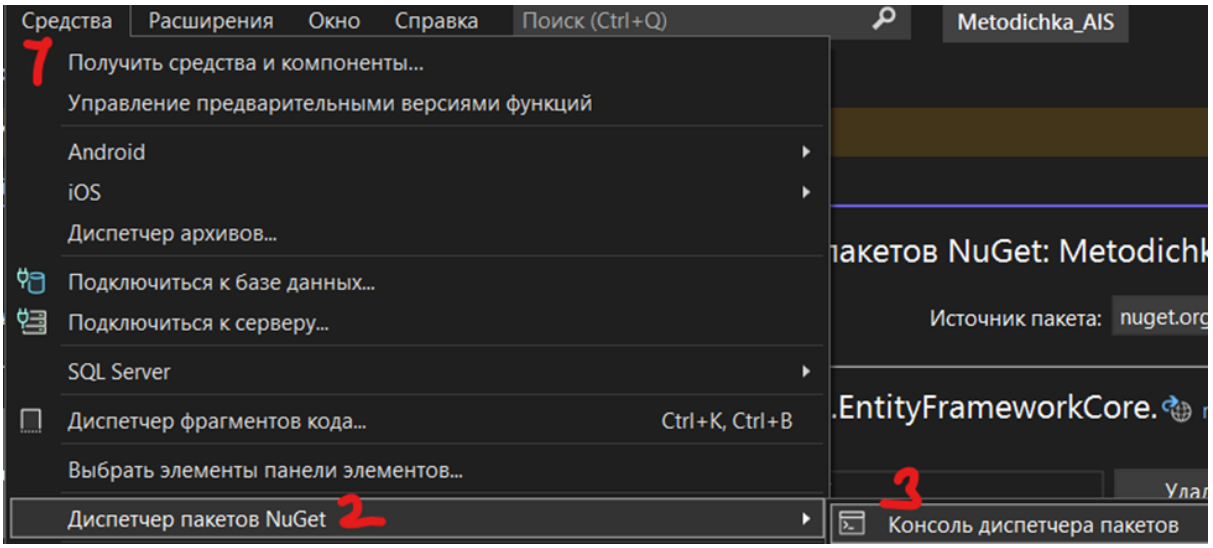
И пакет Microsoft.EntityFrameworkCore.Tools (Версии также как и в прошлом)



3. Подключение базы

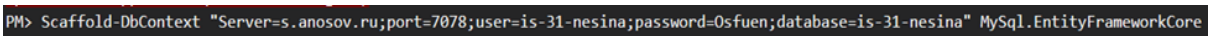
Открываем вкладку «Средства», выбираем «Диспетчер пакетов NuGet» и там «Консоль диспетчера пакетов»

.



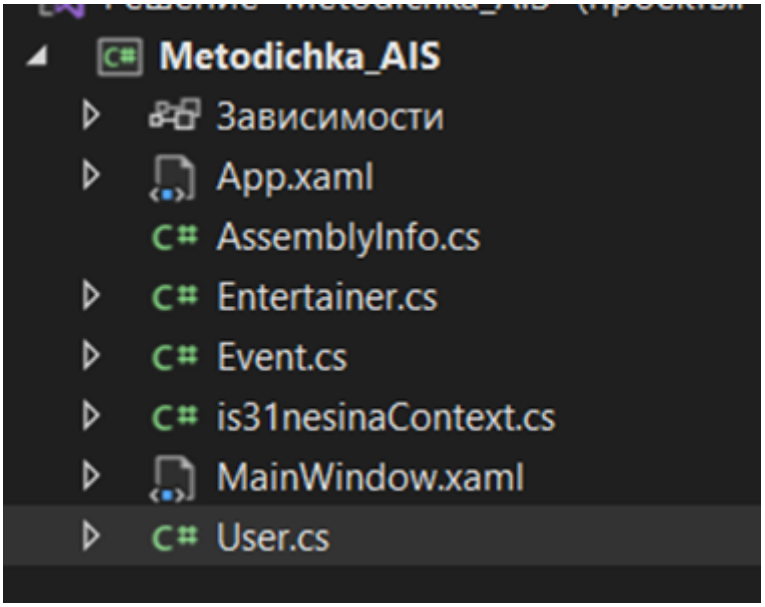
Снизу откроется консоль диспетчера пакетов, в которую вписываем следующее:

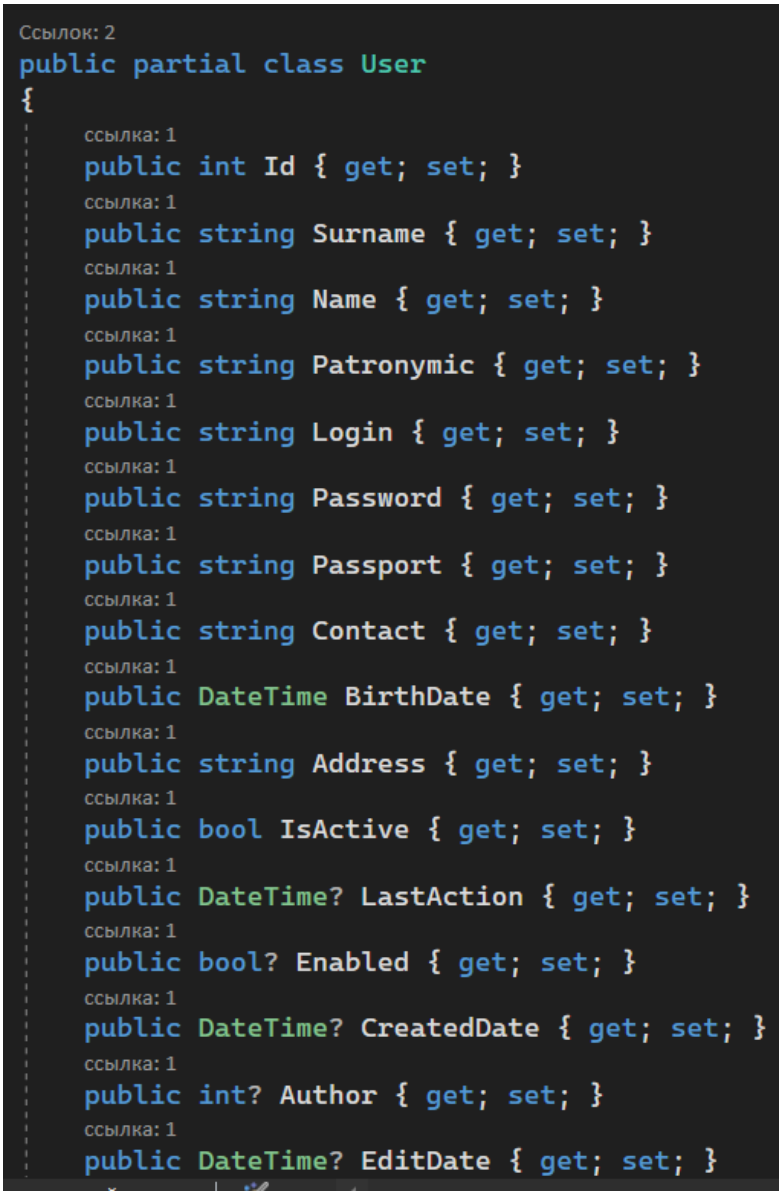
```
Scaffold-DbContext "Server=s.anosov.ru;port=7078;user=is-31-nesina;password=Osfuen;database=is-31-nesina" MySql.EntityFrameworkCore
```



Данные для подключения вписываете свои

После этого произойдет генерация классов для каждой таблицы:

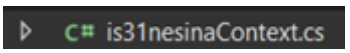


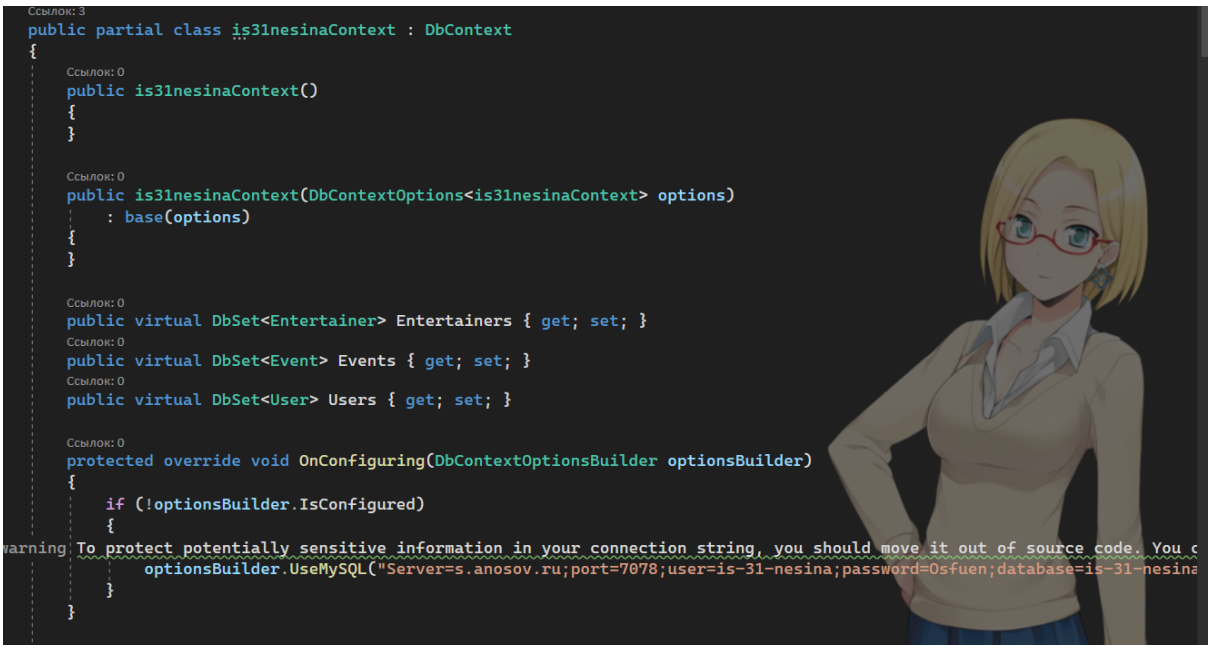


```
Ссылок: 2
public partial class User
{
    ссылка: 1
    public int Id { get; set; }
    ссылка: 1
    public string Surname { get; set; }
    ссылка: 1
    public string Name { get; set; }
    ссылка: 1
    public string Patronymic { get; set; }
    ссылка: 1
    public string Login { get; set; }
    ссылка: 1
    public string Password { get; set; }
    ссылка: 1
    public string Passport { get; set; }
    ссылка: 1
    public string Contact { get; set; }
    ссылка: 1
    public DateTime BirthDate { get; set; }
    ссылка: 1
    public string Address { get; set; }
    ссылка: 1
    public bool IsActive { get; set; }
    ссылка: 1
    public DateTime? LastAction { get; set; }
    ссылка: 1
    public bool? Enabled { get; set; }
    ссылка: 1
    public DateTime? CreatedDate { get; set; }
    ссылка: 1
    public int? Author { get; set; }
    ссылка: 1
    public DateTime? EditDate { get; set; }
```

Как мы видим, для каждого столбца сгенерировалось публичное поле соответствующего типа с геттерами и сеттерами.

Также генерируется класс контекста базы данных, с ее конфигурацией:



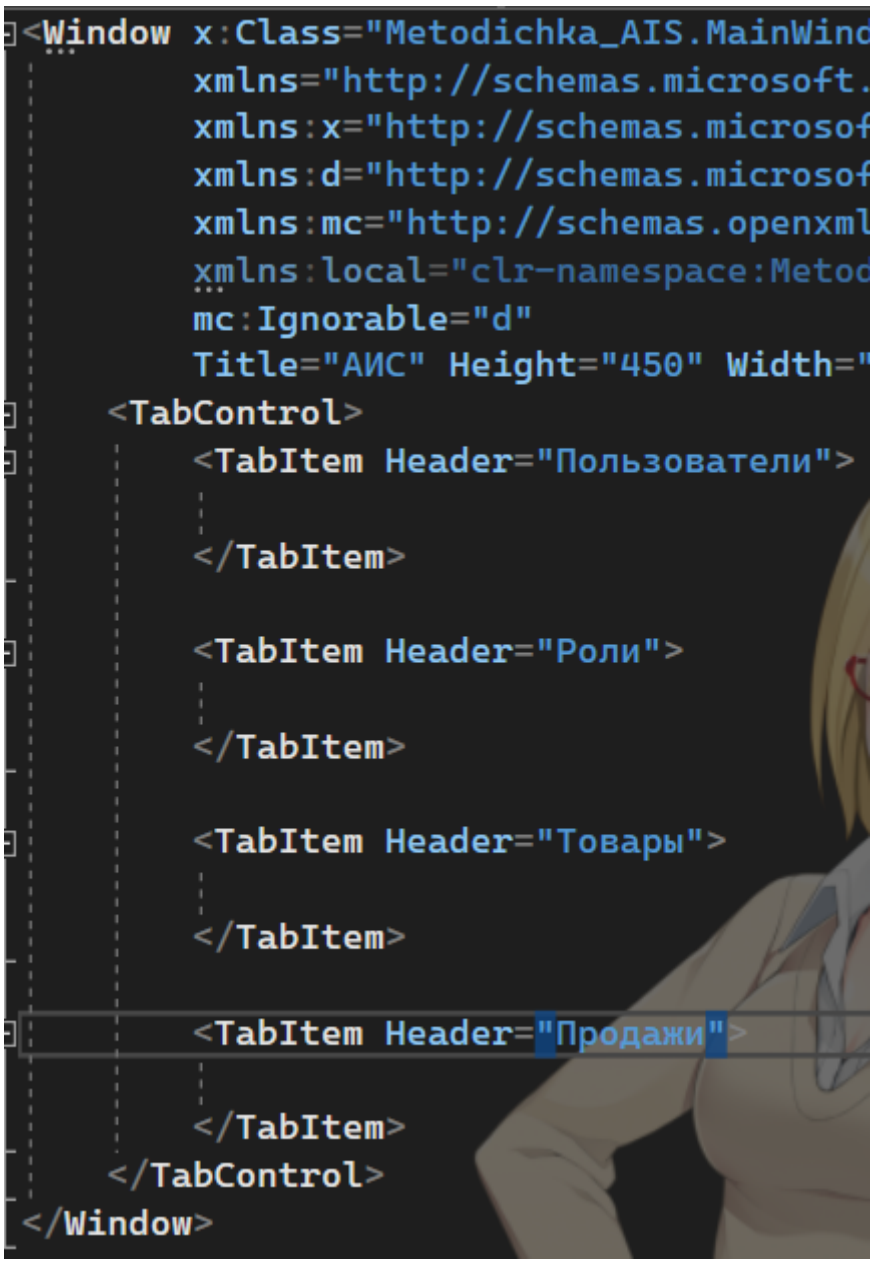


Здесь можно переконфигурировать базу, подключиться к другой и т.д.

Если в базу вносятся конструктивные изменения, необходимо внести соответствующие изменения в код конфигурации или добавить класс таблицы, в случае если была добавлена новая.

4. Разработка интерфейса

В коде главного окна убираем тег “Grid” и вместо него добавляем “TabControl”, этот тег добавляет пространство для вкладок. Добавляем и их тоже, для каждой таблицы (тег “TabItem”). И вписываем атрибут “Header”, который содержит название вкладки:



Здесь я покажу оформление одной вкладки, остальные, нужные вам, делайте по подобию.

Добавляем тег “DockPanel” в во вкладку. По сути, это коробка, в которую мы сложим панель с кнопками, таблицу и панель поиска, дабы они не валялись беспорядочно и были распределены по своим местам. Особенность DockPanel в том, что у элементов, находящихся в нем, можно прописывать сторону, в которой мы бы хотели расположить элемент (Top, Bottom, Left, Right)

```
<TabItem Header="Пользователи">
    <DockPanel>

    </DockPanel>
</TabItem>
```

Далее нужно добавить DataGrid для вывода таблицы.

Здесь важно понимать, что DataGrid не хранит значения, он только их выводит. Для хранения мы создадим отдельные коллекции, данные с которых будем перенаправлять в DataGrid.

```
<TabItem Header="Пользователи">
    <DockPanel>
        <DataGrid Name="usersDG" DockPanel.Dock="Bottom"></DataGrid>
    </DockPanel>
</TabItem>
```

Также для него я указал атрибуты “Name” и “DockPanel.Dock”, по Name мы будем обращаться к таблице, а второй нужен для позиционирования.

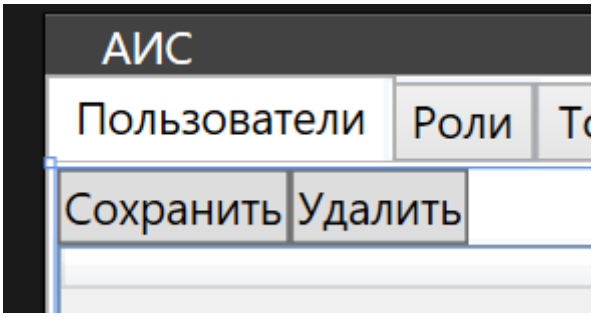
Добавляем StackPanel с кнопками. Это тот же самый DockPanel, с той лишь разницей, что нельзя указывать сторону, в которой расположить элемент, но зато можно указать ориентацию элементов, содержащихся в нем и сгруппировать их, это нам сейчас и нужно.

```
<TabItem Header="Пользователи">
    <DockPanel>
        <StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
            <Button Content="Сохранить"/>
            <Button Content="Удалить"/>
        </StackPanel>

        <DataGrid Name="usersDG" DockPanel.Dock="Bottom"></DataGrid>
    </DockPanel>
</TabItem>
```

С кнопками, думаю, все и так понятно, Content содержит в себе то, что будет написано на ней.

Панель поиска сделаем в самом конце, пока трогать его не будем.

Сейчас все выглядит примерно так:



Не очень красиво, правда?

Давайте сделаем отступы для элементов

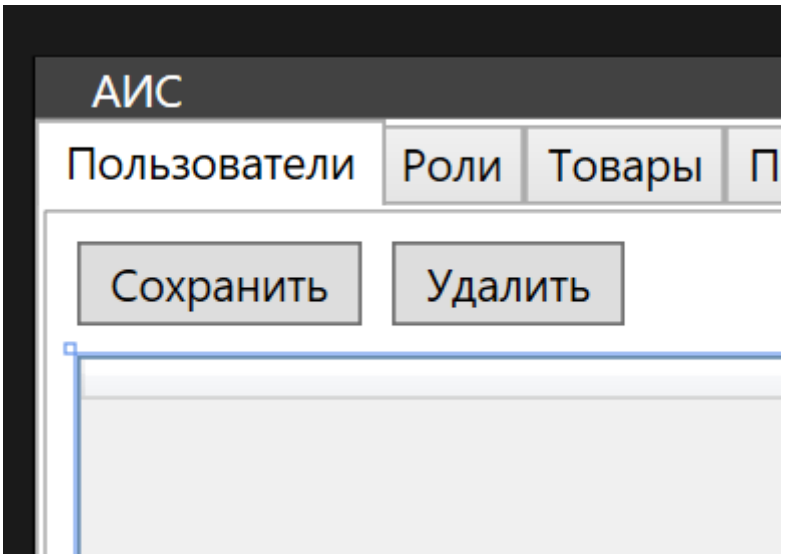
```
<TabItem Header="Пользователи">
    <DockPanel>
        <StackPanel
            Orientation="Horizontal"
            DockPanel.Dock="Top"
            Margin="8"
            >

            <Button
                Content="Сохранить"
                Margin="0 0 8 0"
                Padding="8 2 8 2"
                />
            <Button
                Content="Удалить"
                Padding="8 2 8 2"
                />
        </StackPanel>

        <DataGrid
            Name="usersDG"
            DockPanel.Dock="Bottom"
            Margin="8 0 8 8"
            >
        </DataGrid>
    </DockPanel>
```

Margin - Внешний отступ, т.е. отступ от других элементов

Padding - Внутренний отступ, т.е. отступ от внутренних элементов, например, текста от стенок у кнопки

Значения идут так -> Лево, Верх, Право, Низ. Если оно одно, то для всех сторон отступ одинаковый

Теперь это выглядит так:

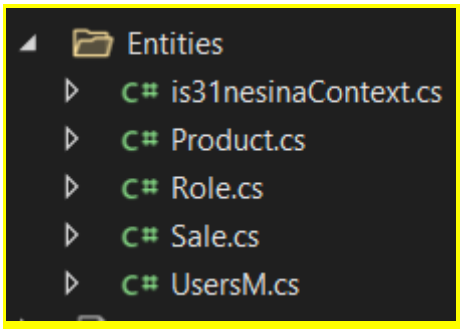


5. Программный код

5.1. Таблицы

Итак, интерфейс мы сверстали, теперь нужно привести его в действие и вывести таблицы.

Если вы вынесли классы таблиц в отдельную папку, что я настоятельно советую сделать для удобства:



Стоит прописать следующую строку в файл, где вы собираетесь их использовать:

```
using System.Windows.Shapes;
using Metodichka_AIS.Entities;
```

В классе MainWindow.xaml.cs создаем объект типа вашего контекста БД для взаимодействия с данными из неё:

```
Ссылок: 2
public partial class MainWindow : Window
{
    is31nesinaContext dbContext = new is31nesinaContext(); // Контекст БД
```

MainWindow.xaml.cs - класс окна, которое мы до этого верстали, здесь прописывается логика его работы

Также сразу создадим переменную для хранения имени текущей таблицы:

```
public partial class MainWindow : Window
{
    private is31nesinaContext _dbContext = new is31nesinaContext(); // Контекст БД
    private string _currentTable; // Имя текущей таблицы
```

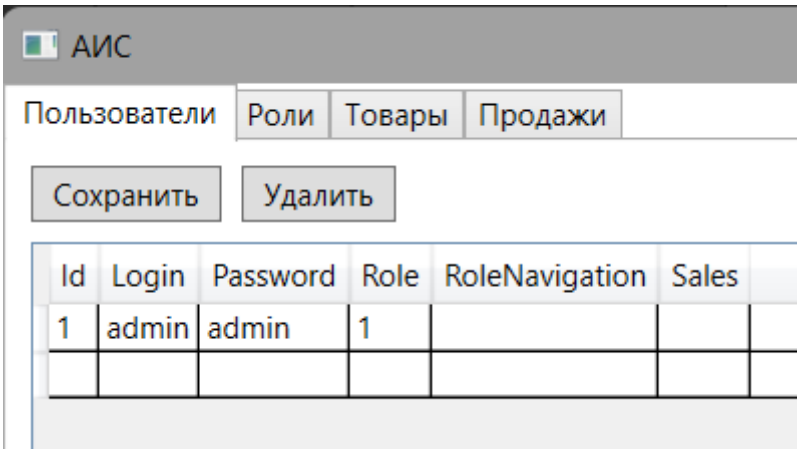
Теперь напишем метод, который будет обновлять данные в таблице:

```
/// <summary>
/// Обновляет данные в переданной таблице
/// </summary>
/// <param name="tableName">Имя таблицы</param>
ссылка: 1
private void RefreshTable(string tableName) {
    switch (tableName) { //Проверяем пришедшую переменную tableName
        case "Пользователи": //В случае если имя таблицы - Пользователи
            _dbContext.UsersMs.Load(); //Загружаем данные таблицы из БД
            //Устанавливаем загруженные данные таблицы из БД как источник для
            //вывода в DataGrid. Вспоминайте, DataGrid не хранит данные, он их
            //только выводит
            usersDG.ItemsSource = _dbContext.UsersMs.Local.ToObservableCollection();
            break;
    }
}
```

Теперь давайте воспользуемся созданным методом, прописав в конструкторе окна таблицу по умолчанию и запустив его с ней:

```
Ссылок: 0
public MainWindow()
{
    InitializeComponent();

    //Устанавливаем текущей таблицей "Пользователи"
    _currentTable = "Пользователи";
    //Обновляем текущую таблицу
    RefreshTable(_currentTable);
}
```

Вот что получилось:



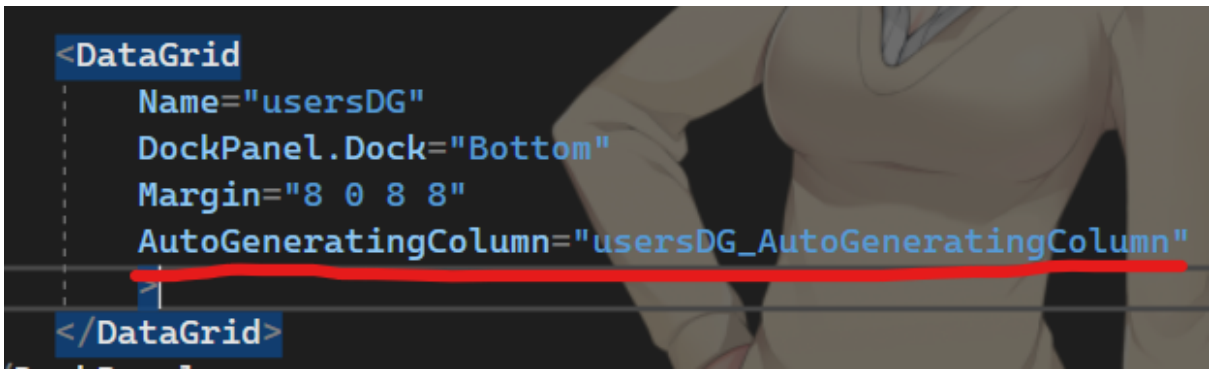
Отлично, не правда ли?)

Но есть одно “но”, вы наверное уже его заметили. Точнее их даже два.

Во-первых, все столбцы на английском, а во-вторых, у нас тут два каких-то левых столбца.

Язык мы сейчас поменяем, а вот с лишними столбцами уже сложнее. Почему они появились? А все из-за того, что EntityFramework создает поля не только для столбцов таблицы, но и для внешних ключей, причем в обе стороны. Нам нужно их отсюда убрать, а также в поле “Роль” выводить не ID подвязанного элемента, а его name, или что мы там хотим.

Для исправления этих “но”, нам нужно обработать событие при генерации столбцов и изменять их название в этот момент на русское, а лишние скрывать.



После этого в классе окна появляется метод, обрабатывающий это событие:

```
ссылка: 1
private void usersDG_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)
{

}
```

В этот метод при срабатывании события приходят “sender” и “e”, в первом хранится DataGrid, который его вызвал, а во втором вся информация о столбце и событии в целом.

Нам нужно получить имя столбца:

```
ссылка: 1
private void usersDG_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)
{
    string headerName = e.Column.Header.ToString();
}
```

Теперь в конструкции “Switch” проверяем каждый столбец по имени и меняем его, а ненужные столбцы скрываем:

```
ссылка: 1
private void usersDG_AutoGeneratingColumn(object sender,
{
    //Имя столбца
    string headerName = e.Column.Header.ToString();

    //Проверяем имя столбца
    switch (headerName) {
        case "Login":
            //Меняем имя
            e.Column.Header = "Логин";
            break;
        case "Password":
            e.Column.Header = "Пароль";
            break;
        case "Role":
            e.Column.Header = "Роль";
            break;
        case "RoleNavigation":
            //Скрываем поле
            e.Column.Visibility = Visibility.Collapsed;
            break;
        case "Sales":
            e.Column.Visibility = Visibility.Collapsed;
            break;
    }
}
```

Итак, теперь все столбцы выводятся на русском языке. Теперь решим вторую проблему. Нам нужно сделать так, чтобы в столбце, в котором идет подвязка значения из другой таблицы, был выбор значения из той самой таблицы, и чтобы там выводилось значения Name, а не ID, как сейчас. Но по стандарту там генерируется обычное текстовое поле, а для выбора нам нужно ComboBox поле. Поэтому мы схитрим: в момент генерации столбца мы перехватим его, скроем, и создадим свой, уже типа ComboBox!

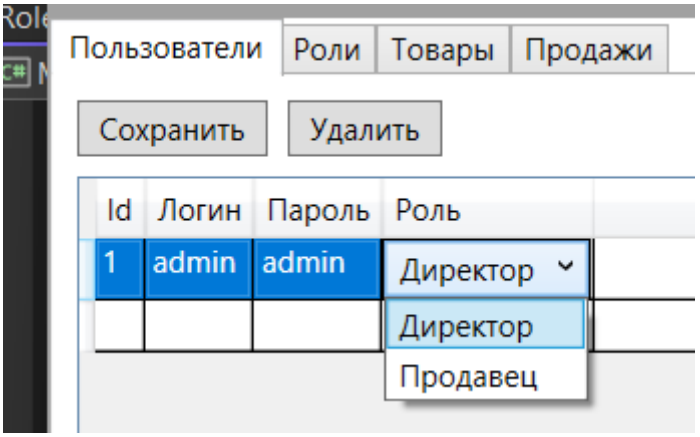
Делается это примерно так:

```
case "Role":
    //Скрываем столбец
    e.Column.Visibility = Visibility.Collapsed;

    _dbContext.Roles.Load(); //Подгружаем данные из таблицы Roles

    Binding binding = new Binding(); //Создаем новый биндинг для подвязки роли
    binding.Path = new PropertyPath("RoleId"); //В путь подвязки указываем поле RoleId

    //Создаем новый столбец типа ComboBox для
    //возможности выбора роли и настраиваем его
    DataGridComboBoxColumn col = new DataGridComboBoxColumn {
        Header = "Роль", //Название столбца
        DisplayMemberPath = "Name", //Отображаем именно поле Name, а не ID
        SelectedValuePath = "Id", //А выбираем по ID
        ItemsSource = _dbContext.Roles.ToArray(), //Подвязываем эти данные в выпадающий список выбора
        SelectedValueBinding = binding //Устанавливаем созданный ранее биндинг к столбцу
    };

    ((DataGrid)sender).Columns.Add(col); //Добавляем созданный столбец в DataGrid
    break;
```

Все это делается в том же месте, где до этого переводили столбцы на русский и скрывали ненужные.

Все закомменчено, так что в объяснении не нуждается.

Проверяем:



Теперь сделаем так, чтобы при переключении вкладок обновлялись соответствующие таблицы.

Добавляем обработчик события GotFocus каждой вкладке:

```
bControl>
 <TabItem Header="Пользователи" Name="usersTab" GotFocus="usersTab_GotFocus">
    <DockPanel>
```

Это событие срабатывает когда мы фокусируемся на элементе, т.е. когда мы нажмем на вкладку.

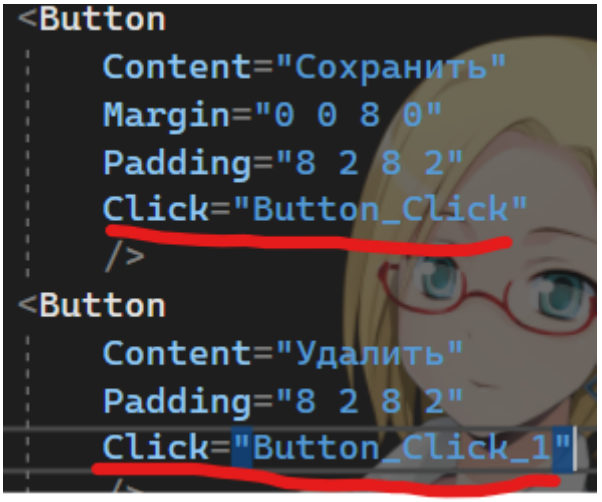
Напишем обработчик:

```
Ссылок: 0
private void usersTab_GotFocus(object sender, RoutedEventArgs e)
{
    _currentTable = ((TabItem)sender).Header.ToString();
    RefreshTable(_currentTable);
}
```

Здесь мы меняем название текущей таблицы на Header вкладки, которая вызвала это событие (sender). И обновляем таблицу.

5.2. Кнопки

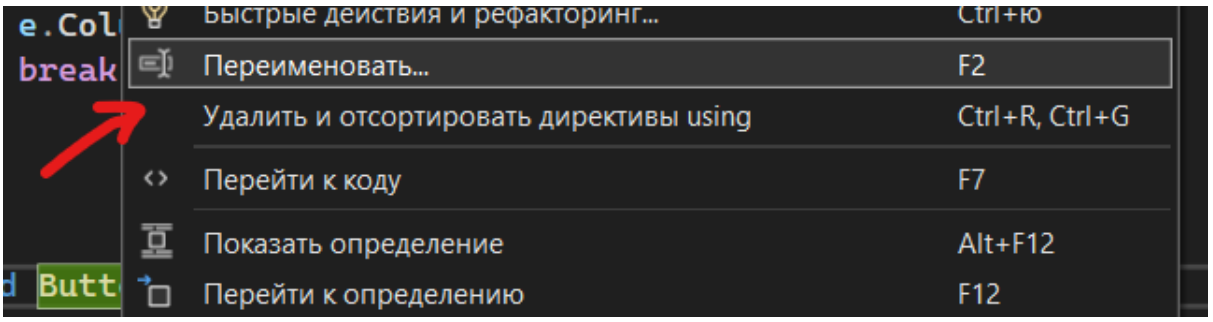
Добавим функционал для кнопок, для этого добавим им обработчики событий на нажатие:



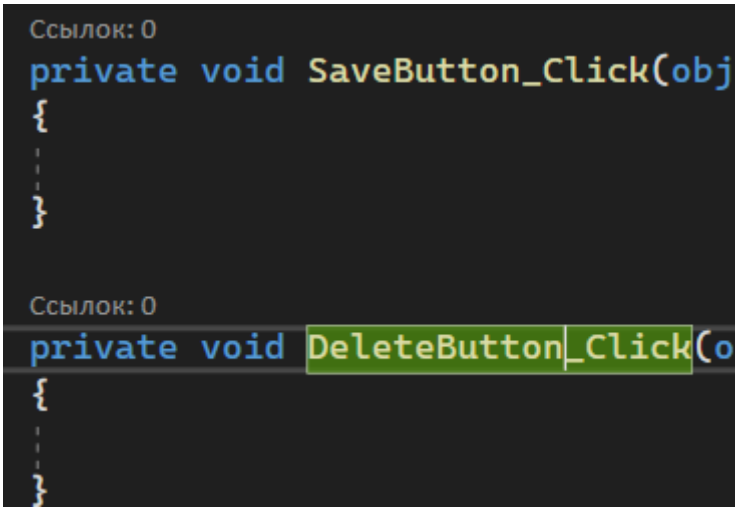
В классе теперь видим:

```
Ссылок: 0
private void Button_Click(object sender, RoutedEventArgs e)
{

}

Ссылок: 0
private void Button_Click_1(object sender, RoutedEventArgs e)
{

}
```

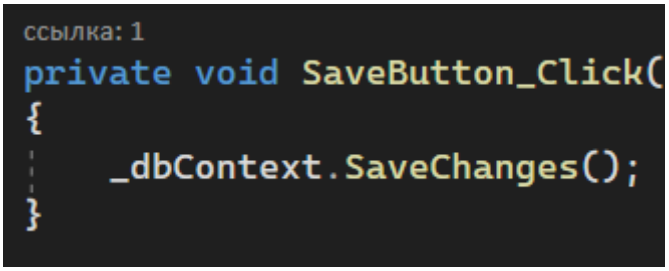
Давайте переименуем, чтобы понимать какой метод какую кнопку обрабатывает:



ОБЯЗАТЕЛЬНО ДЕЛАЙТЕ ЭТО ЧЕРЕЗ “ПЕРЕИМЕНОВАТЬ”, ИНАЧЕ НИЧЕГО НЕ ПЕРЕИМЕНУЕТСЯ



Для сохранения созданных строк просто пишем это:



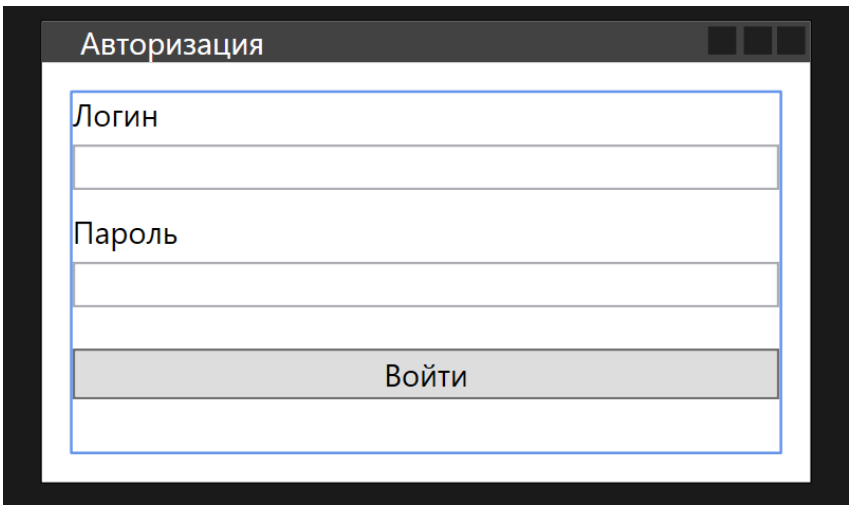
А для удаления делаем так:

```
private void DeleteButton_Click(object sender, RoutedEventArgs e)
{
    switch (_currentTable) {
        case "Пользователи":
            _dbContext.UsersMs.Local.Remove(usersDG.SelectedItem as UsersM);
            break;
    }
}
```

Здесь мы просто удаляем выбранный элемент в текущей таблице, представив его в виде соответствующего класса.

5.3. Авторизация

Создаем окно для авторизации:




```
    Title="Авторизация" Height="180" Width="300">
<StackPanel Margin="12">
    <TextBlock Text="Логин" Margin="0 0 0 4"/>
    <TextBox Name="loginTextBox"/>

    <TextBlock Text="Пароль" Margin="0 8 0 4"/>
    <TextBox Name="passwordTextBox"/>

    <Button Content="Войти" Margin="0 16 0 0"/>
</StackPanel>
```

По Name будем обращаться к текстовым полям.

Теперь нужно сделать так, чтобы это окно появлялось при запуске программы. Делается это просто:

```
public MainWindow()
{
    new LoginWindow().ShowDialog();

    InitializeComponent();

    //Устанавливаем текущей таблицей
    _currentTable = "Пользователи";
    //Обновляем текущую таблицу
    RefreshTable(_currentTable);
}
```

Перед инициализацией главного окна мы создаем новое окно авторизации и открываем его с помощью ShowDialog.

В класс окна авторизации добавляем следующие переменные:

```
public partial class LoginWindow : Window
{
    private is31nesinaContext _dbContext = new is31nesinaContext(); // Контекст БД
    private bool _isLogin = false; //Залогинились ли
    public static UsersM CurrentUser; //Текущий пользователь
```

Контекст для взаимодействия с базой, булевая переменная, хранящая информацию о том, залогинился ли пользователь, и статический объект, хранящий текущего пользователя.

Далее создаем обработчик события нажатия на кнопку “Войти” (Процесс создания уже знаком, пояснять не буду, покажу только дальнейший код):

```
ссылка: 1
private void LoginButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        UsersM user = _dbContext.UsersMs.Where(
            (usr) => usr.Login == loginTextBox.Text && usr.Password == passwordTextBox.Text
            ).Single();
        MessageBox.Show($"Привет, {user.Login}!", "Успешно!");

        _isLogin = true;
        CurrentUser = user;
        Close(); //Закрываем окно
    }
    catch {
        MessageBox.Show("Ошибка!", "Неверный логин или пароль!");
    }
}
```

Здесь мы пробуем получить из БД пользователя с логином и паролем совпадающим с введенными. И если такой нашелся, мы выводим сообщение об успешной авторизации и закрываем это окно. А иначе выводим сообщение об ошибке.

После этого нужно повесить обработчик на закрытие окна, чтобы если пользователь авторизовался, пропустить его в программу, а иначе просто завершить работу:

```
    xmlns:local="clr-namespace:Metodichka_AIS"
    mc:Ignorable="d"
    Title="Авторизация" Height="200" Width="300" Closed="Window_Closed">
StackPanel Margin="12">
```

Ну и пишем соответствующий код:

```
private void Window_Closed(object sender, EventArgs e)
{
    if (!_isLogin)
        App.Current.Shutdown(); //Завершение работы приложения
}
```

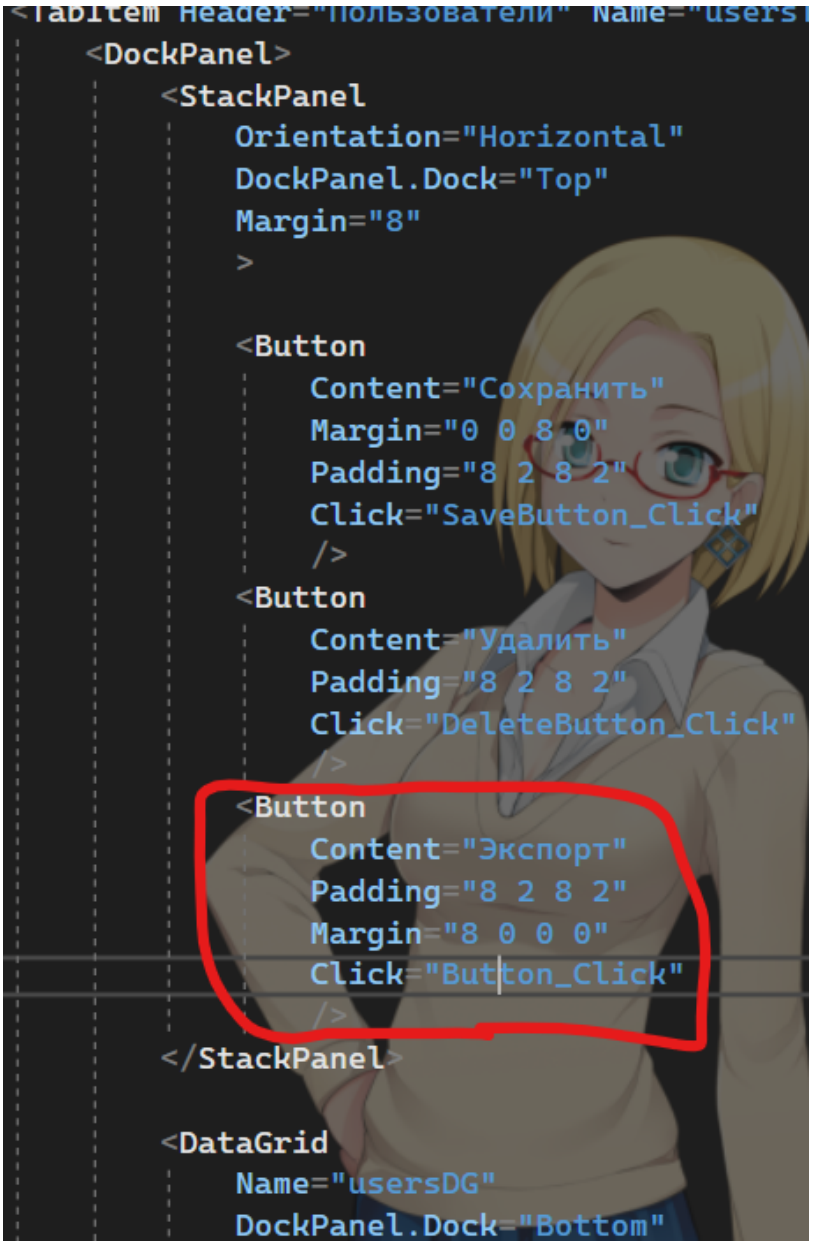
Теперь у нас есть информация о текущем пользователе и мы можем в любом месте задать условия для каждой роли, например:

```
/// <summary>
/// Проверяет роль пользователя и задает параметры отображения элементов для каждого
/// </summary>
ссылка: 1
private void CheckUser() {
    switch (LoginWindow.CurrentUser.RoleId) {
        case 1: //Если директор
            break;
        case 2: //Если продавец
            //Отключаем видимость вкладки "Пользователи"
            usersTab.Visibility = Visibility.Collapsed;
            break;
    }
}
```

Условия устанавливайте свои, по ID ролей. Ну и ограничения для них также думайте сами, по своей предметной области. У любого элемента интерфейса есть Visibility, и если ему задать значение Collapsed, он полностью пропадет из видимости.

5.4. Экспорт в CSV

В панель с кнопками добавляем еще одну кнопку “Экспорт”. Создаем обработчик события на нажатие:



Для выбора файла напишем такой метод:

```
/// <summary>
/// Возвращает полный путь к файлу, выбранному пользователем
/// </summary>
/// <returns>Полный путь к файлу</returns>
ссылка: 1
private string GetUserFile() {
    //Создаем OpenFileDialog для выбора файла
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "CSV Файлы | *.csv"; //Фильтр на CSV файлы
    ofd.Title = "Выберите файл для экспорта";

    //Открываем его, и если выбрали файл, то возвращаем путь до него
    if (ofd.ShowDialog() == true) {
        return ofd.FileName;
    }

    return null; //Иначе вернется null
}
```

И пишем обработку экспорта:

```
ссылка: 1
private void ExportButton_Click(object sender, RoutedEventArgs e)
{
    //Получаем путь к файлу для экспорта
    string filePath = GetUserFile();

    //Если вернулся null (Ничего не выбрали), выходим из метода
    if (filePath == null)
        return;

    //Открываем поток на запись
    StreamWriter file = new StreamWriter(filePath, false);

    //Проверяем какую таблицу будем экспортировать
    switch (_currentTable) {
        case "Пользователи":
            //Сохраняем таблицу в коллекцию table для удобства
            ObservableCollection<UsersM> table = _dbContext.UsersMs.Local.ToObservableCollection();

            file.WriteLine($"ID;Логин;Пароль;РольИД"); //Записываем заголовки
            //Проходим по всем элементам таблицы
            foreach (UsersM elem in table)
            {
                //Записываем каждое поле элемента в файл
                file.WriteLine($"{elem.Id};{elem.Login};{elem.Password};{elem.RoleId}");
            }
            break;
    }

    file.Close(); //Закрываем файл
    MessageBox.Show("Экспорт успешно завершен", "Успешно!");
}
```

5.5. Отчеты

Отчетов можно сделать, конечно, много разных, но для упрощения работы мы просто сделаем отчет, например, по продажам за текущий месяц. По подобию можете сделать что-то похожее по своей предметной области, если нету продаж, суть все равно одна.

Сделаем отдельную вкладку для отчетов:

```
<TabItem Header="Отчеты">
    <DockPanel>
        <StackPanel Orientation="Horizontal"
                    DockPanel.Dock="Top"
                    Margin="8">
            <Button Content="Продажи за текущий месяц"
                    Click="ReportSalesMonthButton_Click"/>
        </StackPanel>

        <DataGrid Name="reportDG"
                  DockPanel.Dock="Bottom"
                  Margin="8 0 8 8"
                  AutoGeneratingColumn="reportDG_AutoGeneratingColumn">

        </DataGrid>
    </DockPanel>
</TabItem>
```

Сюда также добавил кнопку для генерации отчета “Продажи за текущий месяц”.

Пишем обработчик для неё:

```
private void ReportSalesMonthButton_Click(object sender, RoutedEventArgs e)
{
    string reportName = ((Button)sender).Content.ToString();

    switch (reportName) {
        case "Продажи за текущий месяц":
            ObservableCollection<Sale> salesCurMonth = new ObservableCollection<Sale>();

            _dbContext.Sales.Load();
            foreach (Sale sale in _dbContext.Sales.Local.ToObservableCollection()) {
                if (sale.Date.Month == DateTime.Now.Month)
                    salesCurMonth.Add(sale);
            }

            reportDG.ItemsSource = salesCurMonth;

            break;
    }
}
```

И обрабатываем генерацию столбцов отчета также, как и в остальных таблицах, только здесь еще ставим на каждый столбец свойство IsReadOnly в значении true, для того, чтобы пользователь не мог изменять данные в отчете.

```
ссылка: 1
private void reportDG_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)
{
    //Имя столбца
    string headerName = e.Column.Header.ToString();
    e.Column.IsReadOnly = true;

    //Проверяем имя столбца
    switch (headerName)
    {
        case "User":
            e.Column.Visibility = Visibility.Collapsed;
            break;
        case "Product":
            e.Column.Visibility = Visibility.Collapsed;
            break;
        case "Date":
            e.Column.Header = "Дата";
            break;
        case "UserNavigation":
            e.Column.Visibility = Visibility.Collapsed;

            _dbContext.UsersMs.Load(); //Подгружаем данные из таблицы Roles

            Binding binding = new Binding(); //Создаем новый биндинг для подвязки роли
            binding.Path = new PropertyPath("User"); //В путь подвязки указываем поле RoleId

            //Создаем новый столбец типа ComboBox для
            //возможности выбора роли и настраиваем его
            DataGridComboBoxColumn col = new DataGridComboBoxColumn
```

```
        {
            Header = "Пользователь", //Название столбца
            DisplayMemberPath = "Login", //Отображаем именно поле Name, а не ID
            SelectedValuePath = "Id", //А выбираем по ID
            ItemsSource = _dbContext.UsersMs.ToArray(), //Подвязываем эти данные в выпадающий список выбора
            SelectedValueBinding = binding, //Устанавливаем созданный ранее биндинг к столбцу
            IsReadOnly = true
        };

        ((DataGrid)sender).Columns.Add(col); //Добавляем созданный столбец в DataGrid
        break;
    case "ProductNavigation":
        e.Column.Visibility = Visibility.Collapsed;
        _dbContext.Products.Load(); //Подгружаем данные из таблицы Roles

        Binding binding1 = new Binding(); //Создаем новый биндинг для подвязки роли
        binding1.Path = new PropertyPath("Product"); //В путь подвязки указываем поле RoleId

        //Создаем новый столбец типа ComboBox для
        //возможности выбора роли и настраиваем его
        DataGridComboBoxColumn col1 = new DataGridComboBoxColumn
        {
            Header = "Товар", //Название столбца
            DisplayMemberPath = "Name", //Отображаем именно поле Name, а не ID
            SelectedValuePath = "Id", //А выбираем по ID
            ItemsSource = _dbContext.Products.ToArray(), //Подвязываем эти данные в выпадающий список выбора
            SelectedValueBinding = binding1, //Устанавливаем созданный ранее биндинг к столбцу
            IsReadOnly = true
        };

        ((DataGrid)sender).Columns.Add(col1); //Добавляем созданный столбец в DataGrid
        break;
}
```