**MVC 5 Membership Website**

Contents

[Resources placed in folder 5](#_Toc39449188)

[Created the project with 5](#_Toc39449189)

[App\_Start Folder 5](#_Toc39449190)

[BundleConfig.cs 5](#_Toc39449191)

[V1.0 Initial File 5](#_Toc39449192)

[V1.1 Adding Sitejs Bundle 6](#_Toc39449193)

[Content Folder 6](#_Toc39449194)

[Helper Extensions 6](#_Toc39449195)

[Reflection Extensions 6](#_Toc39449196)

[IsPropertyExists Extension 7](#_Toc39449197)

[GetPropertyValue Extension 7](#_Toc39449198)

[ICollection Extensions 7](#_Toc39449199)

[ToSelectListItem Extension 8](#_Toc39449200)

[Convert Extension Method 8](#_Toc39449201)

[Adding the Admin Menu 8](#_Toc39449202)

[\_Layout.cshtml 8](#_Toc39449203)

[V1.0 Initial File 9](#_Toc39449204)

[V1.1 Adding Admin Menu 9](#_Toc39449205)

[V1.2 Adding SiteJs bundle 10](#_Toc39449206)

[\_SiteAdminMenuPartial.cshtml 10](#_Toc39449207)

[V1.0 Admin Menu Initial 10](#_Toc39449208)

[V1.1 Admin Menu – Restricted Access 11](#_Toc39449209)

[Opening and Closing the Menu 12](#_Toc39449210)

[SiteAdminMenu.js 12](#_Toc39449211)

[Creating the Database 12](#_Toc39449212)

[Database Name Web.Config 12](#_Toc39449213)

[Package Manager Console 12](#_Toc39449214)

[Command enable-migrations 13](#_Toc39449215)

[Command update-database [Create Database] 13](#_Toc39449216)

[Adding Table/Entities 14](#_Toc39449217)

[Item Related Tables/Entities 14](#_Toc39449218)

[Product Related Tables/Entities 17](#_Toc39449219)

[Subscription Table/Entity 18](#_Toc39449220)

[ProductItem Table/Entity 19](#_Toc39449221)

[SubscriptionProduct Table/Entity 19](#_Toc39449222)

[UserSubscription Table/Entity 20](#_Toc39449223)

[Code First Approach 21](#_Toc39449224)

[Update Database Command 21](#_Toc39449225)

[Modifying AspNetUser Table 22](#_Toc39449226)

[User Handling – Registeration/Login/Roles 24](#_Toc39449227)

[Altering the Register User Actions/View/Models 24](#_Toc39449228)

[AspNetUsers 24](#_Toc39449229)

[Altering RegisterViewModel 24](#_Toc39449230)

[Altering the RegisterAction 24](#_Toc39449231)

[Altering the Registration View 25](#_Toc39449232)

[Registering the User 25](#_Toc39449233)

[Assigning the Admin Role 26](#_Toc39449234)

[User Id Pick Up 26](#_Toc39449235)

[Add Role 26](#_Toc39449236)

[Add User to Role 26](#_Toc39449237)

[Restrict Access to Admin Menu and Controllers 26](#_Toc39449238)

[Restricting Admin Menu 26](#_Toc39449239)

[Restricting Controller 27](#_Toc39449240)

[Create a General User 27](#_Toc39449241)

[Test Restricted Access 27](#_Toc39449242)

[Areas 27](#_Toc39449243)

[Add Admin Area 27](#_Toc39449244)

[Copying the \_Layout to the Areas Shared Folder 28](#_Toc39449245)

[V 1.0 Initial 28](#_Toc39449246)

[Creating Buttons – will be used on all views 29](#_Toc39449247)

[CreateButton Partial View 29](#_Toc39449248)

[EditButton Partial View 29](#_Toc39449249)

[EditButtonDetail Partial View 29](#_Toc39449250)

[Model 29](#_Toc39449251)

[Partial View 30](#_Toc39449252)

[BackToListButton Partial View 30](#_Toc39449253)

[SmallButton Partial View 31](#_Toc39449254)

[Model 31](#_Toc39449255)

[Partial View 31](#_Toc39449256)

[TableButton Partial 32](#_Toc39449257)

[CSS 33](#_Toc39449258)

[BundleConfig 33](#_Toc39449259)

[Section Entity : Creating Admin User Interface 33](#_Toc39449260)

[Scaffolding the Section Entity 33](#_Toc39449261)

[SectionController 34](#_Toc39449262)

[Authorize Attribute 34](#_Toc39449263)

[Index View 34](#_Toc39449264)

[Edit View 35](#_Toc39449265)

[Details View 36](#_Toc39449266)

[Delete View 37](#_Toc39449267)

[Update \_SiteAdminMenuPartial Section Link using @UrlAction 37](#_Toc39449268)

[Playing with Data 37](#_Toc39449269)

[Adding Controller – Actions - Views for Part / ItemType / ProductType / ProductLinkText Entities 38](#_Toc39449270)

[Authorize Attribute 38](#_Toc39449271)

[Item Entity : Scaffolding 38](#_Toc39449272)

[Controllers - Actions - Views 38](#_Toc39449273)

[Authorize Attribute 38](#_Toc39449274)

[Create Updates 39](#_Toc39449275)

[Controller Action – Create HTTP GET 39](#_Toc39449276)

[Controller Action – Create HTTP POST 39](#_Toc39449277)

[View 39](#_Toc39449278)

[Edit Updates 41](#_Toc39449279)

[Product Entity 41](#_Toc39449280)

[Creating ProductModel 41](#_Toc39449281)

[Conversion Extensions 42](#_Toc39449282)

[Scaffolding Controller/Action for the Product Entity 43](#_Toc39449283)

[Authorize Attribute 44](#_Toc39449284)

[Updating the SiteAdminMenuPartial 44](#_Toc39449285)

[Adding views 44](#_Toc39449286)

[Alter Index Action & View 45](#_Toc39449287)

[Action 45](#_Toc39449288)

[View 45](#_Toc39449289)

[Alter Create Action & View 46](#_Toc39449290)

[Action Create Get 46](#_Toc39449291)

[Action Create Post 47](#_Toc39449292)

[View Create 47](#_Toc39449293)

[Alter Edit Action & View 49](#_Toc39449294)

[Action Edit Get 49](#_Toc39449295)

[Action Edit Post 49](#_Toc39449296)

[View Edit 50](#_Toc39449297)

[Alter Details Action & View 51](#_Toc39449298)

[Action Detail 51](#_Toc39449299)

[View 51](#_Toc39449300)

[Alter Delete Action & View 53](#_Toc39449301)

[Add reference to System.Transactions 53](#_Toc39449302)

[Action Delete Get 53](#_Toc39449303)

[Action Delete Http Post 53](#_Toc39449304)

[View 54](#_Toc39449305)

[Subscription : Scaffolding 55](#_Toc39449306)

[Authorize Attribute 55](#_Toc39449307)

[Delete HTTP Post Action 55](#_Toc39449308)

[ProductItem Entity : Scaffolding 56](#_Toc39449309)

[Creating Controller 56](#_Toc39449310)

[Authorize Attribute 56](#_Toc39449311)

[Create ProductItemModel 56](#_Toc39449312)

[ProductItem to ProductItemModel Conversion Extensions 57](#_Toc39449313)

[Action & Views 59](#_Toc39449314)

[SubscriptionProduct Entity : Scaffolding 59](#_Toc39449315)

[Authorize Attribute 59](#_Toc39449316)

[SubscriptionProduct to SubscriptionProductModel Conversion Extensions 60](#_Toc39449317)

# Resources placed in folder

Training\Books Videos\Building MVC 5 Membership Html 5 Css3

# Created the project with

1. VS 2019
2. Asp.Net Framework 4.7.2
3. MVC 5
4. Individual User Accounts
5. SSL unchecked
6. Microsoft OWIN
7. Razor Pages 3
8. Web Pages 3

# App\_Start Folder



App-start has the config cs files for bundling, filter identity, route and start up files. All if these are classed inside the global.asax file.



Default route is specified inside the RouteConfig.cs file.

## BundleConfig.cs

Here we will create the bundles which will be then added to the \_Layout.cshtml file or individual pages.

### V1.0 Initial File

using System.Web;

using System.Web.Optimization;

namespace Web.Memberships

{

public class BundleConfig

{

// For more information on bundling, visit https://go.microsoft.com/fwlink/?LinkId=301862

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

// Use the development version of Modernizr to develop with and learn from. Then, when you're

// ready for production, use the build tool at https://modernizr.com to pick only the tests you need.

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/Site.css"));

}

}

}

### V1.1 Adding Sitejs Bundle

This entry to create the Site specific js bundle and then add the AiteAdminMenu js to the bundle created via step [Adding SiteAdminMenu.js to BundleConfig](#_Add_SiteAdmin.js_to) below.



Once the bundle is created then Add it to the \_Layout.cshtml file

# Content Folder

This is where we will put in the resources like images, js and css.

Please create the Documents, Images and Logos folder and then moved the files into this folder.

# Helper Extensions

We’ll create some helper extensions which will help us with the major entities. Create a new folder in the root called Extensions.

## Reflection Extensions

We’ll create two extensions here. Create a class with the name ReflectionExtensions. The class has to be public static.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Web.Memberships.Extensions

{

public static class ReflectionExtensions

{

}

}

### IsPropertyExists Extension

/// <summary>

/// Checks if the property exists in type T

/// </summary>

/// <typeparam name="T">The type</typeparam>

/// <param name="item">The T item</param>

/// <param name="propertyName">The propertyName in T</param>

/// <returns>bool</returns>

public static bool IsPropertyExists<T>(this T item, string propertyName)

{

//must have the basics

if (string.IsNullOrWhiteSpace(propertyName) || item == null) return false;

//property name exists

var property = item.GetType().GetProperty(propertyName);

if (property == null) return false;

return true;

}

### GetPropertyValue Extension

/// <summary>

/// Extension method to get the property value from T using reflections

/// </summary>

/// <typeparam name="T">The type</typeparam>

/// <param name="item">The T item</param>

/// <param name="propertyName">The propertyName in T whose value needs to be fetched</param>

/// <returns>string</returns>

public static string GetPropertyValue<T>(this T item, string propertyName)

{

//must have the basics

if (!item.IsPropertyExists(propertyName)) return "";

var value = item.GetType()

.GetProperty(propertyName)

.GetValue(item, null)

.ToString();

return value;

}

## ICollection Extensions

Create a class with the name ICollectionExtensions. The class has to be public static.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Web.Memberships.Extensions

{

public static class ICollectionExtensions

{

}

}

### ToSelectListItem Extension

This extension will convert ICollection<T> to IEnumerable<SelectListItem>.

/// <summary>

/// Converts ICollection<T> to IEnumarable<SelectListItem>. We need this to display items in dropdowns in MVC views

/// </summary>

/// <typeparam name="T">The ICollection T to display in the dropdown</typeparam>

/// <param name="items">The ICollection T items to display in the dropdown</param>

/// <param name="selectedValue">The selectedValue in the dropdown</param>

/// <returns>IEnumerable of SelectListItem</returns>

public static IEnumerable<SelectListItem> ToSelectListItem<T>(this ICollection<T> items, int selectedValue)

{

if (items == null) return null;

var selectListItems = items.Select(x => new SelectListItem()

{

Text = x.GetPropertyValue("Title"),

Value = x.GetPropertyValue("Id"),

Selected = x.GetPropertyValue("Id").Equals(selectedValue.ToString())

});

return selectListItems;

}

## Convert Extension Method

The extensions in this class works with different models. The extensions and models are created under each section. Click the model or the extension to go to the corresponding code if you want to create these before hand.

* With [ProductModel](#_Creating_ProductModel) this is the [extension](#_Convert_Extension_Method)
* With [ProductItemModel](#_Create_ProductItemModel) this is the [extension](#_ProductItem_to_ProductItemModel)
* With [SubscriptionProduct](#_SubscriptionProduct_Entity_:) this is the [extension](#_SubscriptionProduct_to_Subscription)

# Adding the Admin Menu

Admin menu will be created inside the Views\Shared folder in the root and added to the \_layout.cshtml file

## \_Layout.cshtml

All the menus are in this file and we will keep on building on it.

### V1.0 Initial File

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="/Home/Index/">

<img src="~/Content/Logos/membership-icon-30x152.png" class="visible-xs" />

<img src="~/Content/Logos/membership-icon-45x184.png" class="hidden-xs" />

</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

### V1.1 Adding Admin Menu

This step is part of [\_SiteAdminMenuPartial.cshtml](#__SiteAdminMenuPartial.cshtml) below.



### V1.2 Adding SiteJs bundle

We first created the js file to toggle admin menu via step [SiteAdminMenu.js](#_SiteAdminMenu.js) below.

Then we created the [bundle](#_V1.1_Adding_Sitejs) entry above.

And finally we need to add the bundle to the \_Layout.cshtml file.



## \_SiteAdminMenuPartial.cshtml

Create the menu file inside Views\Shared folder and then will add it to the [\_Layout.cshtml (v1.1)](#_V1.1_Adding_Admin) file.

Right click the Shared folder and add view. Under the options select “Create as a partial view” check box and name it per the heading above. This will be a dropdown menu so will need to

1. add the bootstrap class dropdown.
2. and an attribute “data-admin-menu” which we will target via jquery to open the menu.
3. And finally will add the partial view to the [\_layout.cshtml (V1.1)](#_V1.1_Adding_Admin) file.
4. Do note that Admin link will have down arrow so adding a span and applying classes “glyphicon glyphicon-chevron-down”. Important thing to note is that you shouldn’t be placing anything inside the span tag.
5. We’ll then place another UL block and apply the “dropdown-menu” class and will give it the role of “menu”.

### V1.0 Admin Menu Initial

<li class="dropdown" data-admin-menu>

<a href="#">

Admin

<span class="glyphicon glyphicon-chevron-down"></span>

</a>

<ul class="dropdown-menu" role="menu">

<li class="dropdown-header">Minor Entities</li>

<li class="divider" />

<li><a href="/Admin/Section">&nbsp;&nbsp;Section</a></li>

<li><a href="/Admin/Part">&nbsp;&nbsp;Part</a></li>

<li><a href="/Admin/ItemType">&nbsp;&nbsp;Item Type</a></li>

<li><a href="/Admin/ProductType">&nbsp;&nbsp;Product Type</a></li>

<li><a href="/Admin/ProductLinkText"> &nbsp;&nbsp;Product Link Text</a></li>

<li class="divider" />

<li class="dropdown-header">Major Entities</li>

<li class="divider" />

<li><a href="/Admin/Item">&nbsp;&nbsp;Item</a></li>

<li><a href="/Admin/Product">&nbsp;&nbsp;Product</a></li>

<li><a href="/Admin/Subscription"> &nbsp;&nbsp;Subscription</a></li>

<li class="divider" />

<li class="dropdown-header">Connector Entities</li>

<li class="divider" />

<li><a href="/Admin/ProductItem">&nbsp;&nbsp;Product Item</a></li>

<li><a href="/Admin/SubscriptionProduct"> &nbsp;&nbsp;Subscription Product</a></li>

<li class="divider" />

<li class="dropdown-header">Users & Subscriptions</li>

<li class="divider" />

<li><a href="/Account"> &nbsp;&nbsp;Users & Subscriptions</a></li>

</ul>

</li>

### V1.1 Admin Menu – Restricted Access

This step has happened after [user registration](#_User_Handling_–) and we have added an [Admin role](#_Assigning_the_Admin) to the user below. Also check [restrict Admin](#_Restricting_Admin_Menu) menu.

@if (Request.IsAuthenticated && User.IsInRole("Admin"))

{

<li class="dropdown" data-admin-menu>

<a href="#">

Admin

<span class="glyphicon glyphicon-chevron-down"></span>

</a>

<ul class="dropdown-menu" role="menu">

<li class="dropdown-header">Minor Entities</li>

<li class="divider" />

<li><a href="/Admin/Section">&nbsp;&nbsp;Section</a></li>

<li><a href="/Admin/Part">&nbsp;&nbsp;Part</a></li>

<li><a href="/Admin/ItemType">&nbsp;&nbsp;Item Type</a></li>

<li><a href="/Admin/ProductType">&nbsp;&nbsp;Product Type</a></li>

<li><a href="/Admin/ProductLinkText"> &nbsp;&nbsp;Product Link Text</a></li>

<li class="divider" />

<li class="dropdown-header">Major Entities</li>

<li class="divider" />

<li><a href="/Admin/Item">&nbsp;&nbsp;Item</a></li>

<li><a href="/Admin/Product">&nbsp;&nbsp;Product</a></li>

<li><a href="/Admin/Subscription"> &nbsp;&nbsp;Subscription</a></li>

<li class="divider" />

<li class="dropdown-header">Connector Entities</li>

<li class="divider" />

<li><a href="/Admin/ProductItem">&nbsp;&nbsp;Product Item</a></li>

<li><a href="/Admin/SubscriptionProduct"> &nbsp;&nbsp;Subscription Product</a></li>

<li class="divider" />

<li class="dropdown-header">Users & Subscriptions</li>

<li class="divider" />

<li><a href="/Account"> &nbsp;&nbsp;Users & Subscriptions</a></li>

</ul>

</li>

}

## Opening and Closing the Menu

We’ll do it via javascript file

1. Right click the Scripts folder and add javascript file
2. Name it SiteAdminMenu.js.
3. We will toggle the open class when we’ll hover over the menu.
4. Then we’ll create the bundle
5. And finally will add the bundle to the \_Layout.cshtml file

### SiteAdminMenu.js

$(function () {

//target the li that has the attribute data-admin-menu

//we'll toggle the class open, to open and close the menu

$('li[data-admin-menu]').hover(function () {

$(this).toggleClass('open');

});

});

#### Add SiteAdmin.js to BundleConfig

Check [BundleConfig (V1.1)](#_V1.1_Adding_Sitejs) for details

And then will need to add the bundle to the [\_layout.cshtml (V1.2)](#_V1.2_Adding_SiteJs) above.

At this point run the app and hover over/out the Admin menu to see the affect.

# Creating the Database

* Database table classes are in Entities folder.
* Check Entities diagram inside the App\_Code folder for details.
* Please refer to the Admin Menu. We have Major/Minor/Connector entities. Entities diagram clearly depict this schema.
* We’ll be using the localDB which will be hosted inside the App\_Data folder.
* We’ll be following the Code First approach

## Database Name Web.Config

Open the Web.Config and change the DefaultConnection. Name it whatever you like but make sure that you both highlighted pieces.



## Package Manager Console

Next we’ll issue commands to perform the actions to create the DB for us. Open PackageManager Console or type Package and then issue the commands.





### Command enable-migrations



* Above didn’t create any thing in the App\_Data folder but it did add Migration folder in the root with file Migrations.cs file
* For the purpose of tutorial, we’ll do automatic database migrations. Open the Migrations.cs file and make following two changes. Please read the comment for each property.
  + make AutoMaticMigrationsEnabled = true
  + and add AutomaticMigrationDataLossAllowed and make it true as well.



* Migrations help you seed and revert back if needed. If you scroll down you’ll see the Seed method. Will work with it little later.

### Command update-database [Create Database]

Every time you make a change you need to issue this command.



If you click on ShowAllFiles you’ll see the MembershipDB created.



If you double click the MembershipsDB, it will open in the ServerExplorer and you’ll see the following tables added.



## Adding Table/Entities

We’ll create the classes in the Entities folder.

### Item Related Tables/Entities

#### Section Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as Section.

//If we don't do this then the table will get created with name Sections.

[Table("Section")]

public class Section

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

#### Part Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as Part.

//If we don't do this then the table will get created with name Parts.

[Table("Part")]

public class Part

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

#### ItemType Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as ItemType.

//If we don't do this then the table will get created with name ItemTypes.

[Table("ItemType")]

public class ItemType

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

#### Item Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace Web.Memberships.Entities

{

//specify the table name as Item.

//If we don't do this then the table will get created with name Items.

[Table("Item")]

public class Item

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

[MaxLength(2048)]

public string Description { get; set; }

[MaxLength(1024)]

public string Url { get; set; }

[MaxLength(1024)]

[DisplayName("Image Url")]

public string ImageUrl { get; set; }

[AllowHtml]

public string HTML { get; set; }

//only getter

public string HTMLShort => HTML == null || HTML.Length < 50 ? HTML : HTML.Substring(0, 50);

[DefaultValue(0)]

[DisplayName("Wait Days")]

public int WaitDays { get; set; }

public int ProductId { get; set; }

public int ItemTypeId { get; set; }

public int SectionId { get; set; }

public int PartId { get; set; }

public bool IsFree { get; set; }

public ICollection<ItemType> ItemTypes { get; set; }

[DisplayName("Sections")]

public ICollection<Section> Sections { get; set; }

[DisplayName("Parts")]

public ICollection<Part> Parts { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

### Product Related Tables/Entities

#### Product Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as Product.

//If we don't do this then the table will get created with name Products.

[Table("Product")]

public class Product

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

[MaxLength(2048)]

public string Description { get; set; }

[MaxLength(1024)]

public string ImageUrl { get; set; }

public int ProductLinkTextId { get; set; }

public int ProductTypeId { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

#### ProductType Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as ProductLinkText.

//If we don't do this then the table will get created with name ProductLinkTexts.

[Table("ProductLinkText")]

public class ProductLinkText

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(25)]

[Required]

public string Title { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

#### ProductLinkText Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as ProductLinkText.

//If we don't do this then the table will get created with name ProductLinkTexts.

[Table("ProductLinkText")]

public class ProductLinkText

{

//specify the Id as Identity column with sequence starting from 1

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(25)]

[Required]

public string Title { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

### Subscription Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as Subscription.

//If we don't do this then the table will get created with name Subscriptions.

[Table("Subscription")]

public class Subscription

{

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

[MaxLength(2048)]

public string Description { get; set; }

[MaxLength(20)]

[DisplayName("Registration Code")]

public string RegistrationCode { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

### ProductItem Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as ProductItem.

//If we don't do this then the table will get created with name ProductItem.

[Table("ProductItem")]

public class ProductItem

{

//specify the Id as Identity column with sequence starting from 1

[Required]

[Key, Column(Order = 1)]

public int ProductId { get; set; }

[Required]

[Key, Column(Order = 2)]

public int ItemId { get; set; }

[NotMapped]

public int OldProductId { get; set; }

[NotMapped]

public int OldItemId { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

### SubscriptionProduct Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as Part.

//If we don't do this then the table will get created with name Parts.

[Table("SubscriptionProduct")]

public class SubscriptionProduct

{

//we have a composite primary key

[Required]

[Key, Column(Order = 1)]

public int ProductId { get; set; }

[Required]

[Key, Column(Order = 2)]

public int SubscriptionId { get; set; }

[NotMapped]

public int OldProductId { get; set; }

[NotMapped]

public int OldSubscriptionId { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

### UserSubscription Table/Entity

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace Web.Memberships.Entities

{

//specify the table name as UserSubscription.

//If we don't do this then the table will get created with name UserSubscription.

public class UserSubscription

{

//we have a composite primary key

[Required]

[Key, Column(Order = 1)]

public int SubscriptionId { get; set; }

[Required]

[Key, Column(Order = 2)]

[MaxLength(128)]

public string UserId { get; set; }

public DateTime? StartDate { get; set; }

public DateTime? EndDate { get; set; }

}

}

Next we need to tell entity framework to use this class as [Code First](#_Code_First_Approach) to create the table.

## Code First Approach

We need to tell the entity framework to use the classes created for the [tables/entities](#_Adding_Table/Entities) above to use to create the tables.

Open the Models folder and then open Identity Models and add the following properties to ApplicationDbContext class.

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext()

: base("DefaultConnection", throwIfV1Schema: false)

{

}

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

//We need to add property for each class to create the table for as DbSet

//also the property name will be used to query the table using linq.

//Items

public DbSet<Section> Sections { get; set; }

public DbSet<Part> Parts { get; set; }

public DbSet<ItemType> ItemTypes { get; set; }

public DbSet<Item> Items { get; set; }

//Products

public DbSet<Product> Products { get; set; }

public DbSet<ProductType> ProductTypes { get; set; }

public DbSet<ProductLinkText> ProductLinkTexts { get; set; }

//Subscription

public DbSet<Subscription> Subscriptions { get; set; }

//ProductItem

public DbSet<ProductItem> ProductItems { get; set; }

//SubscriptionProduct

public DbSet<SubscriptionProduct> SubscriptionProducts { get; set; }

//UserSubscription

public DbSet<UserSubscription> UserSubscriptions { get; set; }

}

## Update Database Command

Issue update [database-command](#_Command_update-database_[Create) after creating all the [entities](#_Adding_Table/Entities) and [Code First](#_Code_First_Approach) ApplicationDbContext properties to create the database.

Go to server explorer and you’ll see the new tables created.



## Modifying AspNetUser Table

We’ll add four additional columns to it.

* FirstName
* LastName
* IsActive
* RegistrationDate

Open IdentityModel.cs inside the Models folder and update the ApplicationUser class.

using System;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using Web.Memberships.Entities;

public class ApplicationUser : IdentityUser

{

[MaxLength(100)]

[Required]

public string FirstName { get; set; }

[MaxLength(100)]

[Required]

public string LastName { get; set; }

[DefaultValue(false)]

[Required]

public bool IsActive { get; set; }

[DefaultValue(typeof(DateTime), "")]

[Required]

public DateTime RegistrationDate { get; set; }

public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)

{

// Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType

var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);

// Add custom user claims here

return userIdentity;

}

}

Next we need to tell entity framework to update the table.

Open the package manager Console and run “update-database” command just like when we did [Code First](#_Code_First_Approach_1) action when creating the tables.

Open server explorer and refresh the AspNetUsers table to see the columns added.



# User Handling – Registeration/Login/Roles

## Altering the Register User Actions/View/Models

### AspNetUsers

We have already added extra fields to the [ApplicationUser](#_Modifying_AspNetUser_Table) entity above and these are in the database. We’ll need to handle these extra fields in the registration model/actions.

### Altering RegisterViewModel

Go to Models\AccountViewModels.cs class and find the “RegisterViewModel” model. Add the first & last name properties to this model. The remainder of the properties will stay the same.

[Required]

[Display(Name = "First Name")]

public string FirstName { get; set; }

[Required]

[Display(Name = "Last Name")]

public string LastName { get; set; }

### Altering the RegisterAction

Go to Accounts controller and open AccountController.cs file. Locate the Register actions, this is where we are going to make the change.

#### HttpPost Register Action

* We will be receiving the following via the [model above](#_Altering_RegisterViewModel).
  + FirstName
  + LastName
* We will default
  + IsActive to true
  + RegistratinDate to current datetime
  + EmailConfirmed to true

Make the following changes to the ApplicationUser object being built in this action.

var user = new ApplicationUser {

UserName = model.Email,

Email = model.Email,

//Custom field ==> Add

FirstName = model.FirstName,

//Cusotm field ==> Add

LastName = model.LastName,

//Custom field ==> Add and make it true

IsActive = true,

//Custom field ==> assign current date time

RegistrationDate = DateTime.Now,

//this is built in property, since we are not sending the email, make it true for this project purposes

//ApplicationUser is inheriting from IdentityUser. This field is inside the IdentityUser

EmailConfirmed = true

};

### Altering the Registration View

Go to Views\Account and open the RegisterView.chtml. Add the FirstName and LastName before the Email form group. Make sure that it is put after the validation summary.

<div class="form-group">

@Html.LabelFor(m => m.FirstName, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.FirstName, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.LastName, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.LastName, new { @class = "form-control" })

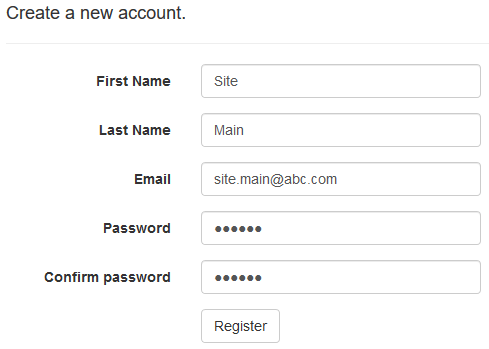
</div>

</div>

## Registering the User

Run the application, click register link and add the user. My user is

* FirstName: Site
* LastName: Main
* Email: [site.main@abc.com](mailto:site.main@abc.com)
* Password: Test12@ ***[must have upper case, lower case, digit and special character]***



Click register button. You’ll get redirected to the site home after registration in logged in state.



Open up the AspNetUsers and you’ll see the new users in there.



Also, log off and then log back in.

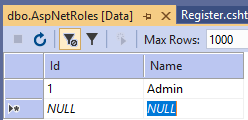
## Assigning the Admin Role

### User Id Pick Up

Pick up the Id of the user from the table. Since it is GUID, it will be different for everyone following this project. In my case it is 527b85ea-06f4-40b8-b988-f361bf9cb5eb.

### Add Role

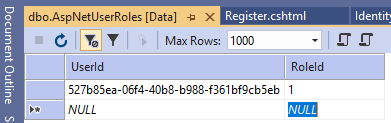
These are different roles that a user can have. Open AspNetRoles table and add a role to it. Take a note of the id that you have entered.



### Add User to Role

Open AspNetUserRoles table and add the following to it. By doing this we have affectively changed the user to have Admin Role. Please make user to pick the

* UserId per the guid that you have in [AspNetUsers](#_User_Id_Pick) table
* RoleId per the id that you have in [AspNetRoles](#_Add_Role) table



## Restrict Access to Admin Menu and Controllers

We have already created the [Admin menu here](#__SiteAdminMenuPartial.cshtml).

And Admin menu controllers are created below. Check different entity sections for more details.

### Restricting Admin Menu

There will be two checks

1. Check user is authenticated
2. Check user has the Admin Role

Go to Views\Shared and then open \_SiteAdminMenuPartial.cshtml

Wrap the menu in the if statement @if (Request.IsAuthenticated && User.IsInRole("Admin")). [Check v1.1](#_V1.1_Admin_Menu) of the menu above.

### Restricting Controller

Add the authorize attribute to all the controllers inside [Admin area](#_Areas) controllers below. Under this check Entity sections.



## Create a General User

We have already created an Admin user. To test the functionality properly we’ll create one general user without the Admin role.

* FirstName: General
* LastName: User
* Email: [general.user@abc.com](mailto:general.user@abc.com)
* Password: Test12@

## Test Restricted Access

LogOff and login with Site Main and General User. With General User you won’t be able to see Admin Menu or access to the controllers.

With General User, try going to any of the restricted controllers directly. You’ll get redirected back to the sign-in page.

# Areas

## Add Admin Area

Add the new areas in VS 2019 by right clicking on the project and then adding a new scaffolding item and selecting area.





* Admin

## Copying the \_Layout to the Areas Shared Folder

Copy the \_Layout from the root Views\Shared\\_Layut.cshtml folder to Areas\Admin\Views\Shared and modify it.

### V 1.0 Initial

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

@Html.Partial("\_SiteAdminMenuPartial");

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@Scripts.Render("~/bundles/sitejs")

@RenderSection("scripts", required: false)

</body>

</html>

# Creating Buttons – will be used on all views

## CreateButton Partial View

Right click the views\shared folder in the root and click add view. Create a partial view with name “\_SiteCreateButtonPartial”. Keep the template Empty (without model) . Add following to it.

<a type="button" class="btn btn-primary btn-sm"

href="@Url.Action("Create")">

<span class="glyphicon glyphicon-plus"></span>

<span>Create New</span>

</a>

We will add this to different pages as

<p>

@Html.Partial("\_SiteCreateButtonPartial")

</p>

## EditButton Partial View

The button will receive the Id as int. Right click on Views\Shared and add a view with the name “\_SiteEditButtonPartial”. Keep the template Empty (without model) . Add following to it.

@model int

<a type="button" class="btn btn-primary btn-sm"

href="@Url.Action("Edit", new { id = Model})">

<span class="glyphicon glyphicon-pencil"></span>

<span>Edit</span>

</a>

Add it to the views as

@Html.Partial("\_SiteEditButtonPartial", Model.Id)

## EditButtonDetail Partial View

### Model

Create a EditButtonModel in Models folder in the root with the name “EditButtonModel”. Add the following to it.

using System.Text;

namespace Web.Memberships.Models

{

public class EditButtonModel

{

public int ItemId { get; set; }

public int ProductId { get; set; }

public int SubscriptionId { get; set; }

public string Link

{

get

{

var s = new StringBuilder("?");

if (ItemId > 0) s.Append($"itemId={ItemId}&");

if (ProductId > 0) s.Append($"productId={ProductId}&");

if (SubscriptionId > 0) s.Append($"subscriptionId={SubscriptionId}&");

return s.ToString().Substring(0, s.Length - 1);

}

}

}

}

### Partial View

Right click the \view\shared folder and click add view. Create a partial view with name “\_SiteEditButtonDetailPartial”. Keep the template Empty (without model) . Add following to it.

@model Web.Memberships.Areas.Admin.Models.EditButtonModel

@\* just appending the link. If we pass in the controller name to the model

then second param would be controller name

and third would be object routeValue as new { a=1,b=2 }

see: https://stackoverflow.com/questions/39095632/how-to-append-a-querystring-to-the-url-created-by-url-action-base-on-a-hidden-fi

\*@

<a type="button" class="btn btn-primary btn-sm"

href="@Url.Action("Edit")@Model.Link">

<span class="glyphicon glyphicon-pencil"></span>

<span>Edit</span>

</a>

And then add the edit button as

@Html.Partial("\_SiteEditButtonPartial", EditButtonModel)

## BackToListButton Partial View

Right click the views\shared folder in the root and click add view. Create a partial view with name “\_SiteBackToListButtonPartial”. Keep the template Empty (without model) . Add following to it.

<a type="button" class="btn btn-primary btn-sm"

href="@Url.Action("Index")">

<span class="glyphicon glyphicon-list"></span>

<span>Back to List</span>

</a>

We will add this to different pages as

<p>

@Html.Partial("\_BackToListButtonPartial")

</p>

## SmallButton Partial View

### Model

Right click the Models folder in the root add class with the name “SmallButtonModel”.

using System.Text;

namespace Web.Memberships.Models

{

public class SmallButtonModel

{

public string Action { get; set; }

public string Text { get; set; }

public string Glyph { get; set; }

public string ButtonType { get; set; }

public int? Id { get; set; }

public int? ItemId { get; set; }

public int? ProductId { get; set; }

public int? SubscriptionId { get; set; }

public string UserId { get; set; }

public string ActionParameters

{

get

{

var param = new StringBuilder("?");

if (Id != null && Id > 0)

param.Append($"id={Id}&");

if (ItemId != null && ItemId > 0)

param.Append($"itemId={ItemId}&");

if (ProductId != null && ProductId > 0)

param.Append($"productId={ProductId}&");

if (SubscriptionId != null && SubscriptionId > 0)

param.Append($"subscriptionId={SubscriptionId}&");

if (UserId != null && !UserId.Equals(string.Empty))

param.Append($"userId={UserId}&");

return param.ToString().Substring(0, param.Length - 1);

}

}

}

}

### Partial View

Right click Views\Shared in the root and add partial view with name “\_SiteSmallButtonPartial”. Keep the template Empty (without model) . Add following to it.

@model Web.Memberships.Models.SmallButtonModel

<a type="button" class="btn @Model.ButtonType btn-sm"

href="@Url.Action(Model.Action)@Model.ActionParameters">

<span class="glyphicon glyphicon-@Model.Glyph"></span>

<span class="sr-only">@Model.Text</span>

</a>

This will be used inside the \_SiteTableButton Partial

## TableButton Partial

Right click Views\Shared folder and create a partial view with name “\_SiteTableButtonsPartial”. Keep the template Empty (without model) . Add following to it.

@using Web.Memberships.Models;

@model Web.Memberships.Models.SmallButtonModel

<td style="width:140px;">

<div class="btn-group siteTableButtons" role="group">

@Html.Partial("\_SiteSmallButtonPartial",

new SmallButtonModel

{

Action = "Edit",

ButtonType = "btn-primary",

Glyph = "pencil",

Text = "Edit button",

Id = Model.Id,

ItemId = Model.ItemId,

ProductId = Model.ProductId,

SubscriptionId = Model.SubscriptionId,

UserId = Model.UserId

})

@if (Model.UserId == null || Model.UserId.Equals(string.Empty))

{

@Html.Partial("\_SiteSmallButtonPartial",

new SmallButtonModel

{

Action = "Details",

ButtonType = "btn-success",

Glyph = "list",

Text = "Detail button",

Id = Model.Id,

ItemId = Model.ItemId,

ProductId = Model.ProductId,

SubscriptionId = Model.SubscriptionId,

UserId = Model.UserId

})

}

@if (Model.UserId != null && !Model.UserId.Equals(string.Empty))

{

@Html.Partial("\_SiteSmallButtonPartial",

new SmallButtonModel

{

Action = "Subscriptions",

ButtonType = "btn-info",

Glyph = "list",

Text = "Subscriptions",

Id = Model.Id,

ItemId = Model.ItemId,

ProductId = Model.ProductId,

SubscriptionId = Model.SubscriptionId,

UserId = Model.UserId

})

}

@Html.Partial("\_SiteSmallButtonPartial",

new SmallButtonModel

{

Action = "Delete",

ButtonType = "btn-danger",

Glyph = "trash",

Text = "Delete button",

Id = Model.Id,

ItemId = Model.ItemId,

ProductId = Model.ProductId,

SubscriptionId = Model.SubscriptionId,

UserId = Model.UserId

})

</div>

</td>

This will be used like

@Html.Partial("\_SiteTableButtonsPartial",

new SmallButtonModel { Id = item.Id })

### CSS

Open Content\Site.css and add the following to it. This will apply the left margin to the second and third buttons specified above.

.btn-group.siteTableButtons > .btn:not(:first-child) {

margin-left: 2px !important;

}

### BundleConfig

Open BundeConfig and fix the Site.css name, it should match exactly the file name.

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/Site.css"));

# Section Entity : Creating Admin User Interface

## Scaffolding the Section Entity

Go to the Admin area and right click on the Controller folder and click add and then click controller.

We’ll create a MVC 5 Controller with views, using Entity Framework option.

* Select the model class
* Select the data context class
* Check the check box for User async controller actions
* Name the controller SectionController and
* Finally click Add



This would create the Controller, Action and associated views. Please spend some time to look at these.



## SectionController

Take some time a look at the controller. We have everything that we need to Create/Edit/Delete/View the Section entity.

HTTPPost methods are also decorated with “ValidateAntiForgeryToken”.

### Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

## Index View

Open the index view

* Remove the default button and
* Add the button we created above

@model IEnumerable<Web.Memberships.Entities.Section>

@using Web.Memberships.Models;

@{

ViewBag.Title = "Index";

}

<h2>Index</h2>

<p>

@Html.Partial("\_SiteCreateButtonPartial")

</p>

<table class="table table-striped table-condensed">

<tr>

<th>

@Html.DisplayNameFor(model => model.Title)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Title)

</td>

@Html.Partial("\_SiteTableButtonsPartial", new SmallButtonModel { Id = item.Id })

</tr>

}

</table>

## Edit View

Open the index view

* Remove the default button and
* Add the button we created above

@model Web.Memberships.Entities.Section

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Section</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.Id)

<div class="form-group">

@Html.LabelFor(model => model.Title, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Title, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-success" />

</div>

</div>

</div>

}

<div>

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

## Details View

Open the index view

* Remove the default button and
* Add the button we created above

@model Web.Memberships.Entities.Section

@{

ViewBag.Title = "Details";

}

<h2>Details</h2>

<div>

<h4>Section</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd>

@Html.DisplayFor(model => model.Title)

</dd>

</dl>

</div>

<p>

@Html.Partial("\_SiteEditButtonPartial", Model.Id)

@Html.Partial("\_SiteBackToListButtonPartial")

</p>

## Delete View

Open the index view

* Remove the default button and
* Add the button we created above

@model Web.Memberships.Entities.Section

@{

ViewBag.Title = "Delete";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Section</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd>

@Html.DisplayFor(model => model.Title)

</dd>

</dl>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

<div class="form-actions no-color">

<input type="submit" value="Delete" class="btn btn-danger btn-sm" /> |

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

}

</div>

## Update \_SiteAdminMenuPartial Section Link using @UrlAction

Open the admin menu in Views\Shared folder and update the Section menu. Currently it was hardcoded value.

<li><a href="@Url.Action("Index", "Section", new { Area = "Admin" })">&nbsp;&nbsp;Section</a></li>

## Playing with Data

Try creating/editing/view/deleting some test data.

# Adding Controller – Actions - Views for Part / ItemType / ProductType / ProductLinkText Entities

Just like the [Section](#_Section_Entity_:) Entity above, scaffold the following entities as well. Create a non plural name for each item.

* Part
* ItemType
* ProductType
* ProductLinkText

Once created, open the views and add the buttons just like the Section entity.

Don’t forget to update the SiteAdminMenu to use @Url.Action rather than hard coded urls. The update should look like

<li><a href="@Url.Action("Index", "Section", new { Area = "Admin" })">&nbsp;&nbsp;Section</a></li>

<li><a href="@Url.Action("Index", "Part", new { Area = "Admin" })">&nbsp;&nbsp;Part</a></li>

<li><a href="@Url.Action("Index", "ItemType", new { Area = "Admin" })">&nbsp;&nbsp;Item Type</a></li>

<li><a href="@Url.Action("Index", "ProductType", new { Area = "Admin" })">&nbsp;&nbsp;Product Type</a></li>

<li><a href="@Url.Action("Index", "ProductLinkText", new { Area = "Admin" })"> &nbsp;&nbsp;Product Link Text</a></li>

## Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

# Item Entity : Scaffolding

## Controllers - Actions - Views

Just like the previous two sections, scaffold the Item entity.

Open the views and change the buttons accordingly.

Don’t forget to update the SiteAdminMenu to use @Url.Action rather than hard coded urls. The update should look like

<li><a href="@Url.Action("Index", "Item", new { Area = "Admin" })">&nbsp;&nbsp;Item</a></li>

We will create dropdowns so would need to modify the view.

### Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

## Create Updates

### Controller Action – Create HTTP GET

Go to Item controller and then to Create Get action. Pull the data for ItemType, Section and Parts.

Make sure that you have created some ItemType, Section and Parts data first.

// GET: Admin/Item/Create

public ActionResult Create()

{

//build model

var model = new Item

{

Parts = db.Parts.ToList(),

ItemTypes = db.ItemTypes.ToList(),

Sections = db.Sections.ToList()

};

//pass the model into the view so that it renders it

return View(model);

}

### Controller Action – Create HTTP POST

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create([Bind(Include = "Id,Title,Description,Url,ImageUrl,HTML,WaitDays,ProductId,ItemTypeId,SectionId,PartId,IsFree")] Item item)

{

if (ModelState.IsValid)

{

db.Items.Add(item);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

item.Parts = db.Parts.ToList();

item.ItemTypes = db.ItemTypes.ToList();

item.Sections = db.Sections.ToList();

return View(item);

}

### View

At the top of the view add

@using Web.Memberships.Extensions;

#### ProductId - Delete

Find and delete the ProductId Group.

#### ItemTypeId – Convert to DropDown

**Before**

<div class="form-group">

@Html.LabelFor(model => model.ItemTypeId, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.ItemTypeId, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.ItemTypeId, "", new { @class = "text-danger" })

</div>

</div>

**After**

<div class="form-group">

@Html.LabelFor(model => model.ItemTypes, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@if (Model.ItemTypes != null && Model.ItemTypes.Any())

{

@Html.DropDownListFor(

model => model.ItemTypeId,

Model.ItemTypes.ToSelectListItem(Model.ItemTypeId),

new { @class = "form-control"})

@Html.ValidationMessageFor(model => model.ItemTypeId, "", new { @class = "text-danger" })

}

</div>

</div>

#### SectionId – Convert to DropDown

Just like above ItemTypeId, change the section to dropdown as well.

<div class="form-group">

@Html.LabelFor(model => model.Sections, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@if (Model.Sections != null && Model.Sections.Any())

{

@Html.DropDownListFor(

model => model.SectionId,

Model.Sections.ToSelectListItem(Model.SectionId),

new { @class = "form-control"})

@Html.ValidationMessageFor(model => model.SectionId, "", new { @class = "text-danger" })

}

</div>

</div>

#### PartId – Convert to DropDown

Just like above ItemTypeId and Sectionid, change the PartId to dropdown as well.

<div class="form-group">

@Html.LabelFor(model => model.Parts, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@if (Model.Parts != null && Model.Parts.Any())

{

@Html.DropDownListFor(

model => model.PartId,

Model.Parts.ToSelectListItem(Model.PartId),

new { @class = "form-control"})

@Html.ValidationMessageFor(model => model.PartId, "", new { @class = "text-danger" })

}

</div>

</div>

#### Html – Convert to TextAreaFor for Multiline

<div class="form-group">

@Html.LabelFor(model => model.HTML, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.TextAreaFor(model => model.HTML, new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.HTML, "", new { @class = "text-danger" })

</div>

</div>

## Edit Updates

Just like what we have done for the Create, make similar changes for the Edit as well.

Fetch Parts, ItemTypes and Section for both Edit HTTP Post and HTTP Get actions.

Also, update the view for the dropdowns and HTML textarea for.

# Product Entity

## Creating ProductModel

Go to Areas\Admin\Model, right click and add class ProductModel

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

using Web.Memberships.Entities;

namespace Web.Memberships.Areas.Admin.Models

{

public class ProductModel

{

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

[MaxLength(2048)]

public string Description { get; set; }

[MaxLength(1024)]

[DisplayName("Image Url")]

public string ImageUrl { get; set; }

public int ProductLinkTextId { get; set; }

public int ProductTypeId { get; set; }

[DisplayName("Product Link Text")]

public ICollection<ProductLinkText> ProductLinkTexts { get; set; }

[DisplayName("Product Type")]

public ICollection<ProductType> ProductTypes { get; set; }

public string ProductType => ProductTypes == null || !ProductTypes.Any() ? String.Empty : ProductTypes.First(pt => pt.Id.Equals(ProductTypeId)).Title;

public string ProductLinkText => ProductLinkTexts == null || !ProductLinkTexts.Any() ? String.Empty : ProductLinkTexts.First(pt => pt.Id.Equals(ProductLinkTextId)).Title;

}

}

## Conversion Extensions

We have created other [extension](#_ToSelectListItem_Extension) above for the dropdowns.

Right click on the Extensions folder and create a new class ConversionExtensions.cs. The class has to be static. At this time only adding Product related extensions here. Later we’ll add more extensions to it.

We’ll create two extension here for the product. We are adding ProductLinkTexts and ProductTypes from the DB via the extension to ProductModel. Here we are Converting Product entity to ProductModel.

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

using Web.Memberships.Areas.Admin.Models;

using Web.Memberships.Entities;

using Web.Memberships.Models;

namespace Web.Memberships.Extensions

{

public static class ConversionExtensions

{

#region Prduct

public static async Task<IEnumerable<ProductModel>> Convert(this IEnumerable<Product> products, ApplicationDbContext db)

{

if (products == null || !products.Any() || db == null)

return new List<ProductModel>();

var texts = await db.ProductLinkTexts.ToListAsync();

var types = await db.ProductTypes.ToListAsync();

var newProducts = products.Select(p => new ProductModel()

{

Id = p.Id,

Title = p.Title,

Description = p.Description,

ImageUrl = p.ImageUrl,

ProductLinkTextId = p.ProductLinkTextId,

ProductTypeId = p.ProductTypeId,

ProductLinkTexts = texts,

ProductTypes = types

});

return newProducts;

}

public static async Task<ProductModel> Convert(this Product product, ApplicationDbContext db)

{

if (product == null || db == null)

return new ProductModel();

var text = await db.ProductLinkTexts.FirstOrDefaultAsync(p => p.Id.Equals(product.ProductLinkTextId));

var type = await db.ProductTypes.FirstOrDefaultAsync(p => p.Id.Equals(product.ProductTypeId));

var model = new ProductModel

{

Id = product.Id,

Title = product.Title,

Description = product.Description,

ImageUrl = product.ImageUrl,

ProductLinkTextId = product.ProductLinkTextId,

ProductTypeId = product.ProductTypeId,

ProductLinkTexts = new List<ProductLinkText>(),

ProductTypes = new List<ProductType>()

};

model.ProductLinkTexts.Add(text);

model.ProductTypes.Add(type);

return model;

}

#endregion

}

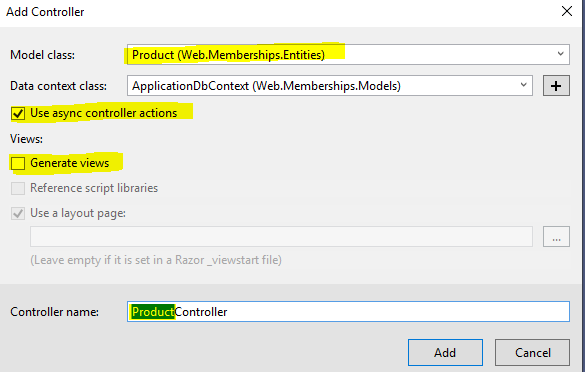
}

## Scaffolding Controller/Action for the Product Entity

Right click the Controllers folder in Areas\Admin\Controller and add a controller Product by selecting “MVC 5 Controller with views, using Entity Framework”. Make sure to select Product model class and name it ProductController. Check [here](#_Scaffolding_the_Section) for more details.

Also note that we’ll create the views manually here.

Take a look at the ProductController for more details.



### Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

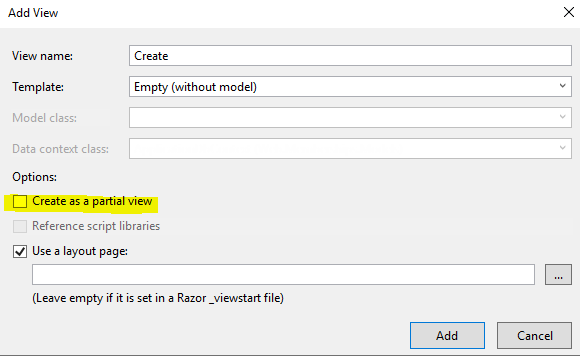
## Updating the SiteAdminMenuPartial

Rather than having a hard coded links, change it as following.

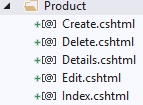
<li><a href="@Url.Action("Index", "Product", new { Area = "Admin" })">&nbsp;&nbsp;Product</a></li>

## Adding views

Open the product controller and go to each action, right click inside action and click AddView.



Once done, you should see the default views generated in the Areas\Admin\Views folder.



## Alter Index Action & View

### Action

Add the using statement at the top

using Web.Memberships.Extensions;

and then change the index action to following. Keep note that here we are converting the Product result to ProductModel and then will send the model to the view.

// GET: Admin/Product

public async Task<ActionResult> Index()

{

var products = await db.Products.ToListAsync();

var model = await products.Convert(db);

return View(model);

}

### View

Change the view as following

@model IEnumerable<Web.Memberships.Areas.Admin.Models.ProductModel>

@using Web.Memberships.Models;

@{

ViewBag.Title = "Index";

}

<h2>Index</h2>

<p>

@Html.Partial("\_SiteCreateButtonPartial")

</p>

<table class="table table-striped table-condensed">

<tr class="success">

<th>

@Html.DisplayNameFor(model => model.Title)

</th>

<th>

@Html.DisplayNameFor(model => model.Description)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductLinkTexts)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductTypes)

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Title)

</td>

<td>

@Html.DisplayFor(modelItem => item.Description)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductLinkText)

(@{@Html.DisplayFor(modelItem => item.ProductLinkTextId)})

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductType)

(@{@Html.DisplayFor(modelItem => item.ProductTypeId)})

</td>

@Html.Partial("\_SiteTableButtonsPartial", new SmallButtonModel { Id = item.Id })

</tr>

}

</table>

## Alter Create Action & View

### Action Create Get

Add following two using statements

using Web.Memberships.Areas.Admin.Models;

This needs to pass ProductModel to the view so change as following

// GET: Admin/Product/Create

public async Task<ActionResult> Create()

{

var model = new ProductModel

{

ProductLinkTexts = await db.ProductLinkTexts.ToListAsync(),

ProductTypes = await db.ProductTypes.ToListAsync()

};

return View(model);

}

### Action Create Post

In case of an error this needs to return a ProductModel as well

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create([Bind(Include = "Id,Title,Description,ImageUrl,ProductLinkTextId,ProductTypeId")] Product product)

{

if (ModelState.IsValid)

{

db.Products.Add(product);

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

//very important to do it as IEnumerable or the dropdown will only show a single item

var products = new List<Product>();

products.Add(product);

var model = await products.Convert(db);

return View(model.First());

}

### View Create

@model Web.Memberships.Areas.Admin.Models.ProductModel

@using Web.Memberships.Extensions;

@{

ViewBag.Title = "Create";

}

<h2>Create</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Product</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.Title, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Title, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ImageUrl, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.ImageUrl, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.ImageUrl, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ProductTypes, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.DropDownListFor(

model => model.ProductTypeId,

Model.ProductTypes.ToSelectListItem(Model.ProductTypeId),

new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.ProductTypeId, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ProductLinkTexts, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.DropDownListFor(

model => model.ProductLinkTextId,

Model.ProductLinkTexts.ToSelectListItem(Model.ProductLinkTextId),

new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.ProductLinkTextId, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-success" />

</div>

</div>

</div>

}

<div>

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

## Alter Edit Action & View

### Action Edit Get

It needs to return the ProductModel

public async Task<ActionResult> Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var product = await db.Products.FindAsync(id);

if (product == null)

{

return HttpNotFound();

}

//very important to do it as IEnumerable or the dropdown will only show a single item

var products = new List<Product>();

products.Add(product);

var model = await products.Convert(db);

return View(model.First());

}

### Action Edit Post

In case of an error, it needs to return the ProductModel as well.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Edit([Bind(Include = "Id,Title,Description,ImageUrl,ProductLinkTextId,ProductTypeId")] Product product)

{

if (ModelState.IsValid)

{

db.Entry(product).State = EntityState.Modified;

await db.SaveChangesAsync();

return RedirectToAction("Index");

}

//very important to do it as IEnumerable or the dropdown will only show a single item

var products = new List<Product>();

products.Add(product);

var model = await products.Convert(db);

return View(model.First());

}

### View Edit

@model Web.Memberships.Areas.Admin.Models.ProductModel

@using Web.Memberships.Extensions;

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Product</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.Title, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Title, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ImageUrl, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.ImageUrl, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.ImageUrl, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ProductTypes, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.DropDownListFor(

model => model.ProductTypeId,

Model.ProductTypes.ToSelectListItem(Model.ProductTypeId),

new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.ProductTypeId, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ProductLinkTexts, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.DropDownListFor(

model => model.ProductLinkTextId,

Model.ProductLinkTexts.ToSelectListItem(Model.ProductLinkTextId),

new { @class = "form-control" })

@Html.ValidationMessageFor(model => model.ProductLinkTextId, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-success" />

</div>

</div>

</div>

}

<div>

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

## Alter Details Action & View

### Action Detail

public async Task<ActionResult> Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var product = await db.Products.FindAsync(id);

if (product == null)

{

return HttpNotFound();

}

var model = await product.Convert(db);

return View(model);

}

### View

@model Web.Memberships.Areas.Admin.Models.ProductModel

@{

ViewBag.Title = "Details";

}

<h2>Details</h2>

<div>

<h4>Product</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd>

@Html.DisplayFor(model => model.Title)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Description)

</dt>

<dd>

@Html.DisplayFor(model => model.Description)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ImageUrl)

</dt>

<dd>

@Html.DisplayFor(model => model.ImageUrl)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ProductTypes)

</dt>

<dd>

@Html.DisplayFor(model => model.ProductType)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ProductLinkTexts)

</dt>

<dd>

@Html.DisplayFor(model => model.ProductLinkText)

</dd>

</dl>

</div>

<p>

@Html.Partial("\_SiteEditButtonPartial", Model.Id)

@Html.Partial("\_SiteBackToListButtonPartial")

</p>

## Alter Delete Action & View

### Add reference to System.Transactions

We’ll be using the transactions to delete the data so add a reference to the System.Transactions in your project. Then add the following using towards the top of the page.

using System.Transactions;

### Action Delete Get

Return the ProductModel

public async Task<ActionResult> Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var product = await db.Products.FindAsync(id);

if (product == null)

{

return HttpNotFound();

}

var model = await product.Convert(db);

return View(model);

}

### Action Delete Http Post

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DeleteConfirmed(int id)

{

var product = await db.Products.FindAsync(id);

//doing it through transactions

//we need to remove the ProductItems and ProductSubscriptions as well.

using (var transaction = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))

{

try

{

var prodItems = db.ProductItems.Where(pi => pi.ProductId.Equals(id));

var prodSubscr = db.SubscriptionProducts.Where(sp => sp.ProductId.Equals(id));

db.ProductItems.RemoveRange(prodItems);

db.SubscriptionProducts.RemoveRange(prodSubscr);

db.Products.Remove(product);

await db.SaveChangesAsync();

transaction.Complete();

}

catch { transaction.Dispose(); }

}

return RedirectToAction("Index");

}

### View

@model Web.Memberships.Areas.Admin.Models.ProductModel

@{

ViewBag.Title = "Delete";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Product</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd>

@Html.DisplayFor(model => model.Title)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Description)

</dt>

<dd>

@Html.DisplayFor(model => model.Description)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ImageUrl)

</dt>

<dd>

@Html.DisplayFor(model => model.ImageUrl)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ProductTypes)

</dt>

<dd>

@Html.DisplayFor(model => model.ProductType)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ProductLinkTexts)

</dt>

<dd>

@Html.DisplayFor(model => model.ProductLinkText)

</dd>

</dl>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-actions no-color">

<input type="submit" value="Delete" class="btn btn-danger btn-sm" />

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

}

</div>

# Subscription : Scaffolding

We’ll be scaffolding the Subscription Controller/Actions/Views. Check [Item](#_Item_Entity_:) for more details.

The controller name will be Subscription and the model will be Subscription. The controller will go in Areas\Admin\Controllers folder.

Once all in place, update the views to use the buttons that we have been using thorough out. Also, open the views in my project and update as necessary.

Don’t forget to update the Subscription link to use @Url.Action in SiteAdminMenuPartial view.

<li><a href="@Url.Action("Index", "Subscription", new { Area = "Admin" })"> &nbsp;&nbsp;Subscription</a></li>

## Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

## Delete HTTP Post Action

We’ll use the transactions to do the delete.

Add using statement at the top

using System.Transactions;

and then update the action method as

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DeleteConfirmed(int id)

{

Subscription subscription = await db.Subscriptions.FindAsync(id);

using (var transaction = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))

{

try

{

var prodSubscr = db.SubscriptionProducts.Where(sp => sp.SubscriptionId.Equals(id));

db.SubscriptionProducts.RemoveRange(prodSubscr);

db.Subscriptions.Remove(subscription);

await db.SaveChangesAsync();

transaction.Complete();

}

catch { transaction.Dispose(); }

}

return RedirectToAction("Index");

}

# ProductItem Entity : Scaffolding

We’ll pass ProductModel between the actions and views. This will also means creating the [conversion extensions just like Product](#_Conversion_Extensions).

## Creating Controller

Right click Areas\Admin\Controllers folder and a controller.

Select Model: ProductItem

Check Views check box as well. We’ll overwrite these. Check the source for the content.

Name the controller ProductItemController.

Put the following using at the top of the controller

using Web.Memberships.Areas.Admin.Models;

Once done, don’t forget to update the SiteAdminMenuPartial to use the @Url.Action.

<li><a href="@Url.Action("Index", "ProductItem", new { Area = "Admin" })">&nbsp;&nbsp;Product Item</a></li>

### Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

## Create ProductItemModel

Right clicks Areas\Admin\Models and create a class with name “ProductItemModel”.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Web;

using Web.Memberships.Entities;

namespace Web.Memberships.Areas.Admin.Models

{

public class ProductItemModel

{

[DisplayName("Product Id")]

public int ProductId { get; set; }

[DisplayName("Item Id")]

public int ItemId { get; set; }

[DisplayName("Product Title")]

public string ProductTitle { get; set; }

[DisplayName("Item Title")]

public string ItemTitle { get; set; }

public ICollection<Product> Products { get; set; }

public ICollection<Item> Items { get; set; }

}

}

## ProductItem to ProductItemModel Conversion Extensions

Don’t forget to look at the [main extensions](#_Convert_Extension_Method_1) and [Product extensions](#_Convert_Extension_Method). Open \Extensions\ConversionExtensions.cs file and add four more extensions to it for conversion from ProductItem to ProductItemModel.

Since we will be using transactions, make sure to add the following using statement

using System.Transactions;

and then add following methods

#region ProductItem

public static async Task<IEnumerable<ProductItemModel>> Convert(this IQueryable<ProductItem> productItems, ApplicationDbContext db)

{

if (productItems == null || !productItems.Any() || db == null)

return new List<ProductItemModel>();

var model = await (from pi in productItems

select new ProductItemModel

{

ItemId = pi.ItemId,

ProductId = pi.ProductId,

ItemTitle = db.Items.FirstOrDefault(i => i.Id.Equals(pi.ItemId)).Title,

ProductTitle = db.Products.FirstOrDefault(p => p.Id.Equals(pi.ProductId)).Title

}).ToListAsync();

return model;

}

public static async Task<ProductItemModel> Convert(this ProductItem productItem, ApplicationDbContext db, bool addListData = true)

{

if (productItem == null || db == null)

return new ProductItemModel();

var model = new ProductItemModel

{

ItemId = productItem.ItemId,

ProductId = productItem.ProductId,

Items = addListData ? await db.Items.ToListAsync() : null,

Products = addListData ? await db.Products.ToListAsync() : null,

ItemTitle = (await db.Items.FirstOrDefaultAsync(i => i.Id.Equals(productItem.ItemId))).Title,

ProductTitle = (await db.Products.FirstOrDefaultAsync(p => p.Id.Equals(productItem.ProductId))).Title

};

return model;

}

public static async Task<bool> CanChange(this ProductItem productItem, ApplicationDbContext db)

{

if (productItem == null || db == null)

return false;

//check that the current is available

var oldPI = await db.ProductItems.CountAsync(pi => pi.ProductId.Equals(productItem.OldProductId) && pi.ItemId.Equals(productItem.OldItemId));

//make sure that the new is not already selected

var newPI = await db.ProductItems.CountAsync(pi => pi.ProductId.Equals(productItem.ProductId) && pi.ItemId.Equals(productItem.ItemId));

return oldPI.Equals(1) && newPI.Equals(0);

}

public static async Task Change(this ProductItem productItem, ApplicationDbContext db)

{

var oldProductItem = await db.ProductItems.FirstOrDefaultAsync(pi => pi.ProductId.Equals(productItem.OldProductId) && pi.ItemId.Equals(productItem.OldItemId));

var newProductItem = await db.ProductItems.FirstOrDefaultAsync(pi => pi.ProductId.Equals(productItem.ProductId) && pi.ItemId.Equals(productItem.ItemId));

if (oldProductItem != null && newProductItem == null)

{

newProductItem = new ProductItem { ItemId = productItem.ItemId, ProductId = productItem.ProductId };

using (var transaction = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))

{

try

{

db.ProductItems.Remove(oldProductItem);

db.ProductItems.Add(newProductItem);

await db.SaveChangesAsync();

transaction.Complete();

}

catch { transaction.Dispose(); }

}

}

}

public static async Task<bool> CanCreate(this ProductItem productItem, ApplicationDbContext db)

{

if (productItem == null || db == null)

return false;

//make sure that the new is not already selected

var newPI = await db.ProductItems.CountAsync(pi => pi.ProductId.Equals(productItem.ProductId) && pi.ItemId.Equals(productItem.ItemId));

return newPI.Equals(0);

}

#endregion

## Action & Views

We have been through this multiple time now. Take a look at the source and modify the following actions\views accordingly. This will also include adding the site buttons that we have been using so far.

* Index Action\View
* Detail Action\View. Will receive the itemId and productId
* Create Action\View. Change Create Get to async
* Edit Action\View. Edit GET will receive the itemId and productid
* Delete Action\View. Delete GET will receive the itemId and productId

# SubscriptionProduct Entity : Scaffolding

We’ll pass ProductModel between the actions and views. This will also means creating the [conversion extensions just like Product](#_Conversion_Extensions) or [conversion extensions just like ProductItem](#_ProductItem_to_ProductItemModel).

* We’ll be creating a controller without the default views. Since we have done this before, try creating the controller with action methods only.
  + The controller will go in Ares\Admin\Controllers.
  + Selectmodel SubscriptionProduct
  + Name the controller as SubscriptionProduct.
* Don’t forget to update the SiteAdminMenuPartial to use the @Url.Action.

<li><a href="@Url.Action("Index", "SubscriptionProduct", new { Area = "Admin" })"> &nbsp;&nbsp;Subscription Product</a></li>

* Create a model to pass the info between the action and view. Name would be “SubscriptionProduct”. Check Ares\Admin\Models folder for more details.
* Next create the extensions, see [below](#_SubscriptionProduct_to_Subscription) for details.
* For the views, look aa the entity information and try create on your own. For reference purposes you can view my project for details.
* Finally work on the actions by receiving and passing SubscriptionProductModel between the actions and views.

## Authorize Attribute

Add the authorize attribute to the controller.

[Authorize(Roles = "Admin")]

## SubscriptionProduct to SubscriptionProductModel Conversion Extensions

Don’t forget to look at the [main extensions](#_Convert_Extension_Method_1) , [Product extensions](#_Convert_Extension_Method) or [ProductItem extensions](#_ProductItem_to_ProductItemModel). Open \Extensions\ConversionExtensions.cs file and add four more extensions to it for conversion from ProductItem to ProductItemModel. Open the extensions class and then look for area “Subscription Product” for more details. Following are the extensions we’ll we need

* Convert
  + Returns Task<IEnumerable<SubscriptionProductModel>>
  + Receives param 1 as this IQueryable<SubscriptionProduct> subscriptionProducts
  + Receivers param 2 as ApplicationDbContext db
* Convert
  + Returns Task<SubscriptionProductModel>
  + Receives param 1 as this SubscriptionProduct subscriptionProduct
  + Receives param 2 as ApplicationDbContext db
  + Receives param 3 as bool addListData = true
* CanChange
  + Returns Task<bool>
  + Receives param 1 as this SubscriptionProduct subscriptionProduct
  + Receives param 2 as ApplicationDbContext db
* Change
  + Receives param 1 as this SubscriptionProduct subscriptionProduct
  + Receives param 2 as ApplicationDbContext db
* CanCreate
  + Returns Task<bool>
  + Receives param 1 as SubscriptionProduct subscriptionProduct
  + Receives param 2 as ApplicationDbContext db