CrowSoft C# Coding Standards

Contents

[ASP.Net Security & Roles 2](#_Toc6558229)

[MySQL Database Changes 2](#_Toc6558230)

[Code Changes on crowsoftmvc 2](#_Toc6558231)

[Changes to Startup.cs 2](#_Toc6558232)

[Limit Access To Controllers 3](#_Toc6558233)

[Administrator Tasks 3](#_Toc6558234)

[Update Users Model 3](#_Toc6558235)

[Changes to add new Menu 5](#_Toc6558236)

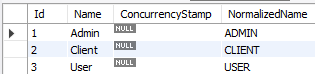
## ASP.Net Security & Roles

In this document, you will find how I added roles to the database, and how I link users to a specific role, and how I added a policy based authentication method. I will also explain in this document how to limit a user not have access to specific views, controllers or functions.

## MySQL Database Changes

The following database changes were required for the Policy Based authentication to work:

* Added two new columns to AspNetRoles table
  + ConcurrencyStamp : longtext
  + NormalizedName : varchar(256)
* The following records where added manually on the AspNetRoles MySQL table
  + Admin, Client and User



* For the security to work, I need to add a new table, called AspNetRoleClaims (Note: This table is not in use, but the application will not work due to the ASP.Net Identity manager searching for this table when adding users to roles in code. Table script is checked into the CrowSoftSQL Script sql file.)

## Code Changes on crowsoftmvc

There was several code changes and new classes and I will break it down in different sections.

### Changes to Startup.cs

To use the out of the box identity role components, it was necessary to add AddRoles<IdentityRole> to the this code, as seen highlited in brown below.

// This is to add the Identity components to the application, AddRoles adds the role components

services.AddDefaultIdentity<crowsoftmvcUser>()

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>();

The following code shows you how I added a build command below to AddMvc to add policy components to the application. This is to authorize a use role into a specific policy.

services.AddMvc(obj =>

{

// This code adds Policy components to the application, to be able to the Authorize users against specific policies

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

}).SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

Policies were added to control who can access the controllers:

* RequireAdminOnly = User with Admin Roles Only
* AllUsers = All user roles, Admin, Client and User
* RequireUserandAdminOnly = CrowSoft Users only, Admin and User

services.AddAuthorization(options =>

{

options.AddPolicy("RequireAdminOnly", policy =>

policy.RequireRole("Admin"));

options.AddPolicy("AllUsers", policy =>

policy.RequireRole("Admin", "Client", "User"));

options.AddPolicy("RequireUserandAdminOnly", policy =>

policy.RequireRole("Admin", "User"));

});

### Limit Access To Controllers

The following code example is needed on the top of the each controller class for the policy to be active on a specific controller. E.g. [Authorize(Policy = "RequireAdminOnly")]

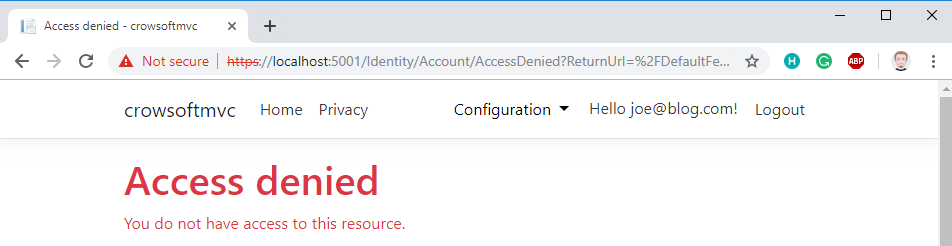
Example of the Authorize Controller:

[Authorize(Policy = "RequireAdminOnly")]

public class DefaultFeatureController : Controller

{

Below is what a user will see when they try to access this controller’s view:



## Administrator Tasks

We require the administrator to be able to view, edit and delete users. The following few headings will go in more detail how this was done.

### Update Users Model

To be able to create a Controller and View from the Identify model, called CrowsoftUser in our project, needed to override the columns to add data annotations for the views to reflect these annotations.

Here is the update to the CrowSoft Class:

// Add profile data for application users by adding properties to the CrowsoftUser class

public class CrowsoftUser : IdentityUser

{

[Key]

public override string Id { get => base.Id; set => base.Id = value; }

[Display(Name = "User Name")]

public override string UserName { get => base.UserName; set => base.UserName = value; }

[HiddenInput(DisplayValue = false)]

public override string NormalizedUserName { get => base.NormalizedUserName; set => base.NormalizedUserName = value; }

[Display(Name = "Email Address")]

[EmailAddress(ErrorMessage = "Invalid Email Address")]

[Required(ErrorMessage = "{0} is required.")]

public override string Email { get => base.Email; set => base.Email = value; }

[HiddenInput(DisplayValue = false)]

public override string NormalizedEmail { get => base.NormalizedEmail; set => base.NormalizedEmail = value; }

[Display(Name = "Confirm Email Address")]

[EmailAddress(ErrorMessage = "Invalid Email Address")]

[Required(ErrorMessage = "{0} is required.")]

public override bool EmailConfirmed { get => base.EmailConfirmed; set => base.EmailConfirmed = value; }

[HiddenInput(DisplayValue = false)]

public override string PasswordHash { get => base.PasswordHash; set => base.PasswordHash = value; }

[HiddenInput(DisplayValue = false)]

public override string SecurityStamp { get => base.SecurityStamp; set => base.SecurityStamp = value; }

[HiddenInput(DisplayValue = false)]

public override string ConcurrencyStamp { get => base.ConcurrencyStamp; set => base.ConcurrencyStamp = value; }

[Display(Name = "Phone Number")]

public override string PhoneNumber { get => base.PhoneNumber; set => base.PhoneNumber = value; }

[HiddenInput(DisplayValue = false)]

public override bool PhoneNumberConfirmed { get => base.PhoneNumberConfirmed; set => base.PhoneNumberConfirmed = value; }

[HiddenInput(DisplayValue = false)]

public override bool TwoFactorEnabled { get => base.TwoFactorEnabled; set => base.TwoFactorEnabled = value; }

[HiddenInput(DisplayValue = false)]

public override DateTimeOffset? LockoutEnd { get => base.LockoutEnd; set => base.LockoutEnd = value; }

[HiddenInput(DisplayValue = false)]

public override bool LockoutEnabled { get => base.LockoutEnabled; set => base.LockoutEnabled = value; }

[HiddenInput(DisplayValue = false)]

public override int AccessFailedCount { get => base.AccessFailedCount; set => base.AccessFailedCount = value; }

}

The following command was executed in the terminal to auto-genrate the Controllers and Views:

dotnet aspnet-codegenerator --project . controller -name CrowsoftUserController -m CrowsoftUser -dc ApplicationDbContext --useDefaultLayout --referenceScriptLibraries

### Changes to add new Menu

I made the following changes in the \_Layout.cshtml to add a menu for Configuration::

<div class="dropdown">

<button class="btn btn-primary dropdown-toggle" data-toggle="dropdown"

style="background-color:white;color:black;border:none !important;outline:none !important">

Configuration

</button>

<div class="dropdown-menu">

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="DefaultFeature" asp-action="Index">Default Features</a>

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-controller="CrowsoftUser" asp-action="Index">Manage Users</a>

</li>

<li class="nav-item">

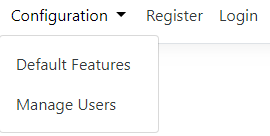
<a class="dropdown-item" href="#">Link 3</a>

</li>

</div>

</div>

This is how the menu looks like:



Below find the view to Manage Users:

