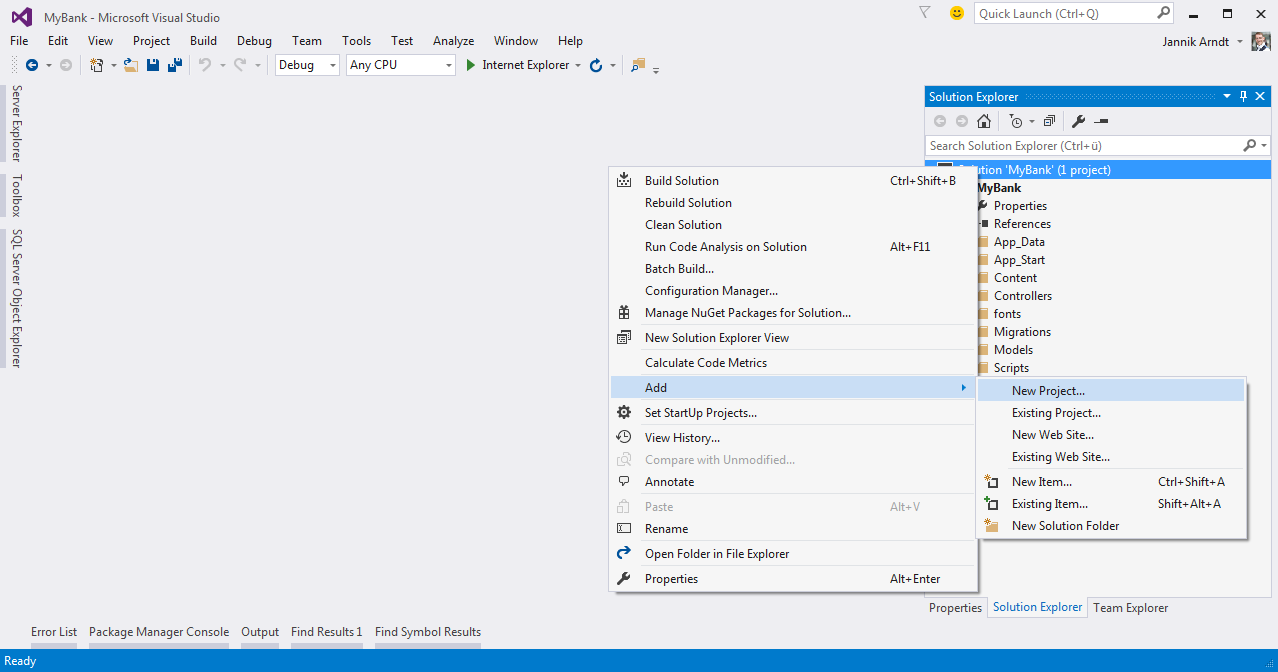
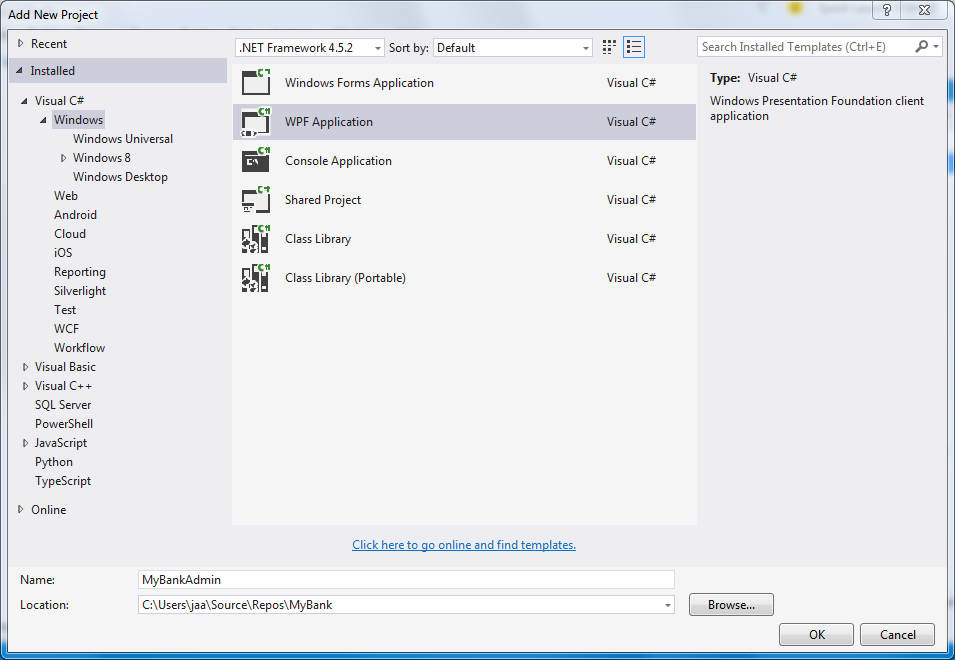
Beispielprojekt „MyBankAdmin“

WPF Windows-Programm mit Entity Framework 6

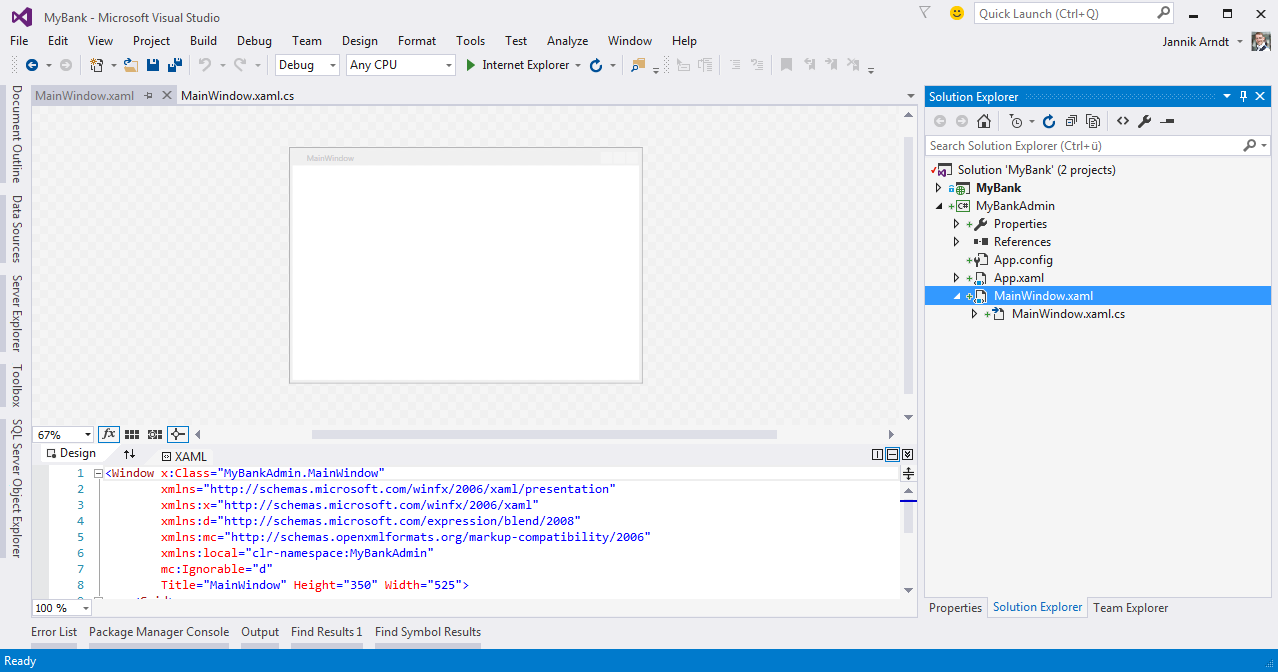
Dieses Projekt baut auf das ASP.NET MVC-Beispielprojekt „MyBank“ auf und benutzt dessen Datenbank. Zunächst fügen wir der MyBank-Solution ein neues Projekt hinzu:

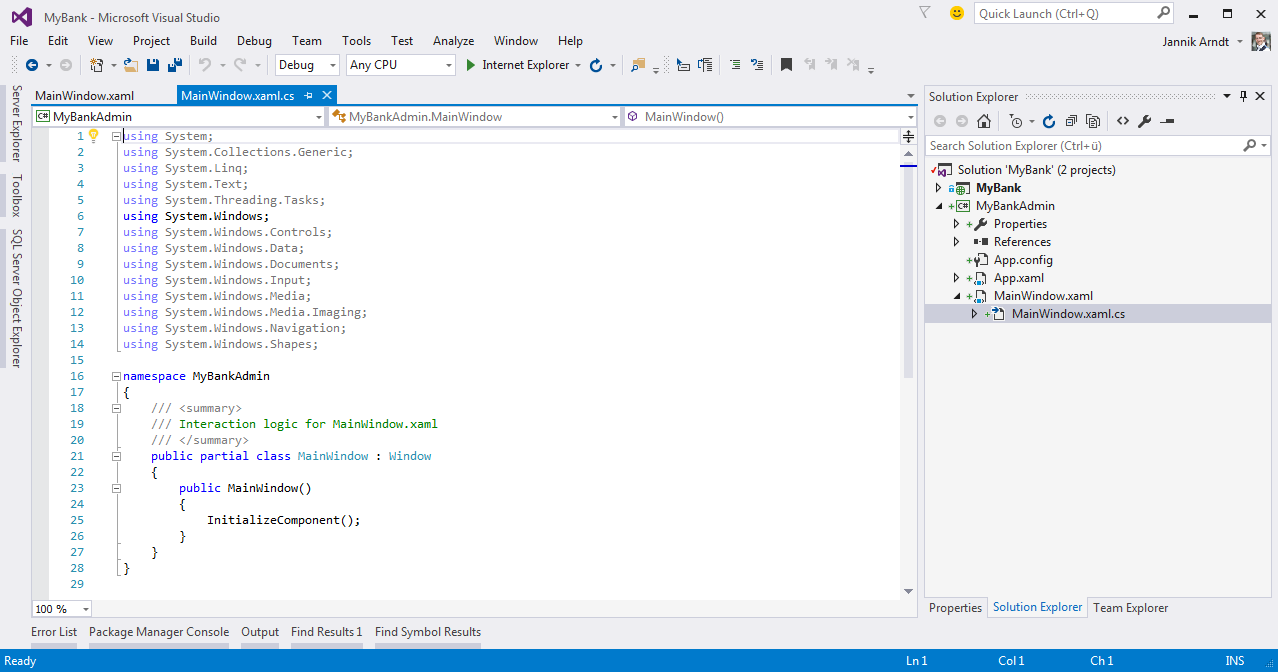


Ein WPF-Projekt auswählen:



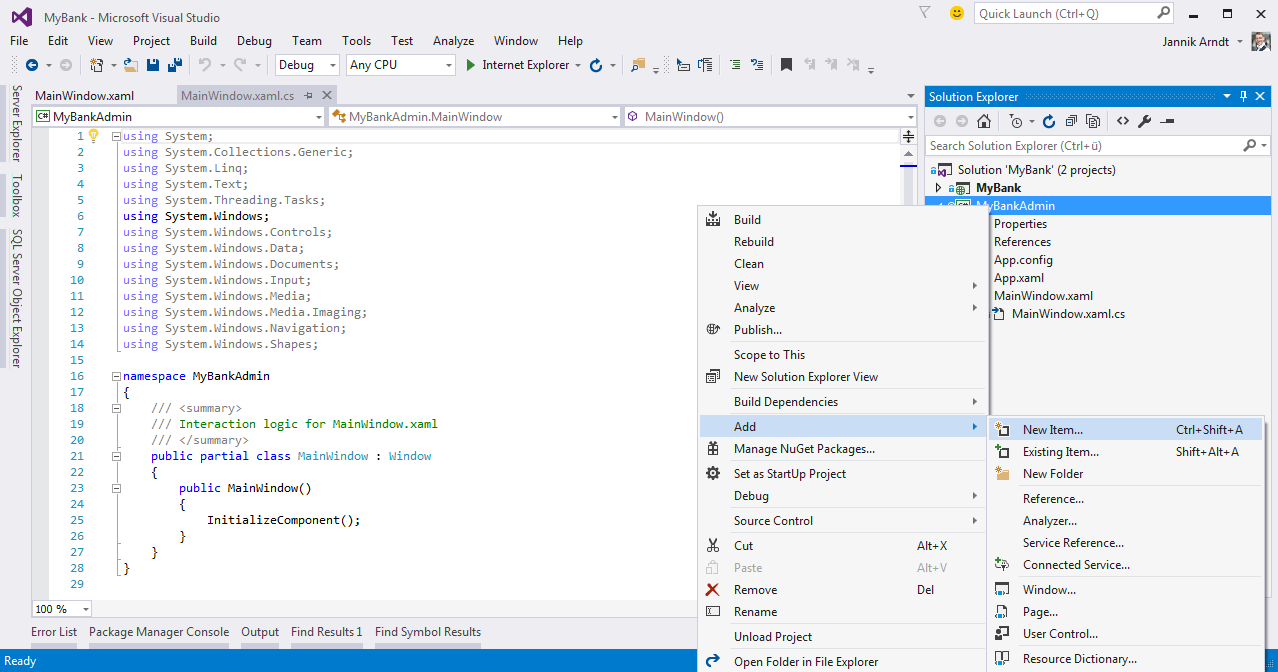
Ein Fenster besteht aus zwei Dateien: der XAML-Datei für die Oberfläche und der xaml.cs für das Backend:

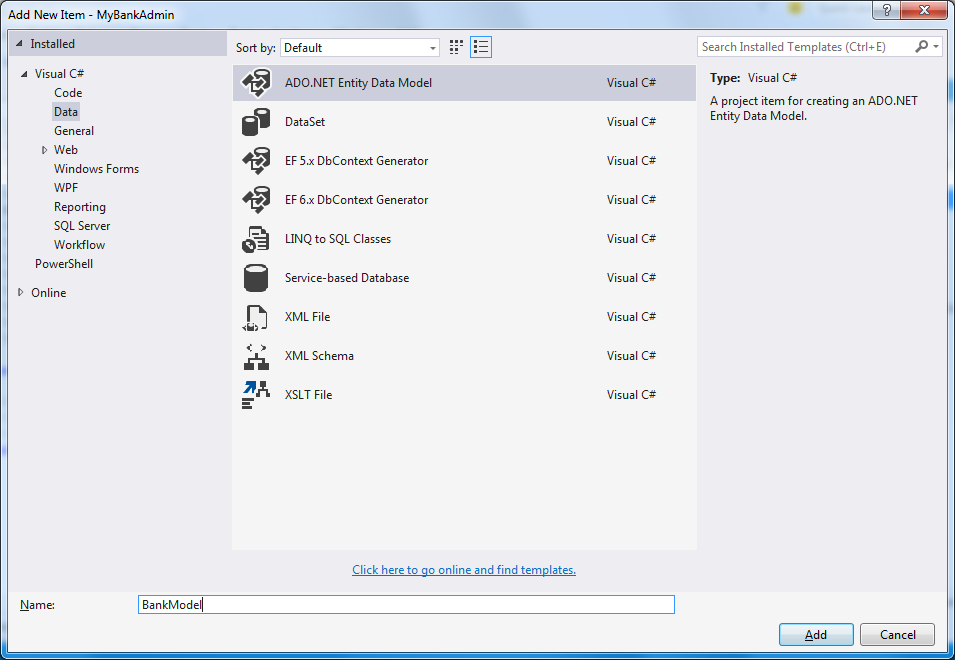


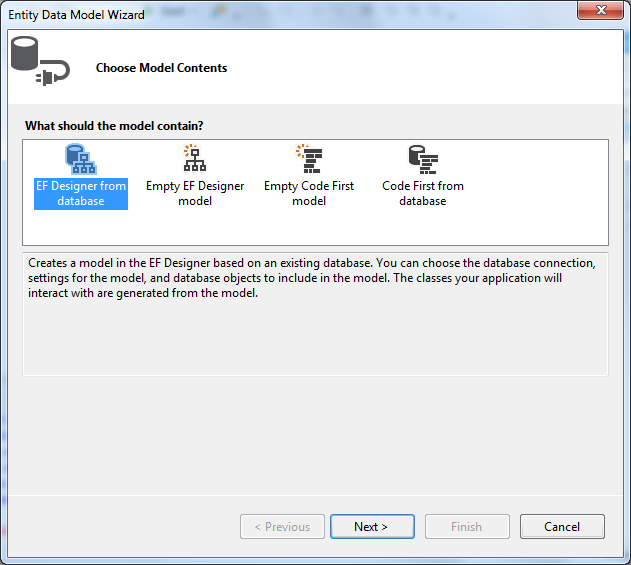


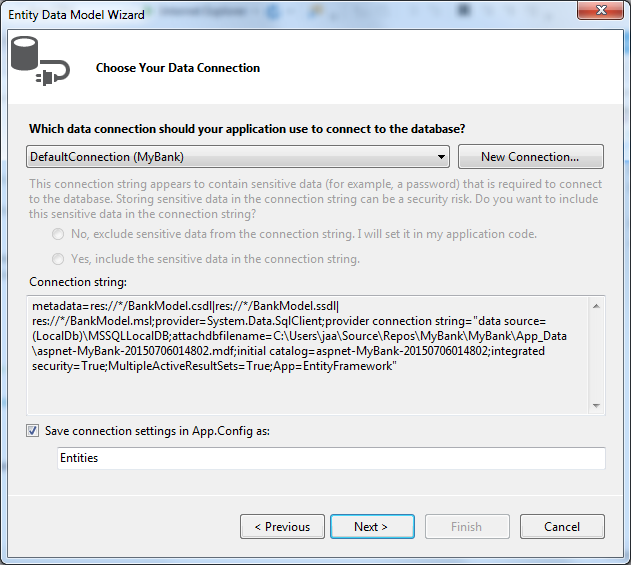
# Datenbankanbindung einfügen

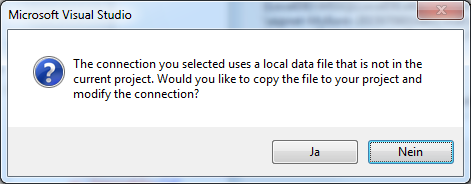
Jetzt fügen wir die Datenquelle aus dem MVC-Beispielprojekt hinzu:

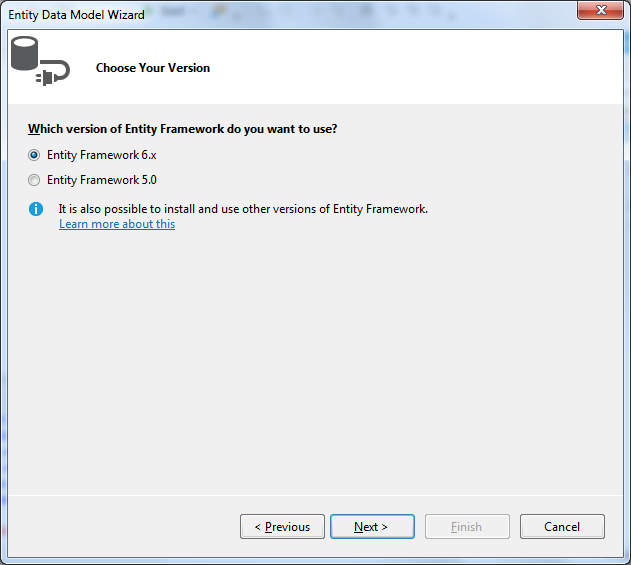


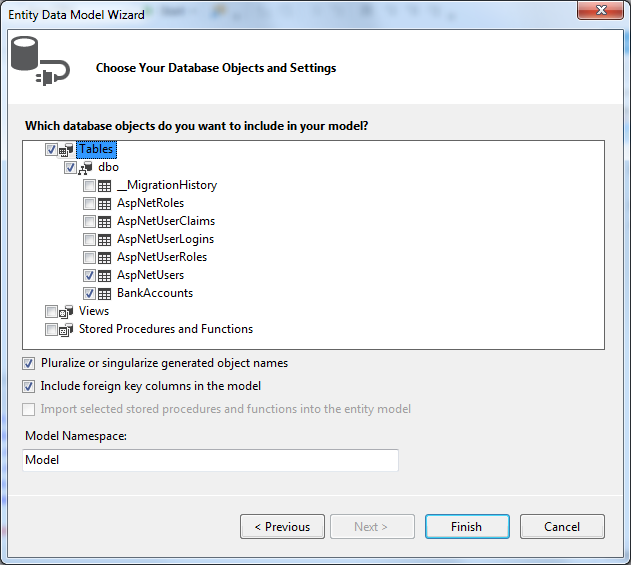




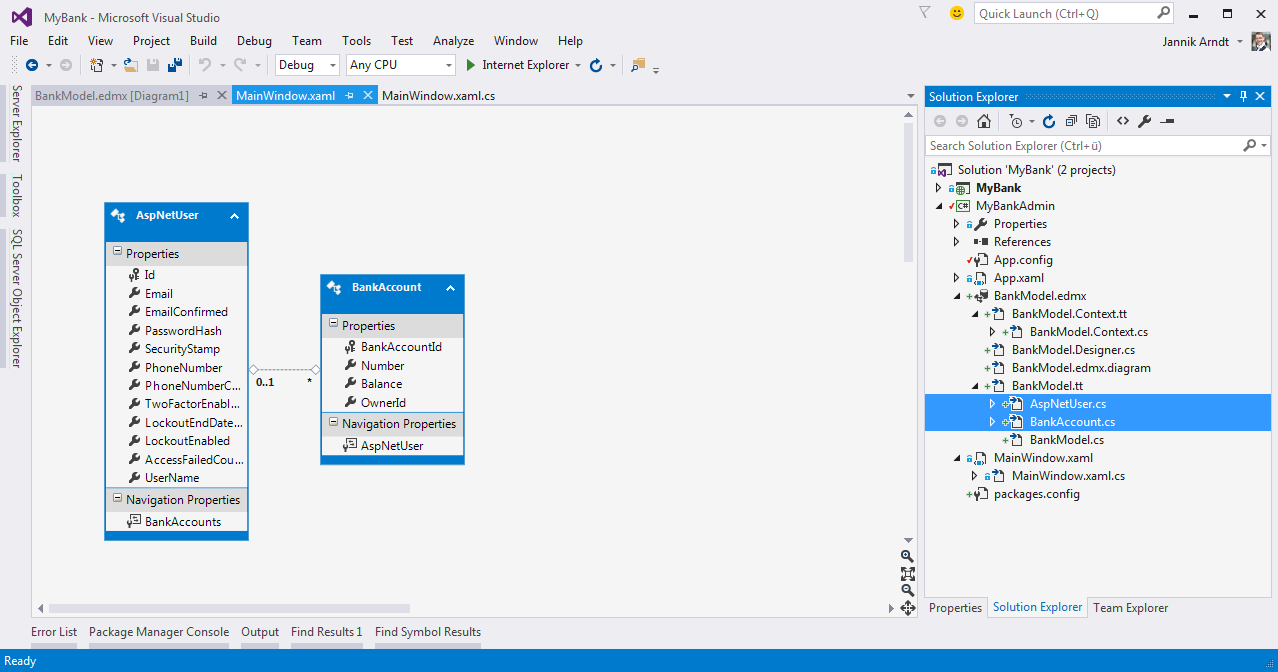




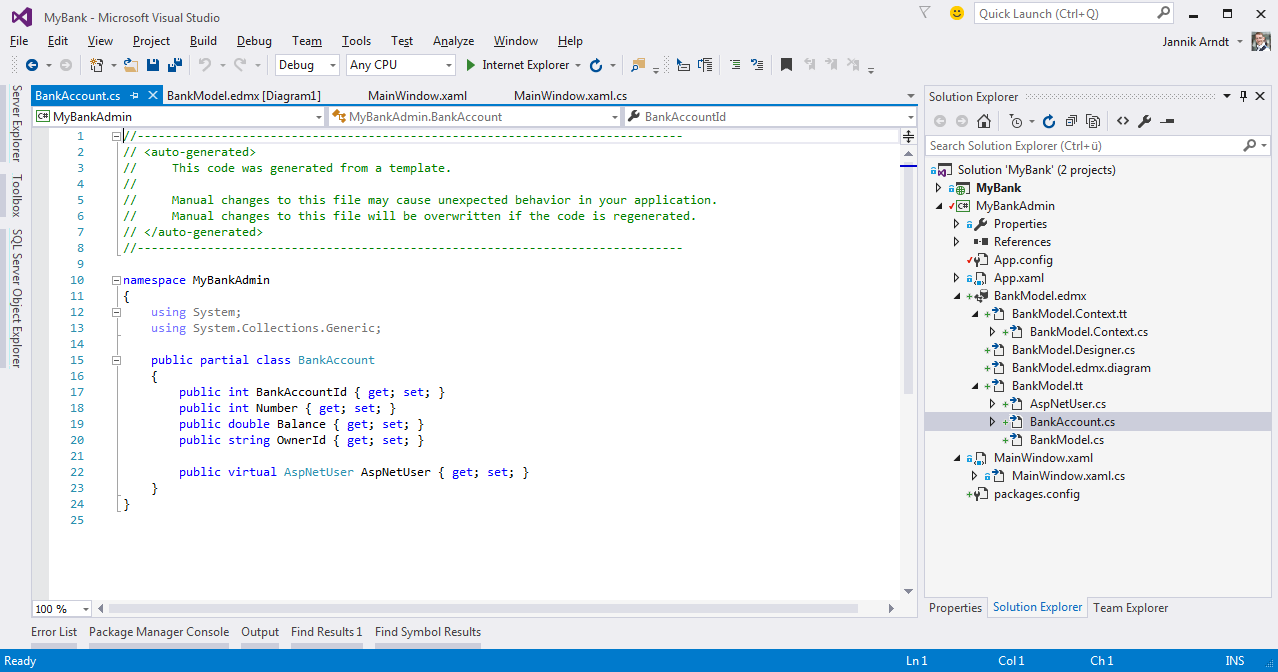




Die beiden generierten Klassen werden als edmx-Diagramm angezeigt und sind im Solution Explorer in dieser edmx-Datei zu finden:



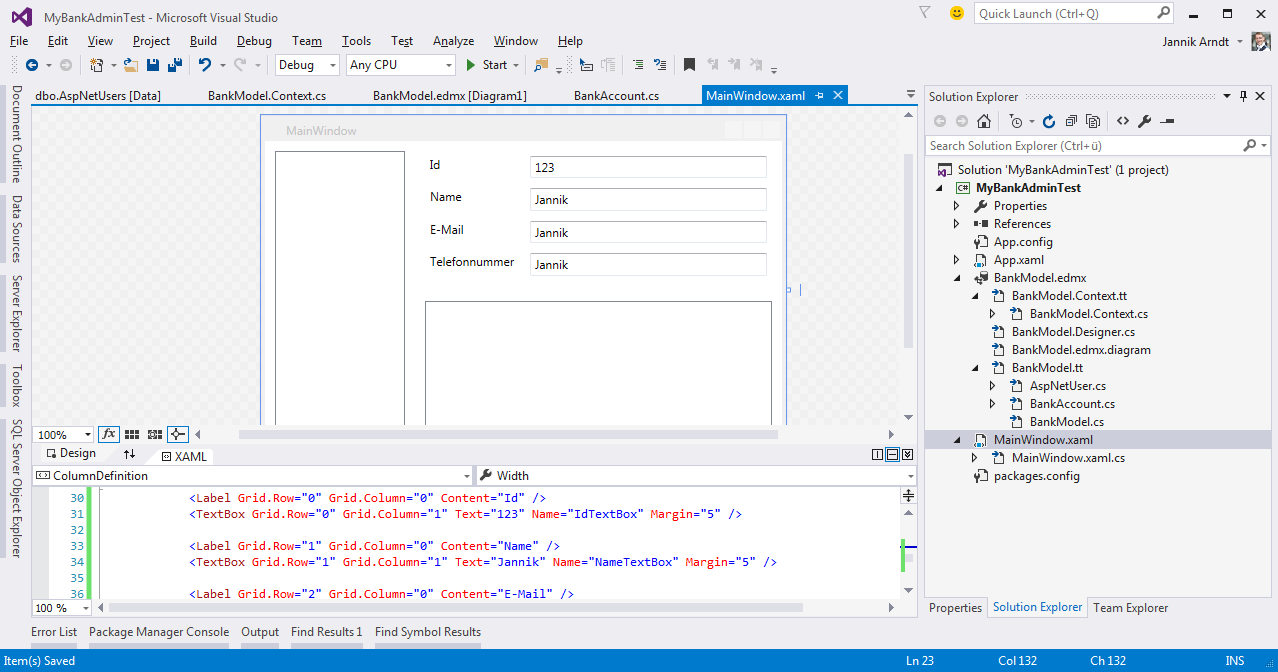
In der Datei weist ein Hinweis darauf hin, dass diese Klasse generiert ist. Sie kann zwar beliebig verändert werden, sollte sie jedoch neu generiert werden, z.B. weil sich die Datenbank geändert hat, werden diese Änderungen überschrieben.



# Einträge anzeigen

Nun können wir in der MainWindow.xaml eine Oberfläche bauen, um die Daten aus der Datenbank anzeigen zu können. Ziel ist eine Liste aller Benutzer, wenn man einen Benutzer auswählt sollen Details und eine Auflistung seiner Konten angezeigt werden.

XAML-Dateien werden im Split-View angezeigt, oben eine Vorschau, unten der Code. Über den Reiter „Toolbox“ kann man GUI-Elemente grafisch bearbeiten, um die volle Kontrolle zu behalten kann man die GUI aber auch im Code erstellen.



Das root-Element ist ein Window-Tag, darunter ist bereits ein Grid-Element vorgegeben. Wir unterteilen unser Grid in zwei Hälften, links die Kundenübersicht, rechts die Details.

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="150" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<!-- Links Kundenliste -->

<!-- Rechts Details -->

</Grid>

Die Kundenliste ist ein ListBox-Element, die Detail-Ansicht ein weiteres Grid mit Label- und TextBox-Elementen. Für die Auflistung der Konten benutzen wir ein DataGrid-Element:

<ListBox Grid.Row="0" Grid.Column="0" Name="UsersListBox" Margin="10" />

<Grid Grid.Row="0" Grid.Column="1" Margin="10" Name="UserItemsGrid">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="4\*" />

</Grid.RowDefinitions>

<Label Grid.Row="0" Grid.Column="0" Content="Id" />

<TextBox Grid.Row="0" Grid.Column="1" Name="IdTextBox" Margin="5" />

<Label Grid.Row="1" Grid.Column="0" Content="Name" />

<TextBox Grid.Row="1" Grid.Column="1" Name="NameTextBox" Margin="5" />

<Label Grid.Row="2" Grid.Column="0" Content="E-Mail" />

<TextBox Grid.Row="2" Grid.Column="1" Name="EmailTextBox" Margin="5" />

<Label Grid.Row="3" Grid.Column="0" Content="Telefonnummer" />

<TextBox Grid.Row="3" Grid.Column="1" Name="PhoneTextBox" Margin="5" />

<Label Grid.Row="4" Grid.Column="0" Content="Konten" />

<DataGrid Grid.Row="4" Grid.Column="1" Name="AccountsDataGrid" Margin="5" />

</Grid>

Zunächst füllen wir die Benutzerliste aus dem Backend („Code Behind“). Dazu fügen wir der MainWindow-Klasse eine Entities-Property namens „Bank“ hinzu, die unsere Datenbank repräsentiert. Im Konstruktor wird diese initialisiert und die Benutzer der Liste hinzugefügt:

public partial class MainWindow : Window

{

public Entities Bank { get; set; }

public MainWindow()

{

InitializeComponent();

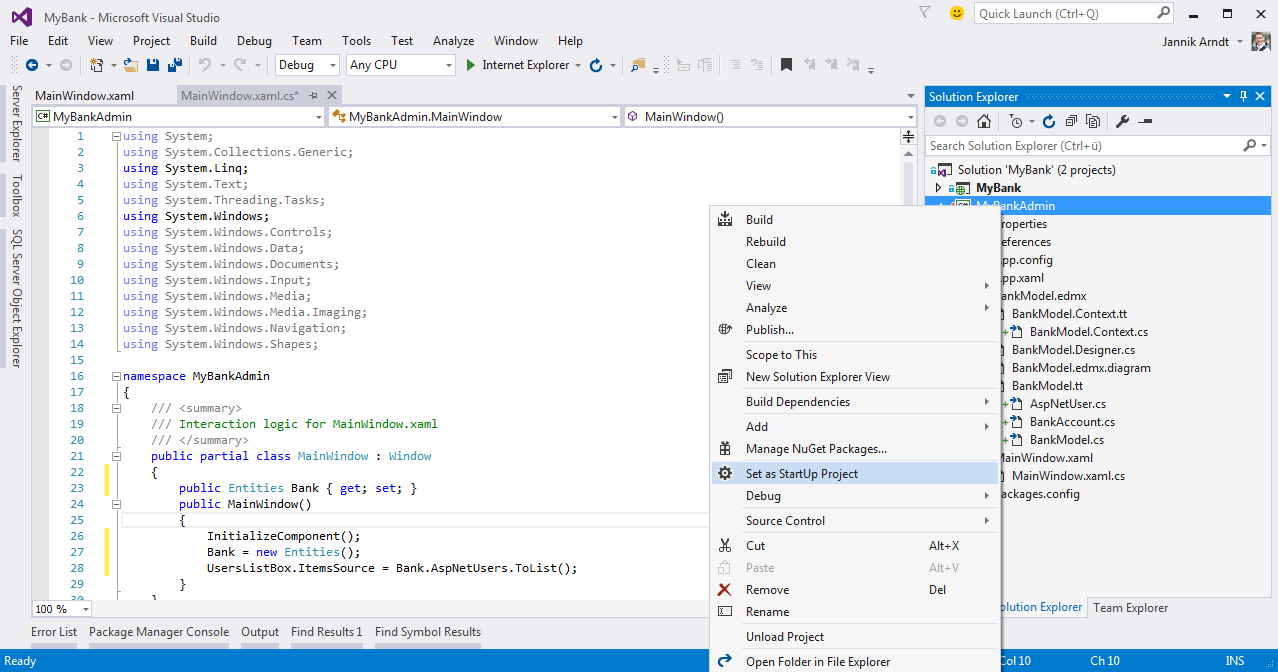
Bank = new Entities();

UsersListBox.ItemsSource = Bank.AspNetUsers.ToList();

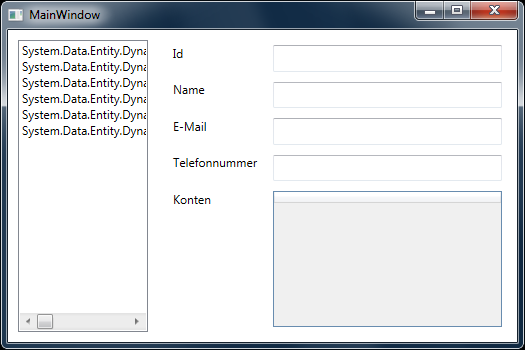
}

}

Zeit für einen ersten Test! Im Solution Explorer müssen das Projekt noch als StartUp Project festlegen:



Mit einem Klick auf „Start“ erscheint das Programm:



Die Benutzer werden in der Liste angezeigt, allerdings das, was die ToString()-Methode liefert. Dies lässt sich über das Attribut DisplayMemberPath kontrollieren:

<ListBox Grid.Row="0" Grid.Column="0" Name="UsersListBox" Margin="10" DisplayMemberPath="UserName" />

Als nächstes sollen die Details beim Klick auf einen Namen angezeigt werden. Wenn man im Designer auf die ListBox doppelklickt, dann wird dem XAML-Element das Attribut SelectionChanged="UsersListBox\_SelectionChanged" hinzugefügt. Die entsprechende Methode wird im Code Behind automatisch generiert. Als Parameter bekommt sie einen „sender“ und ein Event. Das sender-Objekt ist die ListBox, von der man nach dem casten auch das selectedItem erhält:

var user = (sender as ListBox).SelectedItem as AspNetUser;

Die Properties des user-Objektes können nun den TextBoxen und dem AccountsDataGrid zugewiesen werden:

private void UsersListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

var user = (sender as ListBox).SelectedItem as AspNetUser;

IdTextBox.Text = user.Id;

NameTextBox.Text = user.UserName;

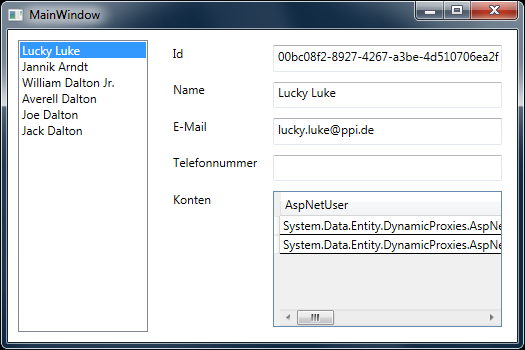
EmailTextBox.Text = user.Email;

PhoneTextBox.Text = user.PhoneNumber;

AccountsDataGrid.ItemsSource = user.BankAccounts;

}

Das führt zu:



Standardmäßig wird im DataGrid für alle Properties eine Spalte generiert. Dies kann man mit AutoGenerateColumns=“False“ deaktivieren, muss dann aber die Spalten, die man behalten möchte, selbst definieren:

<DataGrid Grid.Row="4" Grid.Column="1" Name="AccountsDataGrid" Margin="5" AutoGenerateColumns="False">

<DataGrid.Columns>

<DataGridTextColumn Binding="{Binding Path=Number}" Header="Kontonummer" Width="\*" />

<DataGridTextColumn Binding="{Binding Path=Balance, ConverterCulture='de-DE', StringFormat={}{0:C}}" Header="Saldo" Width="\*" />

</DataGrid.Columns>

</DataGrid>

# Bindings zwischen Model und View

Dem DataGrid wird im Code Behind eine Liste als ItemsSource übergeben, im XAML-Code werden Spalten an bestimmte Properties gebunden, den Rest erstellt das Framework von alleine. Diese Bindings funktionieren jedoch nicht nur für einzelne Elemente, sondern auch für ganze Blöcke. Zum Beispiel können wir dem Grid, in dem wir alle Details zum Benutzer anzeigen, das User-Objekt als Context übergeben. Die Zuweisung in der SelectionChanged-Methode besteht dann nur noch aus einer Zeile:

UserItemsGrid.DataContext = (sender as ListBox).SelectedItem as AspNetUser;

In der XAML-Datei bekommen die TextBox-Elemente ein Attribut Text="{Binding UserName}" und das DataGrid das Attribut ItemsSource="{Binding BankAccounts}".

# Einträge bearbeiten

Wenn man einen Eintrag bearbeitet, einen anderen Benutzer auswählt und dann zum bearbeiteten Benutzer zurückkehrt, dann sieht man, dass die Änderung bleibt. Das liegt daran, dass die Textfelder per Bindings an die Objekte im Speicher gebunden sind und dieses Binding in beide Richtungen definiert ist. Allerdings werden die Änderungen nicht in der Datenbank persistiert.

Hierfür benötigt man aber nicht viel Code, es reicht, in die SelectionChanged-Methode die Zeile

Bank.SaveChanges();

mit aufzunehmen.

Versucht man jedoch, einen Saldo via Doppelklick zu ändern, wird eine InvalidOperationException geworfen, mit der Info, dass „EditItem“ für diese Ansicht nicht zulässig sei. Hintergrund ist, dass das Entity Framework in der Klasse AspNetUsers die Property

public virtual ICollection<BankAccount> BankAccounts { get; set; }

generiert hat, die als HashSet initialisiert wird:

this.BankAccounts = new HashSet<BankAccount>();

Ändert man diese beiden Einträge zu List<BankAccount> um, so wird das Bearbeiten plötzlich möglich.

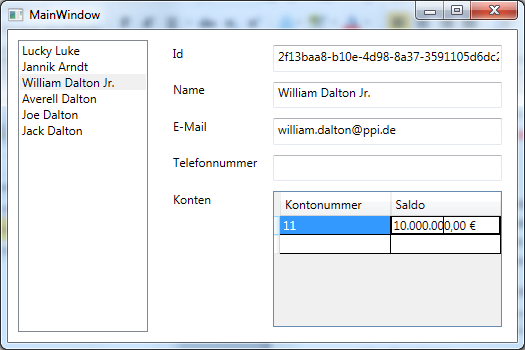
Die Änderungen werden allerdings erst in die Datenbank persistiert, wenn man einen anderen Benutzer auswählt. Mit einem Doppelklick auf das DataGrid erzeugt man die Methode SelectionChanged, in der man, wie in der SelectionChanged der ListBox, ein Speichern anstoßen kann:

private void AccountsDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

Bank.SaveChanges();

}



Damit ist auch die Erstellung der Admin-Software erfolgreich abgeschlossen.