Пояснительная записка к разработанному шаблонному приложению на WPF для подготовки к демо экзамену.

Разработчик-автор: Ершов Дмитрий Алексеевич, студент группы 18ИП1.

Целевая аудитория: типчики из шараги (не преподаватели, кроме Егора Олеговича, он свой).

Цель: оставить наследие и помочь челам разобраться в быстром создании приложения и реализации функциональности.

Тема шаблонного проекта: корпоративная сеть.

Структура сети: сеть состоит из пользователей разных ролей, минимальный перечень ролей: модераторы (админы) и обычные пользователи. В данной сети пользователи могут добавлять друг друга в друзья, а также удалять из друзей.

Минимальный функционал:

* Модераторы могут просматривать пользовательские учетные данные;
* Модераторы могут удалять/добавлять пользователей любых ролей;
* Модераторы могут повышать/уменьшать роль пользователя;
* Пользователи могут добавлять/удалять друзей;
* Пользователи просматривают список друзей;
* Пользователи применяют сортировку/фильтрацию/поиск к списку друзей для поиска нужного друга.

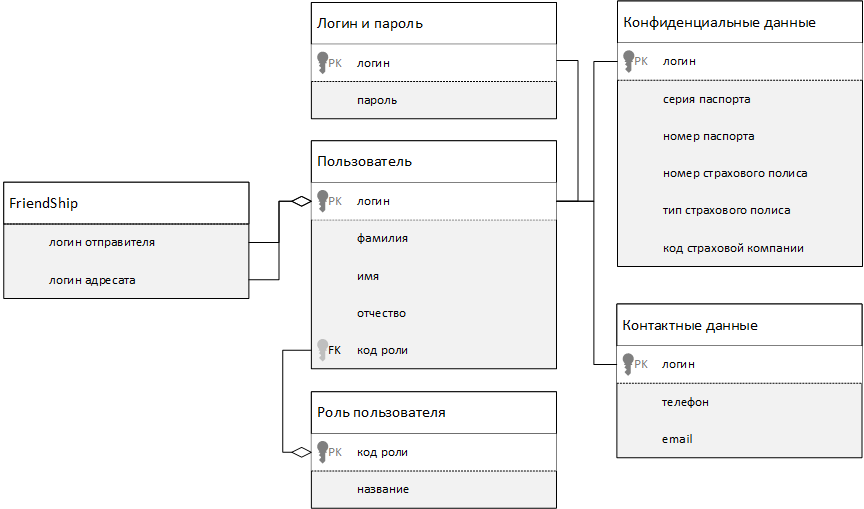
Пояснение к внедрению/разработке функционала:

Каждая новая функциональность будет разработана в отдельной ветке и после успешного завершения будет слита в master, для подробностей смотрите историю коммитов гита.

(команда *$ git log --pretty=format:"%h %s" --graph*).

Эталонная диаграмма ERD (базы данных, нотация UML):

Рекомендую в любой АИС делать такую структуру базы данных, связанных с пользователем, вам начислят дополнительные баллы за безопасную декомпозицию и расширяемость.



На начальных этапах проектирования не пихайте палки себе в колеса. Не создавайте трудностей в дальнейшем, потому что в будущем для расширения и модификации вам потребуется значительно больше времени.

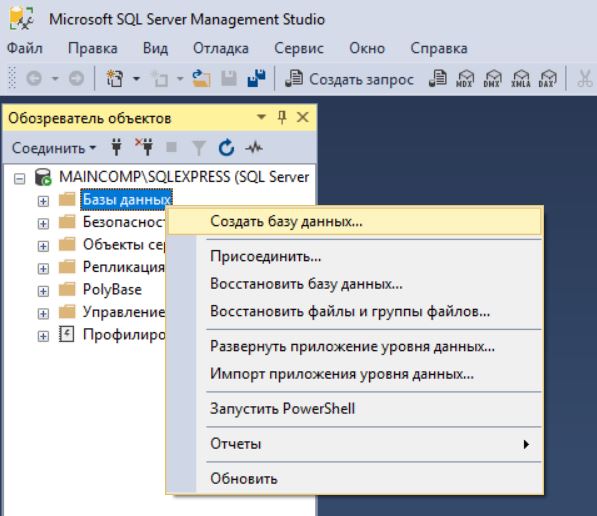
Ладно, лирическое отступление закончилось, начнем с базы данных.

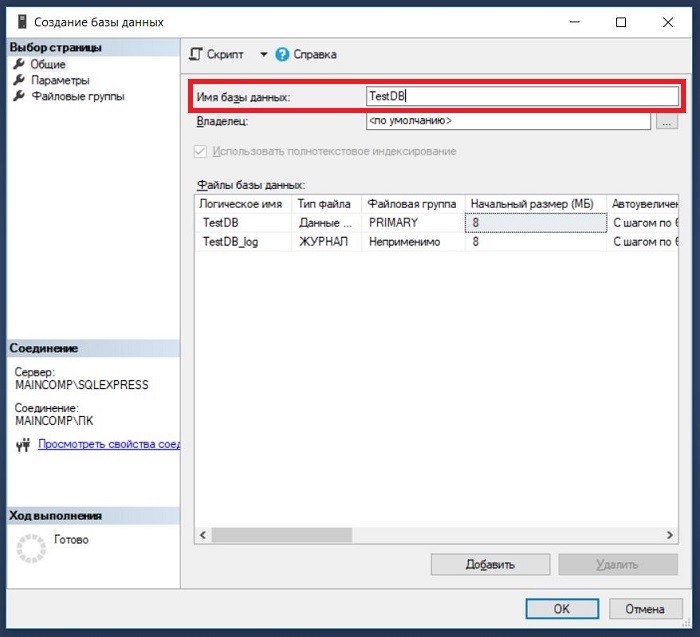
Первым делом вам необходимо создать базу данных или развернуть ее из скрипта, затем следует нормализовать/привести данные из ресурсов под ранее созданную базу данных.

Создавать базу данных исключительно из конструктора или из готового файла, данного в ресурсах, и никак иначе, потому что необходимо беречь время.

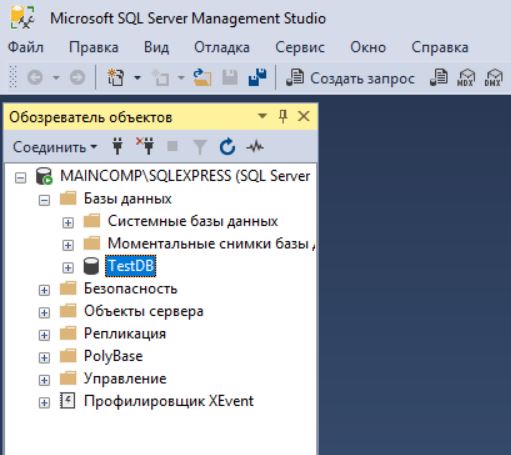
Создать базу данных можно двумя способами:

1. Создать ручками



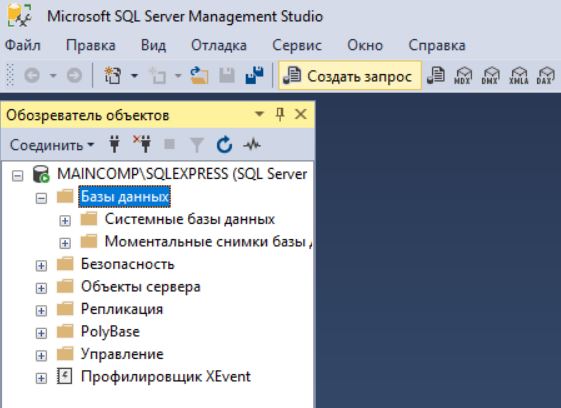


В результате откроется окно «Создание базы данных». Здесь обязательно нужно заполнить только поле «Имя базы данных», остальные параметры настраиваются по необходимости. После того, как Вы ввели имя БД, нажимайте «ОК».

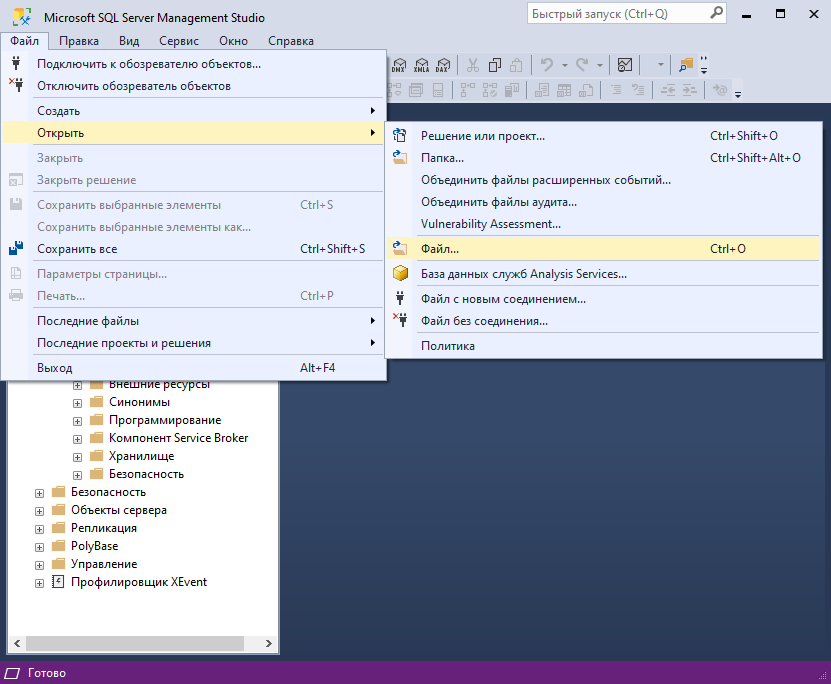


Если БД с таким именем на сервере еще нет, то она будет создана, в обозревателе объектов она сразу отобразится

1. Развернуть скрипт базы



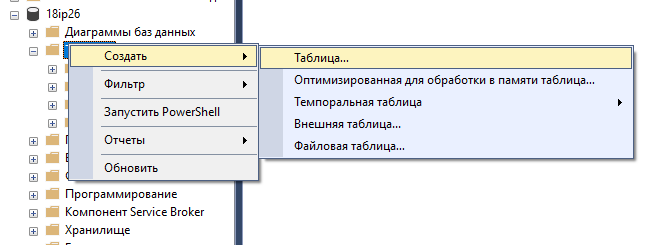
Сначала открываем редактор SQL запросов, для этого щелкаем на кнопку «Создать запрос» на панели инструментов. Затем вводим/вставляем нужный запрос создания базы данных и выполняем. Если скрипт базы данных уже дан, то его можно открыть и выполнить.



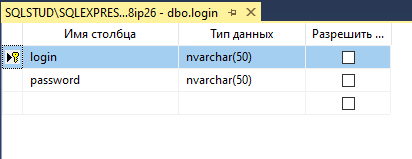
Создание таблиц

Создать таблицы можно двумя способами:

1. Создать с помощью скрипта (см. предыдущий рисунок)
2. Создать ручками

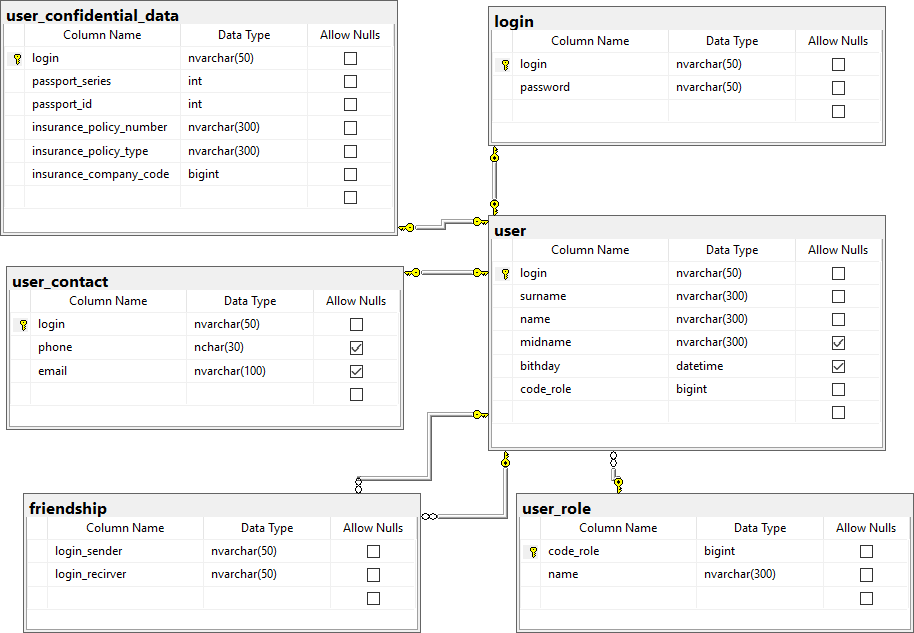


Выделяем контейнер "Таблицы" правой кнопкой мыши и выбрать пункт "Создать" и "Таблица..."



Созданная таблица "Логин и пароль"

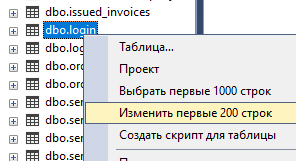
Итогом у вас должна получиться следующая схема базы данных в MS SQL Server Management Studio.



Приступим к импорту данных в базу, для этого необходимо привести данные в общему виду в Excel и потом импортировать данные. Кто не знает, как избавляться от порченых данных следует разобраться в Excel иначе ***гг вп изи катка***.

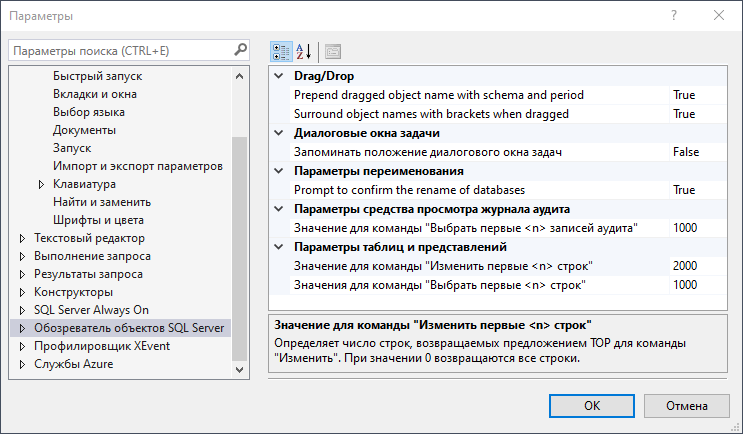
Все данные необходимые для импорта располагаются по пути *./data-files/import-data.xlsx*

Самый быстрый способ заполнить таблицы любыми данными это скопировать их и вставить в таблицу через изменение данных таблицы.



Совет

Для вставки большого количества данных в таблицу, используя способ продемонстрированном на рисунке 8, которые более 200 строк, то вам необходимо зайти в «Сервис/Параметры» и в открывшемся окне открыть раздел «Обозреватель объектов SQL Server».

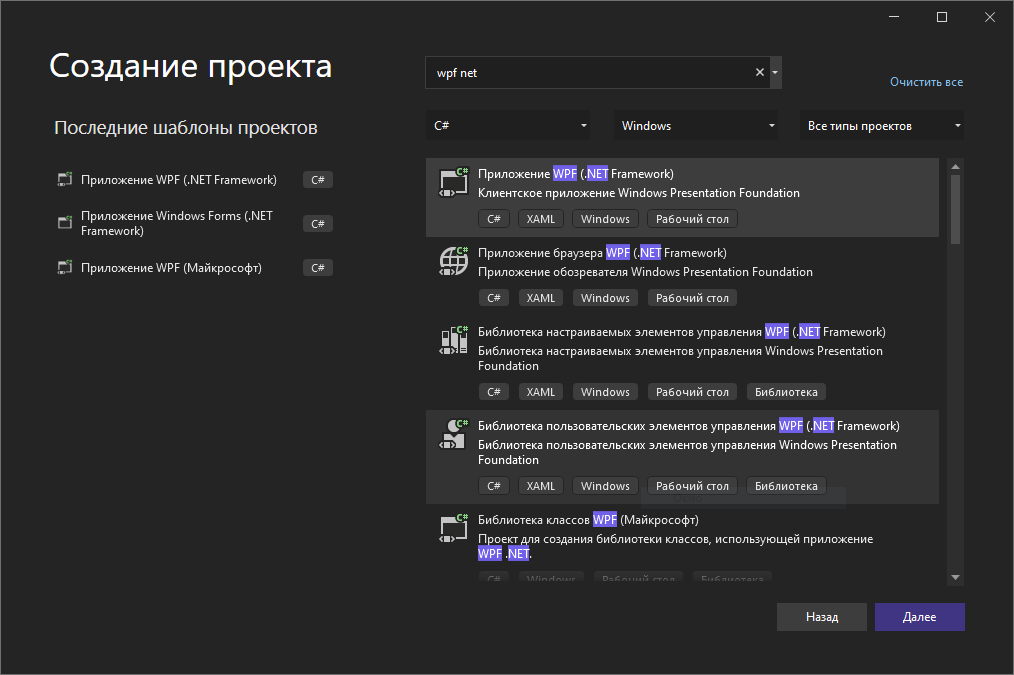


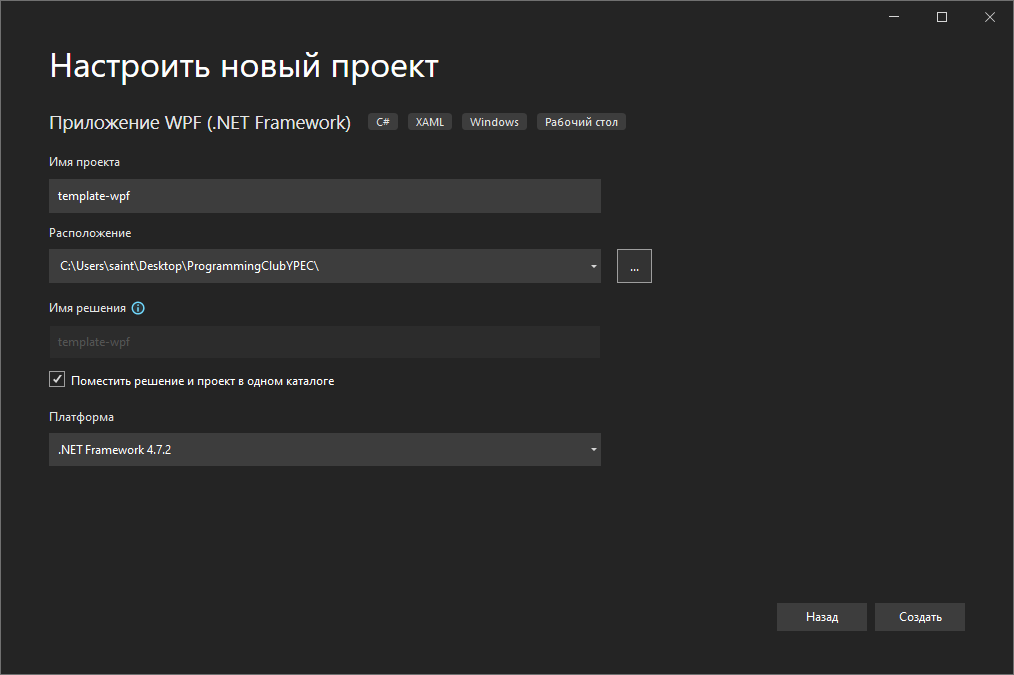
Приступим к реализации каркаса приложения

Для этого нам понадобиться создать приложение WPF в Visual Studio, внизу приведены скрины шагов для создания проекта.

Добро пожаловать в нашу адаптированную версию Шрека.

Глава нашей истории 1: построим мост к Фионе.





Итогом данных телодвижений у вас должен будет создаться проект в указанной папке, если вам не хочется иметь много мусора в корне, тогда уберите галку с чек бокса, если вам пофиг тогда ставьте.

Небольшой обзор созданного проекта по умолчанию

Студия создает сразу же форму MainWindow и делает ее запускаемой, также создает дополнительные конфигурационные файлы для проекта.

Стилистика написания фронта используя язык разметки xaml от майков.

Никогда не пишите разметку в одну строчку – это вам будет только мешать. Объявление и изменение значений полей с аналогичной контекстной нагрузкой пишите в одну строчку. Если более 4 полей в строке, то вы что-то делаете неправильно. Когда объявляете новые UI элементы то между их тегами делайте пустую строчку. Чтобы интерфейс был адаптивным присваивайте элементам расположения внутри макетов. Пример приведен ниже.

<Window x:Class="template\_wpf.MainWindow"

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

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

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

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

xmlns:local="clr-namespace:template\_wpf"

mc:Ignorable="d"

Title="MainWindow"

Height="450" Width="800"

MinHeight="450" MinWidth="800"

>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition />

</Grid.RowDefinitions>

<Frame x:Name="PageFrame"

MinHeight="300" MinWidth="600"

Grid.Column="0" Grid.ColumnSpan="3"

Grid.Row="0" Grid.RowSpan="3"

Content="{Binding CurrentPage}"

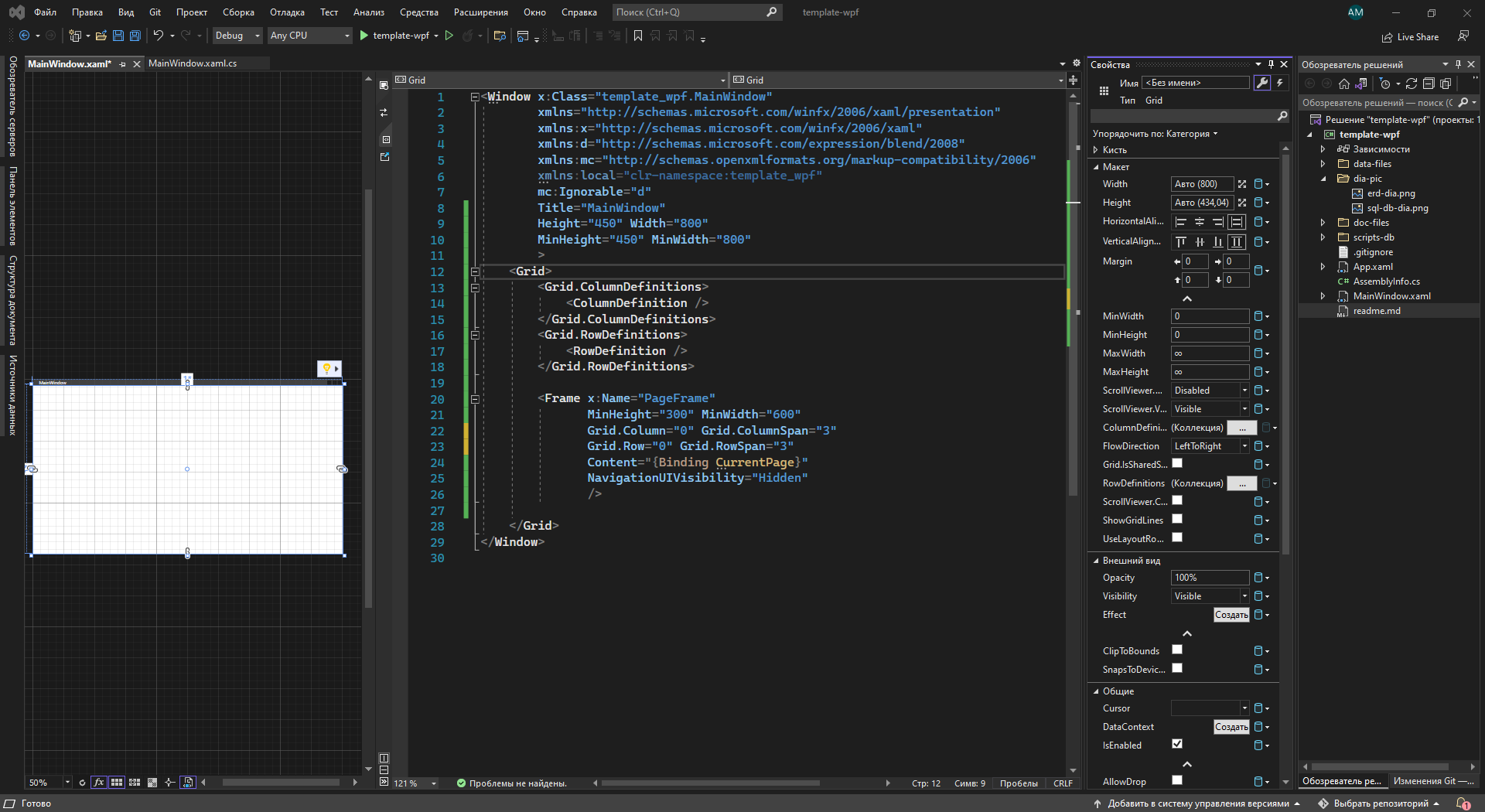
NavigationUIVisibility="Hidden"

/>

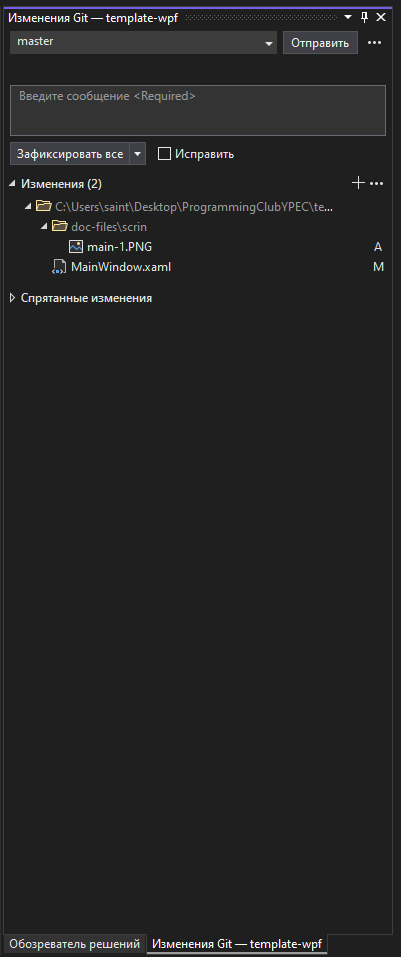
</Grid>

</Window>

Разместите инструменты студии в удобном для вас положении, если вы не знаете, что для вас удобно, то советую мою компоновку инструментов. Разрабатывая на WPF можно оставить панель «Свойства» по умолчанию или скрыть ее вообще. Если же разрабатываем на WinForm, то она вам необходима в полном виде, да и там редактора разметки нет. Но WinForm слишком устарела и заставляет писать много кода для функционирования приложения. Смотрите скрин.



Работать с контролем версий можно из студии переключившись на другую вкладку внутри «Обозревателя решений», смотри ниже скрин.



Я думаю, здесь будет излишне пояснять за функциональность данного инструмента, сами посмотрите и потыкаете по кнопочкам. Мне же удобнее через консоль производить фиксацию изменений.

Ладно, поговорили о среде разработки, пора и за код пояснить.

Недавно (месяца 2 назад) на меня снизошло озарение как быстро создавать приложения для ВСР и ДЭ, но потом я забил на это, потому что так лень было в соло писать такие гигантские документы. Наблюдая за положением вещей и нравов в нашем учебном заведении, выходит, что челы не шарят что да как делать и каждая попытка преподавателей провести практики с подобными условиями, как на вышеперечисленных мероприятиях, приводит к анальной и гортанной боли. В итоге надо помочь типам уменьшить их страдания и дать надежды на лучшее будущее.

Ладно, опять попиздели, теперь пора обсудить смысловую нагрузку разработки приложения.

За основу берется парадигма минимального создания окон и распехивание всего функционала по страницам, которые можно будет применять по несколько раз. Также обобщение функционала для того, чтобы было меньше дублирования кода.

Приступим к реализации

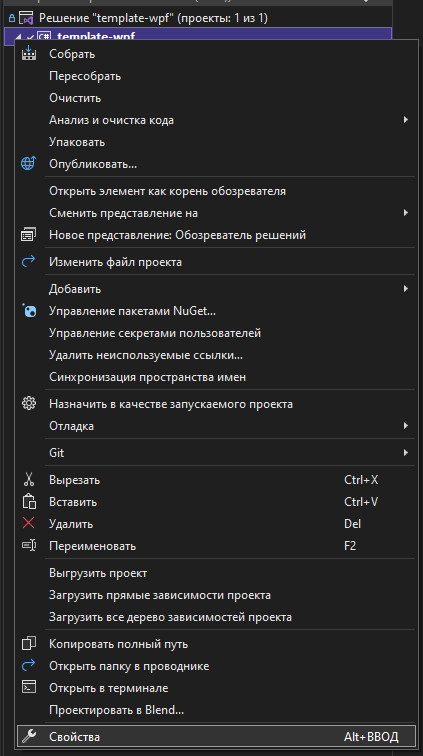
Когда вам придется создавать любое приложение АИС любой предметной области, то вам придется начинать всегда с окна авторизации пользователей. Данное окно будет стартовым и работать только с таблицей login из нашей базы данных.

Ресурсы приложения

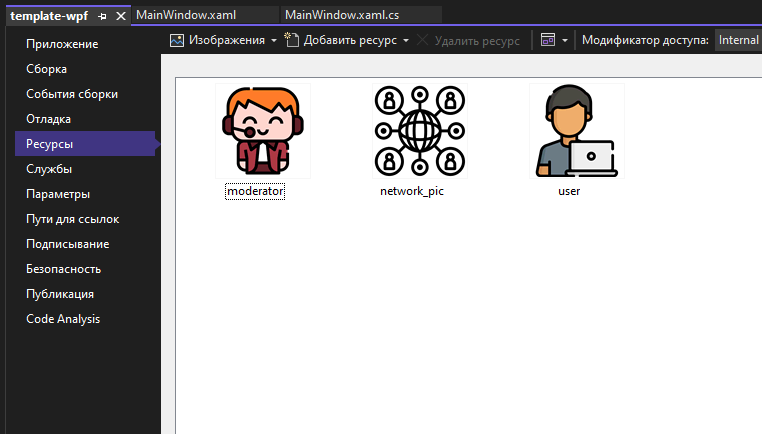
Самая дурная штука на свете – это ресурсы приложения в студии, майки настолько ебанутые, что слов не хватает. Чтобы не было проблем, каждый файл называйте уникально, и даже если у них разные расширения – все равно называйте по-разному, никогда не называйте ресурсы начиная с числа или с других символов.

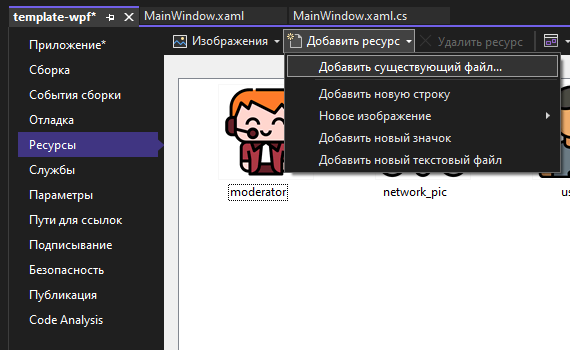
Добавление ресурсов в проект

Внутри «Обозревателя решений» добавляем папку названную «Resources», затем переходим в свойства проекта (см. скрин ниже). Кликаем ПКМ по корню решения и выбираем «Свойства».



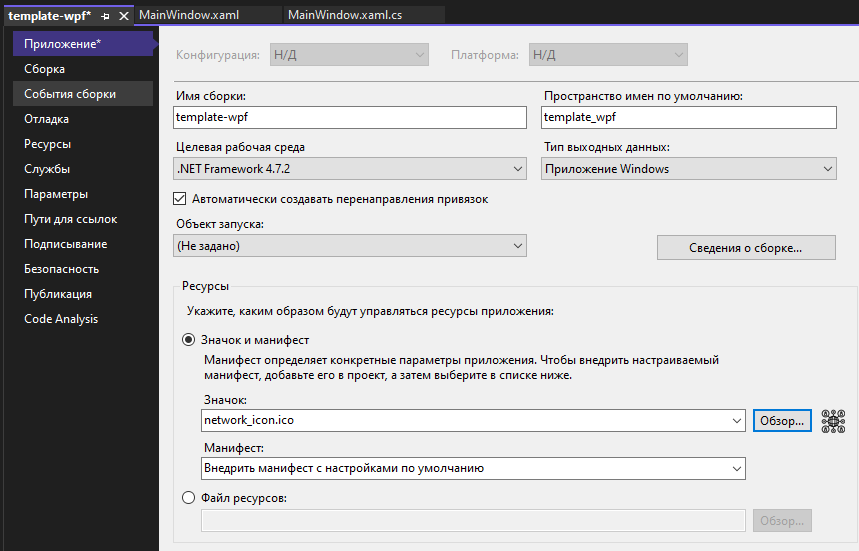
Затем переходим в ресурсы (слева будет список), у разработчиков нет никакой стабильности, интерфейс студии меняется от релиза к релизу – бесит. У меня студия 22 года, в шараге ее обновили, поэтому думаю наши мысли совпадут. Выбираем «Создать или открыть ресурсы сборки», у кого 19 студия, то сразу откроется вкладка.





Переходим на вкладку ресурсов и добавляем существующие файлы, в нашем случае я сразу добавляю иконку приложения в двух форматах, заметьте, когда будете смотреть исходники проекта, то обратите внимание на названия фалов логотипа приложения. Спойлер: *network\_icon.ico & network\_pic.png*. И добавил фотки пользователей для двух ролей.

Дальше вам необходимо установить иконку приложения, для этого переходим в «Приложение/Ресурсы/Значок» и выбираем из тех ресурсов, которые добавили недавно.



Кто дошел до сюда, красавцы, вы сможете получить нихуевые 10 баллов 😊. Это за создание базы данных и импорт данных, также плюсом идет созданная ERD диаграмма и развернули нихуевый (пока что хуевый) проект.

Погнали возьмем на абордаж окно авторизации и переходы между окном авторизации и основным окном.

Создаем окно авторизации, называя ее «signin» и быстренько делаем разметку интерфейса с полями ввода логина и пароля и кнопку авторизации, и не забываем про логотип приложения.

signin.xaml

<Window x:Class="template\_wpf.signin"

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

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

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

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

xmlns:local="clr-namespace:template\_wpf"

mc:Ignorable="d"

Title="signin" Height="450" Width="300"

MinHeight="450" MinWidth="300"

>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.2\*"/>

<ColumnDefinition Width="0.6\*"/>

<ColumnDefinition Width="0.2\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="0.4\*"/>

<RowDefinition Height="0.1\*"/>

<RowDefinition Height="0.1\*"/>

<RowDefinition Height="0.05\*"/>

<RowDefinition Height="0.1\*"/>

<RowDefinition Height="0.2\*"/>

</Grid.RowDefinitions>

<Image x:Name="logo\_app"

Grid.Column="1"

Stretch="Uniform"

/>

<TextBox x:Name="tbLogin"

Width="140" Height="20" Margin="5"

Grid.Row="1" Grid.Column="1"

HorizontalAlignment="Center" VerticalAlignment="Center"

Text="login..." Foreground="Gray"

GotFocus="tbLogin\_GotFocus"

LostFocus="tbLogin\_LostFocus"

TextChanged="tbLogin\_TextChanged"

/>

<PasswordBox x:Name="tbPwd"

Width="140" Height="20"

Grid.Row="2" Grid.Column="1"

HorizontalAlignment="Center" VerticalAlignment="Center"

Margin="5"

Password="password..." Foreground="Gray"

LostFocus="tbPwd\_LostFocus"

GotFocus="tbPwd\_GotFocus"

PasswordChanged="tbPwd\_PasswordChanged"

/>

<Button x:Name="signButton"

Content="Sign in"

Margin="5"

Grid.Row="4" Grid.Column="1"

Click="SignButton\_Click"

/>

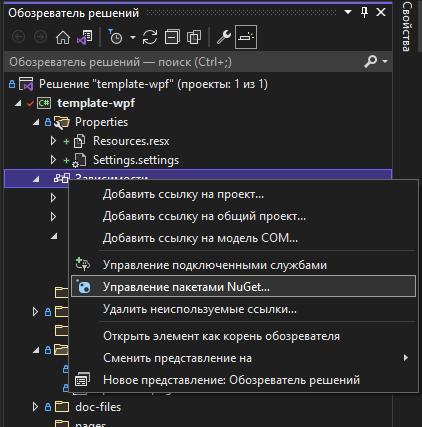
</Grid>

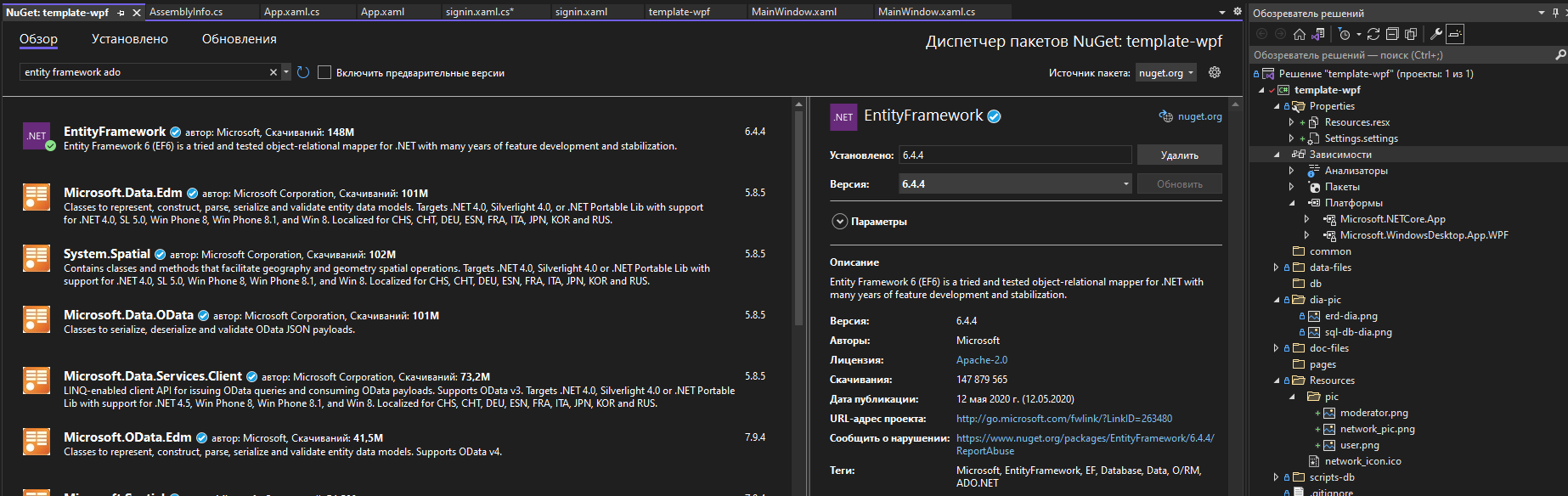
</Window>

Далее, чтобы успешно наращивать функционал авторизации, нам необходимо подключиться к базе данных и для этого дела мы добавляем папочку «db» или как вам угодно, и уже внутри нее используем «Модель ADO.NET EDM». Она упростит нам жизнь по написанию скриптов для получения данных из базы. В шараге она установлена, но может и пропасть. У кого дома не установлена, то устанавливаете ее через NuGet пакеты.

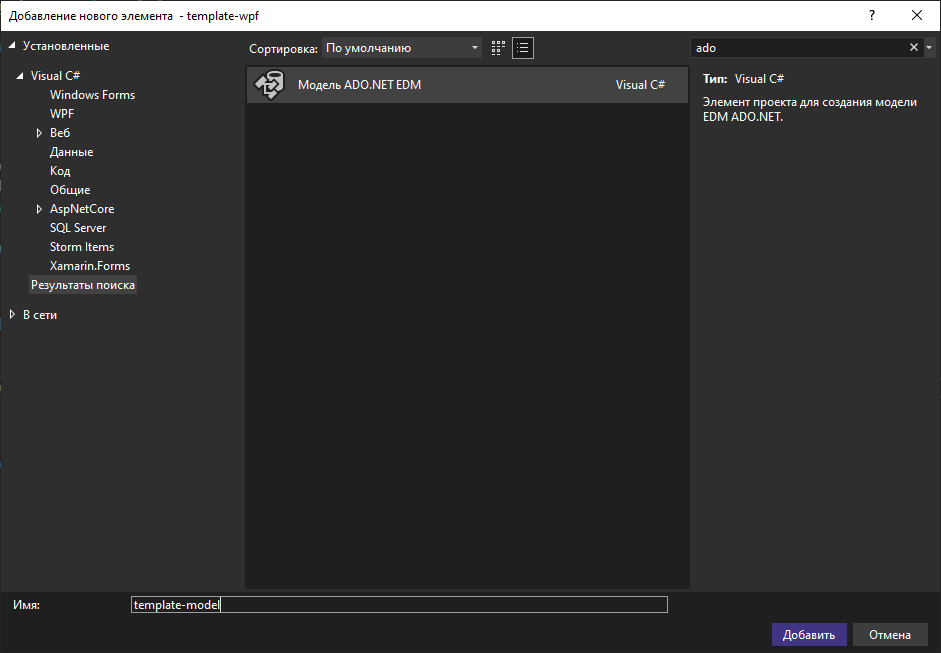
Как установить необходимый пакет

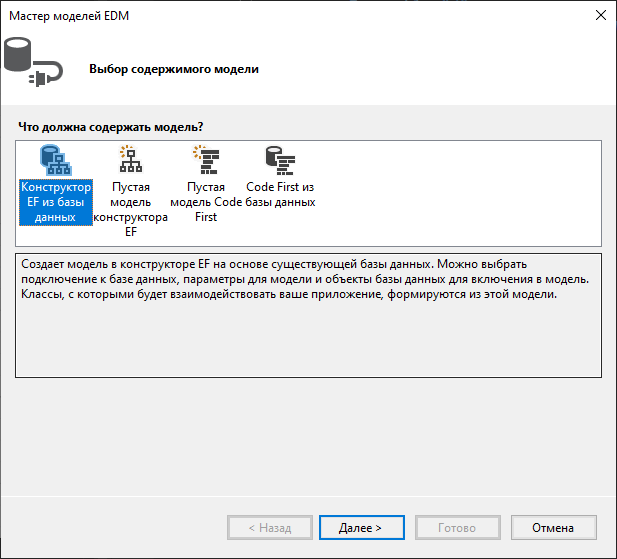
Кликаем ПКМ по зависимостям проекта и выбираем «Управление пакетами NuGet», далее переходим в обзор и вбиваем в строке поиска «entity framework ado» и устанавливаем первый пакет.

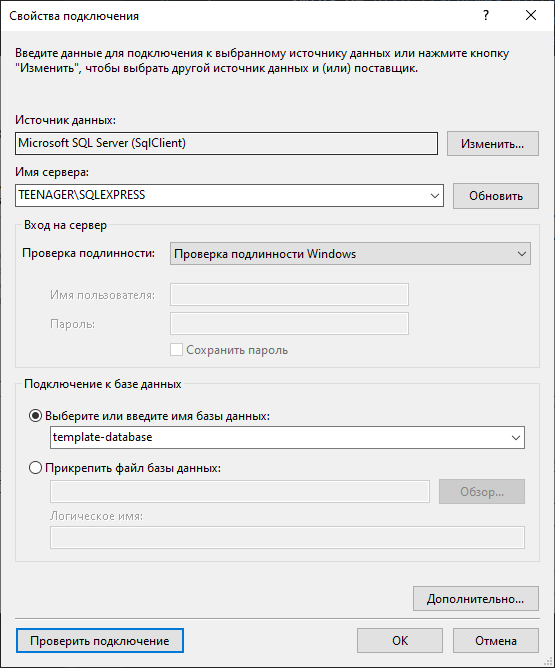


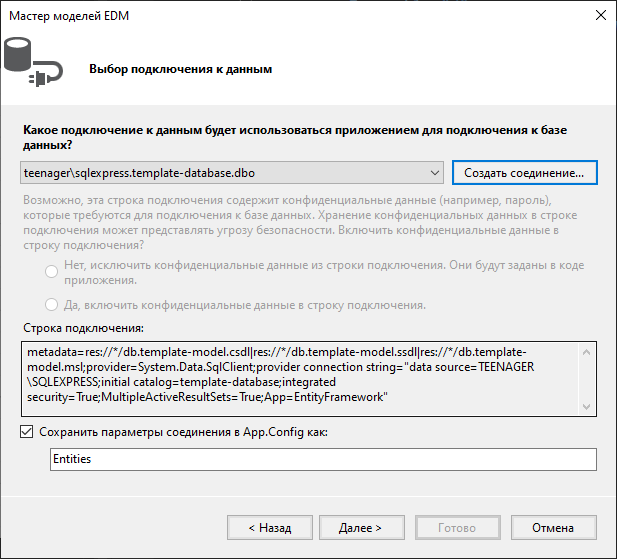


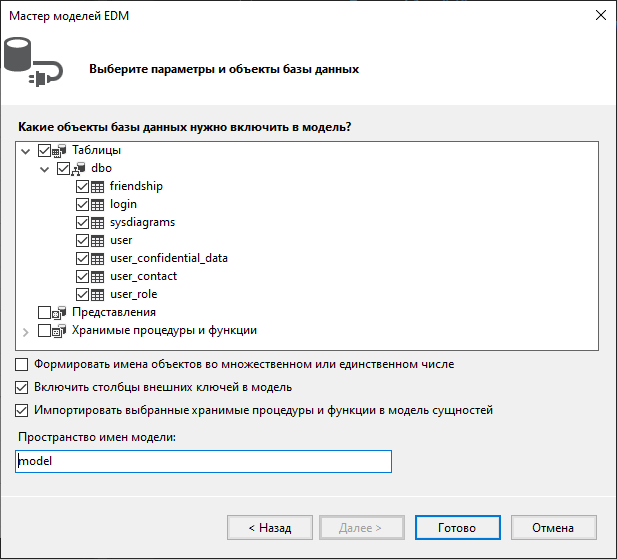
После подключения зависимостей переходим к созданию модели:



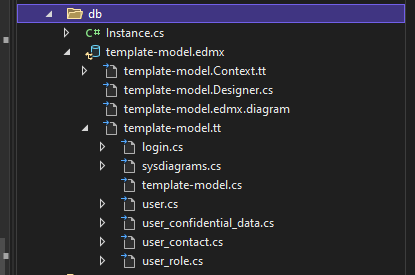








В итоге у вас должны будут создаться классы сущностей базы данных, с помощью которых вы будете обращаться к данным и подтягивать их с помощью LINQ. Как видите на рисунке снизу, нам необходимо создать класс со статическими методами позволяющими взаимодействовать с Entities (классы базы данных).



Instance.cs

internal class Instance

{

private static Entities entities;

public static Entities GetContext()

{

if (entities == null)

entities = new Entities();

return entities;

}

public static bool loginValidated(login login)

{

return !string.IsNullOrEmpty(login.login1) && !string.IsNullOrEmpty(login.password);

}

}

В данном классе мы реализуем паттерн синглтон, он правда вафельный, но тут он нам необходим, потому что позволит уменьшить количество созданий экземпляра базы данных, что повысит производительность. Также создаем статический метода для проверки корректности введенного логина пользователя при авторизации в системе.

Переходим к реализации функциональности авторизации.

Как вы уже заметили из разметки signin.xaml предоставленного ранее, то сначала разберемся с тем как сделать название внутри полей ввода. Все исходные текста вы можете просмотреть в репозитории под названием template-wpf приложенных к данному документу.

signin.cs

private login \_login;

public signin()

{

InitializeComponent();

logo\_app.Source = new BitmapImage(new Uri(System.IO.Path.GetFullPath("../../Resources/pic/network\_pic.png")));

\_login = new login();

}

В самом начале мы объявляем поле логина, в которой в будущем будем записывать данные, введенные пользователем в текст боксы. Далее идет конструктор окна авторизации и там инициализируем логотип и создаем экземпляр класса для нашего поля.

signin.cs

public void clearFieldBox()

{

tbLogin.Text = string.Empty;

tbPwd.Password = string.Empty;

tbLogin\_LostFocus(this, null);

tbPwd\_LostFocus(this, null);

}

Данный метод предназначен для очищения полей окна авторизации, это сделано для того, когда пользователь закрывает MainWindow и вызывается этот метод и очищает ранее введенные данные пользователем. Можно, конечно, избавиться от данной функции, потому что она создает излишки.

signin.cs

Код представленный ниже функции предназначаются для полей ввода данных.

private void tbPwd\_GotFocus(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(\_login.password))

{

tbPwd.Password = string.Empty;

tbPwd.Foreground = Brushes.Black;

}

}

private void tbPwd\_LostFocus(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(\_login.password))

{

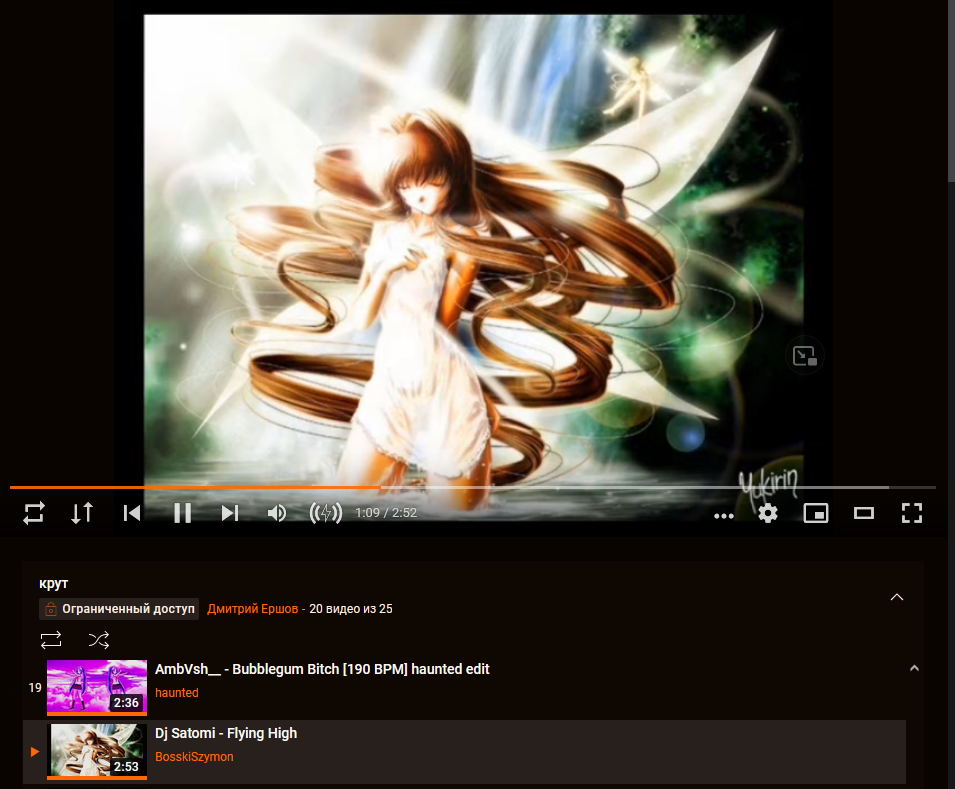
tbPwd.Password = $"password...";

\_login.password = string.Empty;

tbPwd.Foreground = Brushes.Gray;

}

}



Ой….

private void tbLogin\_LostFocus(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(\_login.login1))

{

tbLogin.Text = $"login...";

\_login.login1 = string.Empty;

tbLogin.Foreground = Brushes.Gray;

}

}

private void tbLogin\_GotFocus(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(\_login.login1))

{

tbLogin.Text = string.Empty;

tbLogin.Foreground = Brushes.Black;

}

}

Далее представлены исходные текста функций для отслеживания изменения текста внутри полей ввода.

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

{

if (\_login != null)

\_login.login1 = tbLogin.Text;

}

private void tbPwd\_PasswordChanged(object sender, RoutedEventArgs e)

{

if (\_login != null)

\_login.password = tbPwd.Password;

}

Далее представлены исходные текста кнопки авториазции.

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

{

if (!Instance.loginValidated(\_login))

{

MessageBox.Show("Необходимо заполнить все поля!", "Ошибка ввода", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if ((from l in Instance.GetContext().login

where l.login1 == \_login.login1 && l.password == \_login.password

select l).ToList().Count() != 1)

{

MessageBox.Show("Не удалось авторизироваться!\nЛогин или пароль не корректны.", "Ошибка авторизации", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

Hide();

MainWindow window = new MainWindow(\_login);

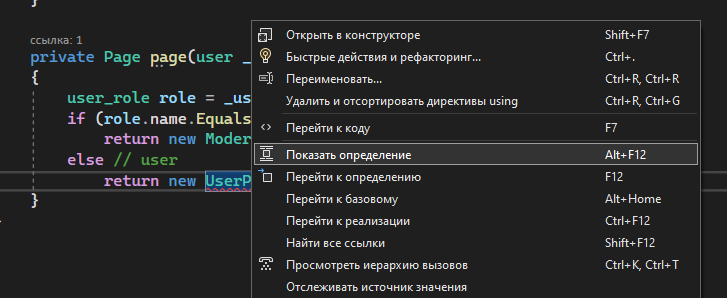
window.Owner = this;

window.Show();

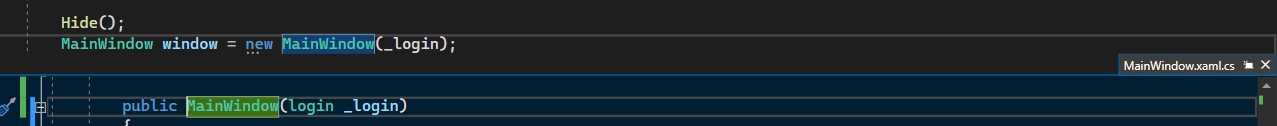
}

В данном методе мы сперва проверяем на корректность логин пользователя, если он не корректен, то выводим диалоговое окно с пояснительным комментарием ошибки и выходим из функции. Если же логин корректен, тогда ищем его в базе данных и если там не одна запись или вообще ее нет, то выводим диалоговое окно с пояснительным комментарием ошибки, в противном случаем все данные верны и существует только один такой логин, тогда мы сворачиваем окно авторизации и создаем экземпляр класса MainWindow и устанавливаем ей свойство Owner и показываем окно.

Для успешной компиляции проекта вам необходимо изменить конструктор класса окна MainWindow. Это можно сделать с помощью такого хука.

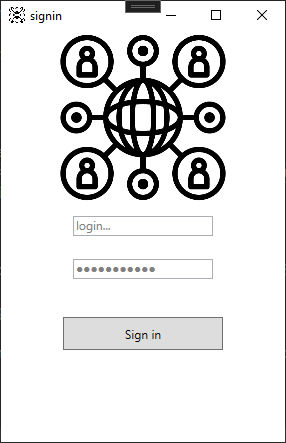


Нажимаем и изменяем параметры класса, правда тут уже я забегаю в будущее, не переживайте мы еще до туда дойдем. А что вам надо продемонстрировано на рисунке ниже.

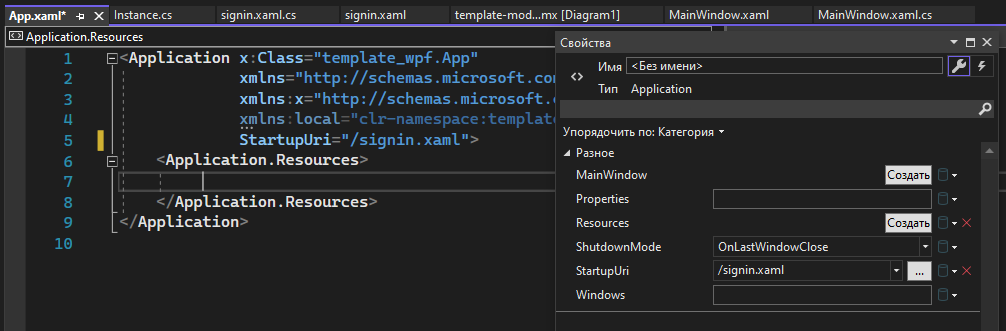


Не забудьте добавить using где располагается класс login.

Итогом копипаста у вас должно получиться следующее окно авторизации.



Но она у вас не запуститься, потому что я вас заскамил и забыл упомянуть один важный факт. Когда вы захотите поменять окно для запуска приложения, то вам следует перейти в файл App.xaml и в нем поменять значение поля StartupUri на необходимое окно. Советую вам изменить это поле в свойствах.



Ладно, пошутили и продолжим дальше шутить, красавцы те, кто допер до сюда, ну я думаю вы нихуя не читаете, ибо кто читает бред шизоида, а пасхалок будет куча, хоть уже почти конец, хотя конец еще не за горами.

Ладно, мы крутые, а теперь перейдем к модернизации нашего основного окна MainWindow.

MainWindow.xaml

<Window x:Class="template\_wpf.MainWindow"

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

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

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

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

xmlns:local="clr-namespace:template\_wpf" xmlns:common="clr-namespace:template\_wpf.common"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800"

MinHeight="450" MinWidth="800"

Closing="Window\_Closing"

>

<Window.Resources>

<ResourceDictionary>

<common:ConvertUserName x:Key="ConvertUserName"/>

</ResourceDictionary>

</Window.Resources>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="50"

MaxWidth="50" MinWidth="50"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="75"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="50"

MaxHeight="50" MinHeight="50"/>

<RowDefinition Height="0.9\*"/>

</Grid.RowDefinitions>

<Rectangle Grid.ColumnSpan="3"

Fill="BlueViolet"/>

<Image x:Name="logoUser"

Grid.Column="0" Grid.Row="0"

Stretch="Uniform"

/>

<TextBlock x:Name="tbNameUser"

Text="{Binding Path=DataContext, RelativeSource={RelativeSource Self}, Converter={StaticResource ConvertUserName}}"

Grid.Column="1" Grid.Row="0"

FontSize="20" TextAlignment="Left"

Margin="10"

VerticalAlignment="Center"

/>

<Button x:Name="exitBtn"

Content="Выход"

Background="LightPink"

Grid.Column="2" Grid.Row="0"

Margin="5, 10"

Click="WindowClose"

/>

<Frame x:Name="PageFrame"

MinHeight="300" MinWidth="600"

Grid.Column="0" Grid.ColumnSpan="3"

Grid.Row="1" Grid.RowSpan="1"

Content="{Binding CurrentPage}"

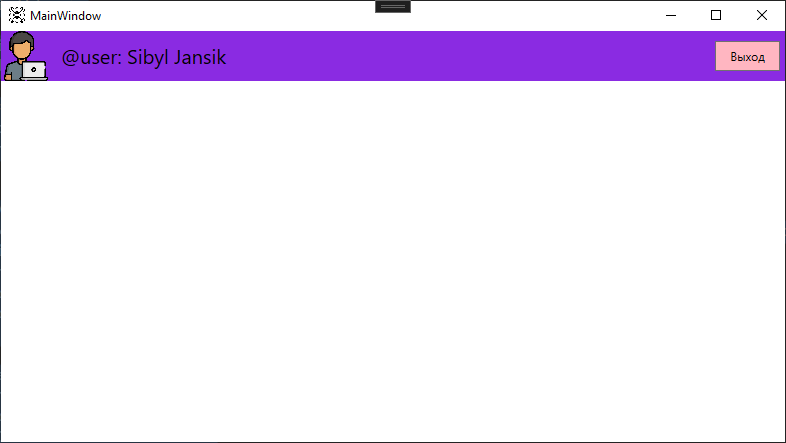
NavigationUIVisibility="Hidden"

/>

</Grid>

</Window>

У вас выйдет вот такое окно:



Концепция нашего окна и последующих созданных окон – будут иметь страницы, занимающие большую полезную область самой формы. На данном окне будет отображаться картинка пользователя, его никнейм с указанием названия роли, кнопка выхода и страница, занимающая всю нижнею часть окна.

Для поддержки концепции страничного приложения будем использовать по кд следующий код:

MainWindow.cs

private Page \_currentPage;

public Page CurrentPage

{

get { return \_currentPage; }

set

{

if (\_currentPage == value)

return;

\_currentPage = value;

OnPropertyChanged("CurrentPage");

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

Он нам позволяет реализовать смену страниц формы и хранения ее. Меньше вопросов больше без осмысленного копипаста. Если интересно что это, то гуглите.

Ладно, перейдем к конструктору нашего окна

private List<string> userRoleNameList;

public MainWindow(login \_login)

{

InitializeComponent();

DataContext = Instance.GetContext().user.Where(p => p.login == \_login.login1).FirstOrDefault();

userRoleNameList = Instance.GetContext().user\_role.Select(p => p.name).ToList();

logoUser.Source = pictureLogo(((user)DataContext).user\_role);

CurrentPage = page((user)DataContext);

PageFrame.Content = CurrentPage;

}

Конструктор окна у нас ищет в базе данных пользователя и присваивает его DataContext окна, из которого в будущем будут подтягиваться данные и изменяться, далее инициализируем список названий ролей, чтобы не харкодить названия ролей в коде. Затем создаем картинку используя метод pictureLogo принимающее роль пользователя или можно будет подтягивать из базы данных само название картинки из таблица привязанной к пользователю. Затем мы создаем страницу по роли пользователя.

Далее представлены методы, которые вызывает конструктор окна. Господа, учитесь читать код, пожалуйста, я реально сейчас зашьюсь, если буду так построчно пояснять.

private ImageSource pictureLogo(user\_role role)

{

return new BitmapImage(new Uri(System.IO.Path.GetFullPath($"../../Resources/pic/{role.name}.png")));

}

private Page page(user \_user)

{

user\_role role = \_user.user\_role;

if (role.name.Equals(userRoleNameList.ElementAt(0)))

return new ModeratorPage(\_user, this);

else // user

return new UserPage(\_user, this);

}

Вот последние два метода нашего основного окна.

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

Owner.Show();

(Owner as signin).clearFieldBox();

}

private void WindowClose(object sender, RoutedEventArgs e)

{

Close();

}

Как вы могли заметить из xaml файла вам не хватает одного класса из common, а именно как конвертировать объект к нужному типу в Binding. Буду краток, создайте класс в common под названием «ConvertUserName.cs» и реализуйте там интерфейс IValueConverter.

[ValueConversion(typeof(user), typeof(string))]

public class ConvertUserName : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

user employee = (user)value;

if (employee == null)

return "";

return $"@{employee.user\_role.name.Trim()}: {employee.surname.Trim()} {employee.name.Trim()} {(employee.midname == null ? "" : employee.midname.Trim())}";

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

return value;

}

}

Тут мы просто конвертируем тип и возвращаем нужный нам формат данных, такая тема работает со всеми Binding’ами для любых пользовательских элементов. Ну и теперь можем запускать сборку и радоваться жизни.

Заебись, все сделали, теперь можем радоваться крутому ентерпрайс приложению, разрешаю позапускать по вводить данные логинов и паролей из базы данных или из файла Excel. Круто, кто молодец? Правильно, вы молодцы!!!

Глава нашей супер-пупер истории 2: шести-головый змей горыныч.