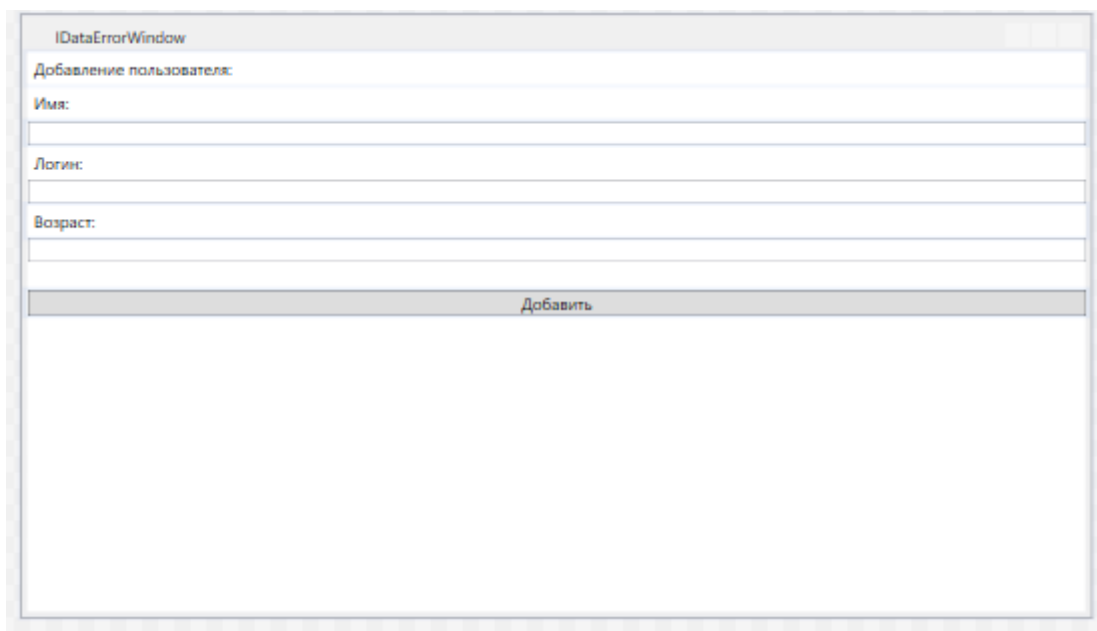


Механизмы валидации. Часть 2.

Продолжение предыдущей работы

Реализация IDataErrorInfo

Создадим окно, аналогичное двум предыдущим:



За основу возьмем подход из <https://metanit.com/sharp/wpf/14.php>

Реализуем интерфейс:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;

public partial class User: IDataErrorInfo
{
    public string this[string columnName] => throw new NotImplementedException();

    public int UserId { get; set; }
    public string Username { get; set; }
    public string Name { get; set; }
    public Nullable<int> Age { get; set; }

    public string Error => throw new NotImplementedException();
}
```

Далее опишем индексатор:

```
public string this[string columnName]
{
    get
    {
        if (columnName == "Name")
        {
            if (Name.Trim() == "")
```

```

        {
            return "Имя не может быть пустым!";
        }
    }

    if (columnName == "Username")
    {
        if (Username.Trim().Length < 3)
        {
            return "Имя пользователя должно содержать хотя бы 3 символа!";
        }

        if (Username.Trim().Any(ch => !char.IsLetter(ch)))
        {
            return "Имя пользователя должно содержать только буквы!";
        }
    }

    if (columnName == "Age")
    {
        if (Age < 0 || Age > 100)
        {
            return "Возраст не может быть меньше 0 или больше 100!";
        }
    }

    return "";
}

```

Реализация интерфейса готова, можно выполнять привязку.

Привязка к имени (не забудьте по DataContext):

```
<TextBox Name="NameTextBox" Text="{Binding Name, ValidatesOnDataErrors=True}" />
```

Здесь мы устанавливаем правило `DataErrorValidationRule`, что заставляет привязку считаться с нашей реализацией `IDataErrorInfo`. Более детализированный способ сделать то же самое (по аналогии с предыдущим заданием):

```

<TextBox Name="NameTextBox">
    <TextBox.Text>
        <Binding Path="Name">
            <Binding.ValidationRules>
                <DataErrorValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>

```

Самостоятельная работа

- Выполнить привязку ко всем полям ввода;
- При возникновении ошибки кнопка не должна работать;
- Реализовать вывод сообщений об ошибках (либо с помощью `ToolTip`, либо в отдельное поле выше кнопки);

The screenshot shows a window titled "IDataErrorWindow" with a blue title bar. Inside, there's a form titled "Добавление пользователя:". It has three input fields: "Имя:" (Name), "Логин:" (Login), and "Возраст:" (Age). The "Логин:" field contains the text "admin". The "Возраст:" field contains the number "37". Below the "Имя:" field, there is a red error message: "Имя не может быть пустым!". At the bottom of the form, there is a button labeled "Добавить".

Валидация на стороне EF

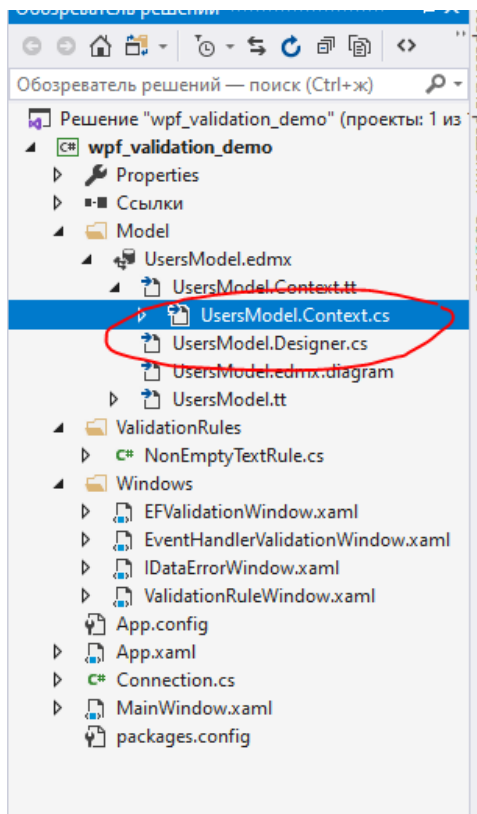
За основу возьмем

<https://www.entityframeworktutorial.net/EntityFramework4.3/validate-entity-in-entity-framework.aspx>

Сперва нужно создать еще одно окно:

The screenshot shows a new window titled "EFValidationWindow". It contains a form titled "Добавление пользователя:". There are three input fields: "Имя:", "Логин:", and "Возраст:". Below the "Возраст:" field, there is a button labeled "Добавить".

Далее, опишем изменим наш DbContext:



Добавим сборки:

```
// добавим сборки
using System.Collections.Generic;
using System.Data.Entity.Validation;
using System.ComponentModel.DataAnnotations;

public partial class ValidationUsersEntities : DbContext
{
    public ValidationUsersEntities()
        : base("name=ValidationUsersEntities")
    {
    }
}
```

Переопределим метод ValidateEntity:

```
// переопределим метод DbEntityValidationResult
protected override DbEntityValidationResult ValidateEntity(DbEntityEntry entityEntry, IDictionary<object, object> items)
{
    // если работаем с User
    if (entityEntry.Entity is User)
    {
        var errorList = new List<DbValidationError>();

        // для свойства Name запрещаем пустые значения
        if (entityEntry.CurrentValues.GetValue<string>("Name") == "")
        {
            errorList.Add(new DbValidationError("Name", "Имя не может быть пустым!"));
        }

        // проверяем значение для Age
        if (entityEntry.CurrentValues.GetValue<int>("Age") < 0)
        {
            errorList.Add(new DbValidationError("Age", "Возраст не может быть отрицательным!"));
        }

        // .. дополнить логику валидации (Username и Age)

        return new DbEntityValidationResult(entityEntry, errorList);
    }

    return base.ValidateEntity(entityEntry, items);
}
```

Данный метод будет вызван для проверки значений, непосредственно перед выполнением SaveChanges.

Реализуем привязку и обработчик добавления в последнем окне:

XAML

```
<StackPanel Name="stackPanel">
    <Label Content="Добавление пользователя: " />
    <Label Content="Имя: " />
    <TextBox Name="NameTextBox" Text="{Binding Name}" />

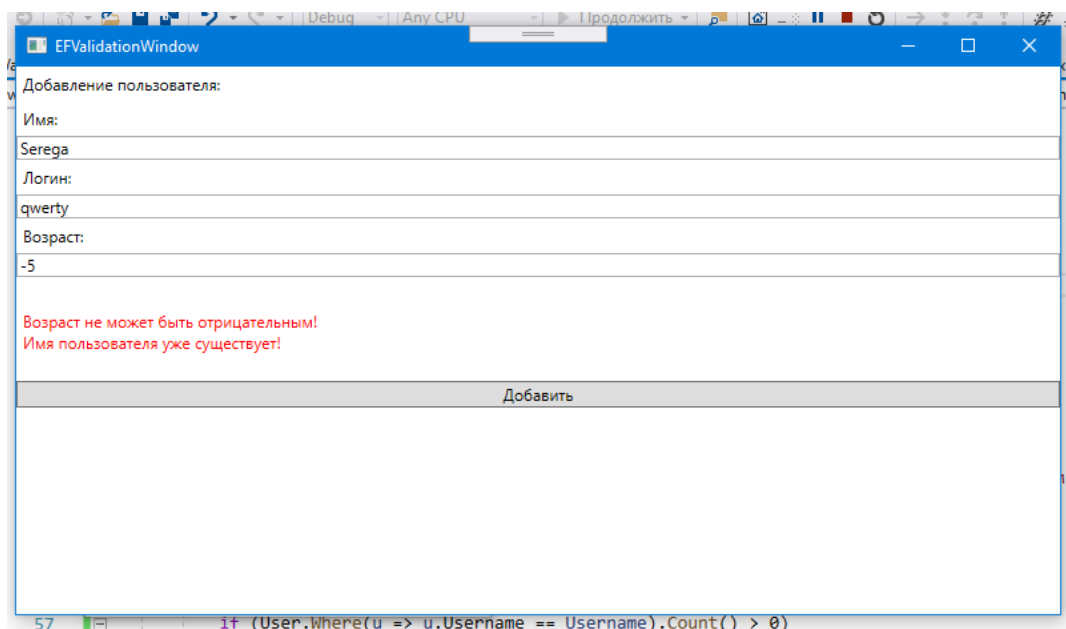
    </TextBox>
    <Label Content="Логин: " />
    <TextBox Name="UsernameTextBox" Text="{Binding Username}" />
    <Label Content="Возраст: " />
    <TextBox Name="AgeTextBox" Text="{Binding Age}" />
    <Label Height="20" />
    <Label Name="ErrorListLabel" Foreground="Red" />
    <Button Content="Добавить" Name="AddUserButton" Click="AddUserButton_Click" />
</StackPanel>
```

Обработчик:

```
private void AddUserButton_Click(object sender, RoutedEventArgs e)
{
    ErrorListLabel.Content = "";
    Connection.Database.User.Add(User);
    try
    {
        Connection.Database.SaveChanges();
    }
    catch (DbEntityValidationException validationException)
    {
        Connection.Database.User.Remove(User);

        foreach (DbEntityValidationResult validationResult in validationException.EntityValidationErrors)
        {
            foreach (var error in validationResult.ValidationErrors)
            {
                ErrorListLabel.Content += $"{error.ErrorMessage}\n";
            }
        }
    }
}
```

Результат (не забудьте «доделать» DbContext):



```
57 if (User.Where(u => u.Username == Username).Count() > 0)
```