**MVC 5 Membership Website**

Contents

[Resources placed in folder 8](#_Toc40828375)

[Created the project with 8](#_Toc40828376)

[App\_Start Folder 8](#_Toc40828377)

[RouteConfig.cs 8](#_Toc40828378)

[BundleConfig.cs 9](#_Toc40828379)

[V1.0 Initial File 9](#_Toc40828380)

[V1.1 Adding Sitejs Bundle 9](#_Toc40828381)

[V1.2 Adding Navbar.css Bundle 10](#_Toc40828382)

[V1.3 Adding Thumbnails.css Bundle 10](#_Toc40828383)

[V1.4 Adding ProductContent.css Bundle 10](#_Toc40828384)

[V1.5 Adding Carret.css Bundle 10](#_Toc40828385)

[V1.6 Adding SiteCarret.js Bundle 10](#_Toc40828386)

[V1.7 Adding JWPlayers.js Bundle 11](#_Toc40828387)

[V1.8 Adding RegisterCode.css Bundle 11](#_Toc40828388)

[V1.9 Adding SiteRegisterCode.js Bundle 11](#_Toc40828389)

[Content Folder 11](#_Toc40828390)

[Adding the Admin Menu 11](#_Toc40828391)

[\_Layout.cshtml 11](#_Toc40828392)

[V1.0 Initial File 11](#_Toc40828393)

[V1.1 Adding Admin Menu 12](#_Toc40828394)

[V1.2 Adding SiteJs bundle 12](#_Toc40828395)

[V1.3 Making Adjustments 13](#_Toc40828396)

[\_SiteAdminMenuPartial.cshtml 13](#_Toc40828397)

[V1.0 Admin Menu Initial 13](#_Toc40828398)

[V1.1 Admin Menu – Restricted Access 14](#_Toc40828399)

[Opening and Closing the Menu 14](#_Toc40828400)

[SiteAdminMenu.js 14](#_Toc40828401)

[Creating the Database 15](#_Toc40828402)

[Database Name Web.Config 15](#_Toc40828403)

[Package Manager Console 15](#_Toc40828404)

[Command enable-migrations 15](#_Toc40828405)

[Command update-database [Create Database] 16](#_Toc40828406)

[Adding Table/Entities 16](#_Toc40828407)

[Item Related Tables/Entities 17](#_Toc40828408)

[Product Related Tables/Entities 19](#_Toc40828409)

[Subscription Table/Entity 20](#_Toc40828410)

[ProductItem Table/Entity 21](#_Toc40828411)

[SubscriptionProduct Table/Entity 21](#_Toc40828412)

[UserSubscription Table/Entity 22](#_Toc40828413)

[Code First Approach 22](#_Toc40828414)

[Update Database Command 23](#_Toc40828415)

[Modifying AspNetUser Table 24](#_Toc40828416)

[Helper Extensions 25](#_Toc40828417)

[Reflection Extensions 25](#_Toc40828418)

[IsPropertyExists Extension 25](#_Toc40828419)

[GetPropertyValue Extension 26](#_Toc40828420)

[ICollection Extensions 26](#_Toc40828421)

[ToSelectListItem Extension 26](#_Toc40828422)

[Convert Extension Method 27](#_Toc40828423)

[Identity Extensions 27](#_Toc40828424)

[HttpContext Extensions 27](#_Toc40828425)

[Thumbnail Extensions 27](#_Toc40828426)

[Section Extensions 27](#_Toc40828427)

[Comparers 27](#_Toc40828428)

[ThumbnailEqualityComparer 27](#_Toc40828429)

[ProductSectionEqualityComparer 27](#_Toc40828430)

[User Handling – Registeration/Login/Roles 27](#_Toc40828431)

[Altering the Register User Actions/View/Models 27](#_Toc40828432)

[AspNetUsers 27](#_Toc40828433)

[Altering RegisterViewModel 27](#_Toc40828434)

[Altering the RegisterAction 28](#_Toc40828435)

[Altering the Registration View 28](#_Toc40828436)

[Registering the User 28](#_Toc40828437)

[Assigning the Admin Role 29](#_Toc40828438)

[User Id Pick Up 29](#_Toc40828439)

[Add Role 29](#_Toc40828440)

[Add User to Role 30](#_Toc40828441)

[Restrict Access to Admin Menu and Controllers 30](#_Toc40828442)

[Restricting Admin Menu 30](#_Toc40828443)

[Restricting Controller 30](#_Toc40828444)

[Create a General User 30](#_Toc40828445)

[Test Restricted Access 31](#_Toc40828446)

[Editing / Altering User Info 31](#_Toc40828447)

[Layout Changes 31](#_Toc40828448)

[Navbar.css 31](#_Toc40828449)

[Views\Shared\\_Layout.cshtml 31](#_Toc40828450)

[From 31](#_Toc40828451)

[To 31](#_Toc40828452)

[Adding Google Fonts 32](#_Toc40828453)

[\Admin\Views\Shares\\_Layout.cshtml 32](#_Toc40828454)

[LogOff Action Update 32](#_Toc40828455)

[\_LoginPartial.cshtml 32](#_Toc40828456)

[Home Index View 33](#_Toc40828457)

[JW Player Setup 33](#_Toc40828458)

[Basic Setup 33](#_Toc40828459)

[JWPlayer.js 34](#_Toc40828460)

[Areas 34](#_Toc40828461)

[Add Admin Area 34](#_Toc40828462)

[Copying the \_Layout to the Areas Shared Folder 35](#_Toc40828463)

[V 1.0 Initial 35](#_Toc40828464)

[Creating Buttons – will be used on all views 36](#_Toc40828465)

[CreateButton Partial View 36](#_Toc40828466)

[EditButton Partial View 36](#_Toc40828467)

[EditButtonDetail Partial View 36](#_Toc40828468)

[Model 36](#_Toc40828469)

[Partial View 37](#_Toc40828470)

[BackToListButton Partial View 37](#_Toc40828471)

[SmallButton Partial View 37](#_Toc40828472)

[Model 37](#_Toc40828473)

[Partial View 38](#_Toc40828474)

[TableButton Partial 38](#_Toc40828475)

[CSS 39](#_Toc40828476)

[BundleConfig 40](#_Toc40828477)

[Section Entity : Creating Admin User Interface 40](#_Toc40828478)

[Scaffolding the Section Entity 40](#_Toc40828479)

[SectionController 41](#_Toc40828480)

[Authorize Attribute 41](#_Toc40828481)

[Index View 41](#_Toc40828482)

[Edit View 42](#_Toc40828483)

[Details View 42](#_Toc40828484)

[Delete View 43](#_Toc40828485)

[Update \_SiteAdminMenuPartial Section Link using @UrlAction 44](#_Toc40828486)

[Playing with Data 44](#_Toc40828487)

[Adding Controller – Actions - Views for Part / ItemType / ProductType / ProductLinkText Entities 44](#_Toc40828488)

[Authorize Attribute 44](#_Toc40828489)

[Item Entity : Scaffolding 44](#_Toc40828490)

[Controllers - Actions - Views 44](#_Toc40828491)

[Authorize Attribute 45](#_Toc40828492)

[Create Updates 45](#_Toc40828493)

[Controller Action – Create HTTP GET 45](#_Toc40828494)

[Controller Action – Create HTTP POST 45](#_Toc40828495)

[View 45](#_Toc40828496)

[Edit Updates 47](#_Toc40828497)

[Product Entity 47](#_Toc40828498)

[Creating ProductModel 47](#_Toc40828499)

[Conversion Extensions 47](#_Toc40828500)

[Scaffolding Controller/Action for the Product Entity 49](#_Toc40828501)

[Authorize Attribute 49](#_Toc40828502)

[Updating the SiteAdminMenuPartial 49](#_Toc40828503)

[Adding views 49](#_Toc40828504)

[Alter Index Action & View 50](#_Toc40828505)

[Action 50](#_Toc40828506)

[View 50](#_Toc40828507)

[Alter Create Action & View 51](#_Toc40828508)

[Action Create Get 51](#_Toc40828509)

[Action Create Post 52](#_Toc40828510)

[View Create 52](#_Toc40828511)

[Alter Edit Action & View 53](#_Toc40828512)

[Action Edit Get 53](#_Toc40828513)

[Action Edit Post 54](#_Toc40828514)

[View Edit 54](#_Toc40828515)

[Alter Details Action & View 55](#_Toc40828516)

[Action Detail 55](#_Toc40828517)

[View 55](#_Toc40828518)

[Alter Delete Action & View 56](#_Toc40828519)

[Add reference to System.Transactions 56](#_Toc40828520)

[Action Delete Get 56](#_Toc40828521)

[Action Delete Http Post 57](#_Toc40828522)

[View 57](#_Toc40828523)

[Subscription : Scaffolding 58](#_Toc40828524)

[Authorize Attribute 58](#_Toc40828525)

[Delete HTTP Post Action 59](#_Toc40828526)

[ProductItem Entity : Scaffolding 59](#_Toc40828527)

[Creating Controller 59](#_Toc40828528)

[Authorize Attribute 59](#_Toc40828529)

[Create ProductItemModel 60](#_Toc40828530)

[ProductItem to ProductItemModel Conversion Extensions 60](#_Toc40828531)

[Action & Views 62](#_Toc40828532)

[SubscriptionProduct Entity : Scaffolding 62](#_Toc40828533)

[Authorize Attribute 62](#_Toc40828534)

[SubscriptionProduct to SubscriptionProductModel Conversion Extensions 62](#_Toc40828535)

[Editing / Altering the User Info 63](#_Toc40828536)

[UserViewModel 63](#_Toc40828537)

[Identity Extensions 64](#_Toc40828538)

[Controller to Handle User Subscription 65](#_Toc40828539)

[UserController 65](#_Toc40828540)

[Index Action 66](#_Toc40828541)

[Index View 66](#_Toc40828542)

[Create Action 68](#_Toc40828543)

[Create View 69](#_Toc40828544)

[Edit Action 70](#_Toc40828545)

[Edit View 71](#_Toc40828546)

[Delete Action 72](#_Toc40828547)

[Delete View 73](#_Toc40828548)

[User Subscription 75](#_Toc40828549)

[Display/Handling Products 79](#_Toc40828550)

[Product Link Text – Admin Console 79](#_Toc40828551)

[Product Types – Admin Console 79](#_Toc40828552)

[Product – Admin Console 79](#_Toc40828553)

[Subscription – Admin Console 79](#_Toc40828554)

[Connect Product to Subscription – Admin Console 79](#_Toc40828555)

[Connect Users and Subscriptions – Admin Console 80](#_Toc40828556)

[Thumbnails 80](#_Toc40828557)

[Thumbnails Model 80](#_Toc40828558)

[ThumbnailArea Model 80](#_Toc40828559)

[Styling Thumbnails 80](#_Toc40828560)

[Thumbnails Partial View 80](#_Toc40828561)

[Fetching Subscriptions and Products 81](#_Toc40828562)

[Thumbnail Comparer 81](#_Toc40828563)

[Thumbnail Extensions 81](#_Toc40828564)

[HttpContext Extensions 82](#_Toc40828565)

[HomeModel 83](#_Toc40828566)

[Updating Site Index Page 83](#_Toc40828567)

[Updating Index Action of Home Controller 84](#_Toc40828568)

[Run the App 85](#_Toc40828569)

[Product Content 86](#_Toc40828570)

[Adding Data 86](#_Toc40828571)

[ProductItemRow Class 87](#_Toc40828572)

[ProductSection Class 87](#_Toc40828573)

[ProductSectionModel 87](#_Toc40828574)

[ContentViewModel 88](#_Toc40828575)

[ProductSectionEquality Comparer 88](#_Toc40828576)

[Section Extensions 88](#_Toc40828577)

[ProductContent Controller 90](#_Toc40828578)

[Styling the Index View 92](#_Toc40828579)

[Carret Styling 92](#_Toc40828580)

[Carret Javascript 93](#_Toc40828581)

[Run the App 93](#_Toc40828582)

[Product Content Details 94](#_Toc40828583)

[Register a Subscription 96](#_Toc40828584)

[\_SiteRegisterCodePartial 96](#_Toc40828585)

[Adding the \_SiteRegisterCodePartial view to Index view 97](#_Toc40828586)

[Styling the \_SiteRegisterCodePartial 97](#_Toc40828587)

[Subscription Extensions 97](#_Toc40828588)

[RegisterCode Controller 98](#_Toc40828589)

[Javascript SiteRegisterCode.js 99](#_Toc40828590)

[Add the Register Url in Home\Index View 100](#_Toc40828591)

[Run the App 100](#_Toc40828592)

# 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.



## RouteConfig.cs

Default route is specified inside the RouteConfig.cs file.

Go through the lectures for routing or take a quick look at the following for more details:

<https://www.c-sharpcorner.com/UploadFile/bhushangawale/attribute-based-routing-in-Asp-Net-mvc-5/>

Since we will be using an attribute-based route for [Product Content Details](#_Details_Action), make sue to add the highlighted row before the default route.



## 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

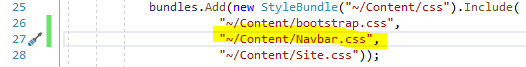
This entry to create the Site specific js bundle and then add the AiteAdminMenu js to the bundle created via step [Adding SiteAdminMenu.js to BundleConfig](#_Add_SiteAdmin.js_to) below.



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

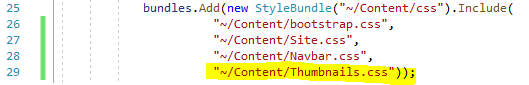
### V1.2 Adding Navbar.css Bundle

CSS added din this [step](#_Navbar.css)



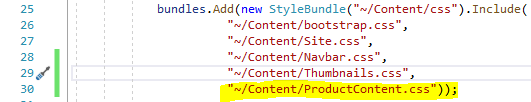
### V1.3 Adding Thumbnails.css Bundle

CSS added in this [step](#_Styling_Thumbnails)



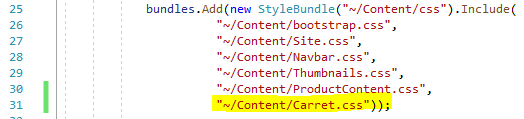
### V1.4 Adding ProductContent.css Bundle

CSS added in this [step](#_Styling_the_Index)



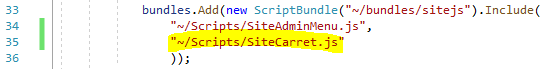
### V1.5 Adding Carret.css Bundle

CSS added in this [step](#_Carret_Styling)



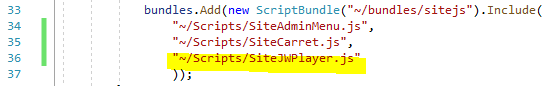
### V1.6 Adding SiteCarret.js Bundle

Javascript added in this [step](#_Carret_Javascript)



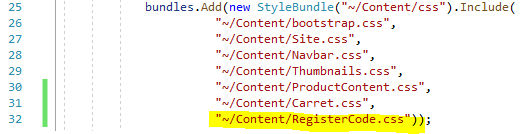
### V1.7 Adding JWPlayers.js Bundle

Javascript added in this [step](#_JWPlayer.js)



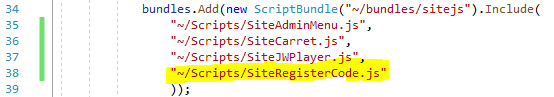
### V1.8 Adding RegisterCode.css Bundle

Css added in this [step](#_Styling_the__SiteRegisterCodePArtia).



### V1.9 Adding SiteRegisterCode.js Bundle

Javascript added in this [step](#_Javascript_SiteRegisterCode.js)



# 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.

# 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.



### V1.3 Making Adjustments

Check the layout changes under this [section](#_Layout_Changes).

## \_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.



# 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)
* With [UserSubscription](#_User_Subscription) this is the [extension](#_Subscription_Extensions)

## Identity Extensions

* With [UserViewModel](#_UserViewModel) this is the [extension](#_Identity_Extensions).
* This is used with Editing/Altering the user info.

## HttpContext Extensions

We are getting the User Id from Owin Context. The GetUserId method is used to get the users id. [Click here](#_HttpContext_Extensions) to view the extension. It is used inside the home controller [index](#_Updating_Index_Action) action.

## Thumbnail Extensions

These extensions are used to get the users subscriptions and products. [Click here](#_Thumbnail_Extensions) to see it in action.

## Section Extensions

There work with the [ProductionContent](#_Product_Content) below. [Click here](#_Section_Extensions) to see it in action.

# Comparers

These are comparers that are used to make our life easier. Create a folder in the root called “Comparers” and we’ll put the files here.

## ThumbnailEqualityComparer

[Click here](#_Thumbnail_Comparer) to view its details.

## ProductSectionEqualityComparer

[Click here](#_ProductSectionEquality_Comparer) to view its details.

# 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" })

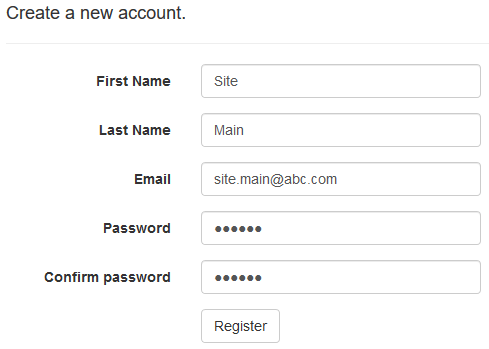
</div>

</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.

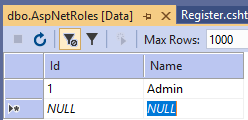
## Assigning the Admin Role

### 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

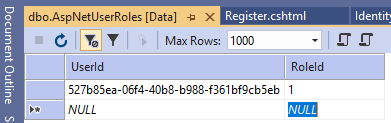
These are different roles that a user can have. Open AspNetRoles table and add a role to it. Take a note of the id that you have entered.



### 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.

## Editing / Altering User Info

Some of the things that we’ll need to complete this task are not yet complete. Due to this reason [this section is detailed below](#_Altering_the_User).

# Layout Changes

We’ll alter the view a bit. For this look at the following for more details.

## Navbar.css

We’ll add a new css for the navbar. Check \Content folder to view the content of the css file. We’ll add it to the bundle config [[V1.2](#_V1.2_Adding_Navbar.css)] as well.

## Views\Shared\\_Layout.cshtml

### From

<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>

@Html.Partial("\_SiteAdminMenuPartial");

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

### To

<div class="navbar 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 navbar-inverse"></span>

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

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

@Html.Partial("\_SiteAdminMenuPartial");

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

## Adding Google Fonts

Pick the fonts you are interested in and add to the head section.

<head>

<meta charset="utf-8" />

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

<link href='https://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'>

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

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

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

</head>

## \Admin\Views\Shares\\_Layout.cshtml

We have a separate layout for the admin pages. Copy the above \_layut changes to the admin area \_layout as well.

## LogOff Action Update

Open Account controller and then go to LogOff action. In current state if you logoff from the admin pages then you’ll get into an error on redirection after logoff. Here add Area = “” to the RedirectToAction.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("Index", "Home", new { Area = ""});

}

## \_LoginPartial.cshtml

Also, add the Area = “” to the form definition in \_LoginPartial.cshtml file.

using (Html.BeginForm("LogOff", "Account", new { Area = "" }, FormMethod.Post, new { id = "logoutForm", @class = "navbar-right" }))

## Home Index View

Change the default index view template as following. We’ll further modify this later.

@{

ViewBag.Title = "Home Page";

}

<div class="row">

@if (User.Identity.IsAuthenticated)

{

<div class="col-xs-12">

<h2>@User.Identity.Name <span>is logged in!</span></h2>

</div>

}

else

{

@\*left column\*@

<div class="col-lg-9 col-md-8 col-sm-7">

<h2>I am not logged in</h2>

</div>

@\*right column\*@

<div class="col-lg-3 col-md-4 col-sm-5">

<p>Right column</p>

</div>

}

</div>

[Click here](#_Updating_Site_Index) to see thumbnails for products getting added to the index page.

# JW Player Setup

JW Player is used by the [Product Content Details](#_Details_View) below.

## Basic Setup

* Go to <https://www.jwplayer.com/>
* Click Get Started
* I’ll do the setup with free version
* Provide email address to start for free
* You’ll get an activation email so click it. At this point you’ll be asked to create an account
* Then click on the Players section and you’ll get the pre-defined list of players. If none works for your liking then you can create a player
* Click Create Player
  + Name it “My Demo Site Player”
  + Players size make sure responsive 16:9 is selected
  + Select any other settings and then click Save
  + And finally click Close
* The new player should be in the list of the players
* Click your player
* And then pick the player library url
* Go to the \_Layout.cshtml and add script to the head section as V



## JWPlayer.js

Go to the scripts folder and add a new JavaScript file as SiteJWPlayer.js. It will have only one function

function jwVideo(video) {

jwplayer("video").setup({ file: video });

}

Add the JWPlayer to the [BundleConfig](#_V1.7_Adding_JWPlayers.js) (V1.7)

# 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

}

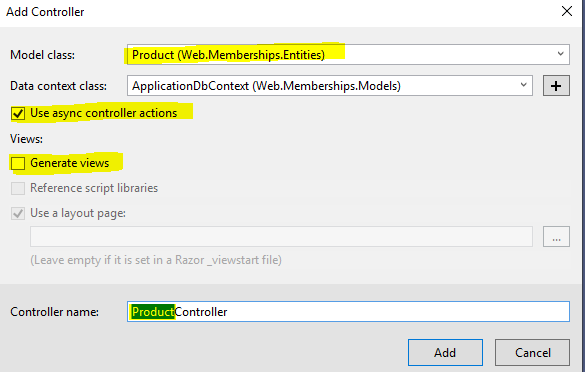
}

## 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")]

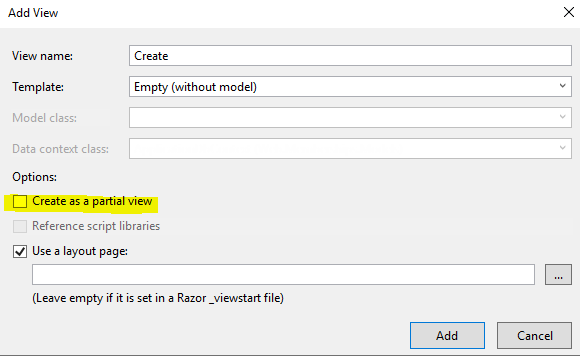
## Updating the SiteAdminMenuPartial

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

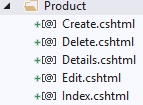
<li><a href="@Url.Action("Index", "Product", new { Area = "Admin" })">&nbsp;&nbsp;Product</a></li>

## 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

# Editing / Altering the User Info

Also check the [registration section](#_User_Handling_–) above for details. We’ll be passing the view model back and forth between the actions and views.

## UserViewModel

Create a new class UserViewModel inside Models folder in the root.

using System.ComponentModel.DataAnnotations;

namespace Web.Memberships.Models

{

public class UserViewModel

{

[Display(Name = "User Id")]

public string Id { get; set; }

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Display(Name = "First Name")]

[StringLength(30, ErrorMessage = "The {0} must be at least {1} characters long", MinimumLength = 2)]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[StringLength(30, ErrorMessage = "The {0} must be at least {1} characters long", MinimumLength = 2)]

public string LastName { get; set; }

[Required]

[StringLength(100)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

public int SubscriptionsCount { get; set; } = 0; //default to 0

}

}

## Identity Extensions

Just like in other sections we’ll create the Identity extension to fetch the user info. Check the rest of the extensions [here](#_Helper_Extensions). Create a class in \Extensions folder with name “IdentityExtensions”. Will create 4 methods to this extension to get the FirstName, LastName, FirstName and LastName, and all Users.

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Security.Principal;

using System.Threading.Tasks;

using System.Web;

using Web.Memberships.Models;

namespace Web.Memberships.Extensions

{

public static class IdentityExtensions

{

public static ApplicationUser GetUserInfo(this IIdentity identity)

{

var db = ApplicationDbContext.Create(); //Create ==> to get the instance

var user = db.Users.FirstOrDefault(u => u.UserName.Equals(identity.Name));

return user;

}

public static string GetUserFirstName(this IIdentity identity)

{

var user = identity.GetUserInfo();

return user != null ? user.FirstName : String.Empty;

}

public static string GetUserLastName(this IIdentity identity)

{

var user = identity.GetUserInfo();

return user != null ? user.LastName : String.Empty;

}

public static string GetUserFullName(this IIdentity identity)

{

var user = identity.GetUserInfo();

return user != null ? $"{user.FirstName} {user.LastName}" : String.Empty;

}

public static async Task GetUsers(this List<UserViewModel> users)

{

var db = ApplicationDbContext.Create();

users.AddRange(await (from u in db.Users

select new UserViewModel

{

Id = u.Id,

Email = u.Email,

FirstName = u.FirstName,

LastName = u.LastName

}).OrderBy(o => o.Email).ToListAsync());

}

}

}

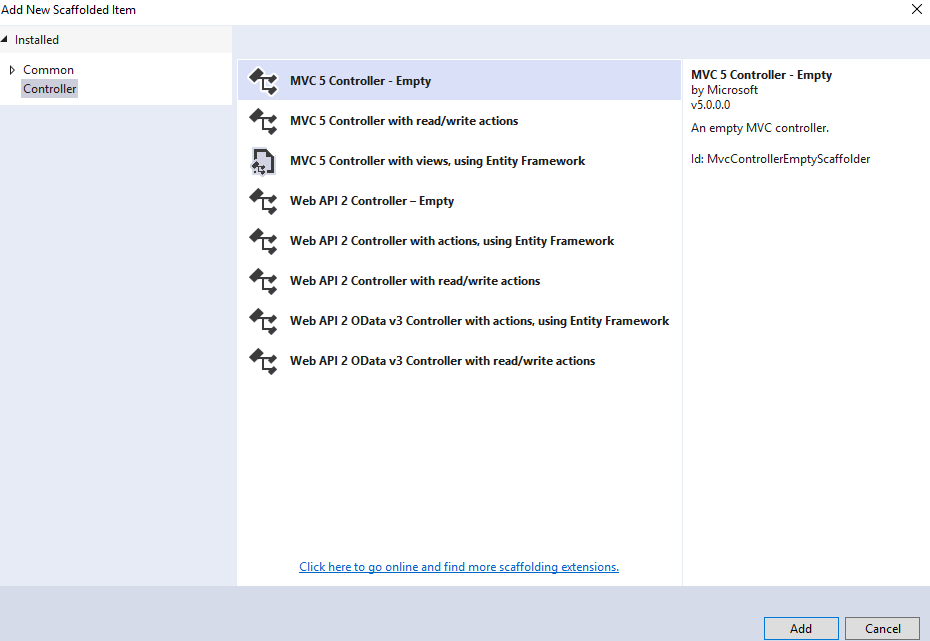
## Controller to Handle User Subscription

We can either create the action inside the Account controller or create a customer controller. I am opting for a User Controller.

### UserController

This time we won’t scaffold but create an empty MVC 5 controller and then we’ll build on top of it.

* Name the controller UserController inside the Controllers folders in the root.
* Also, apply Authorize attribute to it.



using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Web.Memberships.Extensions;

using Web.Memberships.Models;

namespace Web.Memberships.Controllers

{

[Authorize]

public class UserController : Controller

{

private ApplicationUserManager \_userManager;

public UserController()

{

}

public UserController(ApplicationUserManager userManager)

{

UserManager = userManager;

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

// GET: User

public ActionResult Index()

{

return View();

}

}

}

Don’t forget to update the Admin menu link to use @Url.Action

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

### Index Action

The index action will return the site Users. Add the following to the top

using System.Threading.Tasks;

using Web.Memberships.Extensions;

using Web.Memberships.Models;

and then the index action would look like

// GET: Users

public async Task<ActionResult> Index()

{

var users = new List<UserViewModel>();

//use the extension method to get the users

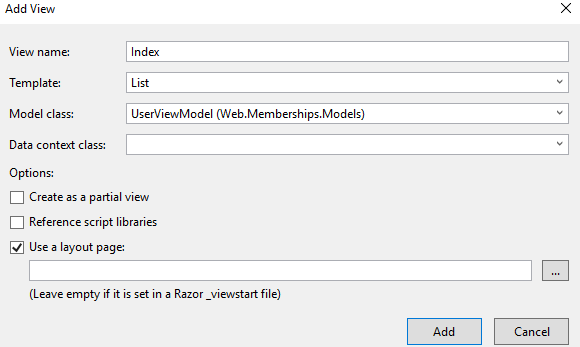
await users.GetUsers();

return View(users);

}

### Index View

Right click any where in the Index action and click Add View. This will scaffold the view for us. We’ll edit it as per our needs.



@model IEnumerable<Web.Memberships.Models.UserViewModel>

@using Web.Memberships.Models;

@{

ViewBag.Title = "Users & Subscriptions";

}

<h2>Users & Subscriptions</h2>

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

{

<p>

@Html.Partial("\_SiteCreateButtonPartial")

</p>

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

<tr class="success">

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

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

</tr>

}

</table>

}

### Create Action

We’ll create two actions

#### HttpGet

[Authorize(Roles = "Admin")]

public ActionResult Create()

{

return View();

}

#### HTTPPost

// POST: /User/Create

[HttpPost]

[Authorize(Roles = "Admin")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create(UserViewModel model)

{

try

{

if (model == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

if (ModelState.IsValid)

{

var user = new ApplicationUser

{

UserName = model.Email,

Email = model.Email,

FirstName = model.FirstName,

LastName = model.LastName,

IsActive = true,

Registered = DateTime.Now,

EmailConfirmed = true

};

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

AddErrors(result);

}

}

catch { }

// If we got this far something failed, re-display the form

return View(model);

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

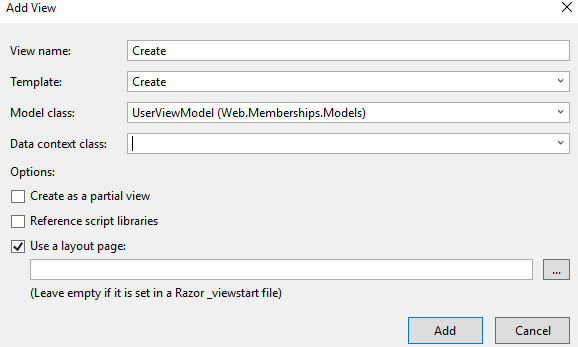
ModelState.AddModelError("", error);

}

}

### Create View

Right click any where in the Create action and click Add View.



@model Web.Memberships.Models.UserViewModel

@{

ViewBag.Title = "Create";

}

<h2>Create</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Create User</h4>

<hr />

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

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

@Html.ValidationMessageFor(model => model.Password, "", 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 btn-sm" />

</div>

</div>

</div>

}

<div>

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

### Edit Action

We’ll create two actions HttpGet and HttpPost

#### HttpGet

[Authorize(Roles = "Admin")]

public async Task<ActionResult> Edit(string userId)

{

if (string.IsNullOrWhiteSpace(userId))

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

ApplicationUser user = await UserManager.FindByIdAsync(userId);

if (user == null)

{

return HttpNotFound();

}

var model = new UserViewModel

{

Email = user.Email,

FirstName = user.FirstName,

LastName = user.LastName,

Id = user.Id,

Password = user.PasswordHash

};

return View(model);

}

#### HttpPost

[HttpPost]

[Authorize(Roles = "Admin")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Edit(UserViewModel model)

{

try

{

if (model == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

if (ModelState.IsValid)

{

var user = await UserManager.FindByIdAsync(model.Id);

if (user != null)

{

user.Email = model.Email;

user.UserName = model.Email;

user.FirstName = model.FirstName;

user.LastName = model.LastName;

if (!user.PasswordHash.Equals(model.Password))

{

user.PasswordHash = UserManager.PasswordHasher.HashPassword(model.Password);

}

var result = await UserManager.UpdateAsync(user);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

AddErrors(result);

}

}

}

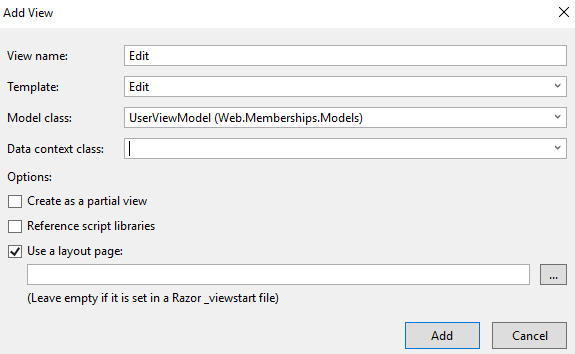
catch { }

return View(model);

}

### Edit View

Right click any where in the Edit action and click Add View.



@model Web.Memberships.Models.UserViewModel

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Edit User</h4>

<hr />

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

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

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

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

</div>

</div>

<div class="form-group">

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

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

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

@Html.ValidationMessageFor(model => model.Password, "", 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 btn-sm" />

</div>

</div>

</div>

}

<div>

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

### Delete Action

Very similar to Edit HttpGet and HttpPost, we’ll create the Delete HttpGet and HttpPost actions.

#### Delete HttpGet

[Authorize(Roles = "Admin")]

public async Task<ActionResult> Delete(string userId)

{

if (string.IsNullOrWhiteSpace(userId))

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

ApplicationUser user = await UserManager.FindByIdAsync(userId);

if (user == null)

{

return HttpNotFound();

}

var model = new UserViewModel

{

Email = user.Email,

FirstName = user.FirstName,

LastName = user.LastName,

Id = user.Id,

Password = "Fake"

};

return View(model);

}

#### Delete HttpPost

Make sure to delete the subscriptions as well.

[HttpPost]

[Authorize(Roles = "Admin")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Delete(UserViewModel model)

{

try

{

if (model == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

if (ModelState.IsValid)

{

var user = await UserManager.FindByIdAsync(model.Id);

if (user != null)

{

var result = await UserManager.DeleteAsync(user);

if (result.Succeeded)

{

//remove subscriptions as well

var db = new ApplicationDbContext();

var subscriptions = db.UserSubscriptions.Where(x => x.UserId.Equals(user.Id));

db.UserSubscriptions.RemoveRange(subscriptions);

await db.SaveChangesAsync(); //persist

return RedirectToAction("Index");

}

AddErrors(result);

}

}

}

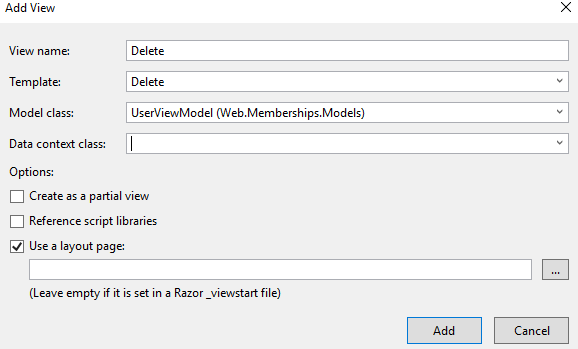
catch { }

return View(model);

}

### Delete View

Right click any where in the Delete action and add view. Select the following and click Add.



@model Web.Memberships.Models.UserViewModel

@{

ViewBag.Title = "Delete";

}

<h2>Delete</h2>

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

<div>

<h4>UserViewModel</h4>

<hr />

<dl class="dl-horizontal">

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dd>

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

</dd>

<dd>

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

</dd>

</dl>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

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

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

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

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

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

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

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

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

}

</div>

### User Subscription

We’ll be creating two view model and extension methods to handle user subscription data.

#### UserSubscriptionModel

Right click on the Model folder in the root and add a class “UserSubscriptionModel”.

using System;

using System.ComponentModel.DataAnnotations;

namespace Web.Memberships.Models

{

public class UserSubscriptionModel

{

public int Id { get; set; }

[MaxLength(255)]

[Required]

public string Title { get; set; }

[MaxLength(2048)]

public string Description { get; set; }

[MaxLength(20)]

public string RegistrationCode { get; set; }

public DateTime? StartDate { get; set; }

public DateTime? EndDate { get; set; }

}

}

#### UserSubscriptionViewModel

Next create a “UserSubscriptionViewModel”. One of its properties would be ICollection<UserSubscriptionModel> created [above](#_UserSubscriptionModel).

#### Subscription Add Actions

We’ll create HttpGet and HttpPost to “Subscription” actions in the UserController.

##### Subscription HttpGet

[Authorize(Roles = "Admin")]

public async Task<ActionResult> Subscriptions(string userId)

{

if (userId == null || userId.Equals(string.Empty))

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var model = new UserSubscriptionViewModel();

var db = new ApplicationDbContext();

//get the current user subscriptions

model.UserSubscriptions = await

(from us in db.UserSubscriptions

join s in db.Subscriptions on us.SubscriptionId equals s.Id

where us.UserId.Equals(userId)

select new UserSubscriptionModel

{

Id = us.SubscriptionId,

StartDate = us.StartDate,

EndDate = us.EndDate,

Description = s.Description,

RegistrationCode = s.RegistrationCode,

Title = s.Title

}).ToListAsync();

//get the ids for the current user subscriptions

var ids = model.UserSubscriptions.Select(us => us.Id);

//get the subscriptions that have not been subscribed to

model.Subscriptions = await db.Subscriptions.Where(s => !ids.Contains(s.Id)).ToListAsync();

//if no subscriptions available then lock the drop down

model.DisableDropDown = model.Subscriptions.Count.Equals(0);

model.UserId = userId;

return View(model);

}

##### Subscription HttpPost

[HttpPost]

[Authorize(Roles = "Admin")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Subscriptions(UserSubscriptionViewModel model)

{

try

{

if (model == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

if (ModelState.IsValid)

{

var db = new ApplicationDbContext();

db.UserSubscriptions.Add(new UserSubscription

{

UserId = model.UserId,

SubscriptionId = model.SubscriptionId,

StartDate = DateTime.Now,

EndDate = DateTime.MaxValue

});

await db.SaveChangesAsync();

}

}

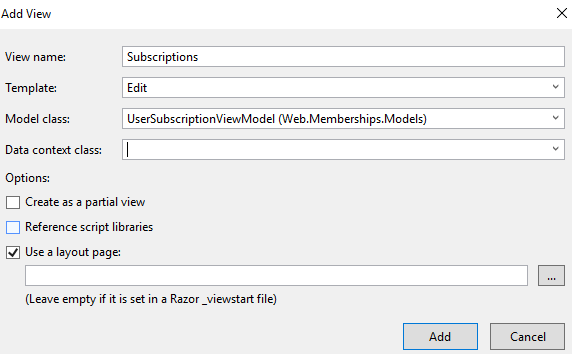
catch { }

return RedirectToAction("Subscriptions", "User", new { Area = "", userId = model.UserId });

}

#### Subscription View

Right click inside the Subscriptions action and click add view.



Make the following changes to the view

@model Web.Memberships.Models.UserSubscriptionViewModel

@using Web.Memberships.Extensions;

@{

ViewBag.Title = "Subscriptions";

}

<h2>Subscriptions</h2>

@\*Currently available subscriptions\*@

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Add Subscription to User</h4>

<hr />

@if (Model.DisableDropDown)

{

<div><strong>User has all the subscriptions</strong></div>

<br />

}

else

{

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

<div class="form-group">

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

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

@Html.DropDownListFor(model => model.SubscriptionId,

Model.Subscriptions.ToSelectListItem(Model.SubscriptionId),

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

@Html.ValidationMessageFor(model => model.SubscriptionId, "", 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 btn-sm" />

</div>

</div>

}

</div>

}

<div>

@Html.Partial("\_SiteBackToListButtonPartial")

</div>

<hr />

<br />

@\*Currently selected extensions\*@

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

<tr class="success">

<th>Id</th>

<th>Title</th>

<th>Code</th>

<th>Start Date</th>

<th>End Date</th>

<th></th>

</tr>

@foreach (var item in Model.UserSubscriptions)

{

<tr>

<td>@Html.DisplayFor(modelItem => item.Id)</td>

<td>@Html.DisplayFor(modelItem => item.Title)</td>

<td>@Html.DisplayFor(modelItem => item.RegistrationCode)</td>

<td>@Html.DisplayFor(modelItem => item.StartDate)</td>

<td>@Html.DisplayFor(modelItem => item.EndDate)</td>

<td>

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

href="@Url.Action("RemoveUserSubscription")?subscriptionId=@item.Id&userId=@Model.UserId">

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

</a>

</td>

</tr>

}

</table>

#### RemoveUserSubscriptions Action

##### HttpGet Action

[Authorize(Roles = "Admin")]

public async Task<ActionResult> RemoveUserSubscription(string userId, int subscriptionId)

{

try

{

if (userId == null || userId.Length.Equals(0) || subscriptionId <= 0)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

if (ModelState.IsValid)

{

var db = new ApplicationDbContext();

var subscriptions = db.UserSubscriptions.Where(

us => us.UserId.Equals(userId) &&

us.SubscriptionId.Equals(subscriptionId));

db.UserSubscriptions.RemoveRange(subscriptions);

await db.SaveChangesAsync();

}

}

catch { }

return RedirectToAction("Subscriptions", "User", new { Area = "", userId = userId });

}

# Display/Handling Products

Following tasks we’ll accomplish by logging in as an admin.

## Product Link Text – Admin Console

Go to Product Link Text and add “Read more +”.

## Product Types – Admin Console

Go to the Product type and add following products “Course”, “Book”, “Articles” and “Misc.”.

## Product – Admin Console

Go to Product and add following products

1. C# For Beginners
   1. Title: C# For Beginners
   2. Description: Content for the book
   3. Image URL: /Content/Images/csharp-for-beginners.png
   4. Product Type: select Book
   5. Product Link Text: select Read more +
2. Unit Testing
   1. Title: Unit Testing
   2. Description: How to unit test in MVC
   3. Image URL: /Content/Images/unit-testing.png
   4. Product Type: select Articles
   5. Product Link Text: select Read more +

## Subscription – Admin Console

Go to Subscriptions and add the following

1. Subscription 1
   1. Title: C# For Beginners Subscriptions
   2. Description: Course material for the C# for beginners course
   3. C#1

## Connect Product to Subscription – Admin Console

Go to Subscription Product and add the following

1. Item 1
   1. Products: C# For Beginners
   2. Subscriptions: C# for Beginners Subscription
2. Item 2
   1. Products: Unit Testing
   2. Subscriptions: C# For Beginners Subscriptions

## Connect Users and Subscriptions – Admin Console

We don’t have a form yet where the users themselves can add the subscriptions so we will at this time do it through the Admin Console. Go to the Users & Subscriptions

1. Click subscription button right next to the admin, select subscription C# For Beginners and click Save.
2. Click subscription button right next to the General user, select subscription C# For Beginners and click Save.

## Thumbnails

On the products page we will be displaying products with thumb nails.

### Thumbnails Model

Create a class in \Models with name “ThumbnailModel”. Check the contents of this class for more details.

### ThumbnailArea Model

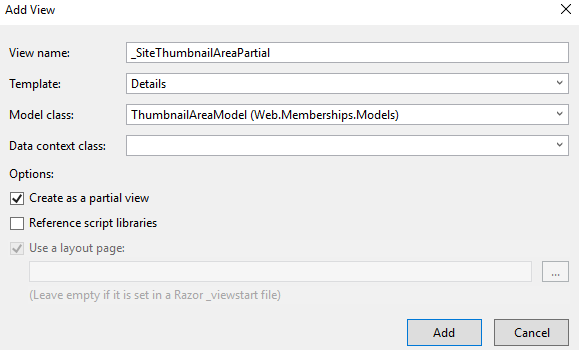
This class will have IEnumerable<ThumbnailModel>. Create a class with name “ThumbnailAreaModel”. In the \Models folder. Check the contents of this class for more details.

### Styling Thumbnails

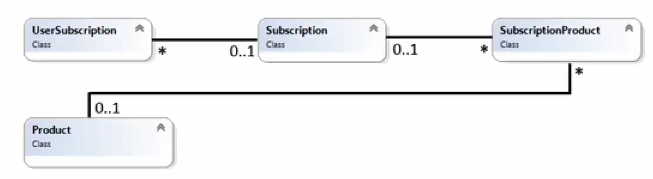
We’ll add the styles to the thumbnails.css and then add it to the [BundleConfig](#_V1.3_Adding_Thumbnails.css) (V1.3). Check the content of \Content\Thumbnails.css for details.

### Thumbnails Partial View

Next we will add the partial view to display each thumb nail. Check \Views\Shared\\_SiteThumbnailAreaPartial.cshtml for details. Change the content of the file once its added.



## Fetching Subscriptions and Products



### Thumbnail Comparer

All of the comparers are [here](#_Comparers). Click a new class with name “ThumbnailEqualityComparer” in folder \Comparers.

using System.Collections.Generic;

using Web.Memberships.Models;

namespace Web.Memberships.Comparers

{

public class ThumbnailEqualityComparer : IEqualityComparer<ThumbnailModel>

{

public bool Equals(ThumbnailModel thumb1, ThumbnailModel thumb2)

{

return thumb1.ProductId.Equals(thumb2.ProductId);

}

public int GetHashCode(ThumbnailModel thumb)

{

return thumb.ProductId;

}

}

}

### Thumbnail Extensions

Our other extensions could be followed [here](#_Helper_Extensions_1). Add new class in the Extensions folder with name “ThumbnailExtensions”.

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using Web.Memberships.Comparers;

using Web.Memberships.Models;

namespace Web.Memberships.Extensions

{

public static class ThumbnailExtensions

{

private static async Task<List<int>> GetSubscriptionIdsAsync(string userId = null, ApplicationDbContext db = null)

{

try

{

if (userId == null) return new List<int>();

if (db == null) db = ApplicationDbContext.Create();

return await (

from us in db.UserSubscriptions

where us.UserId.Equals(userId)

select us.SubscriptionId).ToListAsync();

}

catch { }

return new List<int>();

}

public static async Task<IEnumerable<ThumbnailModel>> GetProductThumbnailsAsync(this List<ThumbnailModel> thumbnails, string userId = null, ApplicationDbContext db = null)

{

try

{

if (userId == null) return new List<ThumbnailModel>();

if (db == null) db = ApplicationDbContext.Create();

var subscriptionIds = await GetSubscriptionIdsAsync(userId, db);

thumbnails = await (

from ps in db.SubscriptionProducts

join p in db.Products on ps.ProductId equals p.Id

join plt in db.ProductLinkTexts on p.ProductLinkTextId equals plt.Id

join pt in db.ProductTypes on p.ProductTypeId equals pt.Id

where subscriptionIds.Contains(ps.SubscriptionId)

select new ThumbnailModel

{

ProductId = p.Id,

SubscriptionId = ps.SubscriptionId,

Title = p.Title,

Description = p.Description,

ImageUrl = p.ImageUrl,

Link = "/ProductContent/Index/" + p.Id,

TagText = plt.Title,

ContentTag = pt.Title

}).ToListAsync();

}

catch { }

return thumbnails.Distinct(new ThumbnailEqualityComparer()).OrderBy(o => o.Title);

}

}

}

## HttpContext Extensions

Our other extensions could be followed [here](#_Helper_Extensions_1). Create a new static class with name “HttpContextExtensions” in \Extensions folder. We’ll get the UserId using the Owin Context.

* Owen content works with claims when handling identities
* The claim we are interested in getting the userid is called name identifier.
  + <http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier>
* It is used inside the home controller [index](#_Updating_Index_Action) action.
* In the \_LoginPartial view we are using User.Identity.GetUserName() to get the logged in users id. This is the email since it is used to login. We are interested in the user GUID so this method is helping with it.

using Microsoft.AspNet.Identity.Owin;

using System.Linq;

using System.Security.Claims;

using System.Web;

namespace Web.Memberships.Extensions

{

public static class HttpContextExtensions

{

private const string nameidentifier = "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier";

public static string GetUserId(this HttpContextBase ctx)

{

var uid = string.Empty;

try

{

var claims = ctx

.GetOwinContext()

.Get<ApplicationSignInManager>()

.AuthenticationManager

.User

.Claims

.FirstOrDefault(claim => claim.Type.Equals(nameidentifier));

if (claims != default(Claim))

uid = claims.Value;

}

catch { }

return uid;

}

}

}

## HomeModel

HomeModel will be used by the Home Index Action and View. Create a new class with name “HomeModel” in the Models folder.

using System.Collections.Generic;

namespace Web.Memberships.Models

{

public class HomeModel

{

public List<ThumbnailAreaModel> ThumbnailsArea { get; set; }

//following fields pulled via User.Identity in Home Controller Index action

public string UserIdentityUserId { get; set; }

public string UserIdentityFirstName { get; set; }

public string UserIdentityLastName { get; set; }

public string UserIdentityId { get; set; }

public string DisplayUserIdentityName => $"{UserIdentityFirstName} {UserIdentityLastName}";

//using the HttpContent GetuserId

public string HttpContextId { get; set; }

}

}

## Updating Site Index Page

We’ll add the thumb nails partial view to the index view. Add the model directive and then call the partial view to Authenticated section. Check index view main change [here](#_Home_Index_View). Check the content of the index view to see the changes.

@model Web.Memberships.Models.HomeModel

@{

ViewBag.Title = "Home Page";

}

<div class="row">

@if (User.Identity.IsAuthenticated)

{

if (Model != null && Model.ThumbnailsArea != null && Model.ThumbnailsArea.Any())

{

foreach (var area in Model.ThumbnailsArea)

{

@Html.Partial("\_SiteThumbnailAreaPartial", area)

}

}

}

else

{

@\*left column\*@

<div class="col-lg-9 col-md-8 col-sm-7">

<h2>I am not logged in</h2>

</div>

@\*right column\*@

<div class="col-lg-3 col-md-4 col-sm-5">

<p>Right column</p>

</div>

}

</div>

## Updating Index Action of Home Controller

For now, we will pass in an empty model to the index view. We’ll get back to this later.

Make sue to use the following using statements.

using Microsoft.AspNet.Identity;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Web.Mvc;

using Web.Memberships.Extensions;

using Web.Memberships.Models;

And then change the Index action as following. At this time you can run the app and see the home page in action.

public async Task<ActionResult> Index()

{

var model = new HomeModel();

/\*

We can get the user's info like following or we'll use the extension we created to get user id.

Leaving both here for

\*/

if (Request.IsAuthenticated)

{

//put a break point and run through following ad check each value for more details.

//getting from User.Identity

model.UserIdentityUserId = User.Identity.GetUserName();

model.UserIdentityFirstName = User.Identity.GetUserFirstName();

model.UserIdentityLastName = User.Identity.GetUserLastName();

model.UserIdentityId = User.Identity.GetUserId();

//getting the id from the HttpContext extension GetUserId

model.HttpContextId = HttpContext.User.Identity.GetUserId();

//get all the thumbnails

var thumbnails = await new List<ThumbnailModel>().GetProductThumbnailsAsync(model.HttpContextId);

if (thumbnails.Any())

{

//thumbnails per area

var area = 4;

var count = thumbnails.Count() / area;

var thumnailsArea = new List<ThumbnailAreaModel>();

for(int i=0; i<= count; i++)

{

var areaData = new ThumbnailAreaModel() {

Title = i.Equals(0) ? "My Content" : string.Empty,

Thumbnails = thumbnails.Skip(i \* area).Take(area)

};

thumnailsArea.Add(areaData);

}

model.ThumbnailsArea = new List<ThumbnailAreaModel>();

model.ThumbnailsArea.AddRange(thumnailsArea);

}

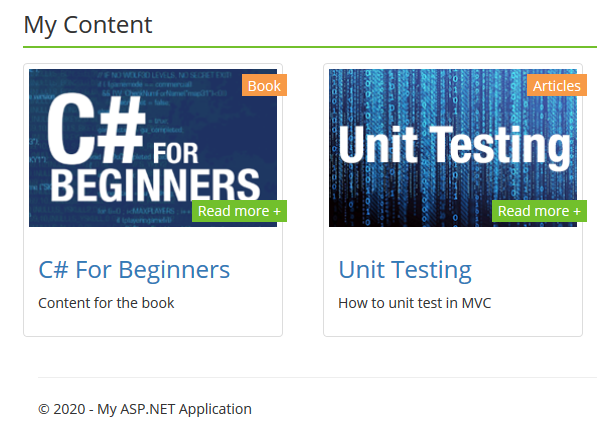
}

return View(model);

}

## Run the App

Run the app, login with users for whom you have assigned some subscriptions and you should see following on the main page.



When clicking the ReadMore or the title, it will take to ProductContent controller index action. Which we’ll tackle [Next](#_Product_Content).

## Product Content

### Adding Data

#### Part

Go to Admin menu and add part as “Part 1”

#### Section

Go to Admin menu and add a section as

1. Module 1
2. Module 2
3. Downlaods

#### Item Type

Go to Admin menu and add Item Types as

1. Video
2. Article
3. Book
4. Download

#### Item

Above part, section, item type is needed to create a part. Here create following items

##### Item 1 - Video

|  |  |
| --- | --- |
| Field | Value |
| Title | Video 1 |
| Description | Description for vide 1 |
| Url | https://www.youtube.com/watch?v=N6srbKfNcV4 |
| Image Url | /Content/Images/video.png |
| HTML | Keep blank |
| Wait Days | 0 |
| Item Types | Video |
| Sections | Module 1 |
| Parts | Part 1 |
| Is Free | Unchecked |

##### Item 2 - Article

|  |  |
| --- | --- |
| Field | Value |
| Title | Article 1 |
| Description | Description for article 1 |
| Url | Keep blank |
| Image Url | /Content/Images/laptop.jpg |
| HTML | <h1>Item article heading</h1>  <h2>Item article text</h1> |
| Wait Days | 0 |
| Item Types | Article |
| Sections | Module 1 |
| Parts | Part 1 |
| Is Free | Unchecked |

##### Item 3 - PDF

|  |  |
| --- | --- |
| Field | Value |
| Title | PDF 1 |
| Description | Description for PDF 1 |
| Url | /Content/documents/test\_pdf.pd |
| Image Url | /Content/Images/pdf.png |
| HTML |  |
| Wait Days | 0 |
| Item Types | Download |
| Sections | Downloads |
| Parts | Part 1 |
| Is Free | Unchecked |

#### Product Item

Go to Admin menu and click Product Item

|  |  |  |
| --- | --- | --- |
| # | Products | Items |
| 1 | C# for Beginners | Video 1 |
| 2 | C# for Beginners | Article 1 |
| 3 | C# for Beginners | PDF 1 |
| 4 | Unit Testing | Video 1 |
| 5 | Unit Testing | Article 1 |
| 6 | Unit Testing | PDF 1 |

#### Product Subscription

Make sure that we have Product Subscription that we added [above](#_Product_–_Admin).

### ProductItemRow Class

We’ll create a new class with name “ProductItemRow” inside the Models folder. Check the contents of the file for more details.

### ProductSection Class

We’ll create a new class with name “ProductSection” inside the Models folder. Check the contents of the file for more details.

### ProductSectionModel

We’ll create a new model class with name “ProductSectionModel” inside the Models folder. Check the contents of the file for more details.

### ContentViewModel

We’ll create a new model class with name “ContentViewModel” inside the Models folder. Check the contents of the file for more details.

### ProductSectionEquality Comparer

All of the comparers are specified [above](#_Comparers). Here we will use an equality comparer to be used in for getting the distinct list. Create a new class with name “ProductSectionEqualityComparer” in the Comparers folder.

using System.Collections.Generic;

using Web.Memberships.Models;

namespace Web.Memberships.Comparers

{

public class ProductSectionEqualityComparer : IEqualityComparer<ProductSection>

{

public bool Equals(ProductSection section1, ProductSection section2)

{

return section1.Id.Equals(section2.Id);

}

public int GetHashCode(ProductSection section)

{

return (section.Id).GetHashCode();

}

}

}

### Section Extensions

All of the sections are specified [above](#_Helper_Extensions_1). Create a new class with name “SectionExtensions” inside the Extensions folder.

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using Web.Memberships.Comparers;

using Web.Memberships.Models;

namespace Web.Memberships.Extensions

{

public static class SectionExtensions

{

public static async Task<ProductSectionModel> GetProductSectionsAsync(int productId, string userId)

{

var db = ApplicationDbContext.Create();

var query = from p in db.Products

join pi in db.ProductItems on p.Id equals pi.ProductId

join i in db.Items on pi.ItemId equals i.Id

join s in db.Sections on i.SectionId equals s.Id

where p.Id.Equals(productId)

orderby s.Title

select new ProductSection

{

Id = s.Id,

ItemTypeId = i.ItemTypeId,

Title = s.Title

};

var sections = await (query).ToListAsync();

foreach (var section in sections)

{

section.Items = await GetProductItemRowsAsync(productId, section.Id, userId);

}

var result = sections.Distinct(new ProductSectionEqualityComparer()).ToList();

var union = result.Where(r => !r.Title.ToLower().Contains("download"))

.Union(result.Where(r => r.Title.ToLower().Contains("download")));

var productQuery = from p in db.Products

where p.Id.Equals(productId)

select p.Title;

var title = await (productQuery).FirstOrDefaultAsync();

var model = new ProductSectionModel

{

Sections = union.ToList(),

Title = title

};

return model;

}

public static async Task<IEnumerable<ProductItemRow>> GetProductItemRowsAsync(

int productId, int sectionId, string userId, ApplicationDbContext db = null)

{

if (db == null) db = ApplicationDbContext.Create();

var today = DateTime.Now.Date;

var query = from i in db.Items

join it in db.ItemTypes on i.ItemTypeId equals it.Id

join pi in db.ProductItems on i.Id equals pi.ItemId

join sp in db.SubscriptionProducts on pi.ProductId equals sp.ProductId

join us in db.UserSubscriptions on sp.SubscriptionId equals us.SubscriptionId

where i.SectionId.Equals(sectionId) &&

//i.ItemTypeId.Equals(itemTypeId) &&

pi.ProductId.Equals(productId) &&

us.UserId.Equals(userId)

orderby i.PartId

select new ProductItemRow

{

ItemId = i.Id,

Description = i.Description,

Title = i.Title,

Link = it.Title.Equals("Download") ? i.Url : "/ProductContent/Content/" + pi.ProductId + "/" + i.Id,

ImageUrl = i.ImageUrl,

ReleaseDate = DbFunctions.CreateDateTime(us.StartDate.Value.Year,

us.StartDate.Value.Month, us.StartDate.Value.Day + i.WaitDays, 0, 0, 0),

IsAvailable = DbFunctions.CreateDateTime(today.Year,

today.Month, today.Day, 0, 0, 0) >= DbFunctions.CreateDateTime(us.StartDate.Value.Year,

us.StartDate.Value.Month, us.StartDate.Value.Day + i.WaitDays, 0, 0, 0),

IsDownload = it.Title.Equals("Download")

};

var items = await (query).ToListAsync();

return items;

}

public static async Task<ContentViewModel> GetContentAsync(int procuctId, int itemId)

{

var db = ApplicationDbContext.Create();

var query = from i in db.Items

join it in db.ItemTypes on i.ItemTypeId equals it.Id

where i.Id.Equals(itemId)

select new ContentViewModel

{

ProductId = procuctId,

HTML = i.HTML,

VideoURL = i.Url,

Title = i.Title,

Description = i.Description

};

var model = await (query).FirstOrDefaultAsync();

return model;

}

}

}

### ProductContent Controller

Right click on the Controller folder and click add controller. In this case we’ll add an empty MVC 4 controller with name “ProductContent”.

This controller needs to be behind login so apply [Authorize] attribute to it.

#### Index Action

Here will get the product sections using the [extension method](#_Section_Extensions) specified above.

using System.Threading.Tasks;

using System.Web.Mvc;

using Web.Memberships.Extensions;

namespace Web.Memberships.Controllers

{

[Authorize]

public class ProductContentController : Controller

{

// GET: ProductContent

public async Task<ActionResult> Index(int id)

{

var userId = Request.IsAuthenticated ? HttpContext.GetUserId() : null;

var sections = await SectionExtensions.GetProductSectionsAsync(id, userId);

return View(sections);

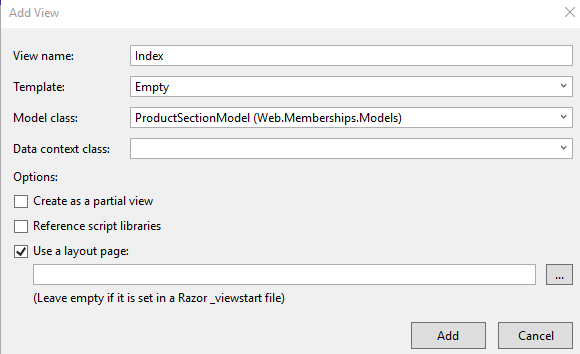
}

}

}

#### Index View

Right click in the index section and click add view. Check the view for more details.



Please note that this view works with following two partial views as well. Open the ProductContent views folder and look at those as well.

#### \_ProductItemRowPartial

The link is to the [details](#_Product_Content_Details) action which we’ll setup later.

@model Web.Memberships.Models.ProductItemRow

<div class="row product-item">

<div class="col-sm-2">

<img src="@Model.ImageUrl" width="140" />

</div>

<div class="col-sm-10 text-color line-bottom min-height">

@if (!Model.IsAvailable)

{

<p>Will be relesed: @Model.ReleaseDate.Value.ToShortDateString()</p>

}

<h2>

@if (Model.IsAvailable)

{

if (Model.IsDownload)

{

<a href="@Model.Url" target="\_blank">@Model.Title</a>

}

else

{

<a href="@Url.Action("Details", "ProductContent", new { Area = "", productId = Model.ProductId, itemId = Model.ItemId })">@Model.Title</a>

}

}

else

{

@Model.Title

}

</h2>

<p>@Model.Description</p>

</div>

</div>

<br />

#### \_ProductSectionPartial

@model IEnumerable<Web.Memberships.Models.ProductSection>

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

{

foreach (var s in Model)

{

<div class="panel panel-success">

<div class="panel-heading">

<h4 class="panel-title">

@\*

SiteCarret.js is used to change the play icon

data attribute is used by javascirpt to to change the title dynamically

\*@

<a data-toggle="collapse" class="panel-carret" href="#collapse@{@s.Id}" title="Click to collapse @s.Title" data="@s.Title.Replace("\"","'")">

<span class="pull-left glyphicon glyphicon-play gly-rotate-90"></span>

@s.Title

</a>

</h4>

</div>

<div id="collapse@{@s.Id}" class="panel-collapse collapse in">

<div class="panel-body">

@if (s.Items != null && s.Items.Any())

{

foreach (var item in s.Items)

{

@Html.Partial("\_ProductItemRowPartial", item)

}

}

</div>

</div>

</div>

<br />

}

}

### Styling the Index View

We’ll add the ProductContent style sheet and then add it to [bundling](#_V1.4_Adding_ProductContent.css) (V1.4) as well. Look at the ProductContent.css for more details.

### Carret Styling

We have added the Carret.css style sheet and then add it to [bundling](#_V1.5_Adding_Carret.css) (V1.5) as well. Look at the Carrent.css for more details.

/\*when expanded\*/

.gly-rotate-90 {

-webkit-transform: rotate(90deg);

-moz-transform: rotate(90deg);

-ms-transform: rotate(90deg);

-o-transform: rotate(90deg);

transform: rotate(90deg);

}

.panel-carret {

display: block;

}

.panel-carret .glyphicon-play.gly-rotate-90 {

margin-top: -1px;

}

.panel-carret .glyphicon-play {

margin-top: -4px;

margin-right: 4px;

}

.panel-carret span {

transition: all 0.3s ease 0s;

transform: rotate(90deg);

}

/\*remove the carret in collapse state\*/

.panel-carret.pressed span {

transform: rotate(0deg);

}

### Carret Javascript

Check SiteCarret.js in \Scripts folder. Then we need to add it to the [BundleConfig](#_V1.6_Adding_SiteCarret.js) (V1.6). This function rotates the play carrot up/down.

/\*

ProductContent

When the panel is collapsed then we want to remove the gly-rotate-90 class. We are using toggleClass function

We also want to toggle pressed class

check Carret.css for details

\*/

$(function () {

$(".panel-carret").click(function (e) {

//pressed class is rotating back to 0deg

$(this).toggleClass("pressed");

//change the title attribute accordingly as well

var state = "collapse"; //default – expand state

if ($(this).hasClass("pressed")) {

state = "expand"; //collapsed state

}

var data = $(this).attr("data");

$(this).attr("title", "Click to " + state + " " + data);

//glyphicon-play is a child element

$(this).children("glyphicon-play").toggleClass("gly-rotate-90");

//stop any default behavior for this a element

e.preventDefault();

});

});

### Run the App

Run the app, login and then go to home page. Click the read more on any product. Here’s how it will look.

Mouse over the green bar to look at the title and how it changes dynamically with expand and collapse state.

|  |  |
| --- | --- |
| Initial | After Click |
|  |  |

### Product Content Details

#### Details Action

Add the details action and it will use the GetContentAsync method that we have created above when creating the [SectionExtensions](#_Section_Extensions). We have created the [ContentViewModel](#_ContentViewModel) class above as well.

Also we will be creating a route here so that rather than

/ProductContent/Details?productId=4&itemId=3

We should use it as following

/ProductContent/Details/4/3

In the \_ProductItemRowPartial we are still using the Url.Action.

Go through the lectures for routing or take a quick look at the following for more details

<https://www.c-sharpcorner.com/UploadFile/bhushangawale/attribute-based-routing-in-Asp-Net-mvc-5/>

Also double check the [RouteConfg.cs](#_RouteConfig.cs) for registering Attribute based routing.

Add the following using statement

using System.Web.Routing;

and then the action code

[Route("ProductContent/Details/{productId:int}/{itemId:int}")]

public async Task<ActionResult> Details(int productId, int itemId)

{

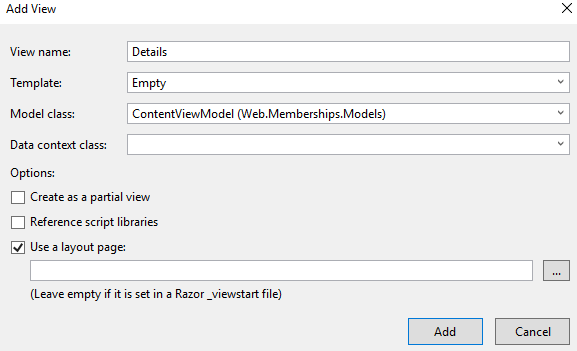
var model = await SectionExtensions.GetContentAsync(productId, itemId);

return View(model);

}

#### Details View

Right click any where in the Detail action and then click Add View.



For the videos, we are working with the JWPlayer, [all setup done](#_JW_Player_Setup) above.

@model Web.Memberships.Models.ContentViewModel

@{

ViewBag.Title = "Details";

}

<div class="pc-headline">

<h2>@Model.Title</h2>

<a class="btn btn-info btn-sm pull-right" href="@Url.Action("Index", "PeoductContent", new { Area = "", id = Model.ProductId })">

<span class="glyphicon glyphicon-arrow-left"></span>

Back

</a>

</div>

<div>@Model.Description</div>

@if (Model.HTML != null)

{

<div class="article">@Html.Raw(Model.HTML)</div>

}

@if (Model.VideoURL != null)

{

<div id="video" class="video-margin">&nbsp;</div>

<div id="hiddenUrl" hidden="hidden">@Model.VideoURL</div>

@\*only fire video play back in case it is a video\*@

@section scripts

{

<script type="text/javascript">

$(function () {

jwVideo($("#hiddenUrl").text());

});

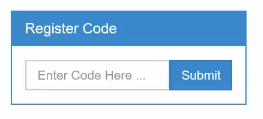
</script>

}

}

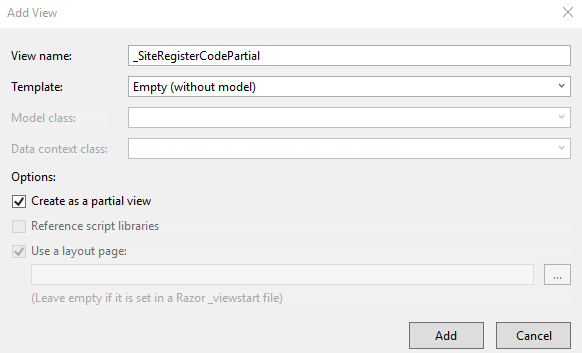
# Register a Subscription

This is a small form that displays on the main page to register.



## \_SiteRegisterCodePartial

We’ll add a partial view to the main view named “\_SiteRegisterCodePartial”. Right click the Views\Shared folder and click Add View



It will create an empty view.

The view needs to display for Authenticated users only.

@if (Request.IsAuthenticated)

{

<div class="panel panel-primary register-code-panel">

<div class="panel-heading">

<h4 class="panel-title">Register Code</h4>

</div>

<div class="panel-body input-group">

<input type="text" class="form-control" placeholder="Enter code here ..." />

@\*To display the button infront of the text box\*@

<span class="input-group-btn">

<button class="btn btn-primary" type="button">Submit</button>

</span>

</div>

<div class="alert alert-danger hidden">Could not register code</div>

</div>

}

### Adding the \_SiteRegisterCodePartial view to Index view

Then will need to add the partial view to site home index view. Add it towards the bottom of the view

@\*Register Code Partial\*@

<div class="row">

<div class="col-sm-3 register-code-area">

@Html.Partial("\_SiteRegisterCodePartial")

</div>

<div class="col-sm-9">

</div>

</div>

## Styling the \_SiteRegisterCodePartial

Right click the content folder and click Add Stylesheet. Add the stylesheet with name “RegisterCode”. After that add it to the [BundleConfig](#_V1.8_Adding_RegisterCode.css) (v1.8). Check the contents of the css file for more details.

## Subscription Extensions

Our current extensions are above and can be looked at by [clicking here](#_Helper_Extensions). We’ll have three extensions to get subscription id and to register.

using System;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using Web.Memberships.Entities;

using Web.Memberships.Models;

namespace Web.Memberships.Extensions

{

public static class SubscriptionExtensions

{

public static async Task<int> GetSubscriptionIdByRegistrationCode(this IDbSet<Subscription> subscription, string code)

{

try

{

if (subscription == null || string.IsNullOrWhiteSpace(code))

return Int32.MinValue;

var subscriptionId = await (

from s in subscription

where s.RegistrationCode.Equals(code)

select s.Id).FirstOrDefaultAsync();

return subscriptionId;

}

catch { return Int32.MinValue; }

}

public static async Task Register(this IDbSet<UserSubscription> userSubscription, int subscriptionId, string userId)

{

try

{

if (userSubscription == null || subscriptionId.Equals(Int32.MinValue) || userId.Equals(string.Empty))

return;

var exist = await Task.Run(() => userSubscription.CountAsync(

s => s.SubscriptionId.Equals(subscriptionId) &&

s.UserId.Equals(userId))) > 0;

if (!exist)

await Task.Run(() => userSubscription.Add(

new UserSubscription

{

UserId = userId,

SubscriptionId = subscriptionId,

StartDate = DateTime.Now,

EndDate = DateTime.MaxValue

}));

}

catch { }

}

public static async Task<bool> RegisterUserSubscriptionCode(string userId, string code)

{

try

{

var db = ApplicationDbContext.Create();

var id = await db.Subscriptions.GetSubscriptionIdByRegistrationCode(code);

if (id <= 0) return false;

await db.UserSubscriptions.Register(id, userId);

if (db.ChangeTracker.HasChanges())

await db.SaveChangesAsync();

return true;

}

catch { return false; }

}

}

}

## RegisterCode Controller

Next we will create a controller and action. Here we’ll be doing an ajax call utilizing jQuery. Create an empty MVC 5 controller with name “RegisterCode” in the main controllers folder. It will have single “Register” action.

using System;

using System.Threading.Tasks;

using System.Web.Mvc;

using Web.Memberships.Extensions;

namespace Web.Memberships.Controllers

{

public class RegisterCodeController : Controller

{

// GET: RegisterCode

public async Task<ActionResult> Register(string code)

{

if (Request.IsAuthenticated)

{

var userId = HttpContext.GetUserId();

var registred = await SubscriptionExtensions.RegisterUserSubscriptionCode(userId, code);

if (!registred) throw new ApplicationException();

return PartialView("\_SiteRegisterCodePartial");

}

return View();

}

}

}

## Javascript SiteRegisterCode.js

Create a new javascript file with name “SiteRegisterCode” in the Scripts folder and add it to the [BundlesConfig](#_V1.9_Adding_SiteRegisterCode.js) (V1.9) as well.

//to keep the code dry, using the vars where possible

var registerCode = {

//we can hard code here or better way is to fill from the home\index view where the partial view is displaying

\_registerCodeUrl: '', //In case to hard code /RegisterCode/Register

//different selectors

\_codeInputSelector: '.register-code-panel input',

\_alertDivSelector: '.register-code-panel .alert',

\_buttonSelector: '.register-code-panel button',

//bootstrap classes

\_alertClass: {

danger: 'alert-danger',

success: 'alert-success',

hidden: 'hidden'

},

//function to display the message

displayMessage: function (success, message) {

var $alertDiv = $(registerCode.\_alertDivSelector);

$alertDiv.text(message);

//defaults

var addClass = registerCode.\_alertClass.success;

var removeClass = registerCode.\_alertClass.danger;

if (!success) {

addClass = registerCode.\_alertClass.danger;

removeClass = registerCode.\_alertClass.success;

}

//hidden class is added by the register function below

$alertDiv.removeClass(removeClass).addClass(addClass).removeClass(registerCode.\_alertClass.hidden);

},

//function to register

register: function () {

//jquery objects

var $alert = $(registerCode.\_alertDivSelector);

var $codeInput = $(registerCode.\_codeInputSelector);

//hide the alert div

$alert.addClass(registerCode.\_alertClass.hidden);

//check have url

if (registerCode.\_registerCodeUrl === "" || registerCode.\_registerCodeUrl.length <= 0) {

registerCode.displayMessage(false, "Could not register. \n\rConnect with the site owner to resolve the issue!");

return;

}

//code val

var code = $codeInput.val();

if (code.length <= 0) {

registerCode.displayMessage(false, "Enter a Code");

return;

}

//post

$.post(registerCode.\_registerCodeUrl, { code: code },

function (data) {

registerCode.displayMessage(true, "The code was successfully added. \n\rPlease reload the page.");

$codeInput.val('');

}).fail(function (xlr, status, error) {

registerCode.displayMessage(false, "Could not register the code");

});

}

};

//click on the button

$(function (e) {

$(registerCode.\_buttonSelector).click(function (e) {

//call the register method

registerCode.register();

e.preventDefault();

});

});

### Add the Register Url in Home\Index View

Open the home index view add the following javascript to it.

@\*

Set the the url to post the registeration code to

The click event is in the SiteRegisterCode.js file

\*@

@section scripts {

<script>

$(function (e) {

registerCode.\_registerCodeUrl = '@Url.Action("Register", "RegisterCode", new { Area = "" })';

});

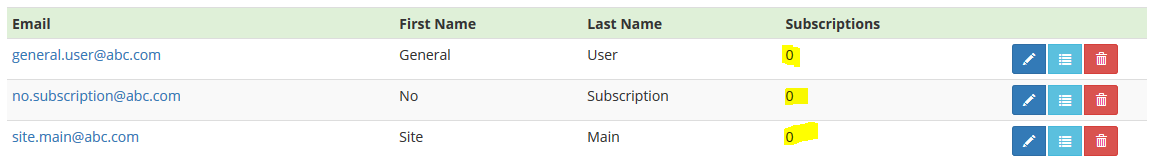
</script>

}

## Run the App

Run the app and login

For this first delete the subscriptions that we have created manually. Go to Admin menu, click User Subscriptions and delete any subscriptions created so far. Make sure to delete the subscription and not the user. After deleting the subscription, you should see 0 under Subscriptions on the User Subscriptions form.



To find the subscription, click Admin menu and then click Subscriptions.