

7.4.0 – Adapting To Other Systems

Introduction

In previous lessons we setup security for our web site using supplied code files. The question is now, what modifications/changes need to be made to add security to a different system? In this course you have your exercise, and project, database systems. Simply copying the code files from eStore to these other systems will not work unless we know what changes need to be made.

IdentityModels.cs

In this code file we added the following code:

Additional Namespaces

In this region we have the following:

```
#region Additional Namespaces
using eStoreWeb.Security;
using System.Data.Entity;
#endregion
```

The eStoreWeb.Security refers to the same namespace in the **SecurityController.cs** file:

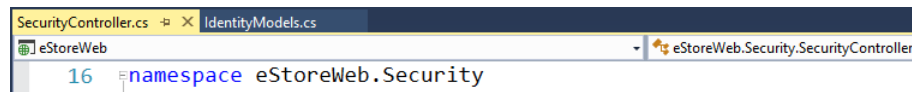


Figure 1: SecurityController Namespace

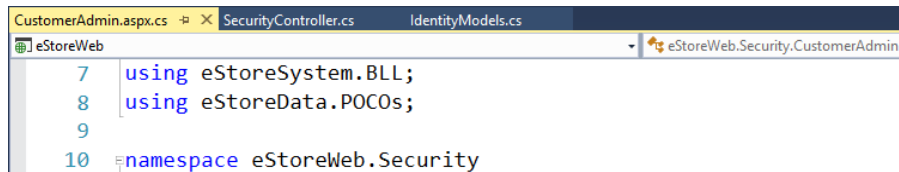
These namespaces **MUST** match, or your code will not compile (i.e. you will have errors). The most important aspect of the namespace name is that the first part refers to the name of the database, or the system, the **SecurityController** belongs to. If we were coding a system for a database called GroceryStore, then the namespace for the **SecurityController** should be like:

```
namespace GroceryStoreWeb.Security
```

Therefore, the **Additional Namespaces** would need to be:

```
#region Additional Namespaces
using GroceryStoreWeb.Security;
using System.Data.Entity;
#endregion
```

This same namespace is found in the security web forms, for our example, eStore, we have the following:

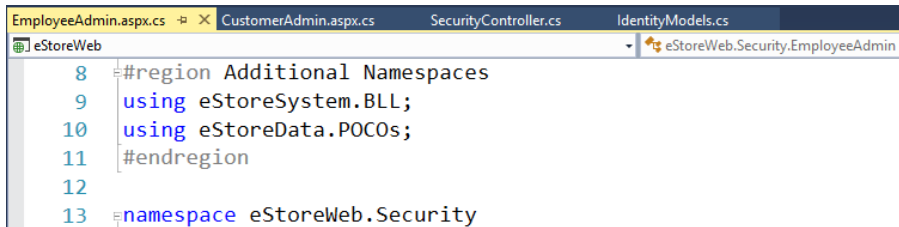


```

7 using eStoreSystem.BLL;
8 using eStoreData.POCOs;
9
10 namespace eStoreWeb.Security

```

Figure 2: CustomerAdmin.aspx.cs Namespace



```

8 #region Additional Namespaces
9 using eStoreSystem.BLL;
10 using eStoreData.POCOs;
11 #endregion
12
13 namespace eStoreWeb.Security

```

Figure 3: EmployeeAdmin.aspx.cs Namespace

These namespaces were added automatically when we created the web forms in the **Security** folder of the web site. By keeping all the namespaces named correctly, the compiler can find all the code, and without any other errors, the web application will run correctly.

Custom Properties

```

#region Custom Properties
//these properties will become attributes on the AspNetUsers Sql Table
//once a numeric datatype is made nullable the default for that data is null.
public int? EmployeeId { get; set; }
public int? CustomerId { get; set; }
#endregion

```

We were able to do this as we had a **Customer** table that had a primary key of **CustomerID**, and we had an **Employee** table with a primary key of **EmployeeID**. Most databases that are used for business systems will have similar tables, with similar primary keys. Thus, we need to verify the names of the primary keys of the users we want to be adding to our **AspNetUsers** table and make any appropriate changes, or additions to the Custom Properties.

SecurityDbContextInitializer.cs

In this file we have the following code:

```

#region Seed the users
//add WebMaster
string adminUser = ConfigurationManager.AppSettings["adminUserName"];
string adminRole = ConfigurationManager.AppSettings["adminRole"];
string adminEmail = ConfigurationManager.AppSettings["adminEmail"];
string adminPassword = ConfigurationManager.AppSettings["adminPassword"];
string adminPhone = ConfigurationManager.AppSettings["adminPhone"];
var userManager = new ApplicationUserManager(new
UserStore<ApplicationUser>(context));
var result = userManager.Create(new ApplicationUser
{
    UserName = adminUser,
    Email = adminEmail,
    PhoneNumber = adminPhone
}

```

```

    }, adminPassword);
    if (result.Succeeded)
        userManager.AddToRole(userManager.FindByName(adminUser).Id, adminRole);
    //Add a customer
    string customerUser = ConfigurationManager.AppSettings["customerUserName"];
    string customerRole = ConfigurationManager.AppSettings["customerRole"];
    string customerEmail = ConfigurationManager.AppSettings["customerEmail"];
    string customerPassword = ConfigurationManager.AppSettings["customerPassword"];
    string customerPhone = ConfigurationManager.AppSettings["customerPhone"];
    result = userManager.Create(new ApplicationUser
    {
        UserName = customerUser,
        Email = customerEmail,
        PhoneNumber = customerPhone
    }, customerPassword);
    if (result.Succeeded)
        userManager.AddToRole(userManager.FindByName(customerUser).Id, customerRole);
#endregion

```

Each of the AppSettings relies on the following which we have in the **Web.config** file:

```

<appSettings >
  <add key="startupRoles" value="Administrators;Employees;Customers;RegisteredUsers" />

  <add key="adminUserName" value="Webmaster" />
  <add key="adminEmail" value="Webmaster@eStore.tba" />
  <add key="adminPassword" value="Pa$$w0rd1" />
  <add key="adminPhone" value="8885551212"/>

  <add key="customerUserName" value="HansenB" />
  <add key="customerEmail" value="HansenB@hotmail.com" />
  <add key="customerPassword" value="Pa$$w0rd1" />
  <add key="customerPhone" value="7805551212"/>

  <add key="adminRole" value="Administrators" />
  <add key="customerRole" value="Customers" />
  <add key="employeeRole" value="Employees" />

  <add key="newUserPassword" value="Pa$$word1" />
</appSettings>

```

Notice the key values here are the same as in the **SecurityContextInitializer.cs** file; these 2 code files work together to set up values in the security tables in the database. What is important here is that we must customize our AppSettings in both files. It is also very important that we ensure that database tables (i.e. the original tables, and not the security tables) have appropriate properties to match what we see in the **SecurityDbContextInitializer.cs** file, such as the code below:

```

result = userManager.Create(new ApplicationUser
{
    UserName = customerUser,
    Email = customerEmail,
    PhoneNumber = customerPhone
}, customerPassword);

```

Email and PhoneNumber fields must exist in the Customer table!

Yes, we can add even more AppSettings in the **Web.config** file and add those to the **SecurityDbContextInitializer.cs** code file. Additionally, you can change the names, and the default passwords in the `<appSettings>`.

SecurityController.cs

In this code file we have the following:

```
[DataObjectMethod(DataObjectMethodType.Select)]
public List<ApplicationUser> ListUsers()
{
    return UserManager.Users.Where(x => x.EmployeeId.HasValue).OrderBy(x =>
x.UserName).ToList();
}

[DataObjectMethod(DataObjectMethodType.Select)]
public List<ApplicationUser> ListCustomers()
{
    return UserManager.Users.Where(x => x.CustomerId.HasValue).OrderBy(x =>
x.UserName).ToList();
} //eom
```

This section of code uses the **Custom Properties** found in the **IdentityModels.cs** code file (see that section in this document). If those properties are different, then this section of code must be modified to match the properties.

Exercise

Following the steps in the previous lessons of this unit, add security to your exercise solution. Make sure you add web page security to the web forms in the **Security** and **Admin** folders.