Role Management in ASP.NET Core

In this tutorial, we will explore one of the ways to implement role authentication and role management in ASP.NET Core.

This tutorial will follow the ASP.NET Core Web Application project created with Inclass4\_WAS. Please create the project files along with the SQL Database connections & tables as stated.

# Add Roles

## Creating a RoleController class

Create a new Empty Controller and name it RoleController that extends the Controller class. We will need to use RoleManager to create a new role, so we need to inject RoleManager dependency into the controller’s constructor. We will need to create two IActionResult methods so we can collect an input from the user and create a role based on their input through a POST method.

|  |
| --- |
| public class RoleController : Controller  {  private RoleManager<IdentityRole> \_roleManager;  public RoleController (RoleManager<IdentityRole> roleManager)  {  \_roleManager = roleManager;  }  [HttpGet]  public IActionResult Index()  {  return View();  }  [HttpPost]  public IActionResult Index(RoleVM roleVM)  {  var newRole = new IdentityRole();  newRole.Id = roleVM.RoleName;  newRole.Name = roleVM.RoleName;  newRole.NormalizedName = roleVM.RoleName;  if (!\_roleManager.RoleExistsAsync(newRole.Name).Result)  {  IdentityResult result = \_roleManager.CreateAsync(newRole).Result;  if (!result.Succeeded)  {  ViewBag.Message = "Error, failed at adding";  } else  {  ViewBag.Message = "Role successfully added.";  }  }  return View();  }  } |

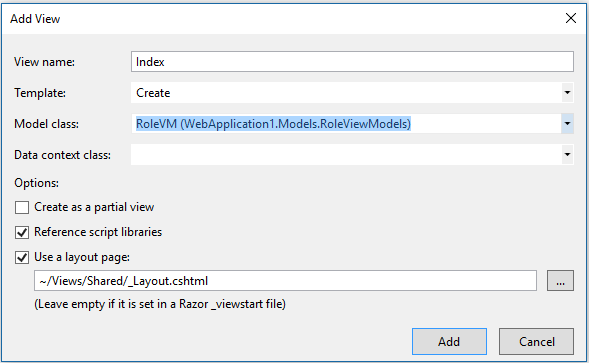
## Create a RoleVM Model

Next, we will need to create a RoleVM model that will be passed to our Post method when we create the role. Create this View Model in ~/Models/RoleViewModels/

|  |
| --- |
| public class RoleVM  {  [Required]  [Display(Name = "User Role")]  public string RoleName { get; set; }  } |

## Add Role Creation View

Create a new View in the ~/Views/Role/ folder. The view should Create a RoleVM model without a reference to a DbContext.



## Add Navigation Links to Role Management

Add this line of code into the nav element in your ~/Views/Shared/\_Layout.cshtml. This will enable us to navigate to our Add Role function easily.

|  |
| --- |
| <li><**a** **asp-area**="" **asp-controller**="Role" **asp-action**="Index">Add Role</**a**></li> |

At this point, we should be able to create a new role if there are no conflicting data that already exists. Try and create a new Role “Admin”.

# Add Users to Roles

## Add UserRoleVM

Create a new view model named UserRoleVM. It will take 2 properties: UserName and RoleName. These properties will be used to assign Role to User.

|  |
| --- |
| public class UserRoleVM  {  [Required]  [Display(Name = "Login Name")]  public string UserName { get; set; }  [Required]  [Display(Name = "Role Name")]  public string RoleName { get; set; }  } |

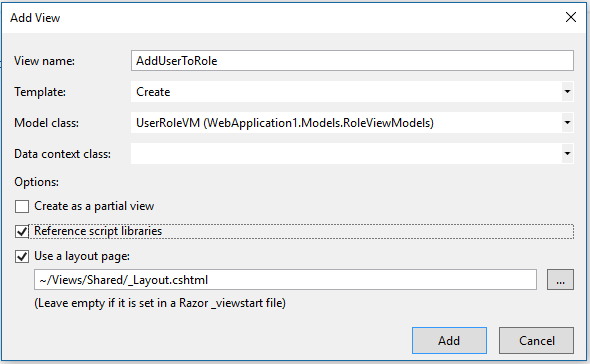
## Integrate AddUserToRole ActionResult Methods

Add the following code to the bottom of your RoleController. The following code will allow us to create a UserRoleVM with our form. From that view model, we can then query the database to make sure that these are valid users and roles. Then we can attempt to link the two.

|  |
| --- |
| [HttpGet]  public IActionResult AddUserToRole()  {  return View();  }  [HttpPost]  public IActionResult AddUserToRole(UserRoleVM userRoleVM)  {  if (ModelState.IsValid) {  using (\_context)  try  {  var user = \_context.Users.Where(u => u.UserName == userRoleVM.UserName).FirstOrDefault();  var role = \_context.Roles.Where(r => r.Name == userRoleVM.RoleName).FirstOrDefault();  var userRole = new IdentityUserRole<string>();  userRole.RoleId = role.Id;  userRole.UserId = user.Id;  \_context.UserRoles.Add(userRole);  \_context.SaveChanges();  ViewBag.Message = "User Successfully Added to Role";  } catch (Exception ex)  {  Console.WriteLine(ex.Message);  ViewBag.Message = "Something went wrong. Please try again.";  }  }  return View();  } |

## Create AddUserToRole View

Next we will need to create a strongly typed view that Creates a UserRoleVM in ~/Views/Role/



## Add Navigation Links to AddUserToRole

Add this line of code with our other line in the nav element located in ~/Views/Shared/\_Layout.cshtml. This will enable us to navigate to our AddUserToRole function easily.

|  |
| --- |
| <li><**a** **asp-area**="" **asp-controller**="Role" **asp-action**="AddUserToRole">Assign role</**a**></li> |

At this point, we should be able to assign a user to a role. We can try to assign ourselves as our recently created “Admin” role.

# Role Based Authentication

## Add Authentication Checks In Controller

Make sure only Admins can access the functions in the RoleController. Add this line before the class declaration of the RoleController. This will provide server side restriction to the contoller for Admin roles only.

|  |
| --- |
| [Authorize(Roles = "Admin")]  public class RoleController : Controller |

## Add Navigation Links to Role Management

Replace our links with one that checks the Roles. This will make it so that Role Management links are only visible to Admin level users.

|  |
| --- |
| @{if(User.IsInRole("Admin"))  {  <li><**a** **asp-area**="" **asp-controller**="Role" **asp-action**="Index">Add Role</**a**></li>  <li><**a** **asp-area**="" **asp-controller**="Role" **asp-action**="AddUserToRole">Assign Role</**a**></li>  }} |

Now if we run the app, our Admin account will have access to our Role Management Tools. However, if we register a new account with no Admin access, we will not be able to see these.