JannikArndt / **MyBank**

Sample application for ASP.NET MVC 5 with Entity Framework 6

21 commits **1** branch **0** releases **1** contributor

Branch: master New pull request Find file Clone or download

JannikArndt Added Validation Latest commit `79f55a1` on Jul 17, 2015

MyBank	Added validation	3 years ago
MyBankAdmin	Changes are now persisted	3 years ago
Tutorial	Added validation	3 years ago
.gitattributes	Initial commit to add default .gitIgnore and .gitAttribute files.	3 years ago
.gitignore	Initial commit to add default .gitIgnore and .gitAttribute files.	3 years ago
MyBank.sln	Added validation	3 years ago
README.md	Didactic Reserve: Added asynchronous calls	3 years ago

README.md

There are two projects in this repository that provide an introduction to .NET technologies:

- "MyBank" is a web app with ASP.NET MVC 5 with Entity Framework 6
- "MyBankAdmin" is a WPF program that accesses the same database via Entity Framework 6. The instructions can be found here or in the folder "Turorial".

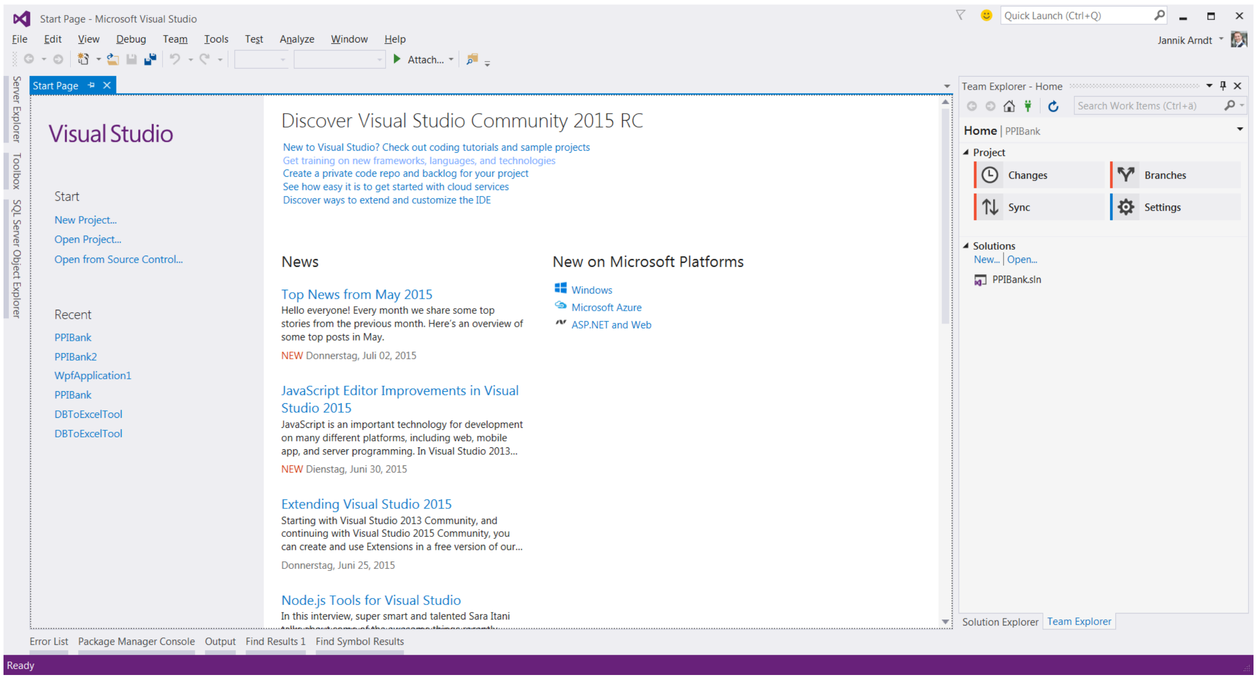
content

Example project "MyBank"

ASP.NET MVC 5 with Entity Framework 6

Create a project

Open Visual Studio 2015:



Discover Visual Studio Community 2015 RC

New to Visual Studio? Check out coding tutorials and sample projects
Get training on new frameworks, languages, and technologies
Create a private code repo and backlog for your project
See how easy it is to get started with cloud services
Discover ways to extend and customize the IDE

News

Top News from May 2015
Hello everyone! Every month we share some top stories from the previous month. Here's an overview of some top posts in May.
NEW Donnerstag, Juli 02, 2015

JavaScript Editor Improvements in Visual Studio 2015
JavaScript is an important technology for development on many different platforms, including web, mobile app, and server programming. In Visual Studio 2013...
NEW Dienstag, Juni 30, 2015

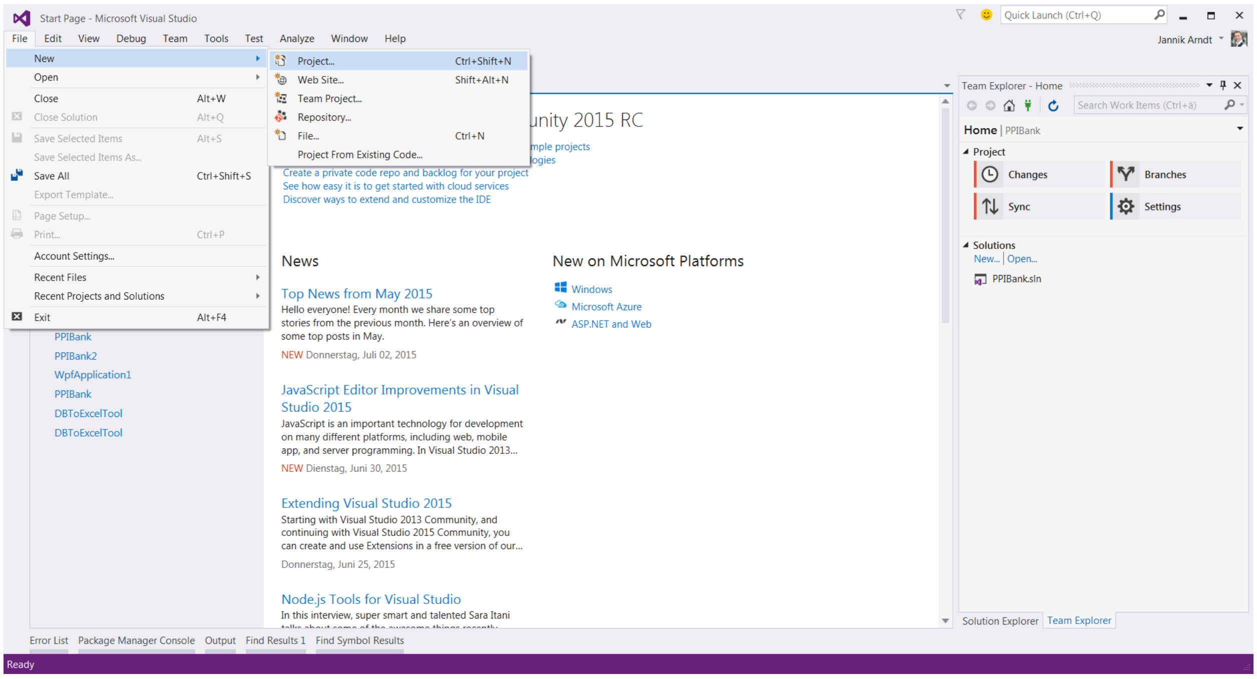
Extending Visual Studio 2015
Starting with Visual Studio 2013 Community, and continuing with Visual Studio 2015 Community, you can create and use Extensions in a free version of our...
Donnerstag, Juni 25, 2015

Node.js Tools for Visual Studio
In this interview, super smart and talented Sara Itani

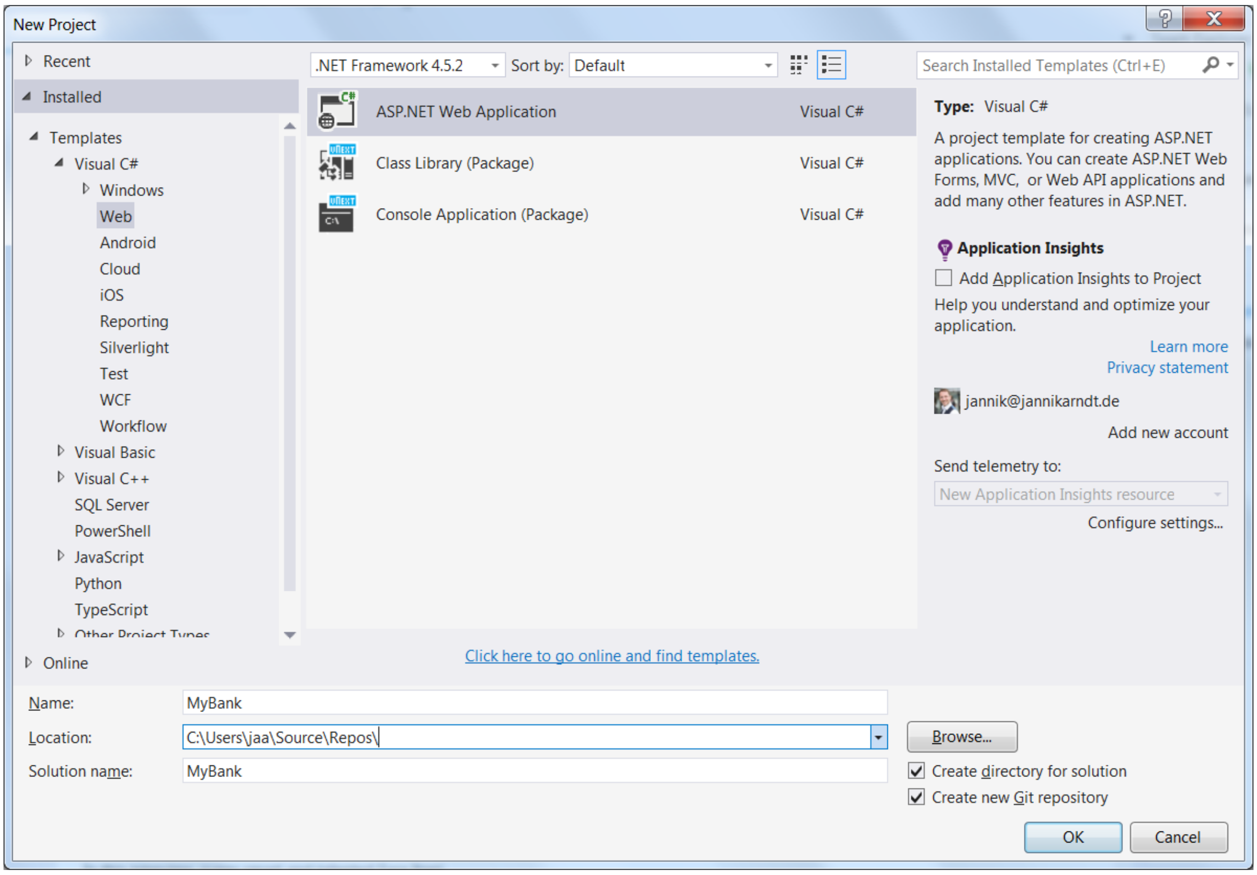
New on Microsoft Platforms

Windows
Microsoft Azure
ASP.NET and Web

Create a new project:



Select ASP.NET Web Application and enter name:


New Project

Recent
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
iOS
Reporting
Silverlight
Test
WCF
Workflow
Visual Basic
Visual C++
SQL Server
PowerShell
JavaScript
Python
TypeScript
Other Project Types
Online

.NET Framework 4.5.2 | Sort by: Default | Search Installed Templates (Ctrl+E)

ASP.NET Web Application	Visual C#
Class Library (Package)	Visual C#
Console Application (Package)	Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to Project

Help you understand and optimize your application.

Learn more
Privacy statement

jannik@jannikarndt.de

Add new account

Send telemetry to:

New Application Insights resource

Configure settings...

Click here to go online and find templates.

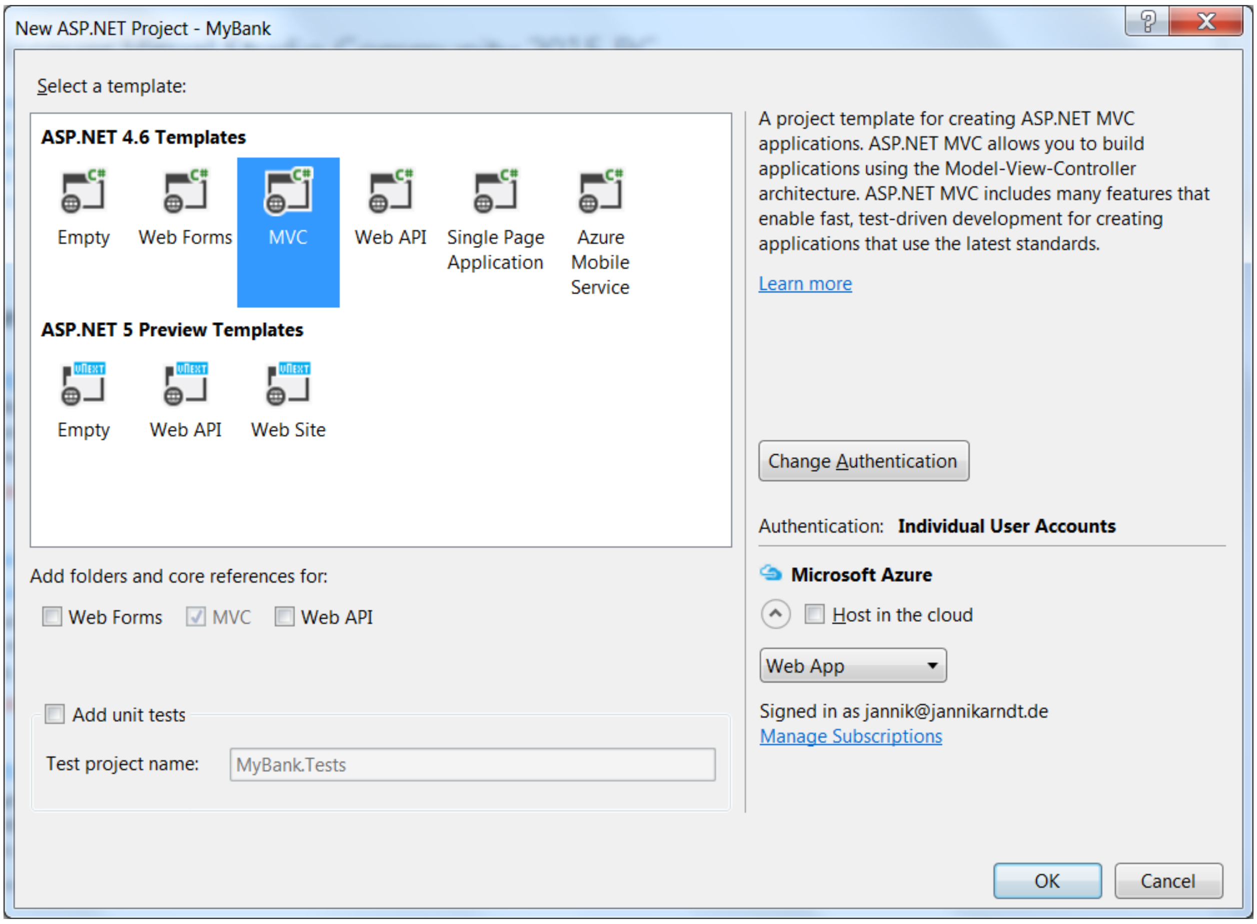
Name: MyBank
Location: C:\Users\jaa\Source\Repos\
Solution name: MyBank

Browse...

☑ Create directory for solution
☑ Create new Git repository

OK | Cancel

Select MVC Template (Authentication from Individual User Accounts):


New ASP.NET Project - MyBank

Select a template:

ASP.NET 4.6 Templates

Empty | Web Forms | MVC | Web API | Single Page Application | Azure Mobile Service

ASP.NET 5 Preview Templates

Empty | Web API | Web Site

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Change Authentication

Authentication: **Individual User Accounts**

Microsoft Azure

☐ Host in the cloud

Web App

Signed in as jannik@jannikarndt.de
Manage Subscriptions

Add folders and core references for:

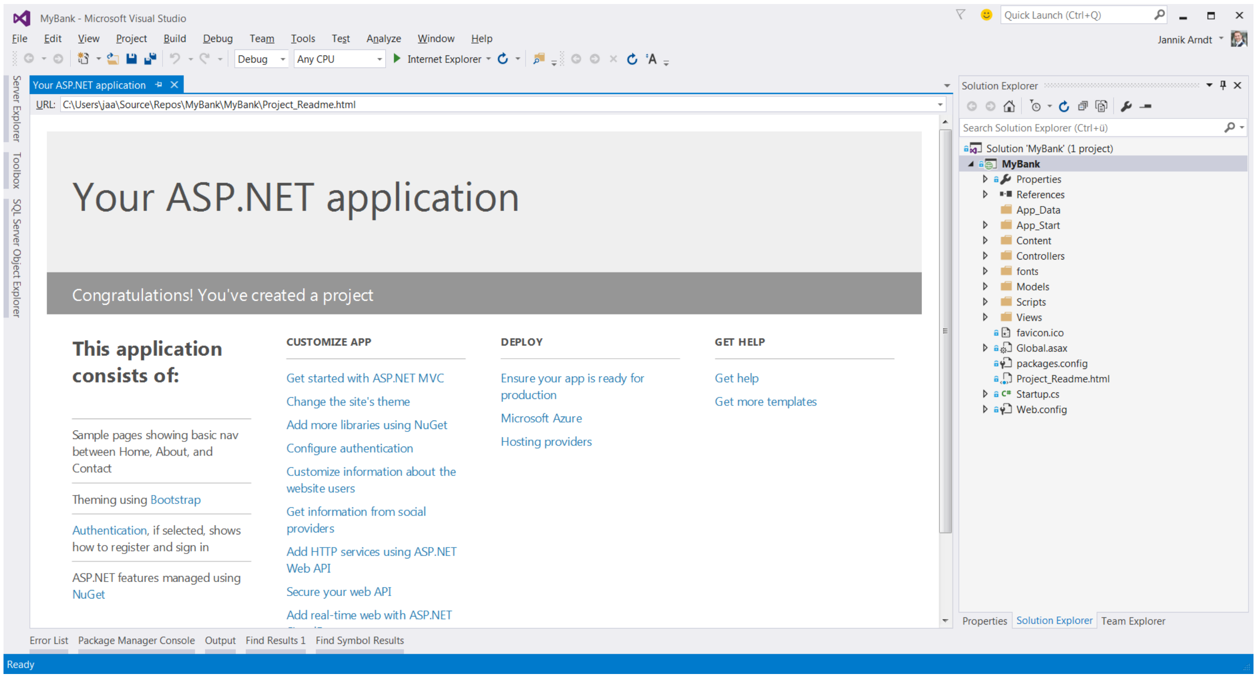
☐ Web Forms ☑ MVC ☐ Web API

☐ Add unit tests

Test project name: MyBank.Tests

OK | Cancel

An empty project has been created:



Your ASP.NET application

Congratulations! You've created a project

This application consists of:

Sample pages showing basic nav between Home, About, and Contact

Theming using Bootstrap

Authentication, if selected, shows how to register and sign in

ASP.NET features managed using NuGet

CUSTOMIZE APP

Get started with ASP.NET MVC
Change the site's theme
Add more libraries using NuGet
Configure authentication
Customize information about the website users
Get information from social providers
Add HTTP services using ASP.NET Web API
Secure your web API
Add real-time web with ASP.NET

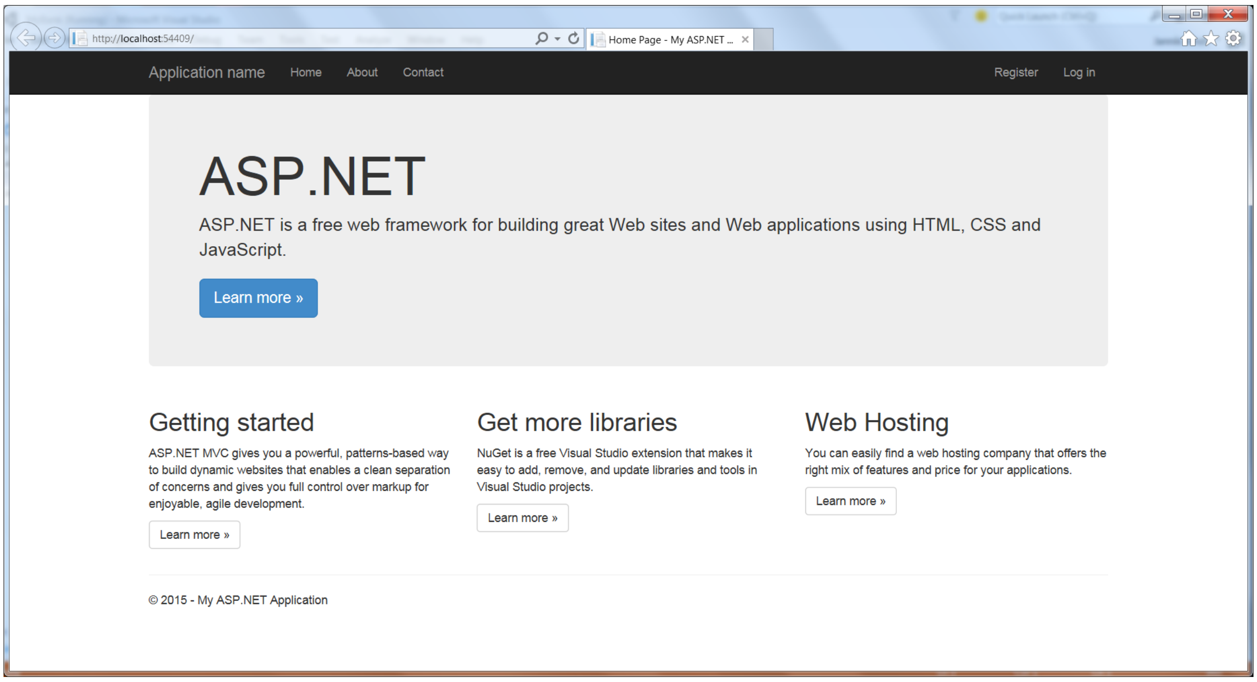
DEPLOY

Ensure your app is ready for production
Microsoft Azure
Hosting providers

GET HELP

Get help
Get more templates

This can already be done by clicking on "Internet Explorer":



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

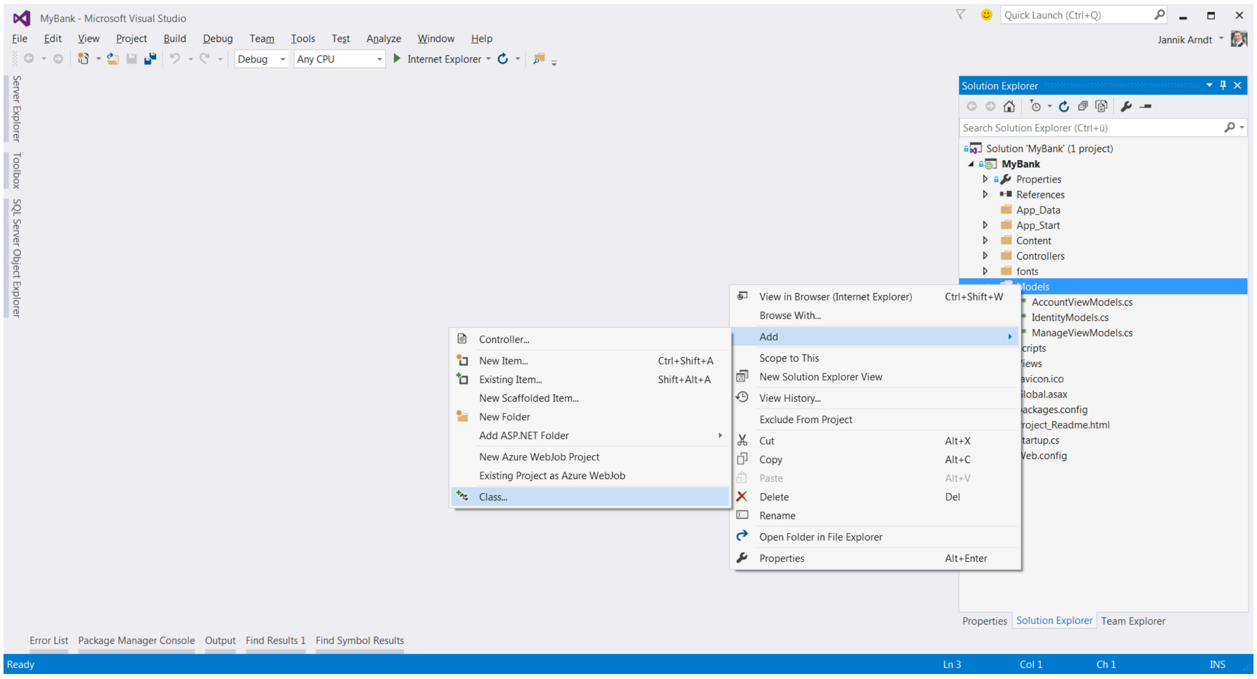
You can easily find a web hosting company that offers the right mix of features and price for your applications.

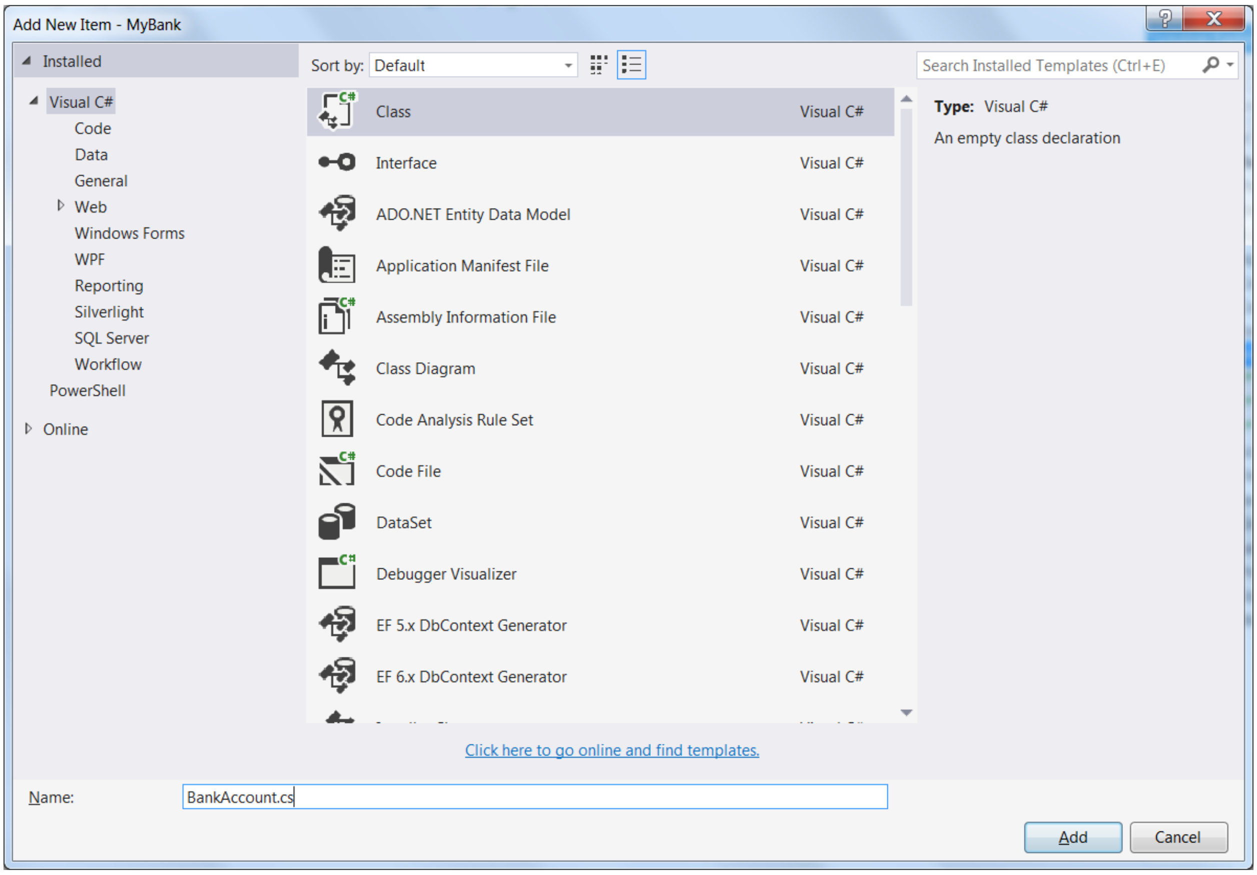
Learn more »

© 2015 - My ASP.NET Application

Create model classes

Stop debugging (Shift + F5 or red square) and insert a new file in the Models folder:



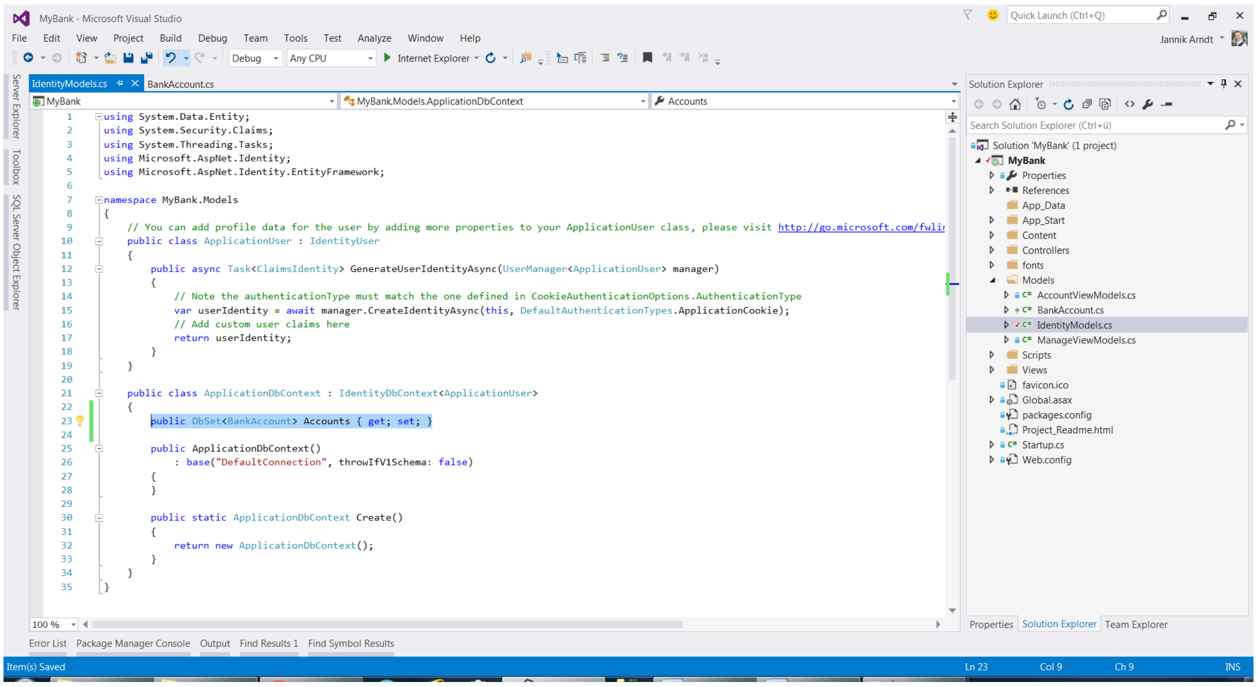

Add New Item - MyBank
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Workflow
PowerShell
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class Visual C#
Interface Visual C#
ADO.NET Entity Data Model Visual C#
Application Manifest File Visual C#
Assembly Information File Visual C#
Class Diagram Visual C#
Code Analysis Rule Set Visual C#
Code File Visual C#
DataSet Visual C#
Debugger Visualizer Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
Type: Visual C#
An empty class declaration
Click here to go online and find templates.
Add
Cancel

Name: BankAccount.cs

Content of the class:

```
namespace  MyBank.Models
{
    /// < summary >
    /// class for an account
    /// </ summary >
    public  class  BankAccount
    {
        /// < summary >
        /// The internal database id
        /// </ summary >
        public int  BankAccountId { get ; set ; }

        /// < summary >
        /// The public account number
        /// </ summary >
        public int  Number { get ; set ; }

        private double  _balance = 0 ;

        /// < summary >
        /// The account balance
        /// </ summary >
        public double  Balance { get { return _balance; } internal set {_balance = value ; }}

        /// < summary >
        /// The ID of the user to whom the account belongs
        /// </ summary >
        public string  OwnerId { get ; set ; }

        /// < summary >
        /// The object of the owner, is automatically assigned by the EF via the ID
        /// </ summary >
        public virtual ApplicationUser  Owner { get ; set ; }

        /// < summary >
        /// An empty constructor for database creation / deserialization
        /// </ summary >
        public  BankAccount ()
        {
        }

        /// < summary >
        /// default constructor
        /// </ summary >
        /// < param  name = " ownerId " > </ param >
        /// < param  name = " number " > </ param >
        public  BankAccount ( string ownerId, int number)
        {
            OwnerId = ownerId;
            Balance = 0 ;
            Number = number;
        }

        /// < summary >
        /// Method to withdraw money
        /// </ summary >
        /// < param  name = " amount " > The sum of the withdrawal </ param >
        /// < returns > True if successful, False if coverage is insufficient </ returns >
        public  bool  Withdraw ( double  amount )
        {
            if (amount> balance)
                 return  false ;

            Balance - = amount;
            return  true ;
        }

        /// < summary >
        /// Deposit method
        /// </ summary >
        /// < param  name = " amount " > </ param >
        public  void  PayIn ( double  amount )
        {
            Balance + = amount;
        }

        /// < summary >
        /// Method of transferring money to another account
        /// </ summary >
        /// < param  name = " amount " > The transfer buzzer </ param >
        /// < param  name = " otherAccount " > The (public) account number of the other account </ param >
        /// < returns > True on success, false if the coverage is insufficient </ returns >
        public  bool Transfer ( double  amount , BankAccount  otherAccount )
        {
            if (amount> balance)
                 return  false ;

            Balance - = amount;
            otherAccount.PayIn (amount);
            return  true ;
        }
    }
}
```

Insert the table in the database context (here in IdentityModels.cs):


```
using System.Data.Entity;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace MyBank.Models
{
    // You can add profile data for the user by adding more properties to your ApplicationUser class, please visit http://go.microsoft.com/fwlir
    public class ApplicationUser : IdentityUser
    {
        public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
        {
            // Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);
            // Add custom user claims here
            return userIdentity;
        }
    }

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public DbSet<BankAccount> Accounts { get; set; }

        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }
}
```

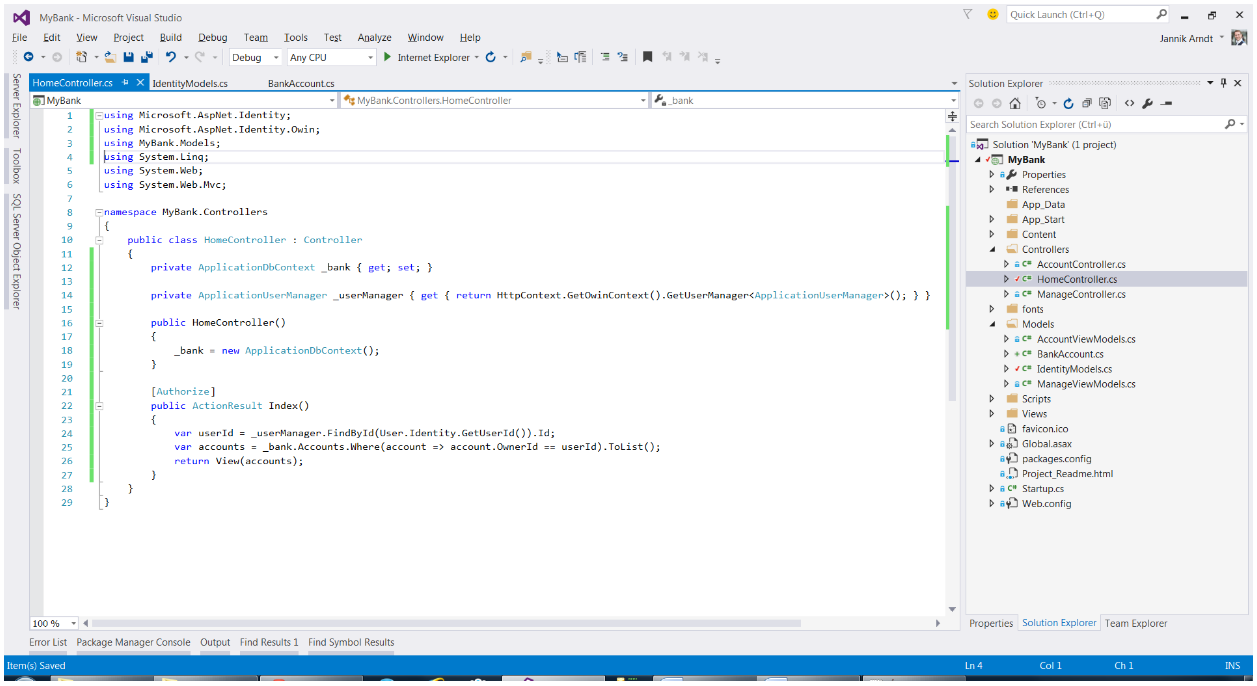
Adaptation of the HomeController (backend for the view):

1. 1.Class variables and constructor:

```
private  ApplicationDbContext \ _bank { get ; set ; }

private  ApplicationUserManager \ _userManager { get { return HttpContext.GetOwinContext (). GetUserManager <Applicat

public home controller ()
{
    \ _bank = new  ApplicationDbContext ();
}
```

1. 2.Index method:

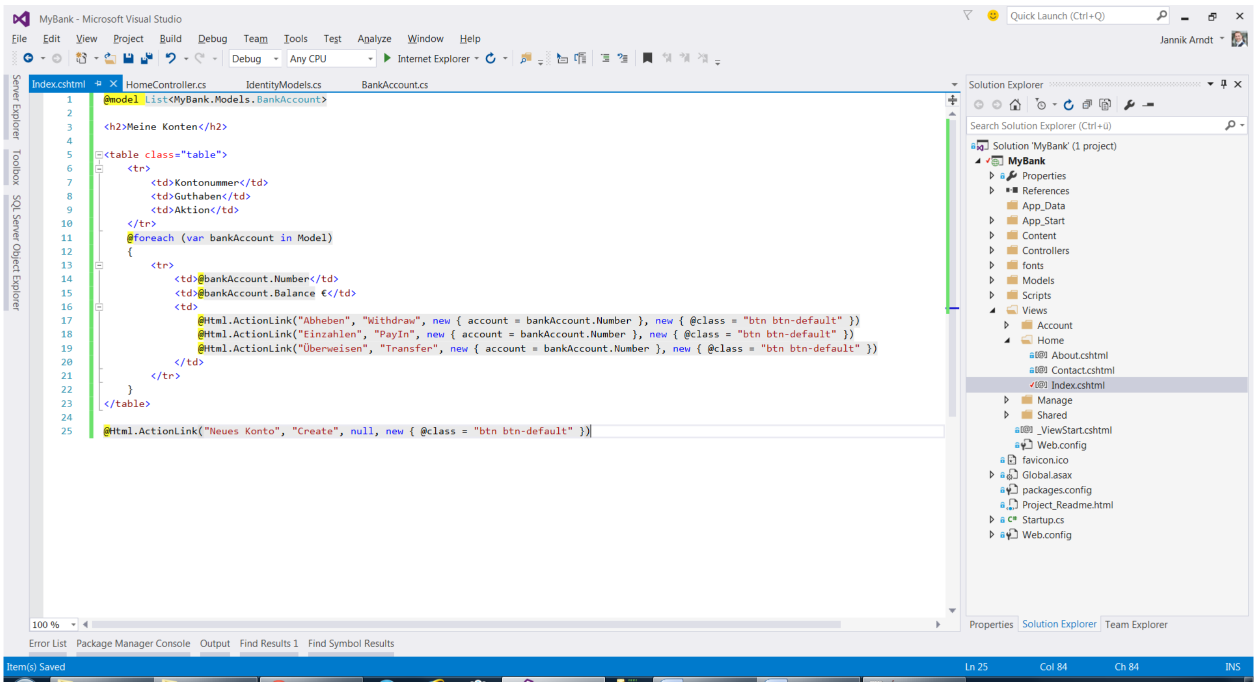
```
[Authorize]
 public  ActionResult  Index ()
{
    var  userId = \ _userManager.FindById (User.Identity.GetUserId ()). Id;
    var  accounts = \ _bank.Accounts.Where (account => account.OwnerId == userId) .ToList ();
    return view (accounts);
}
```



```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using MyBank.Models;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MyBank.Controllers
{
    public class HomeController : Controller
    {
        private ApplicationDbContext _bank { get; set; }

        private ApplicationUserManager _userManager { get { return HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>(); } }

        public HomeController()
        {
            _bank = new ApplicationDbContext();
        }

        [Authorize]
        public ActionResult Index()
        {
            var userId = _userManager.FindById(User.Identity.GetUserId()).Id;
            var accounts = _bank.Accounts.Where(account => account.OwnerId == userId).ToList();
            return View(accounts);
        }
    }
}
```

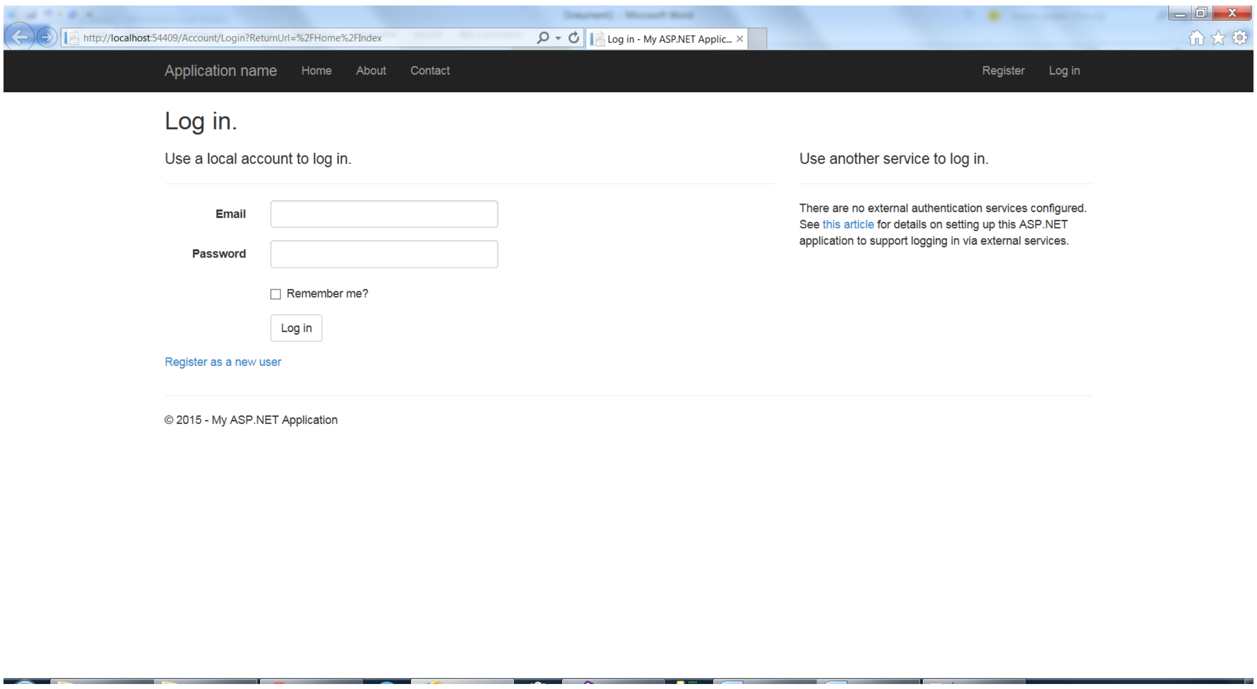
Customizing the view (View> Home> Index.cshtml):

```
@model List < MyBank . Models . Bank Account >

< h2 > My accounts </ h2 >

< table  class = " table " >
    < tr >
        < td > Account number </ td >
        < td > Credit </ td >
        < td > Action </ td >
    </ tr >
    @foreach (var bankAccount in Model)
    {
        < tr >
            < td > @ bankAccount.Number </ td >
            < td > @ bankAccount.Balance € </ td >
            < td >
                @ Html.ActionLink ("Take off", "Withdraw", new {account = bankAccount.Number}, new {@class = "btn btr
                @ Html.ActionLink ("Deposit", "PayIn", new {account = bankAccount.Number}, new {@class = "btn btn-def
                @ Html.ActionLink ("Transfer", "Transfer", new {account = bankAccount.Number}, new {@class = "btn btr
            </ td >
        </ tr >
    }
</ table >

@ Html.ActionLink ("New Account", "Create", null, new {@class = "btn btn-default"})
```



```
@model List<MyBank.Models.BankAccount>

<h2>Meine Konten</h2>

<table class="table">
    <tr>
        <td>Kontonummer</td>
        <td>Guthaben</td>
        <td>Aktion</td>
    </tr>
    @foreach (var bankAccount in Model)
    {
        <tr>
            <td>@bankAccount.Number</td>
            <td>@bankAccount.Balance €</td>
            <td>
                @Html.ActionLink("Abheben", "Withdraw", new { account = bankAccount.Number }, new { @class = "btn btn-default" })
                @Html.ActionLink("Einzahlen", "PayIn", new { account = bankAccount.Number }, new { @class = "btn btn-default" })
                @Html.ActionLink("Überweisen", "Transfer", new { account = bankAccount.Number }, new { @class = "btn btn-default" })
            </td>
        </tr>
    }
</table>

@Html.ActionLink("Neues Konto", "Create", null, new { @class = "btn btn-default" })
```

Click on "Internet Explorer" and a log in dialog will appear, because the index method has the [Authorize] attribute:



http://localhost:54409/Account/Login?ReturnUrl=%2FHome%2FIndex

Log in.

Use a local account to log in.

Email

Password

Remember me?

Log in

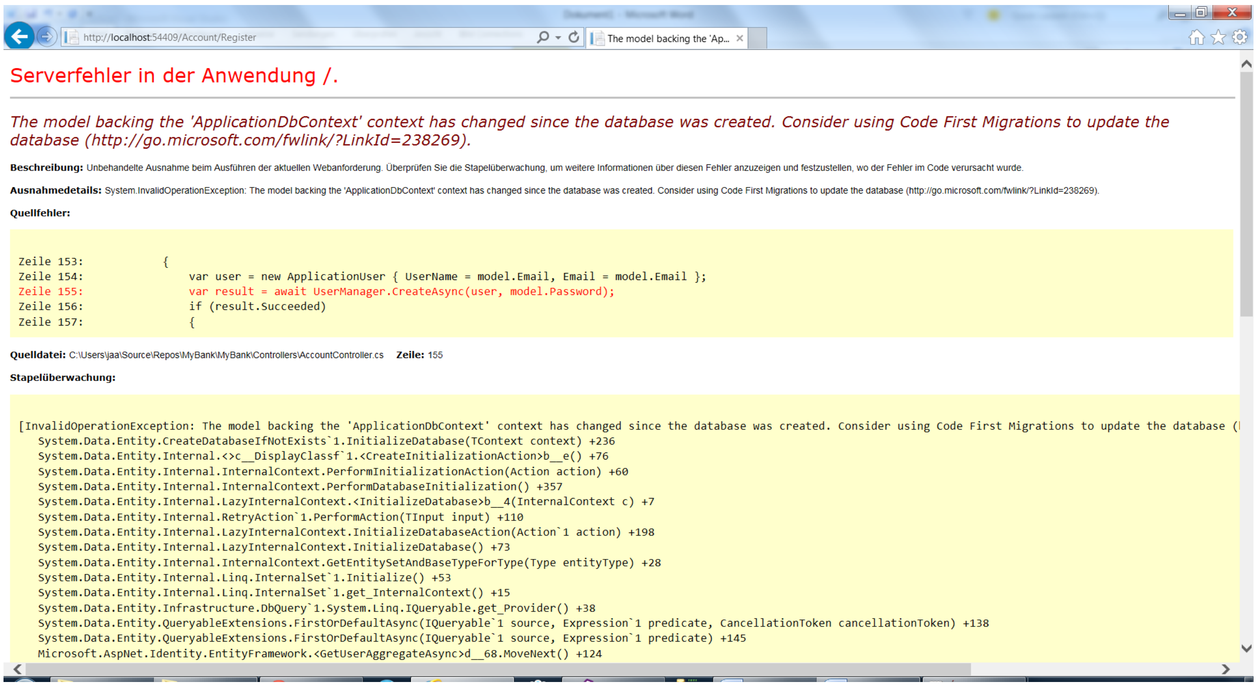
Register as a new user

Use another service to log in.

There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.

© 2015 - My ASP.NET Application

When registering a user, an error message appears:



Serverfehler in der Anwendung /.

The model backing the 'ApplicationDbContext' context has changed since the database was created. Consider using Code First Migrations to update the database (http://go.microsoft.com/fwlink/?LinkId=238269).

Beschreibung: Unbehandelte Ausnahme beim Ausführen der aktuellen Webanforderung. Überprüfen Sie die Stapelüberwachung, um weitere Informationen über diesen Fehler anzuzeigen und festzustellen, wo der Fehler im Code verursacht wurde.

Ausnahmedetails: System.InvalidOperationException: The model backing the 'ApplicationDbContext' context has changed since the database was created. Consider using Code First Migrations to update the database (http://go.microsoft.com/fwlink/?LinkId=238269).

Quellfehler:

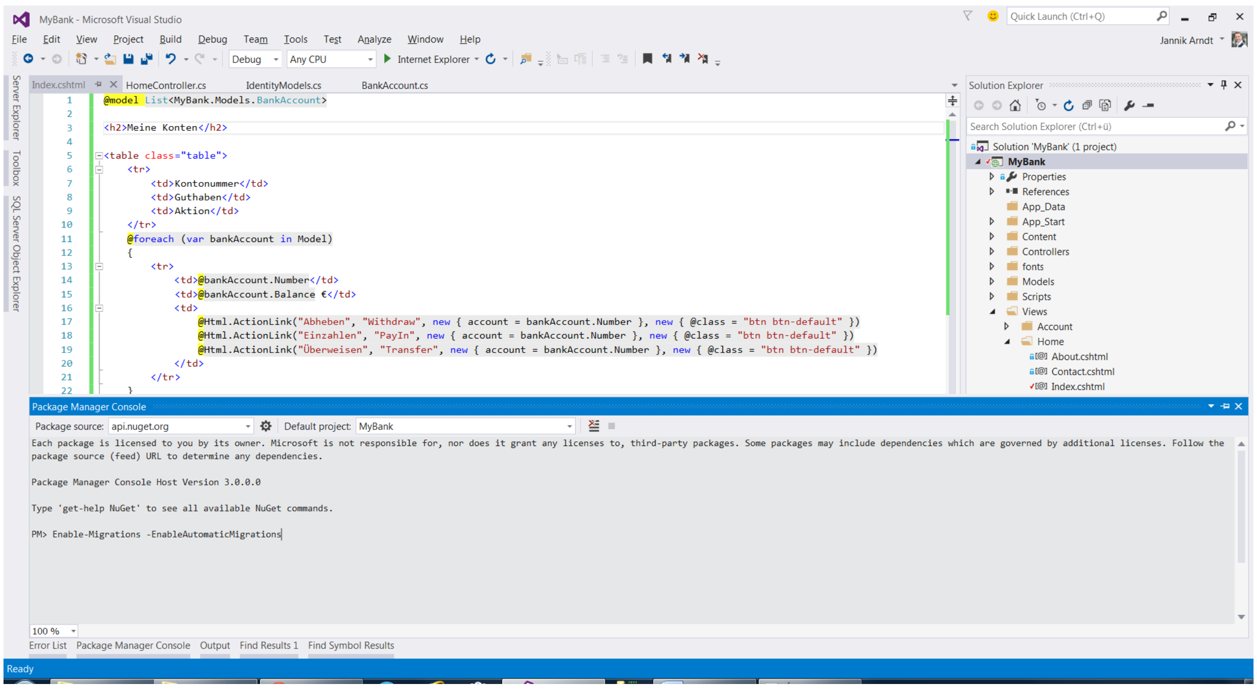
```
Zeile 153:            {
Zeile 154:                var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
Zeile 155:                var result = await UserManager.CreateAsync(user, model.Password);
Zeile 156:                if (result.Succeeded)
Zeile 157:                {
```

Quelldatei: C:\Users\jaa\Source\Repos\MyBank\MyBank\Controllers\AccountController.cs **Zeile:** 155

Stapelüberwachung:

```
[InvalidOperationException: The model backing the 'ApplicationDbContext' context has changed since the database was created. Consider using Code First Migrations to update the database (
   System.Data.Entity.CreateDatabaseIfNotExists`1.InitializeDatabase(TContext context) +236
   System.Data.Entity.Internal.<>c__DisplayClassf`1.<CreateInitializationAction>b__e() +76
   System.Data.Entity.Internal.InternalContext.PerformInitializationAction(Action action) +60
   System.Data.Entity.Internal.InternalContext.PerformDatabaseInitialization() +357
   System.Data.Entity.Internal.LazyInternalContext.<InitializeDatabase>b__4(InternalContext c) +7
   System.Data.Entity.Internal.RetryAction`1.PerformAction(TInput input) +110
   System.Data.Entity.Internal.LazyInternalContext.InitializeDatabaseAction(Action`1 action) +198
   System.Data.Entity.Internal.LazyInternalContext.InitializeDatabase() +73
   System.Data.Entity.Internal.InternalContext.GetEntitySetAndBaseTypeForType(Type entityType) +28
   System.Data.Entity.Internal.Linq.InternalSet`1.Initialize() +53
   System.Data.Entity.Internal.Linq.InternalSet`1.get_InternalContext() +15
   System.Data.Entity.Infrastructure.DbQuery`1.System.Linq.IQueryable.get_Provider() +38
   System.Data.Entity.QueryableExtensions.FirstOrDefaultAsync(IQueryable`1 source, Expression`1 predicate, CancellationToken cancellationToken) +138
   System.Data.Entity.QueryableExtensions.FirstOrDefaultAsync(IQueryable`1 source, Expression`1 predicate) +145
   Microsoft.AspNet.Identity.EntityFramework.<GetUserAggregateAsync>d__68.MoveNext() +124
```

The model has changed, but the database has not been updated yet. So stop debugging and open Package Manager Console:



```
@model List<MyBank.Models.BankAccount>

<h2>Meine Konten</h2>

<table class="table">
    <tr>
        <td>Kontonummer</td>
        <td>Guthaben</td>
        <td>Aktion</td>
    </tr>
    @foreach (var bankAccount in Model)
    {
        <tr>
            <td>@bankAccount.Number</td>
            <td>@bankAccount.Balance €</td>
            <td>
                @Html.ActionLink("Abheben", "Withdraw", new { account = bankAccount.Number }, new { @class = "btn btn-default" })
                @Html.ActionLink("Einzahlen", "PayIn", new { account = bankAccount.Number }, new { @class = "btn btn-default" })
                @Html.ActionLink("Überweisen", "Transfer", new { account = bankAccount.Number }, new { @class = "btn btn-default" })
            </td>
        </tr>
    }
```

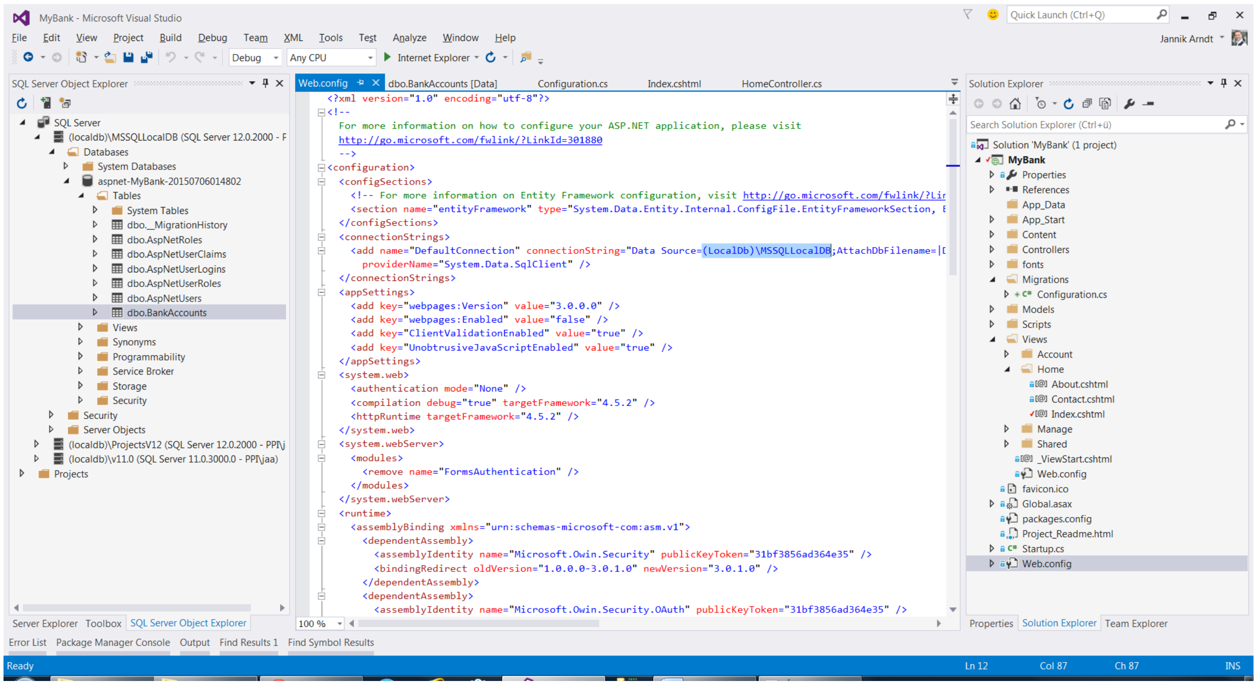
Package source: api.nuget.org   Default project: MyBank

```
Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which are governed by additional licenses. Follow the
package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 3.0.0.0

Type 'get-help NuGet' to see all available NuGet commands.

PM> Enable-Migrations -EnableAutomaticMigrations
```

With the command

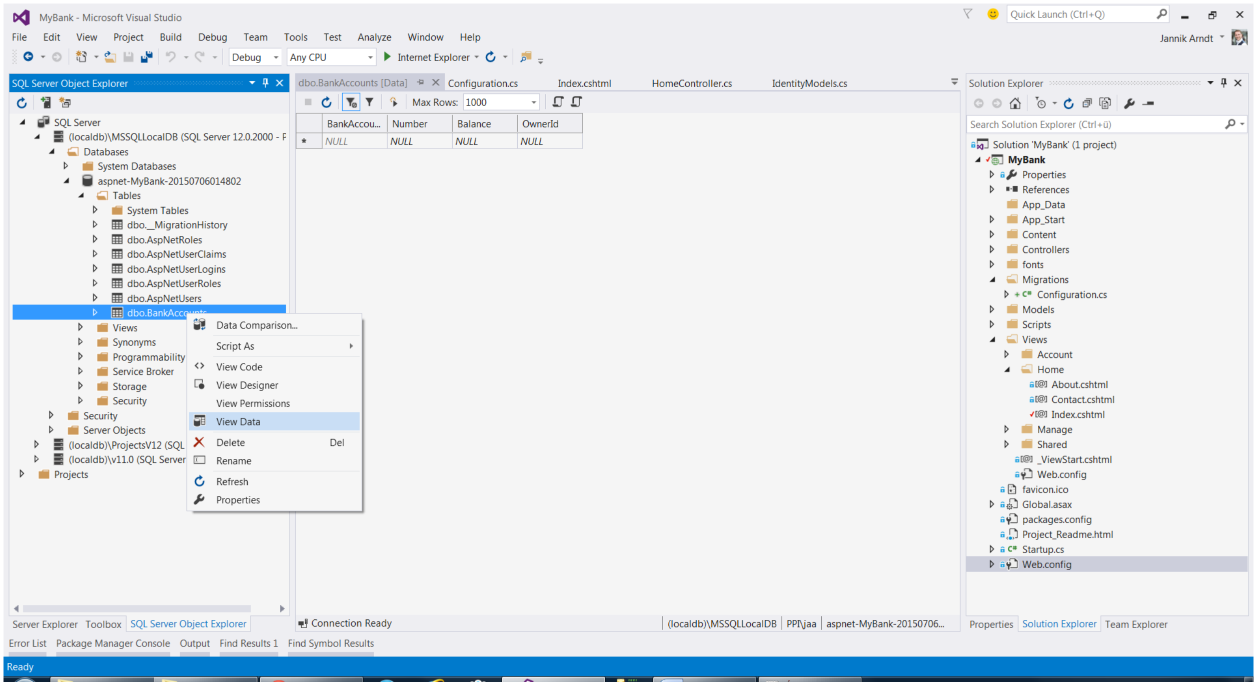
PM> Enable Migrations -EnableAutomaticMigrations

enable automatic migration. In the *SQL Server Object Explorer* (left) you can now look at the freshly created database. The connection to this is usually (localdb) \ MSSQLLocalDB, alternatively the used data source is also entered in the file *Web.config* (middle) in the main directory:



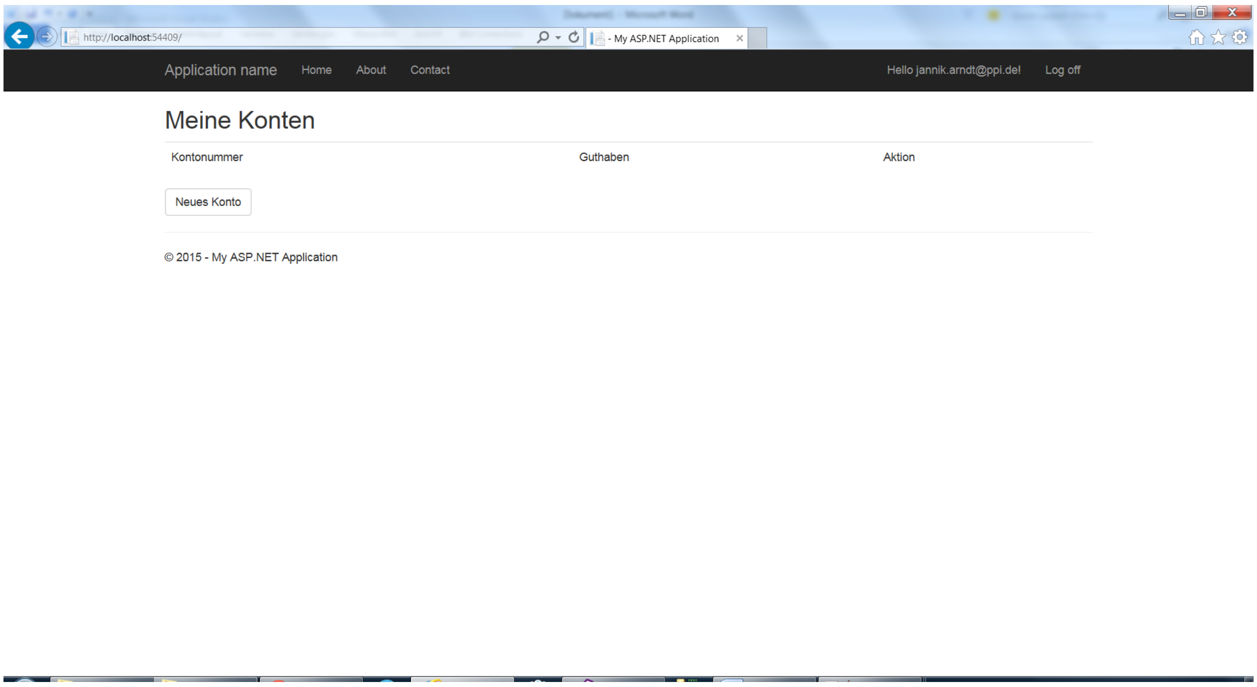
```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?Lir
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, E
  </configSections>
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|I
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
  <system.web>
    <authentication mode="None" />
    <compilation debug="true" targetFramework="4.5.2" />
    <httpRuntime targetFramework="4.5.2" />
  </system.web>
  <system.webServer>
    <modules>
      <remove name="FormsAuthentication" />
    </modules>
  </system.webServer>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.Owin.Security" publicKeyToken="31bf3856ad364e35" />
        <bindingRedirect oldVersion="1.0.0.0-3.0.1.0" newVersion="3.0.1.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.Owin.Security.OAuth" publicKeyToken="31bf3856ad364e35" />
```

However, the call to View Data shows that there is no data yet:



	BankAccou...	Number	Balance	OwnerId
*	NULL	NULL	NULL	NULL

Now back to the application. To do this, start debugging again and register a user. This time it works. All 0 accounts belonging to the user are listed:



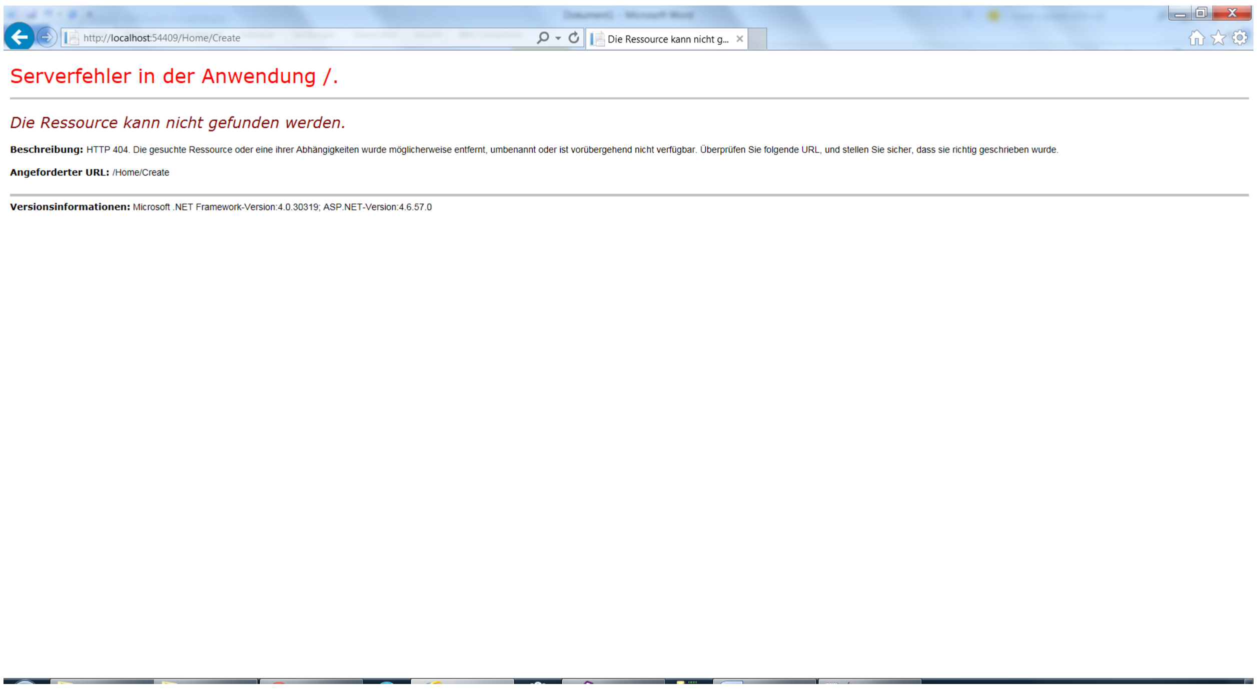
Meine Konten

Kontonummer	Guthaben	Aktion

Neues Konto

© 2015 - My ASP.NET Application

Clicking on *New account* still leads to an error message:


http://localhost:54409/Home/Create
Die Ressource kann nicht g...

Serverfehler in der Anwendung /.

Die Ressource kann nicht gefunden werden.

Beschreibung: HTTP 404. Die gesuchte Ressource oder eine ihrer Abhängigkeiten wurde möglicherweise entfernt, umbenannt oder ist vorübergehend nicht verfügbar. Überprüfen Sie folgende URL, und stellen Sie sicher, dass sie richtig geschrieben wurde.

Angeforderter URL: /Home/Create

Versionsinformationen: Microsoft .NET Framework-Version:4.0.30319; ASP.NET-Version:4.6.57.0

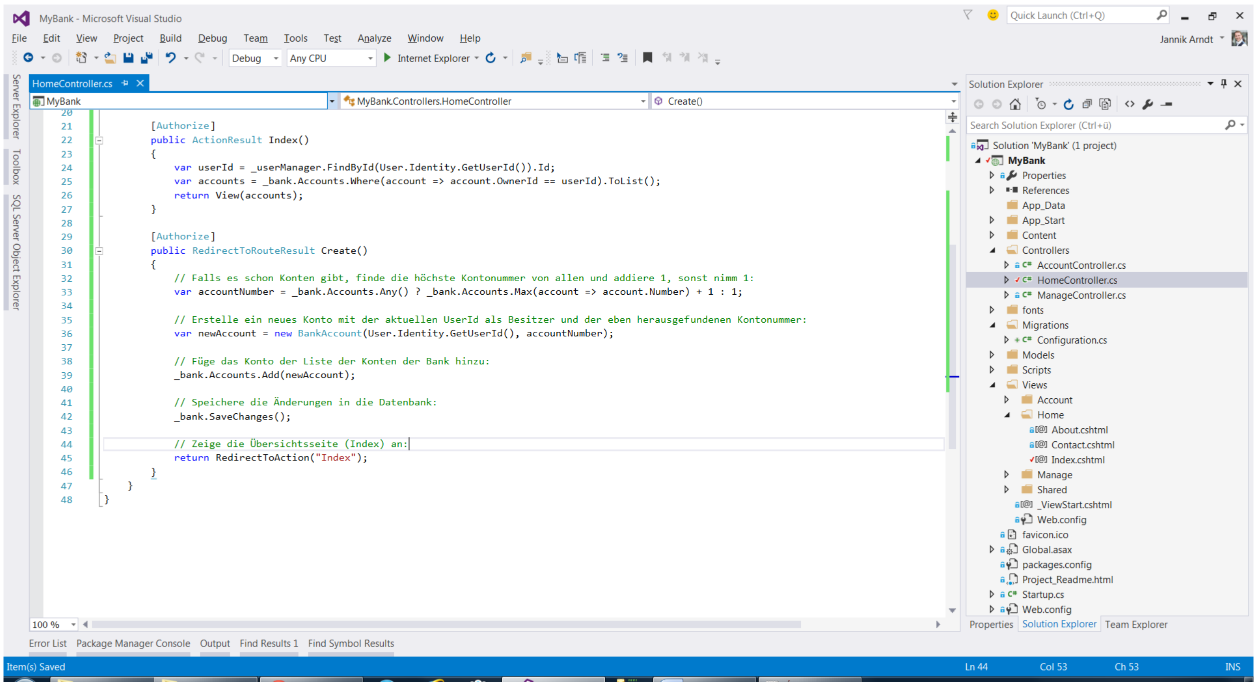
The URL */ Home / Create is* called. *Thus,* a method called *Create ()* is expected in *HomeController.cs* .

Insert the Create method

The HomeController.cs now inserts the method Create ():

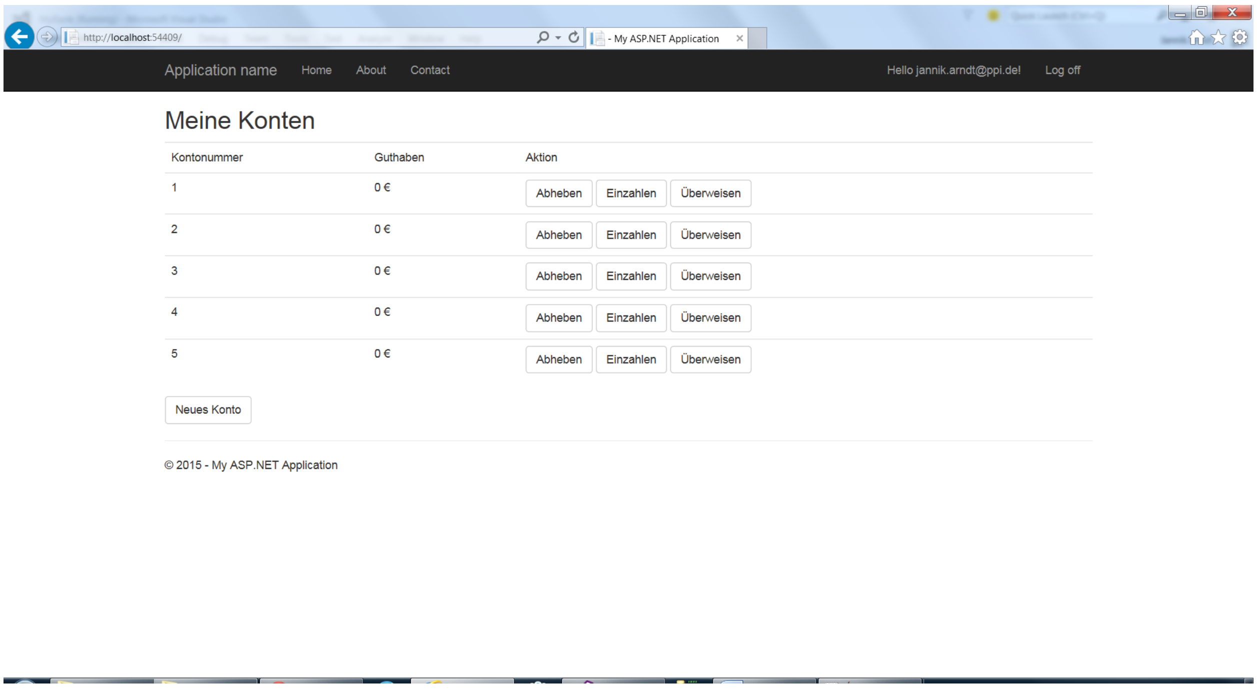
```
[Authorize]
 public  RedirectToRouteResult  Create ()
{
    // If there are already accounts, find the highest account number of all and add 1, otherwise take 1:
    var  accountNumber = _bank.Accounts.Any ()? _bank.Accounts.Max (account => account.Number) + 1 : 1 ;

    // Create a new account with the current UserId as the owner and the account number just found:
    var  newAccount = new  BankAccount (User.Identity.GetUserId (), accountNumber);

    // Add the account to the bank's list of accounts:
    _bank.Accounts.Add (NewAccount);

    // save the changes to the database:
    _bank.SaveChanges ();
```

```
        // Display the index page (index):
        return RedirectToAction ( " Index " );
    }
```



```
        [Authorize]
        public ActionResult Index()
        {
            var userId = _userManager.FindById(User.Identity.GetUserId()).Id;
            var accounts = _bank.Accounts.Where(account => account.OwnerId == userId).ToList();
            return View(accounts);
        }

        [Authorize]
        public RedirectToRouteResult Create()
        {
            // Falls es schon Konten gibt, finde die höchste Kontonummer von allen und addiere 1, sonst nimm 1:
            var accountNumber = _bank.Accounts.Any() ? _bank.Accounts.Max(account => account.Number) + 1 : 1;

            // Erstelle ein neues Konto mit der aktuellen UserId als Besitzer und der eben herausgefundenen Kontonummer:
            var newAccount = new BankAccount(User.Identity.GetUserId(), accountNumber);

            // Füge das Konto der Liste der Konten der Bank hinzu:
            _bank.Accounts.Add(newAccount);

            // Speichere die Änderungen in die Datenbank:
            _bank.SaveChanges();

            // Zeige die Übersichtsseite (Index) an:
            return RedirectToAction("Index");
        }
    }
}
```

By clicking on *New account* a new account will be created and added to the list:



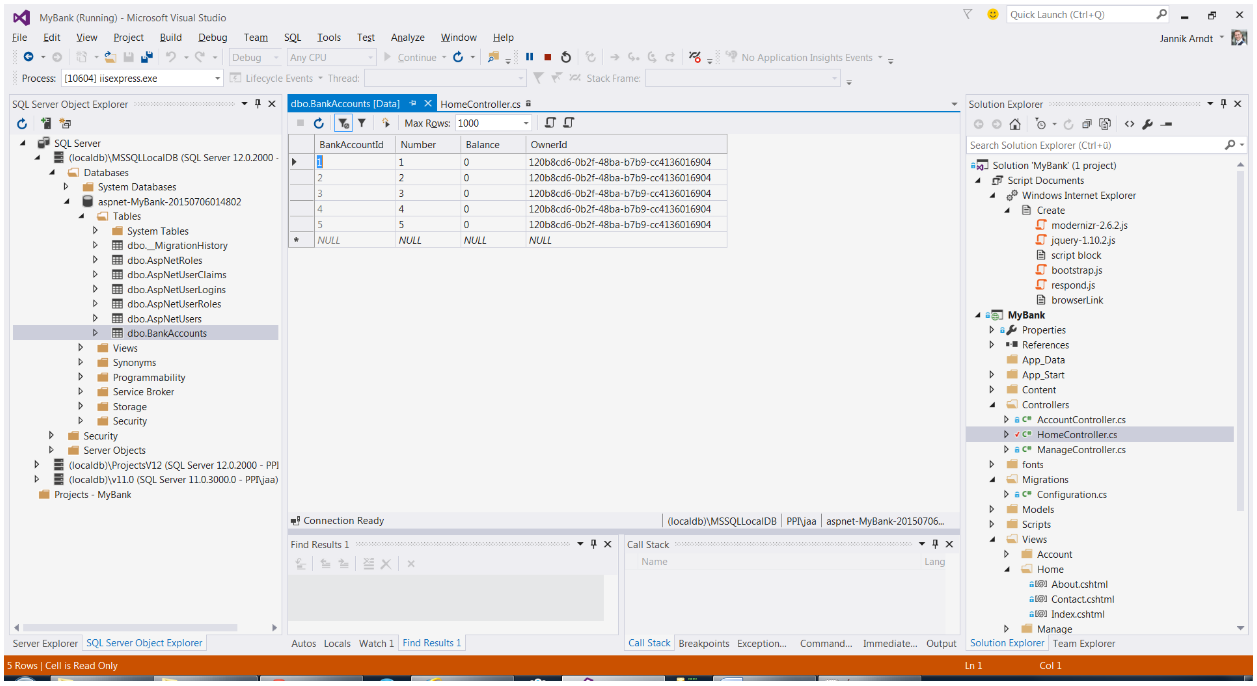
Meine Konten

Kontonummer	Guthaben	Aktion
1	0 €	Abheben Einzahlen Überweisen
2	0 €	Abheben Einzahlen Überweisen
3	0 €	Abheben Einzahlen Überweisen
4	0 €	Abheben Einzahlen Überweisen
5	0 €	Abheben Einzahlen Überweisen

Neues Konto

© 2015 - My ASP.NET Application

These changes can also be found right in the database:



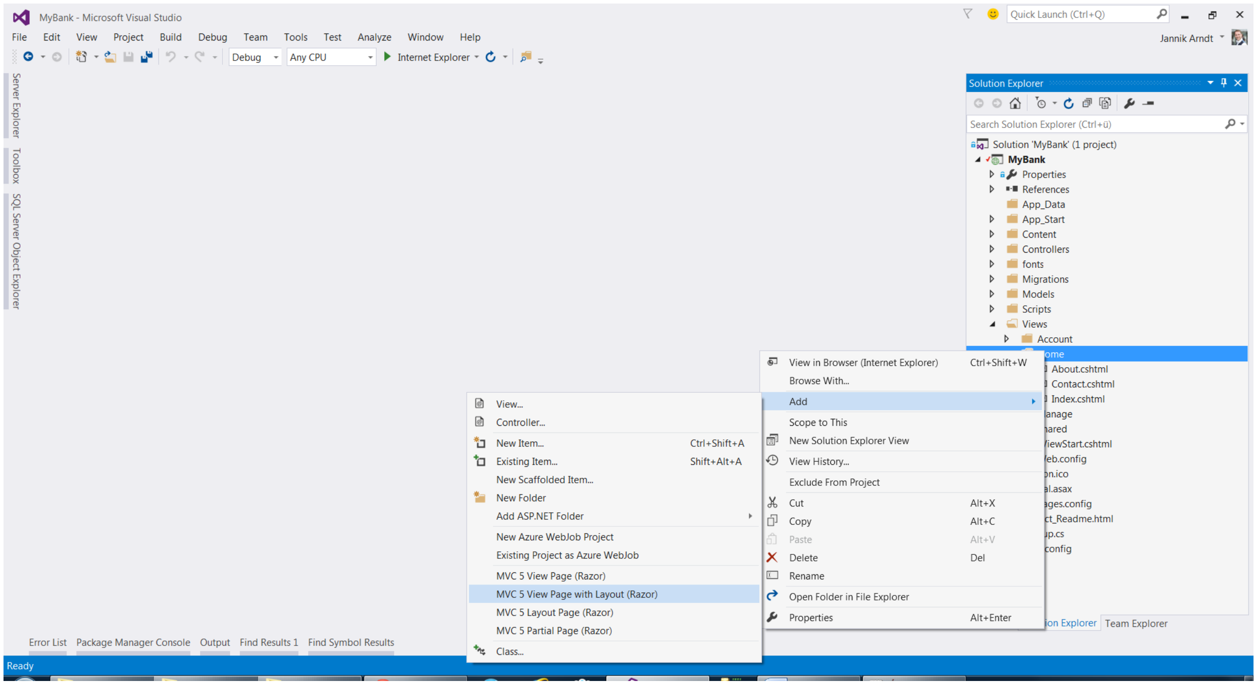
BankAccountId	Number	Balance	OwnerId
1	1	0	120b8cd6-0b2f-48ba-b7b9-cc4136016904
2	2	0	120b8cd6-0b2f-48ba-b7b9-cc4136016904
3	3	0	120b8cd6-0b2f-48ba-b7b9-cc4136016904
4	4	0	120b8cd6-0b2f-48ba-b7b9-cc4136016904
5	5	0	120b8cd6-0b2f-48ba-b7b9-cc4136016904
NULL	NULL	NULL	NULL

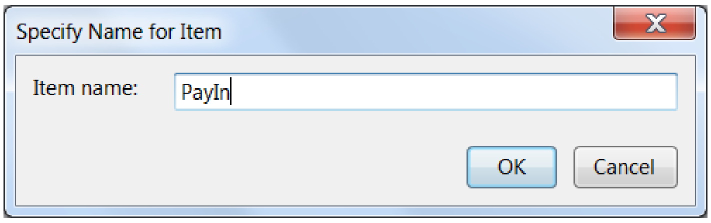
MyBank (Running) - Microsoft Visual Studio
SQL Server Object Explorer
dbo.BankAccounts [Data]
HomeController.cs
Max Rows: 1000
Solution Explorer
Connection Ready
5 Rows | Cell is Read Only

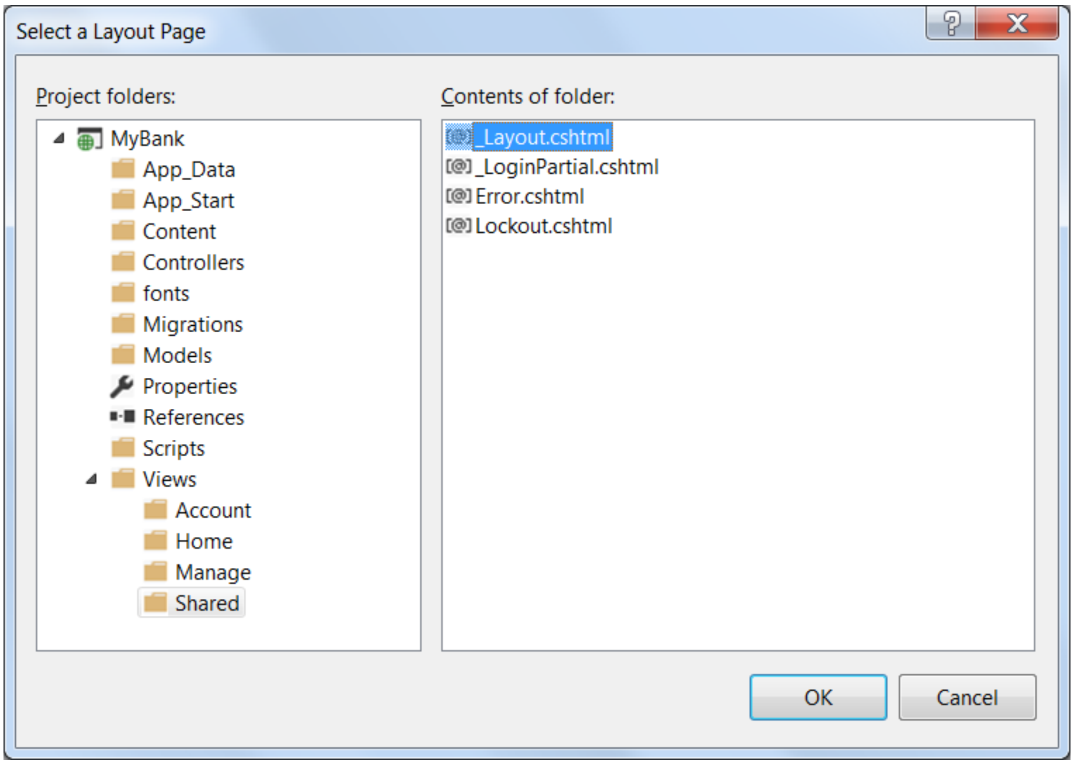
Unfortunately, the methods for depositing, withdrawing and transferring do not lead anywhere.

Other methods and views

The buttons in the list already refer to the methods PayIn, Withdraw and Transfer, which we have to create in the HomeController. However, these actions require additional user input, which means we need a separate view for each method:





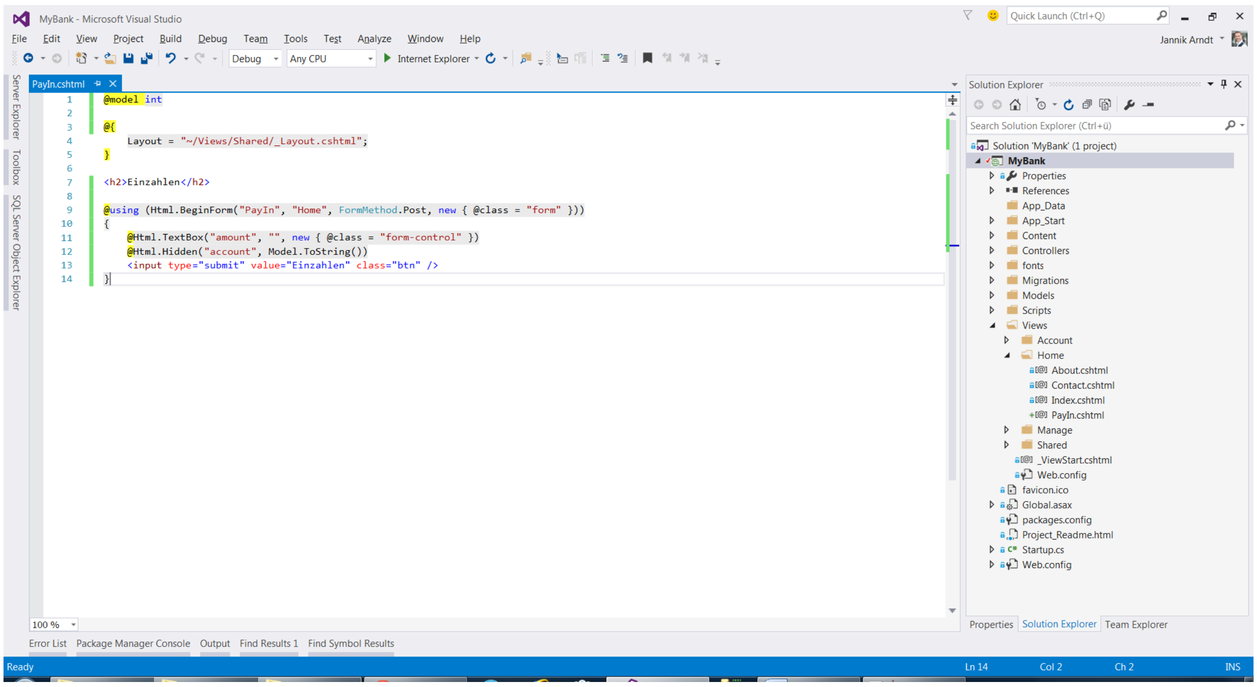

Select a Layout Page

Project folders:

- MyBank
  - App_Data
  - App_Start
  - Content
  - Controllers
  - fonts
  - Migrations
  - Models
  - Properties
  - References
  - Scripts
  - Views
    - Account
    - Home
    - Manage
    - Shared

Contents of folder:

- _Layout.cshtml
- _LoginPartial.cshtml
- Error.cshtml
- Lockout.cshtml


MyBank - Microsoft Visual Studio

PayIn.cshtml

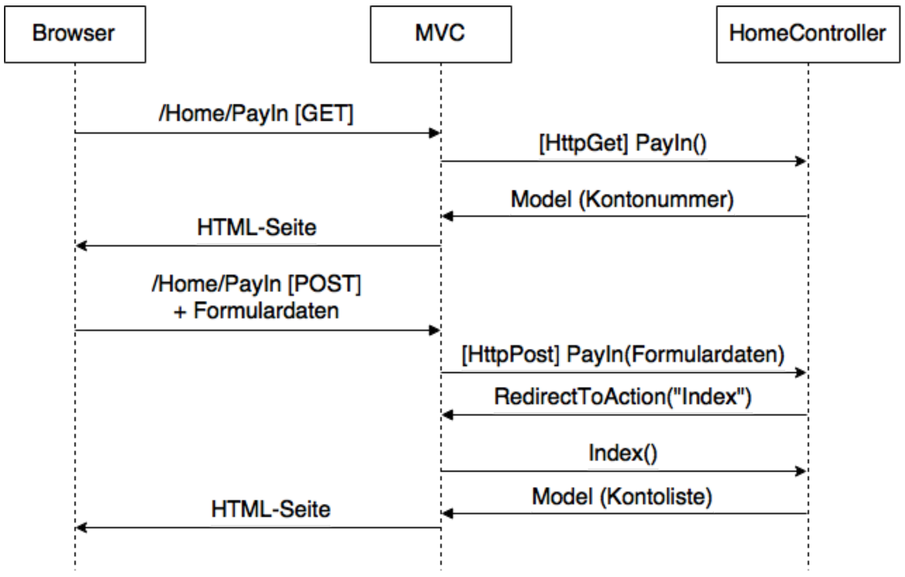
```
@model int

@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Einzahlen</h2>

@using (Html.BeginForm("PayIn", "Home", FormMethod.Post, new { @class = "form" }))
{
    @Html.TextBox("amount", "", new { @class = "form-control" })
    @Html.Hidden("account", Model.ToString())
    <input type="submit" value="Einzahlen" class="btn" />
}
```

View.cshtml:

```
@model int

@ {
    Layout = "~ / Views / Shared / _Layout.cshtml";
}

< h2 > Deposit </ h2 >

@using (Html.BeginForm ("PayIn", "Home", FormMethod.Post, new {@class = "form"}))
{
    @ Html.TextBox ("amount", "", new {@class = "form-control"})
    @ Html.Hidden ("account", Model.ToString ())
    < Br />
    < input  type = " submit "  value = " Deposit "  class = " btn " />
}
```

For each of the actions, we need two calls to the server: the first one generates the view, the second one the data entered (eg the amount) is transmitted and the logic is executed. This can be done using two different methods (eg PayIn and PayInDo), or overloading the same method depending on the type of call (GET or POST) that is chosen.

If the user calls up the URL, the GET method is executed, then he sends the form, this is sent by POST:



```
[HttpGet]
[Authorize]
public  Action  PayIn ( int  account )
{
    return view (account);
}

[HttpPost]
[Authorize]
public  RedirectToRouteResult  PayIn ( int  account , double  amount )
{
    var  myAccount = _bank.Accounts.FirstOrDefault (acc => acc.Number == account);
    if (myAccount == null )
         return RedirectToAction ( " Index " );

    myAccount.PayIn (amount);
    _bank.SaveChanges ();

    return RedirectToAction ( " Index " );
}
[HttpGet]
[Authorize]
public  ActionResult  Withdraw ( int  account )
{
    return view (account);
}

[HttpPost]
[Authorize]
public  RedirectToRouteResult  Withdraw ( int  account , double  amount )
{
    var  myAccount = _bank.Accounts.FirstOrDefault (acc => acc.Number == account);
    if (myAccount! = null )
    {
        if (! myAccount.Withdraw (amount))
             return RedirectToAction ( " Index " );
    }
    else
        return RedirectToAction ( " Index " );

    _bank.SaveChanges ();
    return RedirectToAction ( " Index " );
}

[HttpGet]
[Authorize]
public  ActionResult  Transfer ( int  account )
{
    return view (account);
}

[HttpPost]
[Authorize]
public  RedirectToRouteResult  Transfer ( int  from , int  to , double  amount )
{
    var  sender = _bank.Accounts.FirstOrDefault (acc => acc.Number == from );
    var  receiver = _bank.Accounts.FirstOrDefault (acc => acc.Number == to);
    if (sender! = null && receiver! = null )
    {
        if (! sender.Transfer (amount, receiver))
             return RedirectToAction ( " Index " );
    }
    else
        return RedirectToAction ( " Index " );

    _bank.SaveChanges ();
    return RedirectToAction ( " Index " );
}
```

There is a difference in the Transfer method: Two accounts are accessed here, so the method in the controller has the parameters *from* and *to* , where *from* as otherwise comes from a hidden input field in the view and *to is* entered by the user:

```
@ model int
@ {Layout = " ~ / Views / Shared / _Layout.cshtml " ;}

< h2 > Transfer </ h2>

@ using (Html.BeginForm ( " Transfer " , " Home " , FormMethod.Post, new {@class = " form-horizontal " }))
{
    < label  for = " amount " > amount </ label>
    @ Html.TextBox ( " amount " , " " , new {@class = " form-control " })
    <br />
    < label  for = " to " > recipient </ label>
    @ Html.TextBox ( " to " , " " , new {@class = " form-control " })
    @ Html.Hidden ( " from " , Model.ToString ())
    <br />
    < input  type = " submit "  value = " Transfer " class = " btn btn-primary " />
}
```

Extension to the ViewModel

The methods of the controller pass a model to their associated view, for example, at return view (accounts) ;. But this is currently not a ViewModel, so no specially tailored to the view class. We will change that now. The background is that although we know in the controller whether the calls for withdrawing, withdrawing and transferring have worked or not, but the user can not yet see. So we start by passing a status message as an optional parameter when calling the index method:

```
publicActionResult  Index ( string  message  =  " " , bool  error  =  false , bool  success  =  false )
```

And in the calling methods, an anonymous object is created that contains these parameters:

```
return RedirectToAction ( " Index " , new {message = " The account number " + accountNumber + " was created successfu
```

We also need to create a separate class for the ViewModel:

```
public  ActionResult  Index ( string  message  =  " " , bool  error  =  false , bool  success  =  false )
```

And in the calling methods, an anonymous object is created that contains these parameters:

```
return RedirectToAction ( " Index " , new {message = " The account number " + accountNumber + " was created successfu
```

We also need to create a separate class for the ViewModel:

```
using  System.Collections.Generic ;

namespace  MyBank.Models
{
    public  class  AccountOverviewViewModel
    {
        public List < BankAccount > Accounts { get ; set ; }
         public string  Message { get ; set ; }
         public bool  Error { get ; set ; }
         public bool  Success { get ; set ; }
    }
}
```

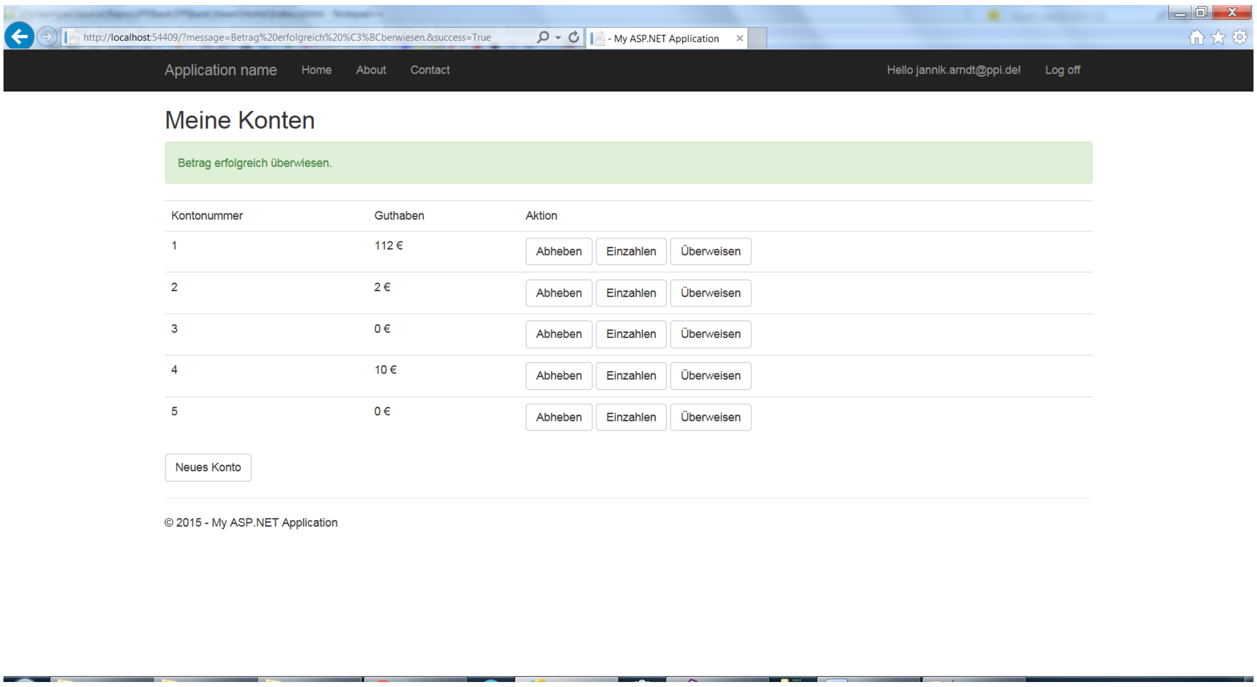
This is now filled in the index method:

```
[Authorize]
 public  ActionResult  Index ( string  message  =  " " , bool  error  =  false , bool  success  =  false )
{
    var  userId = _userManager.FindById (User.Identity.GetUserId ()). Id;

    var  model = new  AccountOverviewViewModel
    {
        Accounts = _bank.Accounts.Where (account => account.OwnerId == userId) .ToList (),
        Message = message,
        Error = error,
        Success = success
    };

    return view (model);
}
```

Now only the view needs to be adjusted:

```
@ model MyBank.Models.AccountOverviewViewModel // other model class

<h2> My accounts </ h2>

@ If (Model.Error)
{
    < div  class = " alert alert-danger " > @ Model.Message </ div>
}
@ If ( Model.Success )
{
    < div  class = " alert alert-success " > @ Model.Message </ div>
}

< table class = "table">
    <Tr>
        <Td> Account number </ td>
        <Td> assets </ td>
        <Td> action </ td>
    </ Tr>
    @ foreach ( var  bankAccount in Model.Accounts) // here now Model.Accounts!
    {
        ...
```

Now success or error messages are displayed after all actions:


http://localhost:54409/?message=Betrag%20erfolgreich%20%C3%BCberwiesen.&success=True
My ASP.NET Application
Application name
Home
About
Contact
Hello jannik.arndt@ppi.de!
Log off

Meine Konten

Betrag erfolgreich überwiesen.

Kontonummer	Guthaben	Aktion
1	112 €	Abheben Einzahlen Überweisen
2	2 €	Abheben Einzahlen Überweisen
3	0 €	Abheben Einzahlen Überweisen
4	10 €	Abheben Einzahlen Überweisen
5	0 €	Abheben Einzahlen Überweisen

Neues Konto

© 2015 - My ASP.NET Application

Didactic Reserve: asynchronous calls

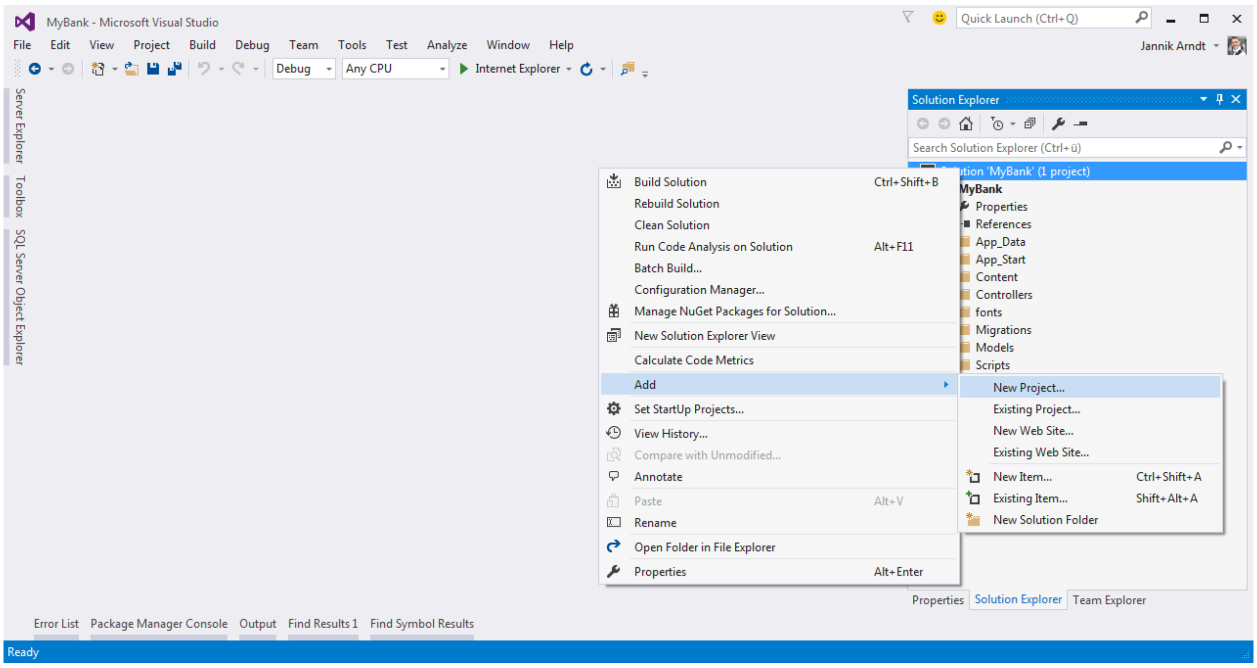
The MVC framework makes it very easy to execute calls to the controller asynchronously, ie while the processor is waiting for a response from, for example, a web service, it can perform other tasks, for example, from other users. The `FindById()` method of the UserManager also offers an asynchronous variant `FindByIdAsync()` . The keyword `await` can be used to signal to the processor that the result of the method will be needed later, so it can not proceed with this task until the result is returned. This also changes the signature of the method slightly, it gets the `async` keyword added and instead of an ActionResults now a `Task<ActionResult>` back:

```
public  async  Task < ActionResult > Index ( string  message  =  " " , bool  error  =  false , bool  success  =  fals
{
    var  user = await _userManager.FindByIdAsync (User.Identity.GetUserId ());
    var  userId = user.Id;

    ...

    return view (model);
}
```

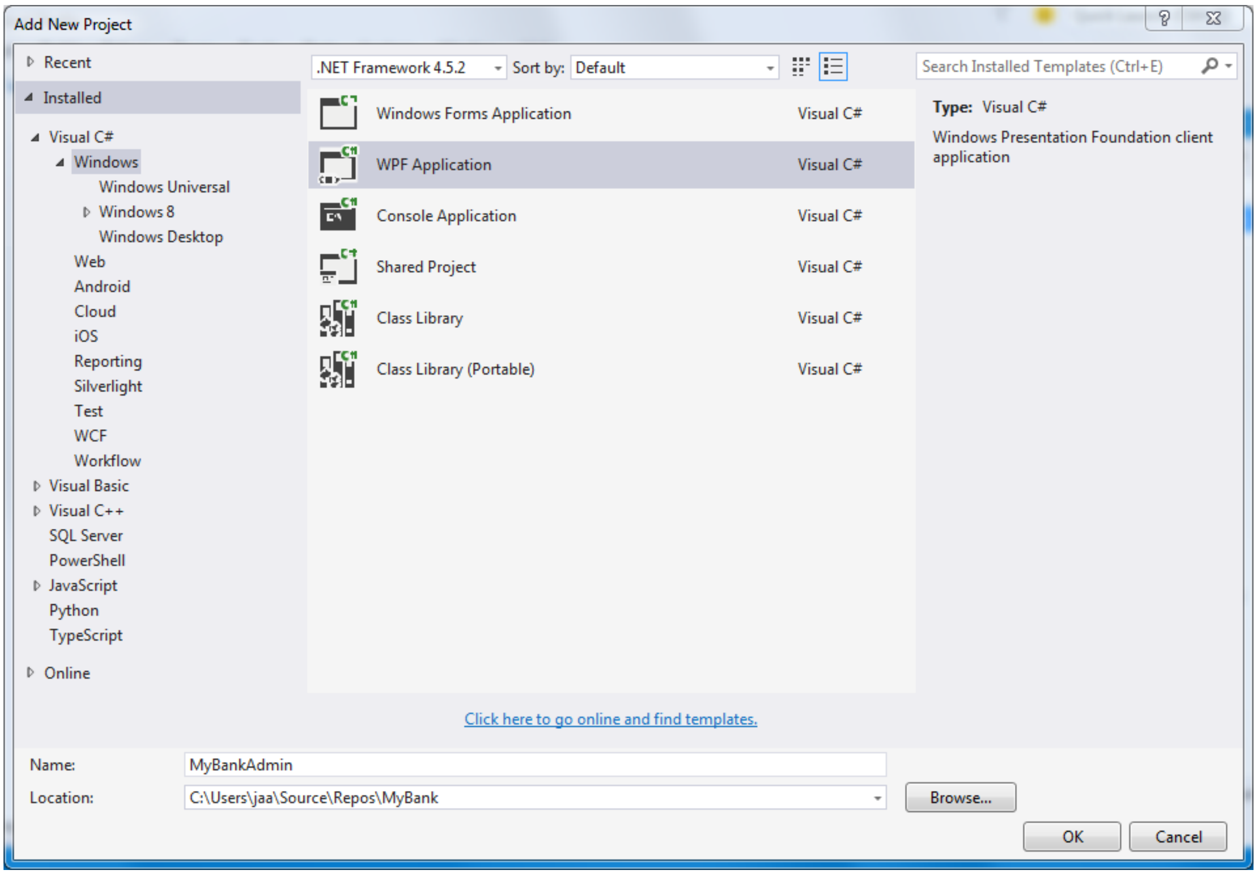
Example project "MyBankAdmin"

WPF Windows program with Entity Framework 6

This project builds on the ASP.NET MVC sample project "MyBank" and uses its database.First we add a new project to the MyBank solution:



Select a WPF project:



.NET Framework 4.5.2 Sort by: Default

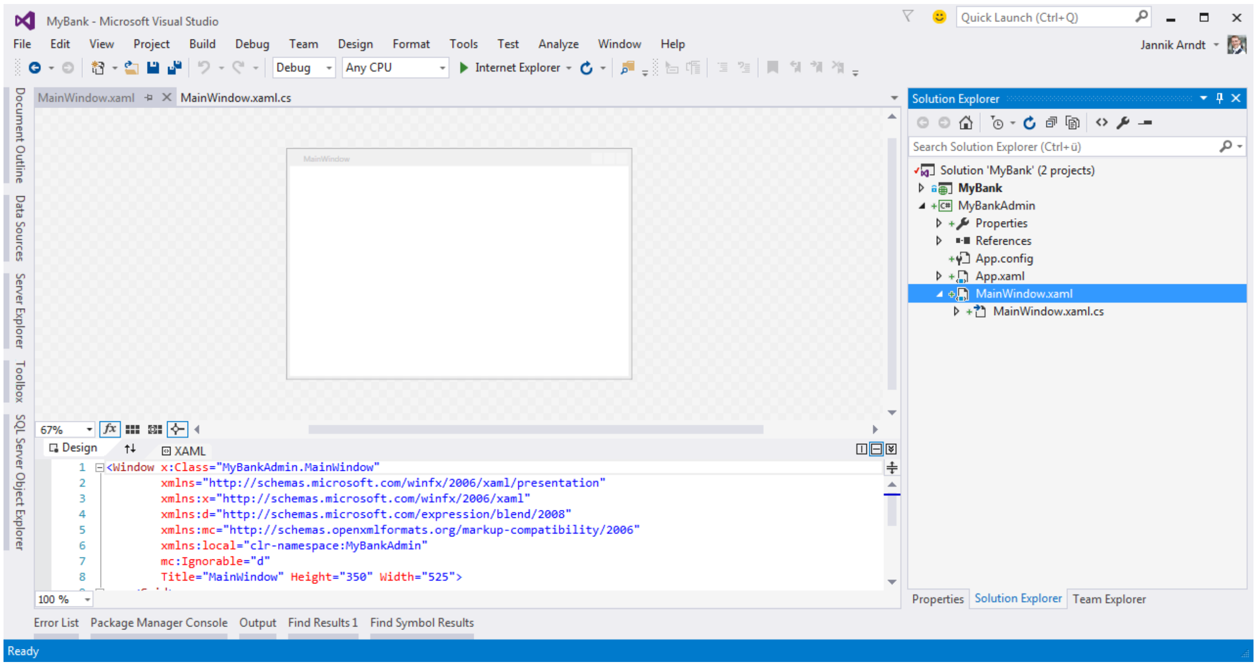
Type: Visual C#

Windows Presentation Foundation client application

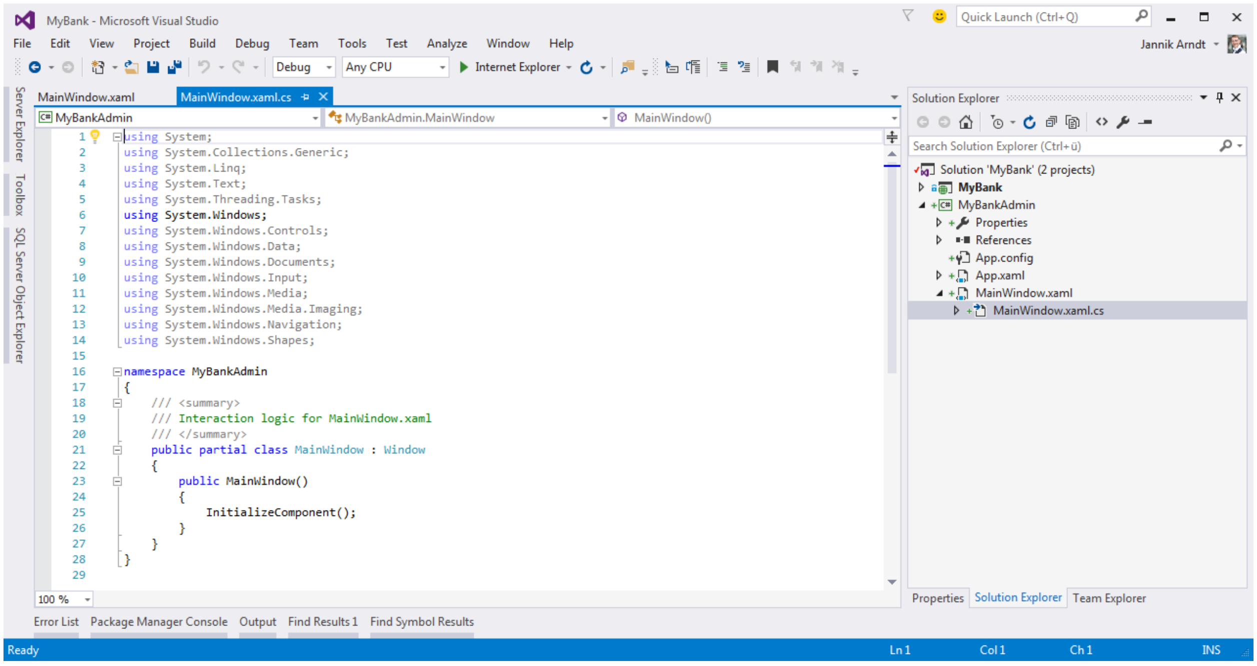
Name: MyBankAdmin

Location: C:\Users\jaa\Source\Repos\MyBank

A window consists of two files: the XAML file for the interface and the xaml.cs for the backend:



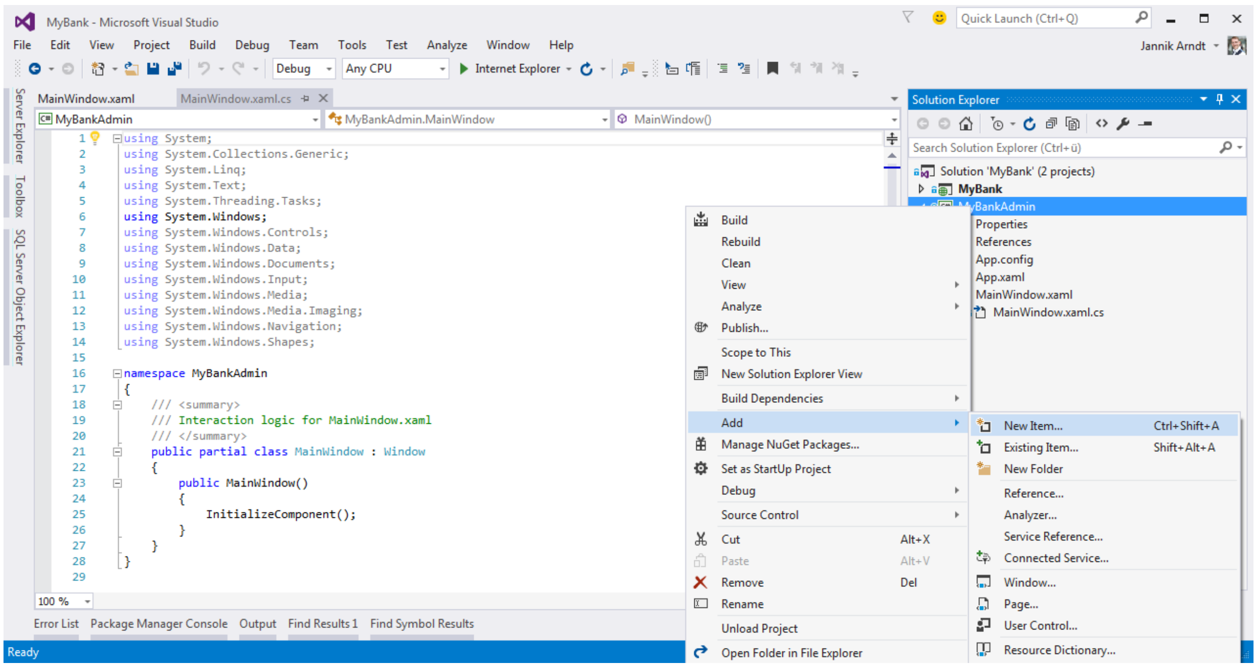
```xml
<Window x:Class="MyBankAdmin.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MyBankAdmin"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
```



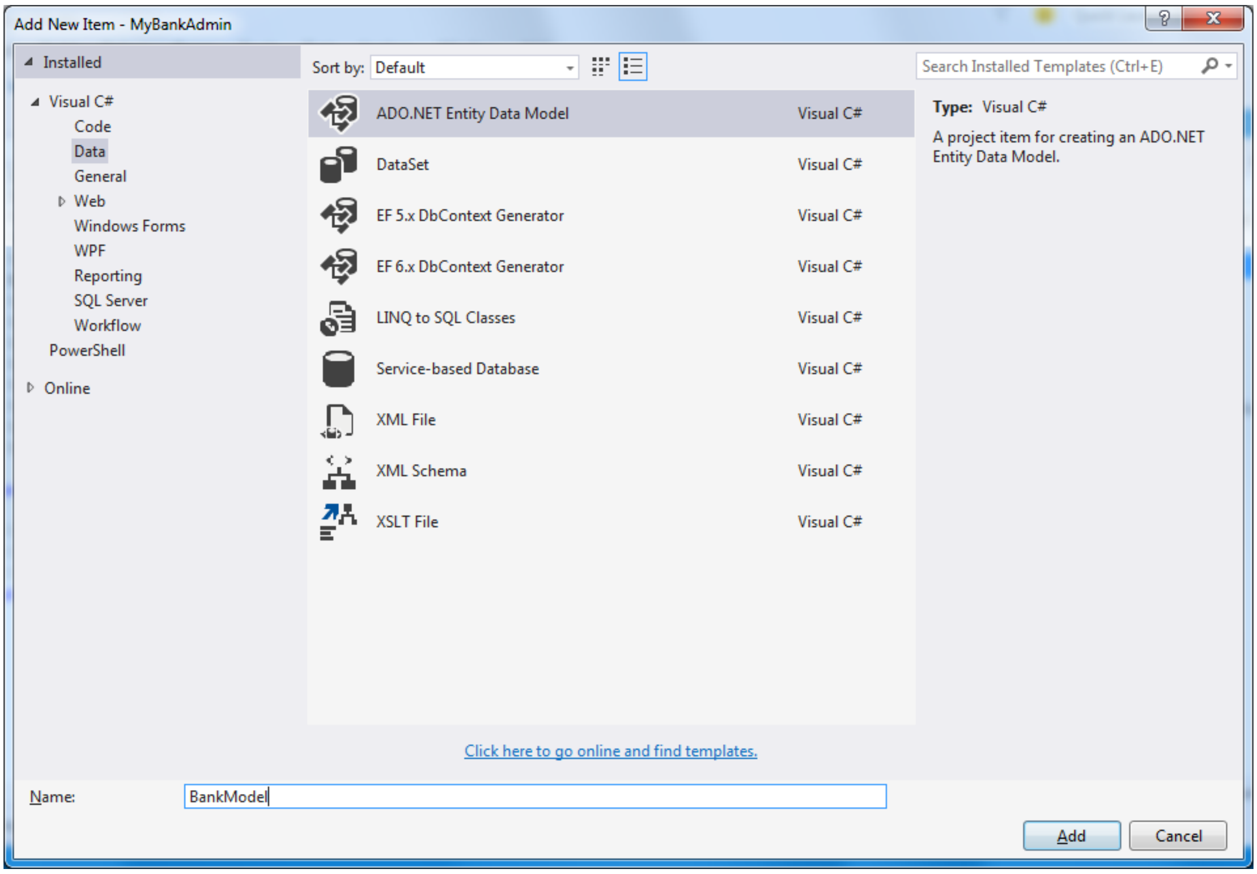
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MyBankAdmin
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Insert database connection

Now we add the data source from the MVC sample project:



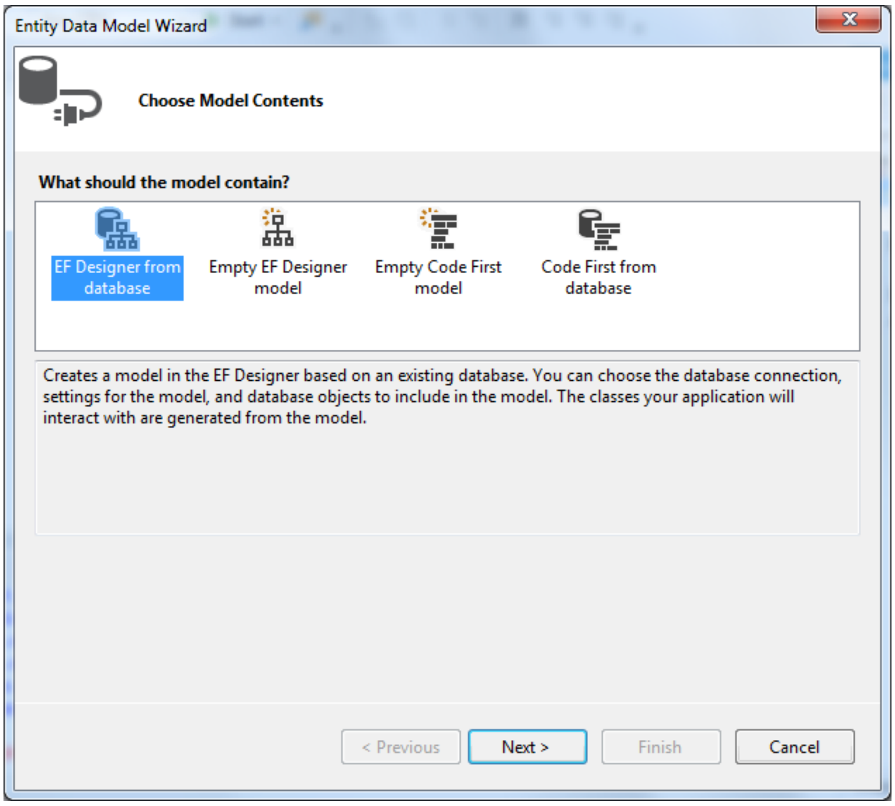
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MyBankAdmin
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```


Add New Item - MyBankAdmin
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
SQL Server
Workflow
PowerShell
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
ADO.NET Entity Data Model Visual C#
DataSet Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
LINQ to SQL Classes Visual C#
Service-based Database Visual C#
XML File Visual C#
XML Schema Visual C#
XSLT File Visual C#
Click here to go online and find templates.
Add
Cancel

Type: Visual C#

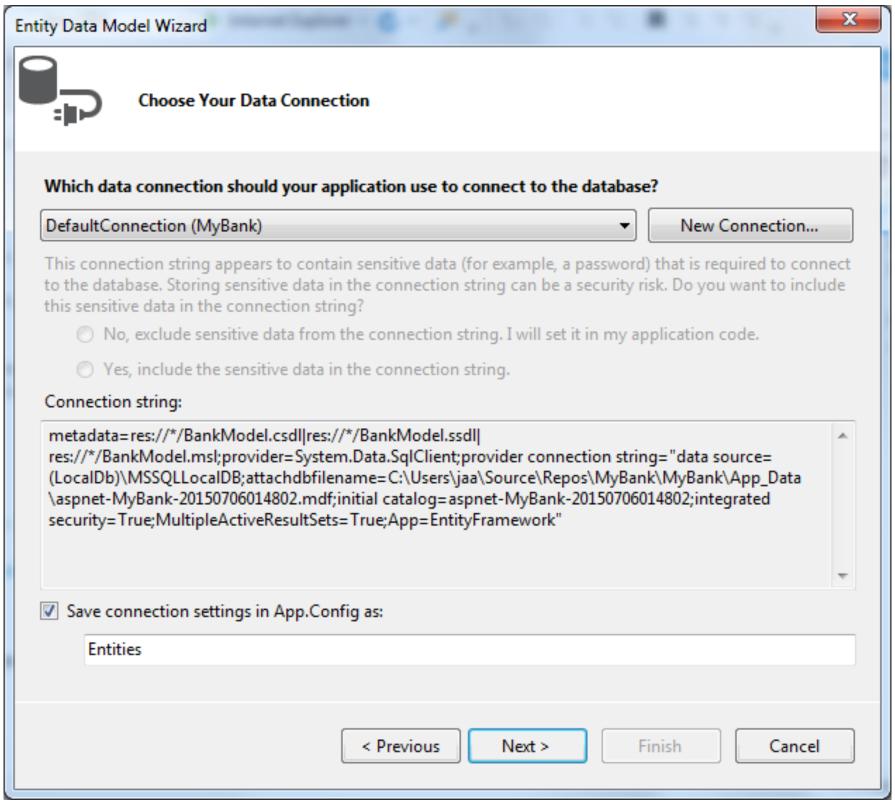
A project item for creating an ADO.NET Entity Data Model.

Name: BankModel



What should the model contain?

Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.



Which data connection should your application use to connect to the database?

DefaultConnection (MyBank)

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

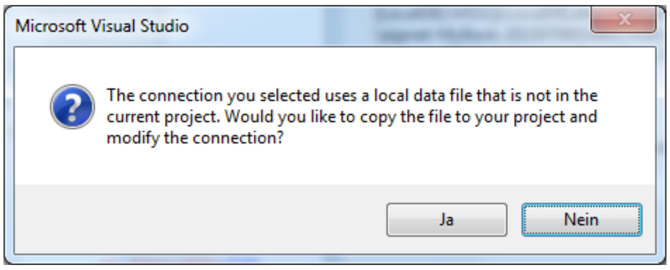
- No, exclude sensitive data from the connection string. I will set it in my application code.
- Yes, include the sensitive data in the connection string.

Connection string:

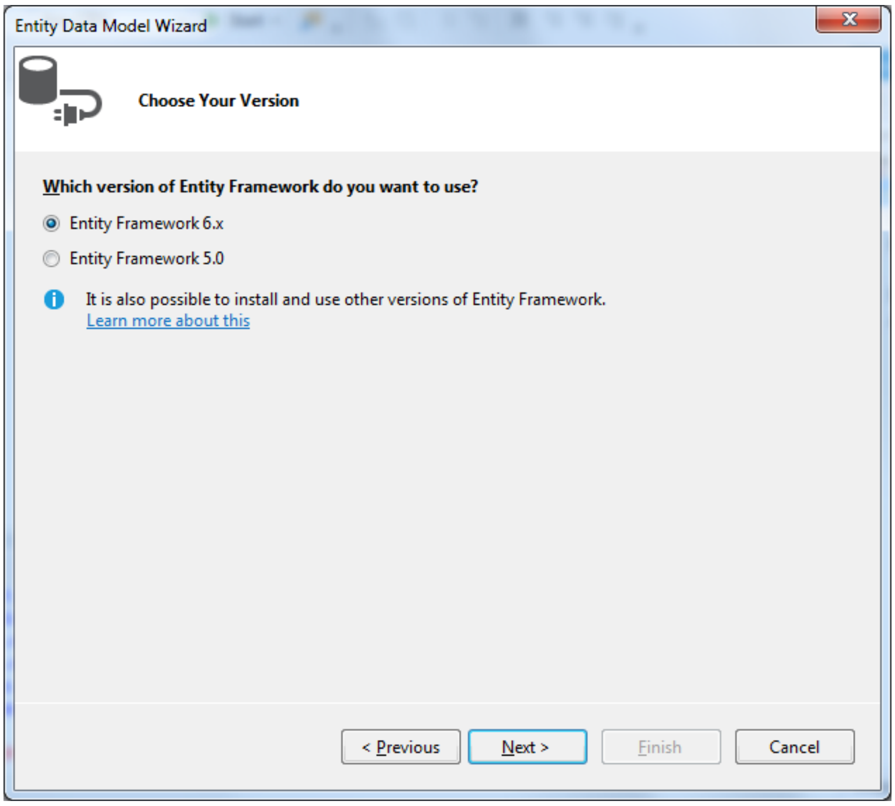
```
metadata=res://*/BankModel.csdl|res://*/BankModel.ssdl|
res://*/BankModel.msl;provider=System.Data.SqlClient;provider connection string="data source=
(LocalDb)\MSSQLLocalDB;attachdbfilename=C:\Users\jaa\Source\Repos\MyBank\MyBank\App_Data
\aspnet-MyBank-20150706014802.mdf;initial catalog=aspnet-MyBank-20150706014802;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☑ Save connection settings in App.Config as:

Entities



The connection you selected uses a local data file that is not in the current project. Would you like to copy the file to your project and modify the connection?


Entity Data Model Wizard

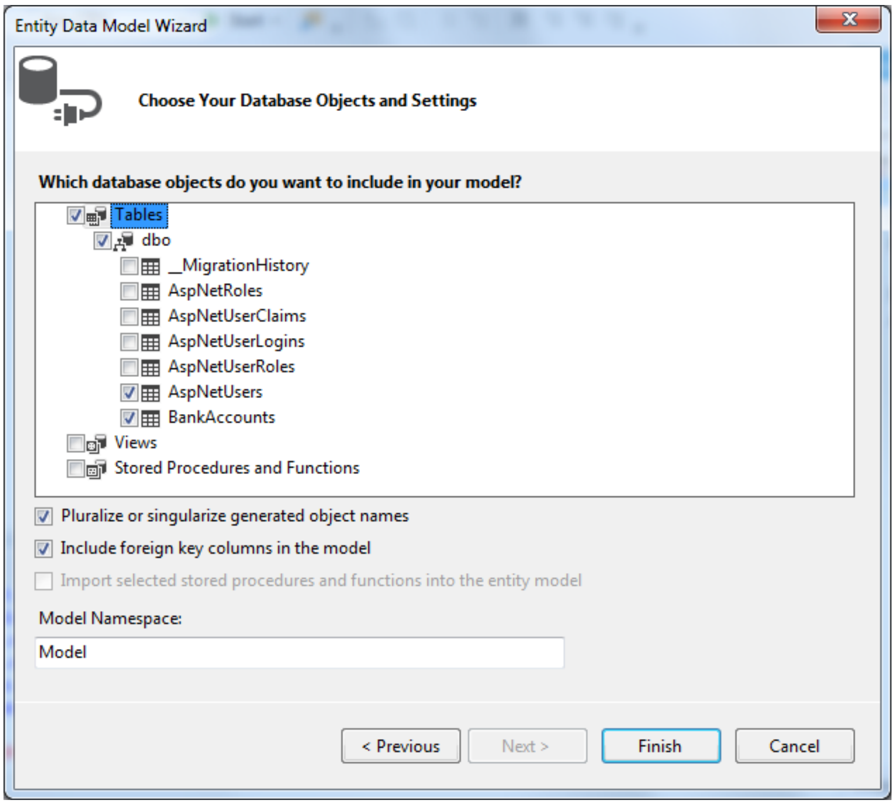
Choose Your Version

Which version of Entity Framework do you want to use?

(●) Entity Framework 6.x

( ) Entity Framework 5.0

It is also possible to install and use other versions of Entity Framework.
Learn more about this


Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- [✓] Tables
  - [✓] dbo
    - [ ] __MigrationHistory
    - [ ] AspNetRoles
    - [ ] AspNetUserClaims
    - [ ] AspNetUserLogins
    - [ ] AspNetUserRoles
    - [✓] AspNetUsers
    - [✓] BankAccounts
- [ ] Views
- [ ] Stored Procedures and Functions

[✓] Pluralize or singularize generated object names

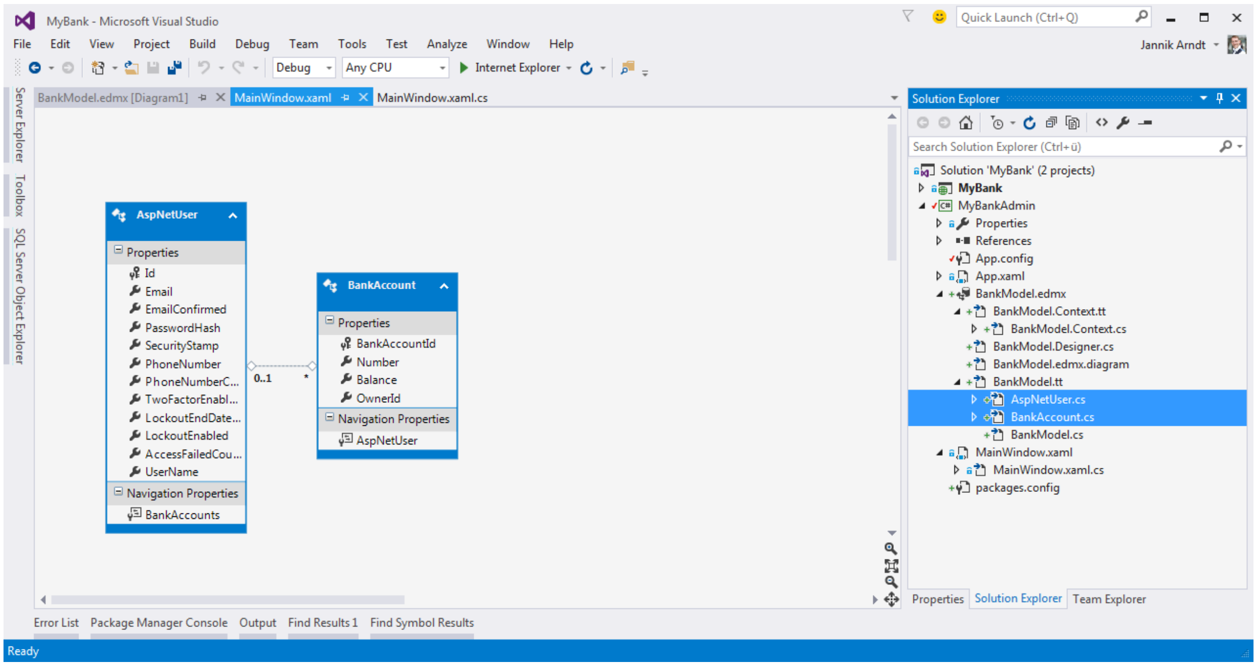
[✓] Include foreign key columns in the model

[ ] Import selected stored procedures and functions into the entity model

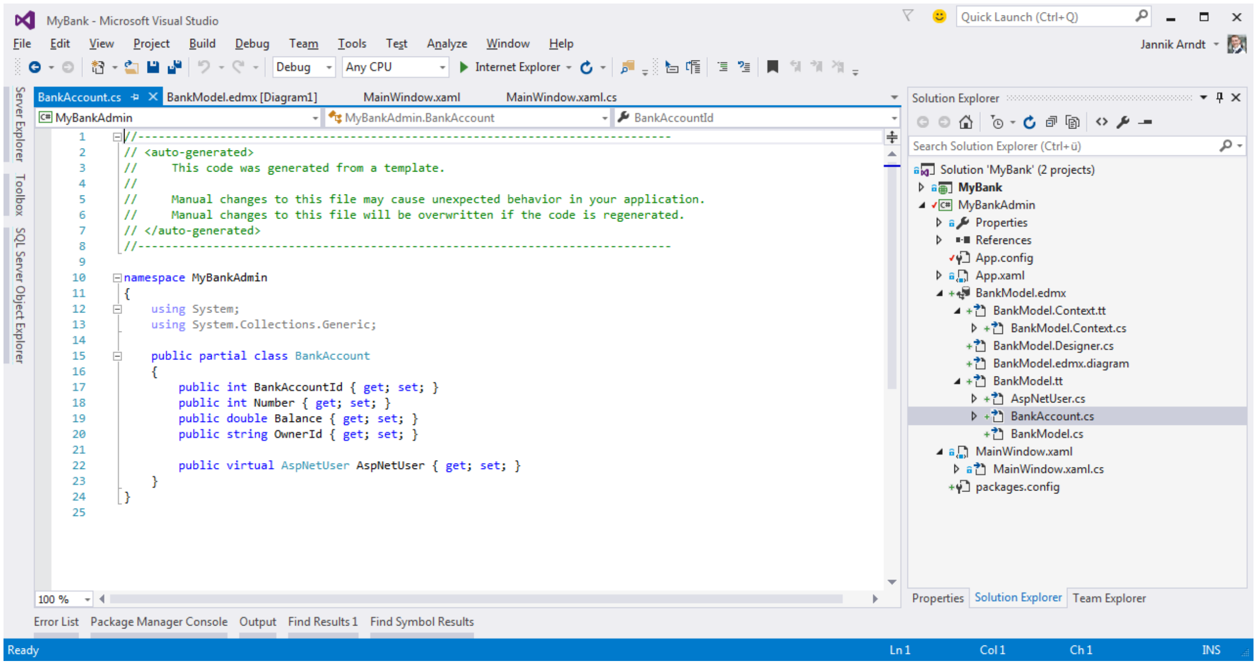
Model Namespace:

Model

The two generated classes are displayed as an edmx diagram and can be found in the Solution Explorer in this edmx file:



There is an indication in the file that this class is generated. It can be changed arbitrarily, but if it is regenerated, eg because the database has changed, these changes will be overwritten.

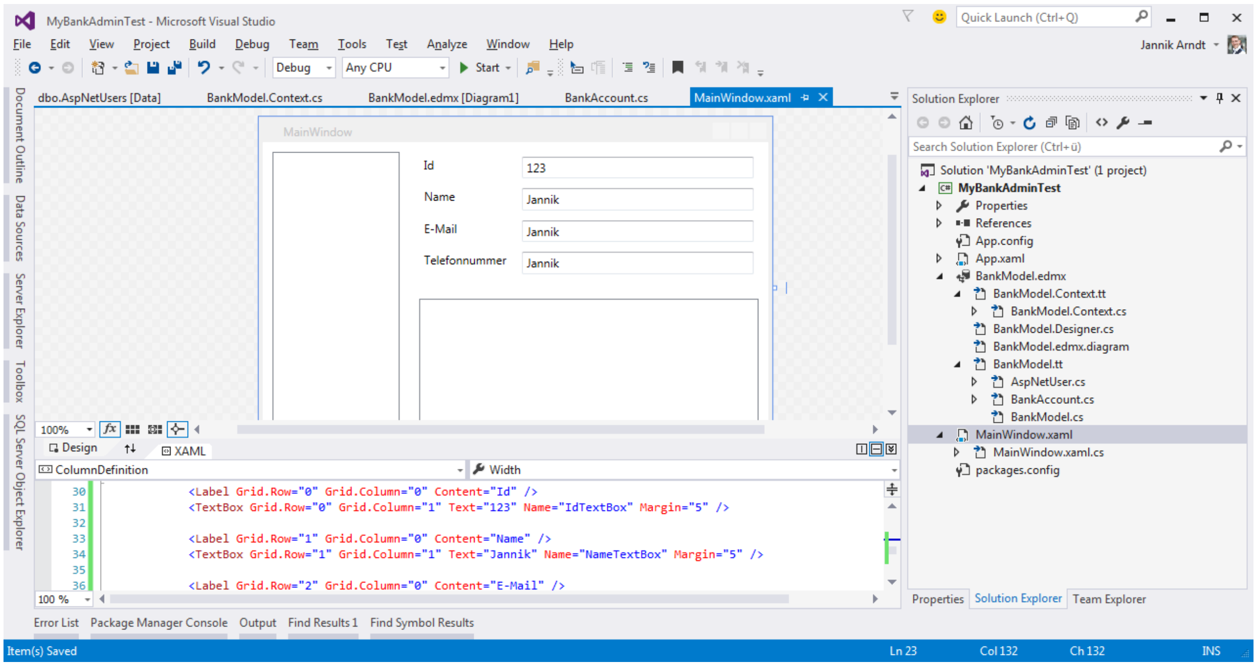


```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace MyBankAdmin
{
    using System;
    using System.Collections.Generic;
    
    public partial class BankAccount
    {
        public int BankAccountId { get; set; }
        public int Number { get; set; }
        public double Balance { get; set; }
        public string OwnerId { get; set; }
    
        public virtual AspNetUser AspNetUser { get; set; }
    }
}
```

show entries

Now we can build a surface in MainWindow.xaml to show the data from the database. Target is a list of all users when selecting a user to display details and a listing of his accounts.

XAML files are displayed in the split view, a preview at the top, the code below. The tab "Toolbox" can be used to graphically edit GUI elements, but in order to keep full control, you can also create the GUI in the code.



Id	123
Name	Jannik
E-Mail	Jannik
Telefonnummer	Jannik

```
            <Label Grid.Row="0" Grid.Column="0" Content="Id" />
            <TextBox Grid.Row="0" Grid.Column="1" Text="123" Name="IdTextBox" Margin="5" />

            <Label Grid.Row="1" Grid.Column="0" Content="Name" />
            <TextBox Grid.Row="1" Grid.Column="1" Text="Jannik" Name="NameTextBox" Margin="5" />

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

The root element is a window tag, under which a grid element is already specified. We divide our grid into two halves, on the left the customer overview, on the right the details.

```
< Grid >
    < Grid .ColumnDefinitions>
        < ColumnDefinition  Width = " 150 " />
        < ColumnDefinition  Width = " \ * " />
    </ Grid .ColumnDefinitions> <! - Links
     Customer List -> <! - Legal
     Details ->
</ Grid >
```

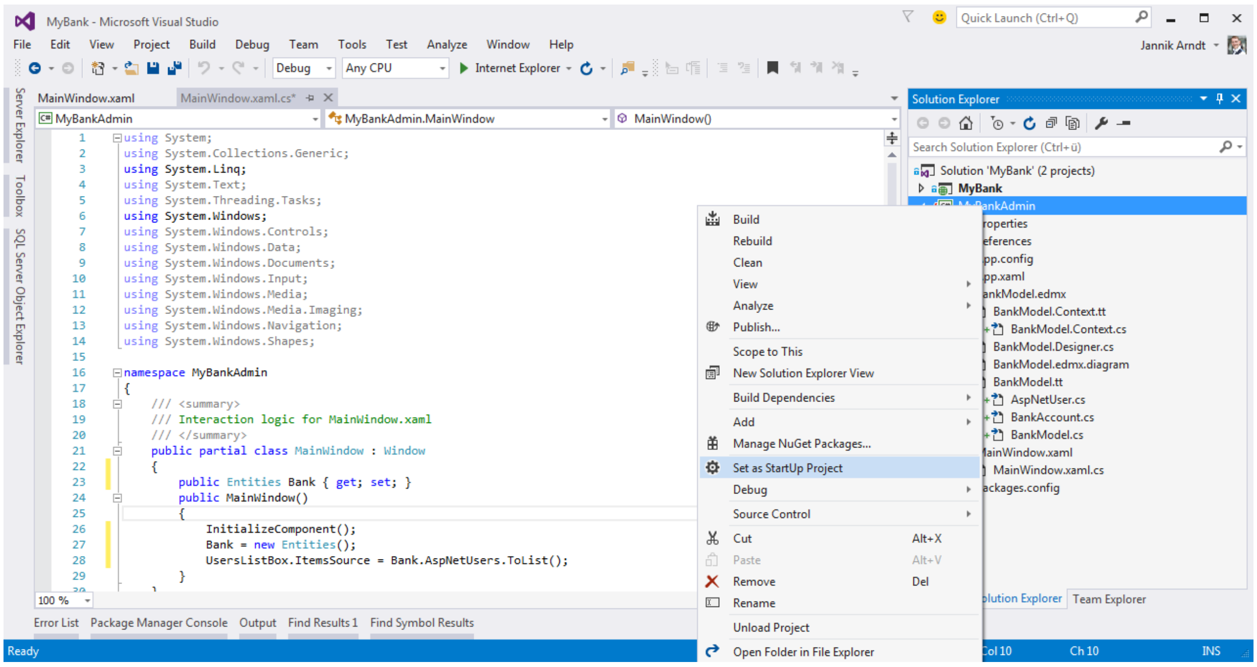
The client list is an `ListBox` element, the detail view another `Grid` with `Label` - and `TextBox` elements. For the listing of the accounts we use an `DataGrid` element:

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

    < Label Grid.Row = " 4 " Grid.Column = " 0 "  Content = " Accounts " />
    < DataGrid Grid.Row = " 4 " Grid.Column = " 1 "  Name = " AccountsDataGrid "  Margin = " 5 " />

</ Grid >
```

First, we populate the user list from the backend ("Code Behind") by adding `MainWindow` a `Entities` property called "Bank," which represents our database, to the class. In the constructor, this is initialized and the users are added to the list:

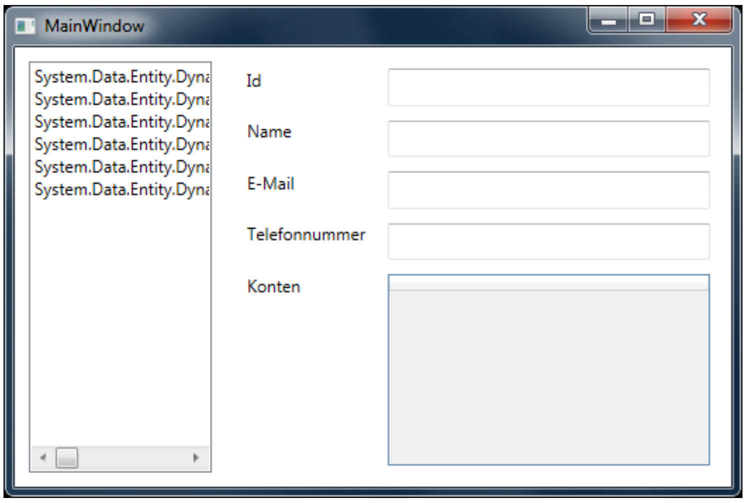
```
public  partial  class  MainWindow : Window
{
    public entities  bank { get ; set ; }

    public  MainWindow ()
    {
        InitializeComponent ();
        Bank = newEntities ();
        UsersListBox.ItemsSource = Bank.AspNetUsers.ToList ();
    }
}
```

Time for a first test! In Solution Explorer you still need to set the project as StartUp Project:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MyBankAdmin
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public Entities Bank { get; set; }
        public MainWindow()
        {
            InitializeComponent();
            Bank = new Entities();
            UsersListBox.ItemsSource = Bank.AspNetUsers.ToList();
        }
```

By clicking on "Start" the program appears:



The users appear in the list, but what the `ToString()` method returns. This can be controlled via the DisplayMemberPath attribute:

```
< ListBox Grid.Row = " 0 " Grid.Column = " 0 "  Name = " UsersListBox "  Margin = " 10 "  DisplayMemberPath = " UserN
```

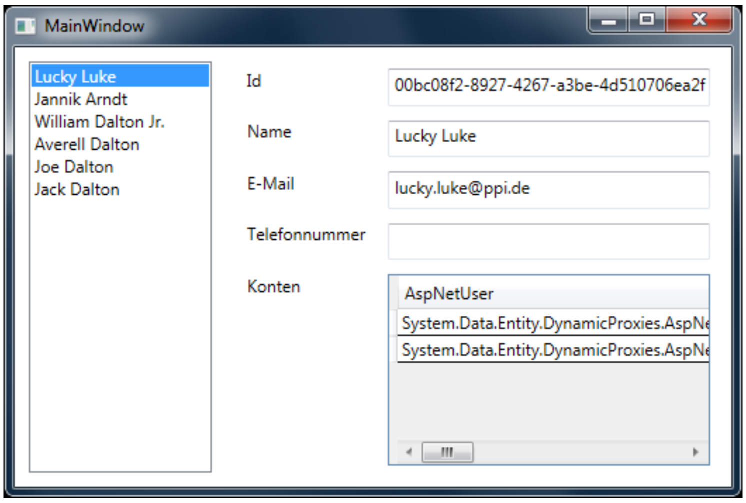
Next, the details should be displayed when clicking on a name. If you double-click on the ListBox in the Designer, the attribute is `SelectionChanged="UsersListBox\_SelectionChanged"` added to the XAML element . The corresponding method is automatically generated in the code behind. As a parameter she gets one `sender` and an event. The `sender` object is the ListBox, from which one gets after the cast also `selectedItem` :

```
var  user = ( sender asListBox) .SelectedItem asAspNetUser;
```

The properties of the user object can now be assigned to the TextBoxes and the AccountsDataGrid:

```
private  void UsersListBox \ _SelectionChanged ( object  sender , SelectionChangedEventArgs  e )
{
    var  user = (sender asListBox). SelectedItem  asAspNetUser ;

    IdTextBox.Text = user.Id;
    NameTextBox.Text = user.UserName;
    EmailTextBox.Text = user.Email;
    PhoneTextBox.Text = user.PhoneNumber;

    AccountsDataGrid.ItemsSource = user.BankAccounts;
}
```

That leads to:



By default, a column is generated in the DataGrid for all properties. You can `AutoGenerateColumns="False"` disable this with , but then you have to define the columns you want to keep:

```
< DataGrid Grid.Row = " 4 " Grid.Column = " 1 "  Name = " AccountsDataGrid "  Margin = " 5 "  AutoGenerateColumns = "
    < DataGrid .Columns>
        < DataGridTextColumn  Binding = " {Binding Path = Number} "  header = " account number "  Width = " \ * " />
        < DataGridTextColumn  Binding = " {Binding Path = Balance, ConverterCulture = 'DE-GB', StringFormat = {} {0:
    </ DataGrid .Columns>
</ DataGrid >
```

Bindings between model and view

In the code behind the DataGrid a list is passed as ItemsSource, in the XAML code columns are bound to certain properties, the rest is created by the framework on its own. However, these bindings not only work for individual elements, but also for entire blocks. For example, we can pass the user object as a context to the grid, where we display all details about the user. The assignment in the SelectionChanged method then only consists of one line:

```
UserItemsGrid.DataContext = (sender asListBox) .SelectedItem asAspNetUser;
```

In the XAML file, the TextBox elements get an attribute `Text="{Binding UserName}"` and the DataGrid the attribute `ItemsSource="{Binding BankAccounts}"` .

Edit entries

If you edit an entry, select another user and then return to the edited user, you will see that the change remains. This is because the text boxes are bound to the objects in memory by bindings and this binding is defined in both directions. However, the changes are not persisted in the database.

But you do not need a lot of code for this, it's enough to put the row in the SelectionChanged method

```
Bank.SaveChanges ();
```

to include.

However, if you `InvalidOperationException` attempt to change a balance by double-clicking, you will be thrown one with the information that "EditItem" is not allowed for this view, because the entity framework in the AspNetUsers class is the property

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

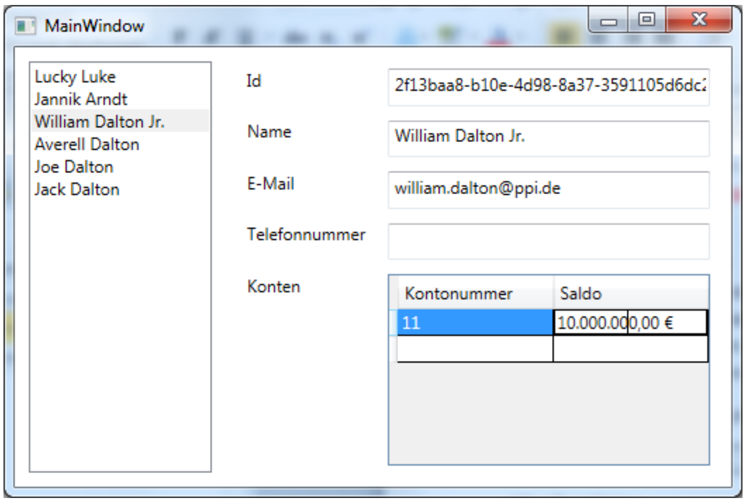
that is initialized as a hash set:

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

If you change these two entries `List<BankAccount>` , editing becomes suddenly possible.

However, the changes will not be persisted to the database unless you select a different user. With a double click on the `DataGrid` one generates the method SelectionChanged in which you, as in `SelectionChanged` the `ListBox` , can initiate a Save:

```
private  void AccountsDataGrid \ _SelectionChanged ( object  sender , SelectionChangedEventArgs  e )
{
    Bank.SaveChanges ();
}
```



Lucky Luke
Jannik Arndt
William Dalton Jr.
Averell Dalton
Joe Dalton
Jack Dalton

Id	2f13baa8-b10e-4d98-8a37-3591105d6dc2
Name	William Dalton Jr.
E-Mail	william.dalton@ppi.de
Telefonnummer	
Konten	

Kontonummer	Saldo
11	10.000.000,00 €

This also completes the creation of the admin software.