

Copyright © 2014 by Software Craftsmanship Guild.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the Software Craftsmanship Guild. For permission requests, write to the Software Craftsmanship Guild, addressed “Attention: Permissions Coordinator,” at the address below.

Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311



Security (ASP.NET Identity)

Software Craftsmanship Guild

Lesson Goals

Learn how to configure the membership api in ASP.NET to manage users and roles.

About Forms Authentication

Forms authentication in ASP .NET is when we keep a database of valid logins, passwords, and application roles such that we can lock down functionality and pages to specific users and roles.

A role is a fancy name for a group of users that we grant permission to perform certain actions. (ex: administrators have rights other users don't have)

How It Works

A user browses to your website. If a controller, page, or action is public, they can browse as normal, however, if they request something that we have tagged as requiring authorization, they will be redirected to a login page.

When the user logs in, the browser creates a time limited cookie with a security token inside it. On each request this cookie is sent back to the web server to ensure they are a valid user.

ASP.NET handles all of the token and cookie manipulation without our intervention.

Create a New Project

To get started:

1. Create a new ASP.NET web application project
2. Make sure that MVC is checked and that individual user accounts is set for the security in the right side bar.
3. Immediately run the site, navigate to the registration page, and register a new account.

Wait, what? I didn't create a database or anything?!

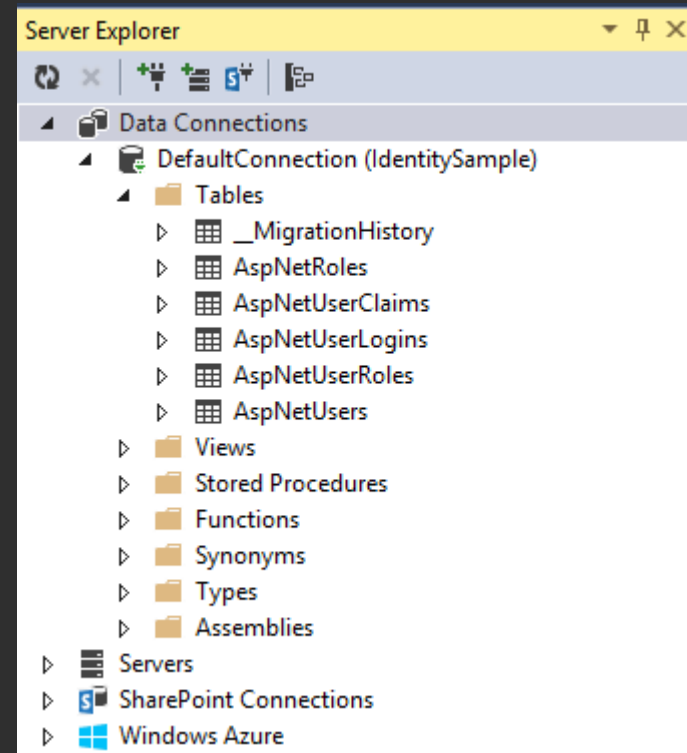
The default settings on the project template will automatically create a database and tables for your project in the local store.

The upside: This is convenient

The downside: In a real app we probably want to put it on a real SQL Server

Regardless, go to the view menu, open the server explorer, data connections, and you should see the default connection for the created database.

Check out the tables that were automatically created.



Entity Framework 6

The new ASP.NET Identity code uses Entity Framework 6.0 to interact with the database as of MVC5.

Open the models folder and you can see the IdentityModel.cs class file. Two classes exist in this file.

ApplicationDbContext is the EntityFramework piece that uses the DefaultConnection string from the web.config file.

ApplicationUser inherits from IdentityUser which has properties for id, user name, password hashes, etc.

If we want to add other information (name, email, etc), we can just add the properties to the ApplicationUser class.

```
using Microsoft.AspNet.Identity.EntityFramework;

namespace IdentitySample.Models
{
    // You can add profile data for the user by adding more properties to your class
    public class ApplicationUser : IdentityUser
    {
    }

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("DefaultConnection")
        {
        }
    }
}
```

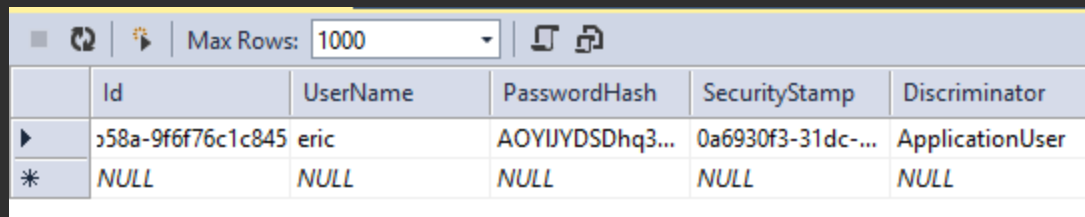
```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;
    AttachDbFilename=|DataDirectory|\aspnet-IdentitySample-20140205103403.mdf;
    Initial Catalog=aspnet-IdentitySample-20140205103403;
    Integrated Security=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```


Back to the Schema

The AspNetUsers table has all of the local logins. If you want people to register on your site and keep their password here, that is the table you will find the data.

Right click the table and show the data.

We also have an AspNetRoles table and a AspNetUserRoles table. Think of a role as a security group. We can create a role called “Administrators” and assign UserIds to that RoleId in the AspNetUserRoles table. This allows us to then lock access to pages and controller methods if we choose to.



■	🔄	🔍	Max Rows: 1000	📄	🔗
	Id	UserName	PasswordHash	SecurityStamp	Discriminator
▶	58a-9f6f76c1c845	eric	AOYIJYDSDhq3...	0a6930f3-31dc-...	ApplicationUser
*	NULL	NULL	NULL	NULL	NULL

Adding More User Information

Let's say we want to add a `FirstName` and a `LastName` to the user information.

First let's add the properties to the `ApplicationUser`.


Open the package manager console and `Enable-Migrations`. Then add migration profile to capture our changes.

Let's tweak the `Up()` method to constrain the size of the columns.

Lastly, update the database

```
public class ApplicationUser : IdentityUser
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

Package Manager Console

Package source:  Default project:

```
PM> Enable-Migrations
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolding an existing database. To use an automatic migration instead, specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project IdentitySample.
```

```
PM> add-migration profile
Scaffolding migration 'profile'.
The Designer Code for this migration file includes logic to calculate the changes to your model when you scaffold. If you want to include in this migration, then you
```

```
public override void Up()
{
    AddColumn("dbo.AspNetUsers", "FirstName", c => c.String(maxLength: 25));
    AddColumn("dbo.AspNetUsers", "LastName", c => c.String(maxLength: 25));
}
```

```
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied.
Applying explicit migrations: [201402060413342_profile].
Applying explicit migration: 201402060413342_profile.
Running Seed method.
```

Adding More User Information: Part 2

Now we need to update the code for the registration and account management pages to include a first name and last name.

Edit the AccountViewModel.cs file accordingly and the Register.cshtml and AccountController.cs.

Then register another user and check your database!

```
public class RegisterViewModel
{
    [Required]
    [Display(Name = "User name")]
    public string UserName { get; set; }

    [Required]
    [Display(Name = "First name")]
    public string FirstName { get; set; }

    [Required]
    [Display(Name = "Last name")]
    public string LastName { get; set; }
}
```

```
<div class="form-group">
    @Html.LabelFor(m => m.FirstName, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.FirstName, new { @class = "form-control" })
    </div>
</div>
<div class="form-group">
    @Html.LabelFor(m => m.LastName, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.LastName, new { @class = "form-control" })
    </div>
</div>
```

```
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser() {
            UserName = model.UserName,
            FirstName=model.FirstName,
            LastName = model.LastName
        };
    }
}
```

Let's make a page just for logged in users

The [Authorize] attribute when placed on a controller method will make that method only available to logged in users.

Let's create a view that just displays the current user information.

```
[Authorize]
public ActionResult ShowUserInformation()
{
    var userStore = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>());
    var user = userStore.FindById(User.Identity.GetUserId());

    return View(user);
}
```

```
@model IdentitySample.Models.ApplicationUser

@{
    ViewBag.Title = "ShowUserInformation";
}

<h2>ShowUserInformation</h2>

<div>
    <h4>ApplicationUser</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.FirstName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.FirstName)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.LastName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.LastName)
        </dd>
    </dl>
</div>
```

Adding Roles

Roles allow us to put users into groups and allow us to lock down features by roles.

Let's create a role called Admin, and put one of our users into that role.

Then create a page that is admin only. Now logged in users can only get to this page if they are a member of the Admin role!

	Id	Name
	1	Admin
▶*	NULL	NULL

	UserId	RoleId
	1cfd3b9c-1eb7-48e5-812c-caa4c4a706d0	1
▶*	NULL	NULL

```
[Authorize(Roles="Admin")]  
public ActionResult AdminOnlyPage()  
{  
    return View();  
}
```

Using Other Providers

The template site comes ready for other authorization providers. To set these up simply edit the Startup_Auth.cs file in the App_Start folder.

For example, to use google authentication for your app, just uncomment the line in the code to the right.

For more auth providers see this article:

<http://www.asp.net/mvc/tutorials/mvc-5/create-an-aspnet-mvc-5-app-with-facebook-and-google-oauth2-and-openid-sign-on>

```
public void ConfigureAuth(IAppBuilder app)
{
    // Enable the application to use a cookie to store information for the s
    app.UseCookieAuthentication(new CookieAuthenticationOptions
    {
        AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
        LoginPath = new PathString("/Account/Login")
    });
    // Use a cookie to temporarily store information about a user logging in
    app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

    // Uncomment the following lines to enable logging in with third party l
    //app.UseMicrosoftAccountAuthentication(
    //    clientId: "",
    //    clientSecret: "");

    //app.UseTwitterAuthentication(
    //    consumerKey: "",
    //    consumerSecret: "");

    //app.UseFacebookAuthentication(
    //    appId: "",
    //    appSecret: "");

    app.UseGoogleAuthentication();
}
```

Conclusion

It's generally better if possible to let external providers manage your accounts. Less we have to worry about.

Don't forget to use SSL/Encryption on the production web server!