# Tuesday, September 9, 2014

Remember what MVC stands for. We typically create the elements of our program in this order. So, if you haven’t done it already, the first thing we need to do is build the Models that will drive our Views.

## Build the Models

1. Right click on the Models folder in Solution Explorer and choose Add > Code File and name the file RolesViewModels.cs.
2. You will need three “using” statement at the top of the file:
   1. using System.Collections.Generic;
   2. using System.ComponentModel.DataAnnotations;
   3. using Microsoft.AspNet.Identity.EntityFramework; // needed for IdentityRole
3. Make sure the namespace indicated in the file is the correct namespace for your models
4. Add two new public classes:
   1. RolesViewModel
   2. UserRolesViewModel
5. In the first of these, create two string attributes:
   1. RoleId
   2. RoleName (include a display property for the role name)
6. In the second model create four attributes – the first two are identical to those in the previous model, the second two are as follows (include a display attribute for Users):
   1. public System.Web.Mvc.MultiSelectList Users { get; set; }
   2. public string[] SelectedUsers { get; set; }
7. Save and close the file

## Add Helper Methods to the ApplicationUser class

1. Open IdentityModels.cs
2. Add an instance of the UserManager to the ApplicationUser class as follows:

private UserManager<ApplicationUser> usermanager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(new ApplicationDbContext()));

1. Add the following methods:

public bool IsInRole(string rolename)

{

var result = usermanager.IsInRole(this.Id, rolename);

return result;

}

public bool AddUserToRole(string rolename)

{

var result = usermanager.AddToRole(this.Id, rolename);

return result.Succeeded;

}

public bool RemoveUserFromRole(string rolename)

{

var result = usermanager.RemoveFromRole(this.Id, rolename);

return result.Succeeded;

}

## Build the Views

1. Add two new folders to the Views folder: Roles and UserRoles
2. Add four new views to the Roles folder (all use the RolesViewModel and do NOT have a database context):
   1. Create (using the Create template)
   2. Edit (using the Edit template)
   3. Delete (using the Delete template)
   4. Index (using the Index template)
3. Modify each view to make them look cleaner and more user friendly (get rid of unnecessary information, change headings, etc). The Delete view should be changed so that it has the same look and feel of the Create and Edit views. The goal is to make all the views look consistent.
4. Add two new action links to the Index view (right above the links to Edit and Delete). Also, uncomment the id specification in each of those links and set the id equal to the role id.
   1. @Html.ActionLink("Assign users", "AssignUsers", "UserRoles", new { id = item.RoleId }, null) |
   2. @Html.ActionLink("View / Unassign users", "UsersInRole", "UserRoles", new { id = item.RoleId }, null) |
   3. These views don’t exist yet, but they will shortly. In fact, let’s create them now.
5. Add these views (AssignUsers and UsersInRole) to the UserRoles view folder. Use the UserRolesViewModel and the Create template.
6. Open the newly created AssignUsers view
   1. Make any cosmetic changes to the beginning of the file (headings, etc)
   2. Modify the BeginForm statement as follows:
      1. @using (Html.BeginForm("AssignUsers", "UserRoles", FormMethod.Post, new { encType = "multipart/form-data", name = "myform" }))
   3. Add HiddenFor elements for both the role id and role name included in the model
   4. Move the Form Group associated with the role id below the one for the role name, and then change it to refer to model.Users instead of role id.
   5. Change the EditorFor statement to the following:
      1. @Html.ListBoxFor(model => model.SelectedUsers, Model.Users as MultiSelectList, new { htmlAttributes = new { @class = "form-control" } })
   6. Change the text on the submit button to “Add User(s)”
   7. Change the Back to List action link to refer to the Index view in the Roles controller (add a third parameter with the name of the controller – “Roles”)
   8. Save this view
7. Open the newly created UsersInRole view
   1. Make the same changes to this view as the previous, with the following exceptions:
      1. Display the role name in heading 2 and remove the Form Group associated with the role name
      2. Make sure you name this form (in the BeginForm statement) the same as the view name
      3. Change the text on the submit button to “Unassign user(s)”
   2. Save and close the views

## Build the Controllers

### RolesController

1. Right click on the Controllers folder and choose Add > Controller > MVC 5 Controller with read/write actions
2. Name the new controller RolesController
3. Add the following using statements to the top of the file
   1. using Microsoft.AspNet.Identity;
   2. using Microsoft.AspNet.Identity.EntityFramework;
   3. using System.Data.Entity;
   4. using BugTracker.Models; (or whatever your namespace is)
4. Delete the Details controller methods
5. Delete the contents of all other methods except the Create GET controller. We leave it alone.
6. Add the [ValidateAntiForgeryToken] attribute to all of the POST controller methods in the file
7. Build the controller according to the code comments on the attached page.

### UserRolesController

1. Add an EMPTY MVC 5 controller named UserRolesController
2. Add using statements for Microsoft.AspNet.Identity and your Models namespace (BugTracker.Models)
3. Create GET and POST method definitions for your two views: AssignUsers and UsersInRole
4. Add the [ValidateAntiForgeryToken] attribute to the two POST controller methods
5. Build the controller according to the code comments on the attached page.

Build and test your project

## RolesController.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using System.Data.Entity;

using BugSplat.Models;

namespace BugSplat.Controllers

{

public class RolesController : Controller

{

// create a private member variable for this controller class (you will need  
 // this throughout the controllers here) that connects to the AspNet tables   
 // (this is represented by the ApplicationDbContext object)

// GET: Roles

public ActionResult Index()

{

// 1) get a list of all roles in the DB

// 2) instantiate the view model as a list of RolesViewModels

// 3) loop through all the roles in the DB and add a new RolesViewModel  
 // object for each one

// send the model to the view

}

// GET: Roles/Create

public ActionResult Create()

{

return View();

}

// POST: Roles/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(RolesViewModel model)

{

// 1) if the model state is valid, continue

// 2) add a new IdentityRole object with the model’s role name to the

// Roles table in the DB - store the return value in a var named result

// 3) if the value of result is not null, save the changes and return

// control to the list view (Index)

// 4) if we got this far, something went wrong. Return the model to the view.

}

// GET: Roles/Edit/5

public ActionResult Edit(string id)

{

// 1) locate the role and get it

// 2) build the view model

// 3) send the model to the view

}

// POST: Roles/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(RolesViewModel model)

{

// 1) locate the role and get it

// 2) change the role name to match what's in the model

// 3) tell the DB the role entry has been modified

// 4) save the changes

// 5) return control to the list view (Index)

}

// GET: Roles/Delete/5

public ActionResult Delete(string id)

{

// 1) locate the role and get it

// 2) build the view model

// 3) send the model to the view

}

// POST: Roles/Delete/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Delete(RolesViewModel model)

{

// 1) locate the role and get it

// 2) remove the role from the DB

// 3) save your changes

// 4) redirect control back to the roles list view

}

}

}

## UserRolesController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using BugSplat.Models;

namespace BugSplat.Controllers

{

public class UserRolesController : Controller

{

// create two private member variables for this controller class (you will need   
 // these throughout the controllers here):

// 1) a connection to the AspNet tables (this is represented by the   
 // ApplicationDbContext object)

// 2) a connection to your BugTracker tables (represented by the   
 // BugTrackerEntities object, unless you called it something different)

// GET: AssignUsers

public ActionResult AssignUsers(string roleId)

{

// 1) locate the role in the DB (you have the id of the role right here)

// 2) instantiate the view model

// 3) add the role ID and Name to the model

// 4) instantiate the user list that is part of the view model

// 5) loop through all of the system users (in the AspNetUsers table) and, as  
 // long as the user is NOT already in the role, add the corresponding   
 // BTUser to the list (you'll need to find the user in the BTUsers   
 // table)

// 6) instantiate the MultiSelectList object (in the model) using the newly   
 // built list with appropriate value and display parameters

// 7) send the model to the view

}

// POST: AssignUsers

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult AssignUsers(UserRolesViewModel model)

{

// check the model state - if it's valid, continue

// check the SelectedUsers attribute of the model - if it's NOT null,  
 // continue

// loop through the elements in model.SelectedUsers (they should be   
 // AspNetUserId's) - for each one in the array,

// 1) locate the user in the AspNetUser table,

// 2) add the user to the role specified in the model

// 3) if the user is already in the "Unassigned" role, remove   
 // him/her from that role

// redirect to the roles list

// if we got here, there's a problem - return the view with the model

}

// GET: UsersInRole

public ActionResult UserInRole(string roleId)

{

// 1) locate the role in the DB (you have the id of the role right here)

// 2) instantiate the view model

// 3) add the role ID and Name to the model

// 4) instantiate the user list that is part of the view model

// 5) loop through all of the system users (in the AspNetUsers table), and if   
 // the user is in the role, add the corresponding BTUser to the list   
 // (you'll need to find the user in the BTUsers table)

// 6) instantiate the MultiSelectList object (in the model) using the newly   
 // built list with appropriate value and display parameters

// 7) send the model to the view

}

// POST: UsersInRole

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult UsersInRole(UserRolesViewModel model)

{

// check the model state - if it's valid, continue

// check the SelectedUsers attribute of the model - if it's NOT null,   
 // continue

// loop through the elements in model.SelectedUsers (they should be   
 // AspNetUserId's) - for each one in the array,

// 1) locate the user in the AspNetUser table,

// 2) remove the user from the role specified in the model

// 3) check the number of roles the user is assigned to   
 // (user.Roles.Count) - if it is zero, add the user to the   
 "Unassigned" role

// return to the current view, passing the role id (this calls the view's   
 // GET controller)

// if we got here, there's a problem - return the view with the model

}

}

}