

Introduction to ASP.NET Identity

Abstract

Background

ASP.NET Membership

ASP.NET Membership was designed to solve requirements common in 2005, which involved Forms Authentication stored in a SQL Server database. The limitations today are:

- The database schema was designed for SQL Server and some methods imply at least a relational database.
- Since the log-in/log-out functionality is based on Forms Authentication, the membership system can't use OWIN which include the support for using external providers like Facebook, Google or Twitter

ASP.NET Simple Membership

The goal of ASP.NET Simple Membership was to make it easy to add membership functionality to a Web Pages application. But as the old Membership system you can't use it with OWIN, and is not extensible.

ASP.NET Universal Providers

ASP.NET Universal Providers were developed to make it possible to persist membership information in Microsoft Azure SQL Database with EF but the Universal Providers are built on the ASP.NET Membership infrastructure so they still carry the same limitations and still uses the Forms Authentication mechanism.

ASP.NET Identity

The assumption that users will log in by entering a user name and password that they have registered in your own application is no longer valid. The web has become more social. Users are interacting with each other in real time through social channels such as Facebook, Twitter, and other social web sites. Developers want users to be able to log in with their social identities so that they can have a rich experience on their web sites. A modern membership system must enable redirection-based log-ins to authentication providers such as Facebook, Twitter, and others.

Design Goals

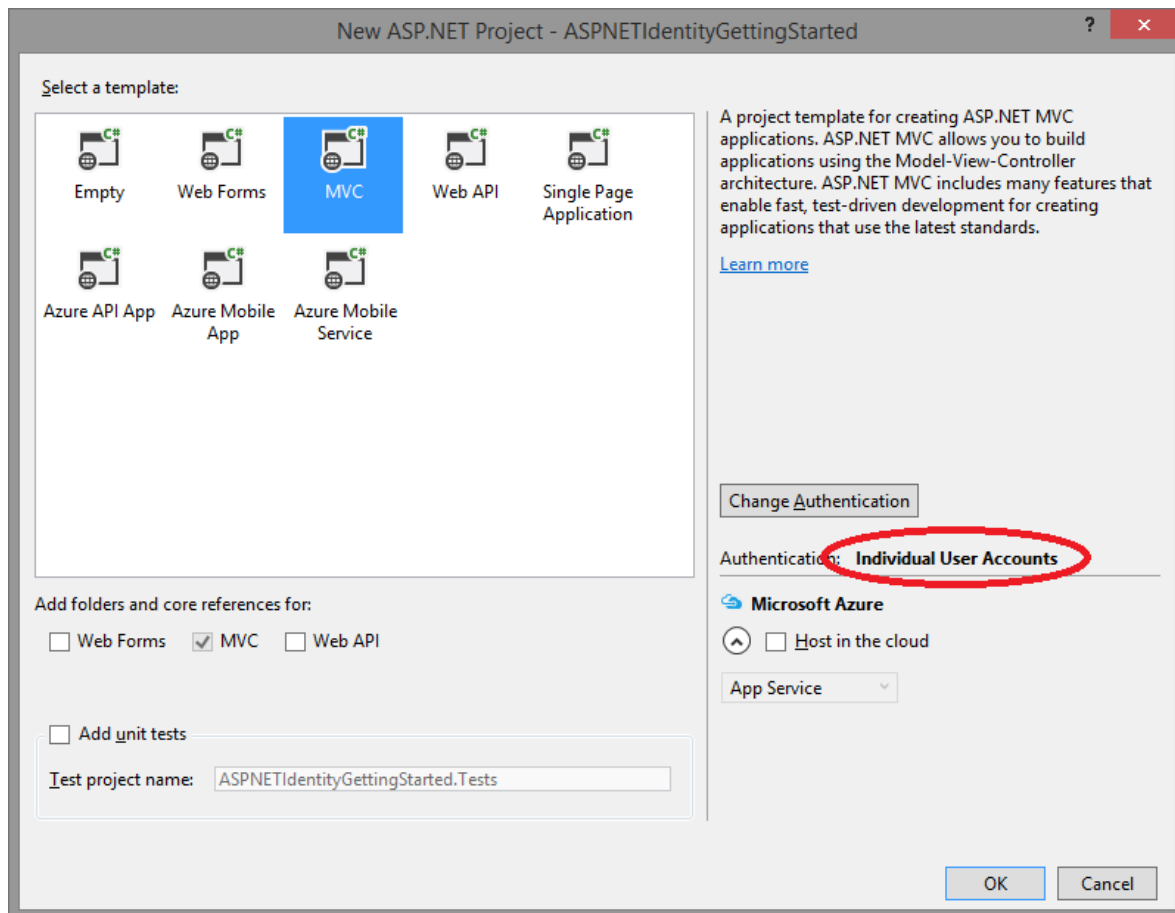
- One ASP.NET Identity System (ASP.NET MVC, Web Forms, Web Pages, Web API, and SignalR).
- Ease of plugging in profile data about the user.
- Persistence control (should be independent?).
- Unit testability.

- Role Provider.
- Claim based.
- Social Login Providers.
- Azure Active Directory.
- OWIN Integration.
- NuGet Package

Getting started with ASP.NET Identity

This getting started will show you the main ASP.NET Identity components and the standard implementation in the Visual Studio 2013 templates.

1. Create an ASP.NET MVC application with Individual Accounts:

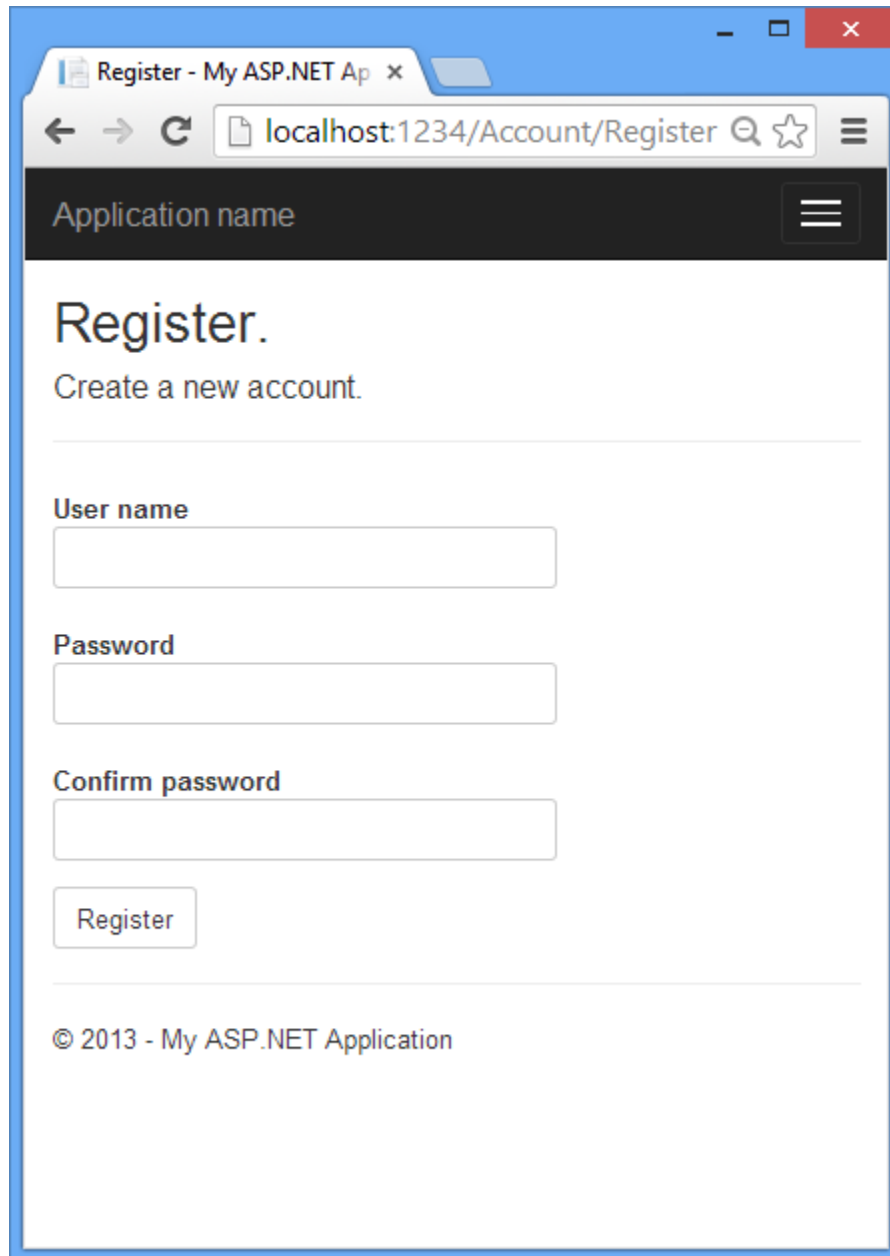


2. The created project contains the following three packages for ASP.NET Identity:

- `Microsoft.AspNet.Identity.Core`
This package has the core interfaces for ASP.NET Identity. This package can be used to write an implementation for ASP.NET Identity that targets different persistence stores as Azure Table Storage, NoSQL databases etc.

- `Microsoft.AspNet.Identity.EntityFramework`
This package has the Entity Framework implementation of ASP.NET Identity which will persist the ASP.NET Identity data and schema to SQL Server.
- `Microsoft.AspNet.Identity.OWIN`
This package contains functionality that is used to plug in OWIN authentication with ASP.NET Identity in ASP.NET Applications. This is used when you add log in functionality to your application and call into OWIN Cookie Authentication middleware to generate a cookie.

3. Creating a user. Launch the application and then click on **Register** link to create a user:



The screenshot shows a web browser window with the title "Register - My ASP.NET Ap". The address bar displays "localhost:1234/Account/Register". The page has a dark header with the text "Application name" and a hamburger menu icon. The main content area is titled "Register." and includes the subtitle "Create a new account." Below this, there are three input fields labeled "User name", "Password", and "Confirm password". A "Register" button is positioned below the "Confirm password" field. At the bottom of the page, the footer text reads "© 2013 - My ASP.NET Application".

When the user clicks the **Register** button, the Register action of the Account controller class creates the user by calling the ASP.NET Identity API, as highlighted below:

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model) {
    if (ModelState.IsValid) {
        var user = new ApplicationUser { UserName = model.UserName };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded) {
            await SignInAsync(user, isPersistent: false, rememberBrowser: false);
            return RedirectToAction("Index", "Home");
        }
        AddErrors(result);
    }
    return View(model);
}
```

4. Log in. If the user was successfully created, she is logged in by the SignInAsync method.

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model) {
    if (ModelState.IsValid) {
        var user = new ApplicationUser() { UserName = model.UserName };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded) {
            await SignInAsync(user, isPersistent: false, rememberBrowser: false);
            return RedirectToAction("Index", "Home");
        }
        AddErrors(result);
    }
    return View(model);
}
```

```
private async Task SignInAsync(ApplicationUser user, bool isPersistent) {
    AuthenticationManager.SignOut(DefaultAuthenticationTypes.ExternalCookie);
    var identity = await UserManager.CreateIdentityAsync(
        user, DefaultAuthenticationTypes.ApplicationCookie);
    AuthenticationManager.SignIn(
```

```

        new AuthenticationProperties() {
            IsPersistent = isPersistent
        }, identity);
    }

```

The highlighted code above in the `SignInAsync` method generates a [ClaimsIdentity](#). Since ASP.NET Identity and OWIN Cookie Authentication are claims-based system, the framework requires the app to generate a `ClaimsIdentity` for the user. `ClaimsIdentity` has information about all the claims for the user, such as what roles the user belongs to. You can also add more claims for the user at this stage.

The highlighted code below in the `SignInAsync` method signs in the user by using the `AuthenticationManager` from OWIN and calling `SignIn` and passing in the `ClaimsIdentity`.

```

private async Task SignInAsync(ApplicationUser user, bool isPersistent) {
    AuthenticationManager.SignOut(DefaultAuthenticationTypes.ExternalCookie);
    var identity = await UserManager.CreateIdentityAsync(
        user, DefaultAuthenticationTypes.ApplicationCookie);
    AuthenticationManager.SignIn(
        new AuthenticationProperties() {
            IsPersistent = isPersistent
        }, identity);
}

```

5. Log off. Clicking the **Log off** link calls the `LogOff` action in the account controller.

```

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff() {
    AuthenticationManager.SignOut();
    return RedirectToAction("Index", "Home");
}

```

The highlighted code above shows the OWIN `AuthenticationManager.SignOut` method. This is analogous to `FormsAuthentication.SignOut` method used by the `FormsAuthentication` module in Web Forms.

Components of ASP.NET Identity

The diagram below shows the components of the ASP.NET Identity system. The packages in blue make up the ASP.NET Identity system. All the other packages and dependencies which are needed to use the ASP.NET Identity system in ASP.NET applications.

