ASP.NET Identity

Working with claims & roles in MVC 5

ASP.NET Identity

In this exercise we will create an application which can create and list claims in an MVC web application

ASP.NET Identity

We start off by creating a MVC project using the Identity template. (Requires MVC 5).

This means we will receive a project which has already been configured to handle ASP.NET Identity.

Now we want to create a page which will allow the user to **add claims** to its **identity**.

ASP.NET Identity

We proceed by creating a claim controller with an ”**Add**” action. The ”**Authorize**” attribute will make sure that we’re logged in to access the controller.

```
public class ClaimController : Controller
    {

     [Authorize]
     public async Task<ActionResult> Add()
        {
            return View();
        }
    }
```

We create a POST method for the **Add** action.

```
[HttpPost]
public async Task<ActionResult> Add(CustomClaimModel model)
{
            var userManager = new ApplicationUserManager(new
        UserStore<ApplicationUser>(new ApplicationDbContext()));

            var userId = User.Identity.GetUserId();

            await userManager.AddClaimAsync(userId, new
           Claim(model.Type, model.Value));

            return View();
}
```

ASP.NET Identity

So what does the action do?

The **ApplicationUserManager** is used to handle identities.

```
var userManager = new ApplicationUserManager(new
        UserStore<ApplicationUser>(new ApplicationDbContext()));
```

Here we add a claim to the currently logged in user using the **ApplicationUserManager**

```
var userId = User.Identity.GetUserId();

await userManager.AddClaimAsync(userId, new
Claim(model.Type, model.Value));
```

Before we create a view for the action we need to create a model. We will call this model "**CustomClaimModel**"

```
public class CustomClaimModel
    {
        public string Type { get; set; }

        public string Value { get; set; }
    }
```

ASP.NET Identity

The next step is to create a view for our ”Add” action. In this view we have to be able to enter a claim type and claim value.

```
@model ModuleExercise2.Models.CustomClaimModel
@{
    ViewBag.Title = "Add";
}

<h2>Add</h2>

@using (Html.BeginForm())
{

    @Html.LabelFor(m => m.Type)
    @Html.TextBoxFor(m => m.Type)

    @Html.LabelFor(m => m.Value)
    @Html.TextBoxFor(m => m.Value)

    <button type="submit">Create</button>
}
```

ASP.NET Identity

Now the user has the ability to add claims to its identity. Next up we are going to list all the users claims.

We are going to this by creating another action which we will call ”List”.

ASP.NET Identity

```
[Authorize]
        public async Task<ActionResult> List()
        {
              var userManager = new ApplicationUserManager(new
        UserStore<ApplicationUser>(new ApplicationDbContext()));
              var userId = User.Identity.GetUserId();

            var claims = await userManager.GetClaimsAsync(userId);

            if (claims == null)
            {
                return View("Add");
            }
            else
            {
                return View(claims);
            }
        }
```

Now we create a view for the ”List” action

```
@using System.Security.Claims
@model IEnumerable<Claim>
<h2>List</h2>
    <div class="panel panel-default">
        <div class="panel-heading">
            Claims
        </div>
        <table class="table table-striped">
            <tr>
                <th>Issuer</th>
                <th>Type</th>
                <th>Value</th>
            </tr>
            @foreach (Claim claim in Model)
            {
                <tr>
                    <td>@claim.OriginalIssuer</td>
                    <td>@claim.Type</td>
                    <td>@claim.Value</td>
                </tr>
            }
        </table>
    </div>
```

ASP.NET Identity

We can now look at our users claims.
It should look something like this

List

Claims

Issuer	Type	Value
LOCAL AUTHORITY	Role	Test
LOCAL AUTHORITY	Test	123456