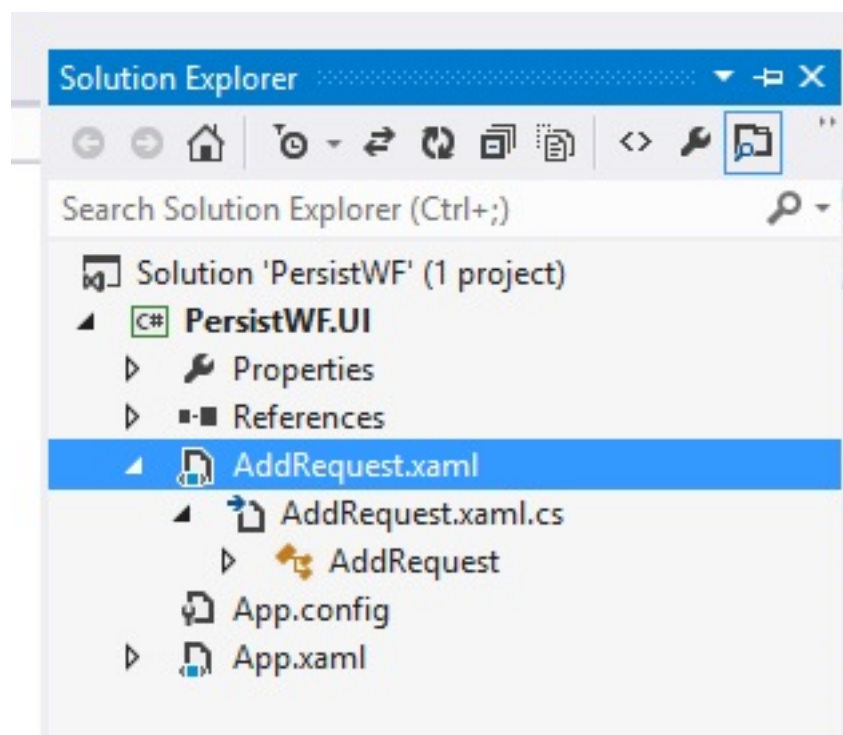


در خیلی از مواقع workflow ها به مرحله‌ای می‌رسند که احتیاج به دستوری از بیرون از فرآیند دارند. در هنگام انتظار، اگر به هر دلیلی workflow از حافظه حذف شود، امکان ادامه فرآیند وجود ندارد. اما می‌توان با Persist (ذخیره) کردن آن، در زمان انتظار و فراخوانی مجدد آن در هنگام نیاز، این ریسک را برطرف نمود.

قصد دارم با این مثال، طریقه persist شدن یک workflow در زمانیکه نیاز به انتظار برای تایید دارد و فراخوانی آن از همان نقطه پس از تایید مربوطه را توضیح دهم.

ساختار اینترفیس کاربری ما WPF می‌باشد. پس در ابتدا یک پروژه از نوع WPF ایجاد می‌کنیم. اسم solution را PersistWF و اسم Project را PersistWF.UI انتخاب می‌کنیم.

در پروژه UI نام فایل MainWindow.xaml را به AddRequest.xaml تغییر می‌دهیم. همچنین اسم کلاس مربوطه را در codebehind



همین طور مقدار StartupUri را هم در app.xaml اصلاح می‌کنیم

```
StartupUri="AddRequest.xaml"
```

Reference های زیر رو هم به پروژه اضافه می‌کنیم

```

•System.Activities
•System.Activities.DurableInstancing
•System.Configuration
•System.Data.Linq
•System.Runtime.DurableInstancing
•System.ServiceModel
•System.ServiceModel.Activities
•System.Workflow.ComponentModel
•System.Runtime.DurableInstancing
•System.Activities.DurableInstancing

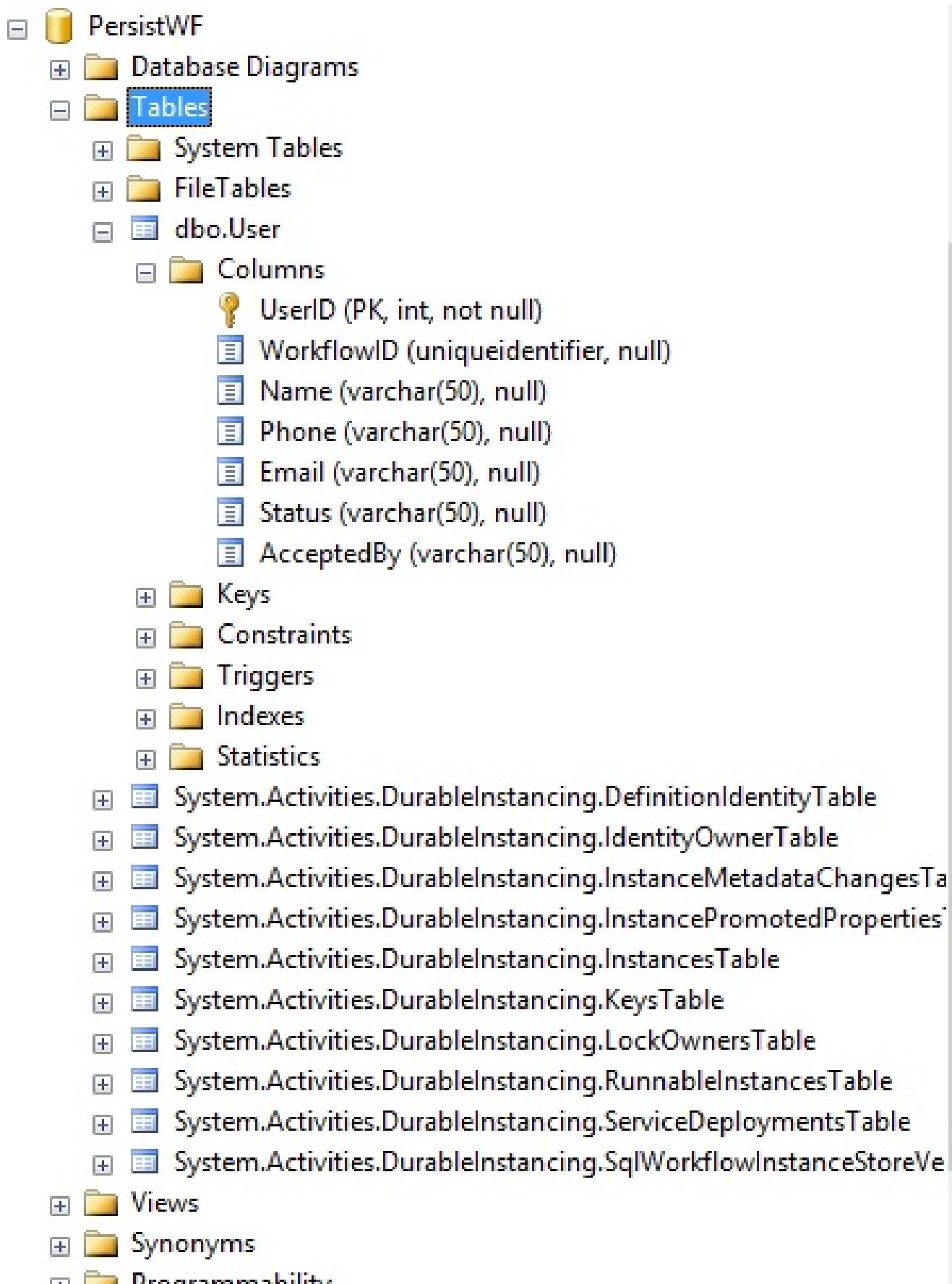
```

قرار است کاربری ثبت نام کند، در فرایند ثبت، منتظر تایید یکی از مدیران قرار می‌گیرد. مدیر، لیست کاربران جدید را می‌بینید، یک کاربر را انتخاب می‌کند؛ مقادیر لازم را وارد می‌کند و سپس پروسه تایید را انجام می‌دهد که فراخوانی فرآیند مربوطه از همان قسمتی است که منتظر تایید مانده است.

برای Persist کردن workflow از کلاس SqlWorkflowInstanceStore استفاده می‌کنم. این شی به connection ای به یک دیتابیس با یک ساختار معین احتیاج دارد. خوشبختانه اسکریپت‌های مورد نیاز این ساختار در پوشه
 [en]\SQL\Microsoft.NET\Framework\v4.0.30319\Windows\Drive] وجود دارند. دو اسکریپت با نام‌های
 SqlWorkflowInstanceStoreSchema و SqlWorkflowInstanceStoreLogic باید به ترتیب در دیتابیس اجرا شوند.

من یک دیتابیس با نام PersistWF ایجاد می‌کنم و اسکریپت‌ها را بر روی آن اجرا می‌کنم. یک جدول هم برای نگهداری کاربران ثبت شده در همین دیتابیس ایجاد می‌کنم.

و شمایل دیتابیس ما پس از اجرا کردن اسکریپت‌ها و ساختن جدول User بدین شکل است:



XAML زیر، ساختار فرم AddRequest می‌باشد که قرار است نقش UI برنامه را ایفا کند. آن را با XAML های پیش فرض عوض کنید.

```

<Window x:Class="PersistWF.UI.AddRequest"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="520" Width="550" Loaded="Window_Loaded">
    <Grid MinWidth="300" MinHeight="100" Width="514">
        <Label Height="30" Margin="5,10,10,10" Name="lblName" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="90" HorizontalContentAlignment="Right">Name:</Label>
        <Label Height="30" Margin="270,10,10,10" Name="lblPhone" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="90" HorizontalContentAlignment="Right">Phone Number:</Label>
        <Label Height="30" Margin="5,40,10,10" Name="lblEmail" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="90" HorizontalContentAlignment="Right">Email:</Label>
        <TextBox Height="25" Margin="100,10,10,10" Name="txtName" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="170" />
        <TextBox Height="25" Margin="365,10,10,10" Name="txtPhone" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="100" />
        <TextBox Height="25" Margin="100,40,10,10" Name="txtEmail" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="300" />
        <Button Height="23" Margin="100,86,0,0" Name="brnRegister" VerticalAlignment="Top"
            HorizontalAlignment="Left" Width="70" Click="brnRegister_Click">Register</Button>
        <ListView x:Name="lstUsers" Margin="10,125,10,10" Height="145" VerticalAlignment="Top"
            ItemsSource="{Binding}" HorizontalContentAlignment="Center"
            SelectionChanged="lstUsers_SelectionChanged" >
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="Current User" Width="480">
                        <GridViewColumn.CellTemplate>
                            <DataTemplate>
                                <StackPanel Orientation="Horizontal">
                                    <TextBlock Text="{Binding Name}" Width="110"/>
                                    <TextBlock Text="{Binding Phone}" Width="70"/>
                                    <TextBlock Text="{Binding Email}" Width="130"/>
                                    <TextBlock Text="{Binding Status}" Width="70"/>
                                    <TextBlock Text="{Binding AcceptedBy}" Width="100"/>
                                </StackPanel>
                            </DataTemplate>
                        </GridViewColumn.CellTemplate>
                    </GridViewColumn>
                </GridView>
            </ListView.View>
        </ListView>
        <Label Height="37" HorizontalAlignment="Stretch" Margin="10,272,5,10" Name="lblSelectedNotes"
            VerticalAlignment="Top" Visibility="Hidden" />
        <Label Height="30" Margin="10,0,0,140" Name="lblAgent" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="40" HorizontalContentAlignment="Left" Visibility="Hidden">Admin
            Name:</Label>
        <TextBox Height="25" Margin="60,0,0,140" Name="txtAcceptedBy" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="190" Visibility="Hidden" />
        <Button Height="25" Margin="270,0,0,140" Name="btnAccept" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="90" Click="btnAccept_Click" Visibility="Hidden">Accept</Button>
        <Label Height="27" HorizontalAlignment="Left" Margin="10,0,0,110" Name="lblEvent"
            VerticalAlignment="Bottom" Width="76">Event Log</Label>
        <ListBox Margin="12,0,5,12" Name="lstEvents" Height="100" VerticalAlignment="Bottom"
            FontStretch="Condensed" FontSize="10" />
    </Grid>
</Window>

```

اگر همه چیز مرتب باشد؛ ساختار فرم شما باید به این شکل باشد

The screenshot shows a WPF application window titled "MainWindow". Inside the window, there is a registration form. The form has four input fields: "Name:", "Phone Number", "Email:", and "Notes:". To the right of the "Notes:" field is a "Register" button. Below the registration fields is a section titled "Current User" which contains a large empty text area. At the bottom of the window is another section titled "Event Log" which also contains a large empty text area.

اکثر workflow ها از activity معروف Wrteline استفاده می کنند که برای نمایش یک رشته به کار می رود. ما هم در workflow مثالمان از این Activity استفاده می کنیم. اما برای اینکه مقادیری که توسط این Activity ایجاد می شوند در کادر event log فرم خودمان نمایش داده شود؛ احتیاج داریم که یک TextWriter سفارشی برای خودمان ایجاد کنیم. اما قبل از آن یک کلاس static در پروژه ایجاد می کنیم که بتوانیم در هر قسمتی، به فرم دسترسی داشته باشیم.

کلاسی را با نام ApplicationInterface به پروژه اضافه کرده و یک Property استاتیک از جنس فرم AddRequest هم برای آن تعریف می کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PersistWF.UI
{
    public static class ApplicationInterface
    {
```

```

    }
    public static AddRequest _app { get; set; }
}

```

به Constructor کلاس موجود در فایل AddRequest.xaml.cs این خط کد رو اضافه می‌کنم

```

public AddRequest()
{
    InitializeComponent();
    ApplicationInterface._app = this;
}

```

این دو متد را هم به این کلاس اضافه می‌کنیم

```

private void AddEvent(string szText)
{
    lstEvents.Items.Add(szText);
}
public ListBox GetEventListBox()
{
    return this.lstEvents;
}

```

متد اول برای اضافه کردن یک event Log و متد دوم هم که کنسول لاگ را در اختیار درخواست کننده‌اش قرار می‌دهد.

و حالا کلاس TextWriter سفارشی‌امان را می‌نویسیم. یک کلاس به نام ListBoxTextWriter به پروژه اضافه می‌کنیم که از TextWriter مشتق می‌شود و محتویات آنرا در زیر می‌بینید:

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;

namespace PersistWF.UI
{
    public class ListBoxTextWriter : TextWriter
    {
        const string textClosed = "This TextWriter must be opened before use";
        private Encoding _encoding;
        private bool _isOpen = false;
        private ListBox _listBox;
        public ListBoxTextWriter()
        {
            // Get the static list box
            _listBox = ApplicationInterface._app.GetEventListBox();
            if (_listBox != null)
                _isOpen = true;
        }
        public ListBoxTextWriter(ListBox listBox)
        {
            this._listBox = listBox;
            this._isOpen = true;
        }
        public override Encoding Encoding
        {
            get
            {
                if (_encoding == null)
                {
                    _encoding = new UnicodeEncoding(false, false);
                }
                return _encoding;
            }
        }
    }
}

```

```

    public override void Close()
    {
        this.Dispose(true);
    }
    protected override void Dispose(bool disposing)
    {
        this._isOpen = false;
        base.Dispose(disposing);
    }
    public override void Write(char value)
    {
        if (!this._isOpen)
            throw new ApplicationException(textClosed); ;
        this._listBox.Dispatcher.BeginInvoke(new Action(() =>
this._listBox.Items.Add(value.ToString())));
    }
    public override void Write(string value)
    {
        if (!this._isOpen)
            throw new ApplicationException(textClosed);
        if (value != null)
            this._listBox.Dispatcher.BeginInvoke(new Action(() =>
this._listBox.Items.Add(value)));
    }
    public override void Write(char[] buffer, int index, int count)
    {
        String toAdd = "";
        if (!this._isOpen)
            throw new ApplicationException(textClosed); ;
        if (buffer == null || index < 0 || count < 0)
            throw new ArgumentOutOfRangeException("buffer");
        if ((buffer.Length - index) < count)
            throw new ArgumentException("The buffer is too small");
        for (int i = 0; i < count; i++)
            toAdd += buffer[i];
        this._listBox.Dispatcher.BeginInvoke(new Action(() => this._listBox.Items.Add(toAdd)));
    }
}
}
}

```

همان طور که می بینید کلاس ListBoxTextWriter از کلاس abstract TextWriter مشتق شده و پیاده سازی از متد Write را فراهم می کند تا یک رشته را به کنترل ListBox اضافه کند. (البته سه تا از این متدها را Override می کنیم تا بتوانیم یک رشته، یک کاراکتر و یا آرایه ای از کاراکترها را به ListBox اضافه کنیم) در constructor پیشفرض از کلاس ApplicationInterface استفاده کردیم تا بتوانیم کنترل lstEvents را از فرم اصلی برنامه به دست بیاوریم. برای Add کردن از Dispatcher و متد BeginInvoke مرتبط با آن استفاده کردیم. این کار، متد را قادر می سازد حتی وقتی که از یک thread متفاوت فراخوانی می شود، کار کند.

حالا می توانیم از این کلاس، به عنوان مقدار خاصیت TextWriter برای WriteLine استفاده کنیم.

به کلاس ApplicationInterface برگردیم تا متد زیر را هم به آن اضافه کنیم

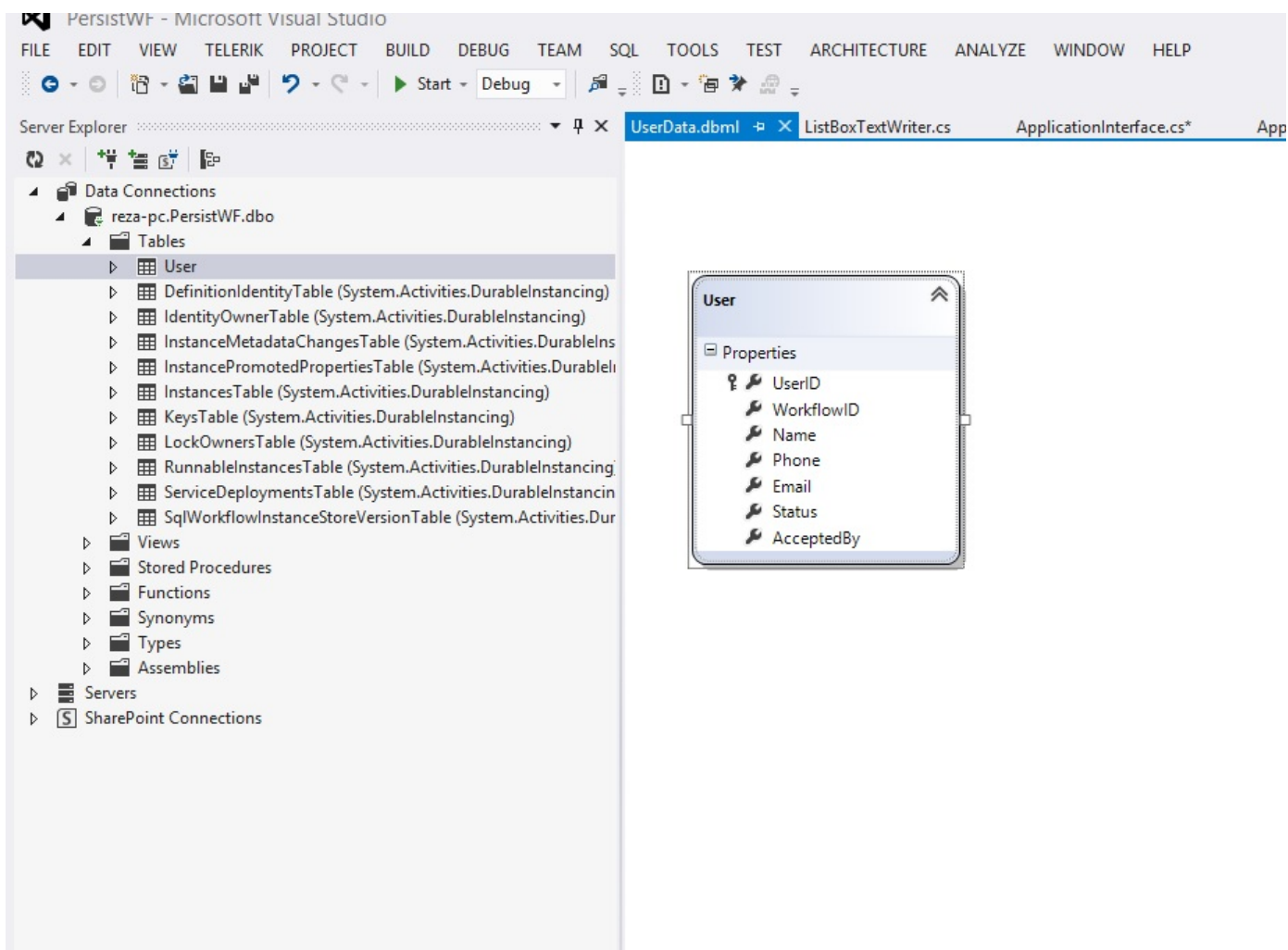
```

public static void AddEvent(String status)
{
    if (_app != null)
    {
        new ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
    }
}

```

این هم از constructor دومی استفاده می کنه برای معرفی ListBox.

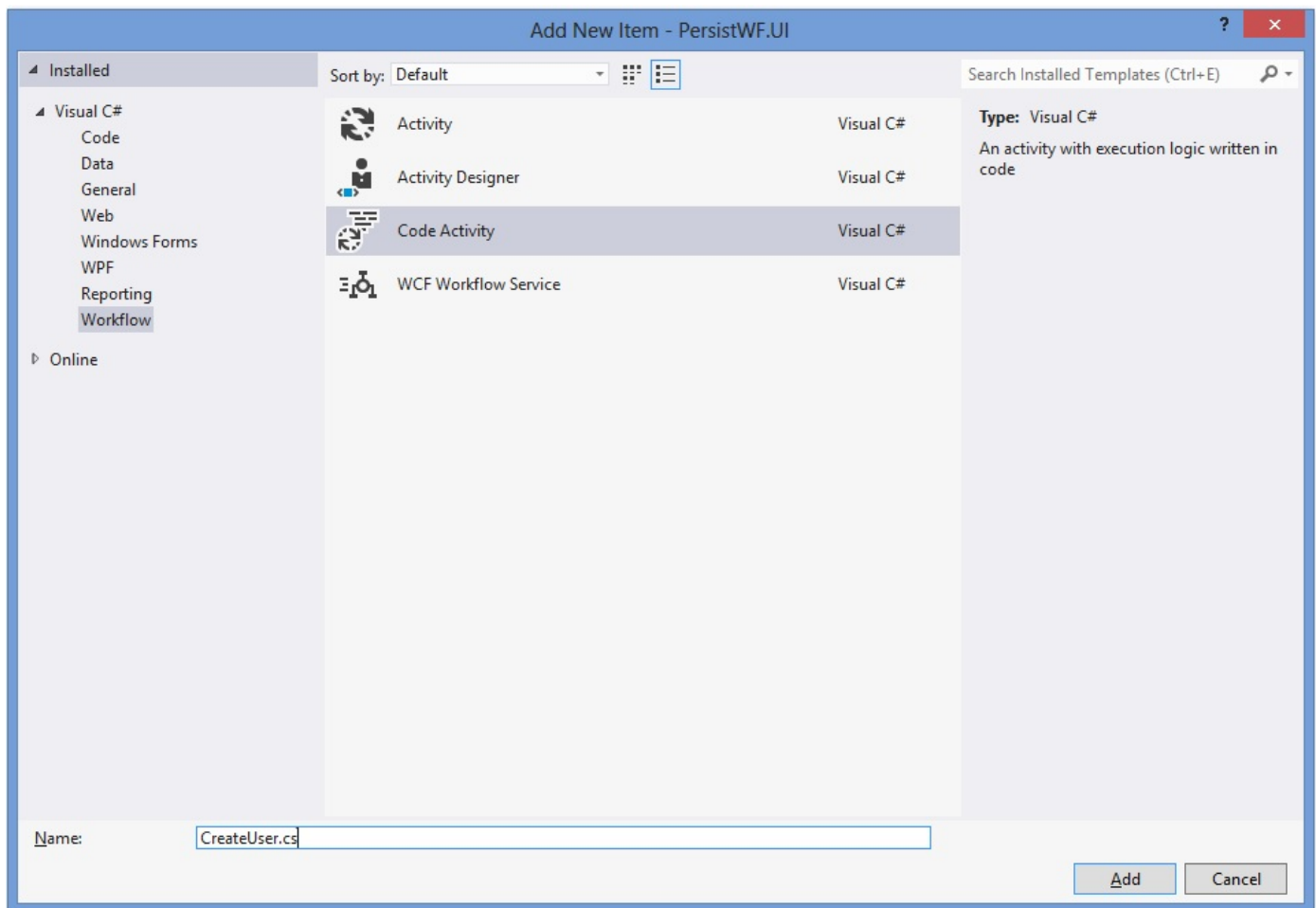
برای ارتباط با دیتابیس از LINQ to SQL استفاده می کنیم تا User رو ذخیره و بازیابی کنیم. به پروژه یک آیتم از نوع LINQ to SQL با نام UserData.dbml اضافه می کنیم. به دیتابیس متصل شده و جدول User رو به محیط Design می کشیم. در ادامه برای شی کلاس SQLWorkflowInstanceStore هم از همین Connectionstring استفاده می کنیم.



برای ایجاد workflow مورد نظر، به دو Activity سفارشی احتیاج داریم که باید خودمان ایجاد نماییم. یک پوشه با نام Activities به پروژه اضافه می‌کنم تا کلاس‌های مورد نظر را آن‌جا ایجاد کنیم.

1. یک Activity برای ایجاد User

این Activity تعدادی پارامتر از نوع InArgument دارد که توسط آن‌ها یک Instance از کلاس User ایجاد می‌کند و در حقیقت آن را به دیتابیس می‌فرستد و ذخیره می‌کند. Connectionstring را هم می‌شود توسط یک آرگومان ورودی دیگر مقدار دهی کرد. یک آرگومان خروجی هم برای این Activity در نظر می‌گیریم تا User ایجاد شده را برگردانیم. روی پوشه‌ی Activities کلیک راست می‌کنیم و Add - NewItem را انتخاب می‌کنیم. از لیست workflow ها Template مربوط به CodeActivity را انتخاب کرده و یک CodeActivity با نام CreateUser ایجاد می‌کنیم



محتویات این کلاس را هم مانند زیر کامل می‌کنیم

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;

namespace PersistWF.UI.Activities
{
    public sealed class CreateUser : CodeActivity
    {
        public InArgument<string> Name { get; set; }
        public InArgument<string> Email { get; set; }
        public InArgument<string> Phone { get; set; }
        public InArgument<string> ConnectionString { get; set; }

        public OutArgument<User> User { get; set; }

        protected override void Execute(CodeActivityContext context)
        {
            // ایجاد کاربر
            User user = new User();
            user.Email = Email.Get(context);
            user.Name = Name.Get(context);
            user.Phone = Phone.Get(context);
            user.Status = "New";
            user.WorkflowID = context.WorkflowInstanceId;
            UserDataDataContext db = new UserDataDataContext(ConnectionString.Get(context));
            db.Users.InsertOnSubmit(user);
            db.SubmitChanges();
        }
    }
}
```

```

        User.Set(context, user);
    }
}

```

متد Execute ، توسط مقادیری که به عنوان پارامتر دریافت شده، یک شی از کلاس User ایجاد می‌کند و به کمک DataContext آنرا در دیتابیس ذخیره کرده و در آخر User ذخیره شده را در اختیار پارامتر خروجی قرار می‌دهد.

1. یک Activity برای انتظار دریافت تایید

این Activity قرار است Workflow را Idle کند تا زمانیکه مدیر دستور تایید را با فراخوانی مجدد workflow از این همین قسمت صادر نماید.

این Activity باید از NativeActivity مشتق شده و برای اینکه workflow را وادار به معلق شدن کند کافی‌است خاصیت CanInduceIdle را با مقدار برگشتی override , true کنیم.

مثل قسمت قبل یک CodeActivity ایجاد می‌کنیم. اینبار با نام WaitForAccept که محتویاتش را هم مانند زیر تغییر می‌دهیم.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
using System.Workflow.ComponentModel;

namespace PersistWF.UI.Activities
{
    public sealed class WaitForAccept<T> : NativeActivity<T>
    {
        public WaitForAccept()
            :base()
        {
        }
        public string BookmarkName { get; set; }
        public OutArgument<T> Input { get; set; }

        protected override void Execute(NativeActivityContext context)
        {
            context.CreateBookmark(BookmarkName, new BookmarkCallback(this.Continue));
        }

        private void Continue(NativeActivityContext context, Bookmark bookmark, object value)
        {
            Input.Set(context, (T)value);
        }
        protected override bool CanInduceIdle
        {
            get
            {
                return true;
            }
        }
    }
}

```

این کلاس را generic نوشتیم تا به جای User بشود هر پارامتر دیگه‌ای را به آن ارسال کرد. در واقع وقتی workflow به این Activity می‌رسد، Idle می‌شود. این activity یک bookmark هم ایجاد می‌کند. ما وقتی workflow را با این bookmark فراخوانی کنیم؛ workflow از همینجا ادامه می‌یابد. فراخوانی bookmark می‌تواند همراه با وارد کردن یک object باشد. متد Continue آن object را به آرگومان خروجی می‌دهد تا مسیر workflow را طی کند. ما User هایی را که به این نقطه رسیدن نمایش می‌دهیم. مدیر اونها را دیده و با مقدار دهی فیلد AcceptedBy، آن User را از اینجا

به workflow می‌فرستد و ما user وارد شده را در ادامه‌ی فرآیند Accept می‌کنیم.

برای ایجاد workflow هم می‌توانید از designer استفاده کنید و هم می‌توانید کد مربوط به workflow را پیاده سازی کنید.

برای پیاده سازی از طریق کد، یک کلاس با نام UserWF ایجاد می‌کنیم و محتویات workflow را مانند زیر پیاده سازی خواهیم کرد:



```
using PersistWF.UI.Activities;
using System;
using System.Activities;
using System.Activities.Statements;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PersistWF.UI
{
    public sealed class UserWF : Activity
    {
        public InArgument<string> Name { get; set; }
        public InArgument<string> Email { get; set; }
        public InArgument<string> Phone { get; set; }
        public InArgument<string> ConnectionString { get; set; }
        public InArgument<TextWriter> Writer { get; set; }

        public UserWF()
        {
            Variable<User> User = new Variable<User> { Name = "User" };
            this.Implementation = () => new Sequence
            {
                DisplayName = "EnterUser",
                Variables = { User },
                Activities = {
                    new CreateUser // 1. ایجاد کاربر با ورود پارامترهای ورودی
                    {
                        ConnectionString = new InArgument<string>(c=> ConnectionString.Get(c)),
                        Email = new InArgument<string>(c=> Email.Get(c)),
                        Name = new InArgument<string>(c=> Name.Get(c)),
                        Phone = new InArgument<string>(c=> Phone.Get(c)),
                        User = new OutArgument<User>(c=> User.Get(c))
                    },
                    new WriteLine // 2. لاگ مربوط به ذخیره کاربر
                    {
                        TextWriter = new InArgument<TextWriter>(c=> Writer.Get(c)),
                        Text = new InArgument<string>(c=> string.Format("User {0} Registered and
waiting for Accept", Name.Get(c) ) )
                    },
                    new InvokeMethod
                    {
                        TargetType = typeof(ApplicationInterface), // 3. برای به روزرسانی لیست
کاربران ثبت شده در نمایش فرم
                        MethodName = "NewUser",
                        Parameters =
                        {
                            new InArgument<User>(env => User.Get(env))
                        }
                    },
                    new WaitForAccept<User> // 4. اینجا فرایند متوقف می‌شود و منتظر تایید مدیر می‌ماند
                    {
                        BookmarkName = "GetAcceptes",
                        Input = new OutArgument<User>(env => User.Get(env))
                    },
                    new WriteLine // 5. لاگ مربوط به تایید شدن کاربر
                    {
                        TextWriter = new InArgument<TextWriter>(c=> Writer.Get(c)),
                        Text = new InArgument<string>(c=> string.Format("User {0} Acceptor by
{1}",Name.Get(c),User.Get(c).AcceptedBy))
                    }
                }
            };
        }
    }
}
```

```
}
```


اگر بخوایم از Designer استفاده کنیم. فرایندمان چیزی شبیه شکل زیر خواهد بود

 Sequence 

▼



CreateUser

▼

 WriteLine

Text

▼

 InvokeMethod 

TargetType


TargetObject

MethodName

▼

WaitForAccept<User>

▼

 WriteLine

Text

▼

به Application بر می‌گردیم تا آن را پیاده سازی کنیم. ابتدا به app.config که اتوماتیک ایجاد شده رفته تا اسم ConnectionString رو به UserGenerator تغییر دهیم. محتویات درون app.config به شکل زیر است.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="UserGenerator"
      connectionString="Data Source=.;Initial Catalog=PersistWF;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

در کلاس AddRequest کد زیر را اضافه می‌کنم. برای نگهداری مقدار connectionString

```
private string _connectionString = "";
```

همچنین کدهای زیر را به رویداد Load فرم اضافه می‌کنم تا مقدار connectionString را از Config بخوانم:

```
Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
ConnectionStringSection css =
(ConfigurationStringsSection)config.GetSection("connectionStrings");
_connectionString = css.ConnectionStrings["UserGenerator"].ConnectionString;
```

خط زیر را هم به کلاس AddRequest اضافه نمایید.

```
private InstanceStore _instanceStore;
```

این ارجاعیه به کلاس InstanceStore که برای Persist و Load کردن workflow از آن استفاده می‌کنیم و کدهای زیر را هم به رویداد Load فرم اضافه می‌کنیم.

```
_instanceStore = new SqlWorkflowInstanceStore(_connectionString);
InstanceView view = _instanceStore.Execute(_instanceStore.CreateInstanceHandle(), new
CreateWorkflowOwnerCommand(), TimeSpan.FromSeconds(30));
_instanceStore.DefaultInstanceOwner = view.InstanceOwner;
```

InstanceStore یک کلاس abstract می باشد که همه‌ی Provider های مربوط به persistence از آن مشتق می‌شوند. در این پروژه من از کلاس SqlWorkflowInstanceStore استفاده کردم تا workflow ها را در دیتابیس SQL Server ذخیره کنم.

برای ایجاد یک Request مقادیر را از فرم دریافت کرده، یک User ایجاد می‌کنیم و آن را در فرآیند به جریان می‌اندازیم. این کار را در رویداد کلیک دکمه Register انجام می‌دهیم

```
private void btnRegister_Click(object sender, RoutedEventArgs e)
{
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("Name", txtName.Text);
    parameters.Add("Phone", txtPhone.Text);
    parameters.Add("Email", txtEmail.Text);
    parameters.Add("ConnectionString", _connectionString);
    parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
    WorkflowApplication i = new WorkflowApplication
    (new UserWF(), parameters);
    // Setup persistence
    i.InstanceStore = _instanceStore;
    i.PersistableIdle = (waiea) => PersistableIdleAction.Unload;
    i.Run();
}
```

پارامترهای ورودی را از روی فرم مقدار دهی می‌کنیم. یک شی از کلاس WorkflowApplication ایجاد می‌کنیم. خاصیت InstanceStore آن را با Store ای که ایجاد کردیم مقدار دهی می‌کنیم. توسط رویداد PersistableIdle فرآیند رو مجبور می‌کنیم به Persist شدن و Unload شدن.

و سپس فرایند را اجرا می‌کنم.

اگر یادتان باشد، در فرآیند، از یک InvoiceMethod استفاده کردیم. متد مورد نظر را هم در کلاس ApplicationInterface.cs ایجاد می‌کنیم.

```
public static void NewUser(User l)
{
    if (_app != null)
        _app.AddNewUser(l);
}
```

همین طور که می‌بینید، یک متد هم در کلاس AddRequest ایجاد می‌شود؛ با این محتوا

```
public void AddNewUser(User l)
{
    this.lstUsers.Dispatcher.BeginInvoke(new Action(() => this.lstUsers.Items.Add(l)));
}
```

این متد فقط یک کاربر را به لیست کاربران اضافه می‌کند. این لیست همه کاربران را نشان می‌دهد. توسط رویداد SelectionChanged این کنترل، کاربر انتخاب شده را بررسی کرده در صورتی که کاربر جدید باشد، امکان تایید شدن را برایش فراهم می‌کنیم؛ که نمایش دکمه تایید است.

```
private void lstUsers_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (lstUsers.SelectedIndex >= 0)
    {
        User l = (User)lstUsers.Items[lstUsers.SelectedIndex];
        lblSelectedNotes.Visibility = Visibility.Visible;
        if (l.Status == "New")
        {
            lblAgent.Visibility = Visibility.Visible;
            txtAcceptedBy.Visibility = Visibility.Visible;
            btnAccept.Visibility = Visibility.Visible;
        }
    }
}
```

```

        else
        {
            lblAgent.Visibility = Visibility.Hidden;
            txtAcceptedBy.Visibility = Visibility.Hidden;
            btnAccept.Visibility = Visibility.Hidden;
        }
    }
    else
    {
        lblSelectedNotes.Content = "";
        lblSelectedNotes.Visibility = Visibility.Hidden;
        lblAgent.Visibility = Visibility.Hidden;
        txtAcceptedBy.Visibility = Visibility.Hidden;
        btnAccept.Visibility = Visibility.Hidden;
    }
}
}

```

و برای رویداد کلیک دکمه تایید کاربر :

```

private void btnAccept_Click(object sender, RoutedEventArgs e)
{
    if (lstUsers.SelectedIndex >= 0)
    {
        User u = (User)lstUsers.Items[lstUsers.SelectedIndex];
        Guid id = u.WorkflowID.Value;
        UserDataDataContext dc = new UserDataDataContext(_connectionString);
        dc.Refresh(RefreshMode.OverwriteCurrentValues, dc.Users);
        u = dc.Users.SingleOrDefault<User>(x => x.WorkflowID == id);
        if (u != null)
        {
            u.AcceptedBy = txtAcceptedBy.Text;
            u.Status = "Assigned";
            dc.SubmitChanges();
            // Clear the input
            txtAcceptedBy.Text = "";
        }
        // Update the grid
        lstUsers.Items[lstUsers.SelectedIndex] = u;
        lstUsers.Items.Refresh();
        WorkflowApplication i = new WorkflowApplication(new UserWF());
        i.InstanceStore = _instanceStore;
        i.PersistableIdle = (waiea) => PersistableIdleAction.Unload;
        i.Load(id);
        try
        {
            i.ResumeBookmark("GetAcceptes", u);
        }
        catch (Exception e2)
        {
            AddEvent(e2.Message);
        }
    }
}

```

کاربر را انتخاب می‌کنم مقادیرش را تنظیم می‌کنیم. آن را ذخیره کرده و workflow را از روی guid مربوط به آن که قبلاً در فرآیند به Entity دادیم، Load می‌کنیم و همانطور که می‌بینید توسط متد ResumeBookmark فرآیند رو از جایی که می‌خواهیم ادامه می‌دهیم. البته می‌توان تایید کاربر را هم در خود فرآیند انجام داد و چون نوشتن Activity مرتبط با آن تقریباً تکراری است با اجازه‌ی شما من اون رو نوشتم و زحمتش با خودتونه.

حالا فقط مانده‌است که همه کاربران را در ابتدای نمایش فرم از دیتابیس فراخوانی کنیم و در لیست نمایش دهیم:

```

private void LoadExistingLeads()
{
    UserDataDataContext dc = new UserDataDataContext(_connectionString);
    dc.Refresh(RefreshMode.OverwriteCurrentValues, dc.Users);
    IEnumerable<User> q = dc.Users;
    foreach (User u in q)
    {

```



```

        AddNewUser(u);
    }
}

```

و فراخوانی این متد را به انتهای رویداد Load صفحه واگذار می‌کنیم.

پروژه رو اجرا کرده و یک کاربر را اضافه می‌کنم. همانطور که می‌دانید این کاربر در فرآیند ایجاد و در دیتابیس ذخیره می‌شود

The screenshot shows a Windows application window titled "MainWindow". It contains a registration form with three input fields: "Name:" with the value "Reza1", "Phone Number" with the value "2231", and "Email:" with the value "reza_bazargan@msn.com". Below these fields is a "Register" button. Underneath the form is a table titled "Current User" with three columns. The first row of data contains the values "Reza1", "2231", and "reza_bazargan@msn.comNew". At the bottom of the window is an "Event Log" section containing a single message: "User Reza1 Registered and waiting for Accept".

Current User		
Reza1	2231	reza_bazargan@msn.comNew

Event Log

User Reza1 Registered and waiting for Accept

برنامه را می‌بندم و دوباره اجرا می‌کنم. کاربر را انتخاب می‌کنم و یک نام برای admin انتخاب و آن را تایید می‌کنم. فرآیند را از bookmark مورد نظر اجرا کرده و به پایان می‌رسد. با بسته شدن برنامه، فرایند Idle و Unload می‌شود و ذخیره آن در sqlserver صورت می‌گیرد.

نظرات خوانندگان

نویسنده: وهاب زاده
تاریخ: ۱۳۹۲/۰۴/۲۱ ۰:۳۵

آقا رضا عالی بود، استفاده کردم.

موفق باشی

نویسنده: بهار
تاریخ: ۱۳۹۲/۰۶/۰۶ ۱۷:۴۹

سلام. ممنون از زحماتی که کشیدید. یک سوال کلید WorkflowID در جدول User کلید خارجی از کدوم یک از جداول WF است؟
توی هیچ کدوم از جدولها نیست!

نویسنده: رضا بازرگان
تاریخ: ۱۳۹۲/۰۷/۲۱ ۲۱:۵۰

ببخشید که دیر شد

شما در هر Activity که ایجاد می کنید با کمک خاصیت Context به خصوصیات workflow جاری دسترسی دارید.

اگه دیده باشید ما یک Activity به نام CreateUser ایجاد کردیم که Entity مربوط به User رو ایجاد می کنه.

درست قبل از اینکه user رو ذخیره کنیم مقدار فیلد WorkflowId اون رو که در واقع شناسه workflow جاری برای این user هست رو به اون اختصاص می دیم.

این فیلد در حقیقت شناسه workflow ایه که این entity در اون جریان داره
شناسه workflow در جدول instancesTable فیلد به نام WorkflowInstanceId