**MVC 5 Membership Website**

Contents

[Resources placed in folder 4](#_Toc38851211)

[Created the project with 4](#_Toc38851212)

[App\_Start Folder 4](#_Toc38851213)

[BundleConfig.cs 4](#_Toc38851214)

[V1.0 Initial File 4](#_Toc38851215)

[V1.1 Adding Sitejs Bundle 5](#_Toc38851216)

[Content Folder 5](#_Toc38851217)

[Helper Extensions 5](#_Toc38851218)

[Reflection Extensions 5](#_Toc38851219)

[IsPropertyExists Extension 6](#_Toc38851220)

[GetPropertyValue Extension 6](#_Toc38851221)

[ICollection Extensions 6](#_Toc38851222)

[ToSelectListItem Extension 7](#_Toc38851223)

[Adding the Admin Menu 7](#_Toc38851224)

[\_Layout.cshtml 7](#_Toc38851225)

[V1.0 Initial File 7](#_Toc38851226)

[V1.1 Adding Admin Menu 8](#_Toc38851227)

[V1.2 Adding SiteJs bundle 8](#_Toc38851228)

[\_SiteAdminMenuPartial.cshtml 9](#_Toc38851229)

[V1.0 Admin Menu Initial 9](#_Toc38851230)

[Opening and Closing the Menu 10](#_Toc38851231)

[SiteAdminMenu.js 10](#_Toc38851232)

[Creating the Database 10](#_Toc38851233)

[Database Name Web.Config 10](#_Toc38851234)

[Package Manager Console 10](#_Toc38851235)

[Command enable-migrations 11](#_Toc38851236)

[Command update-database [Create Database] 11](#_Toc38851237)

[Adding Table/Entities 12](#_Toc38851238)

[Item Related Tables/Entities 12](#_Toc38851239)

[Product Related Tables/Entities 15](#_Toc38851240)

[Subscription Table/Entity 16](#_Toc38851241)

[ProductItem Table/Entity 17](#_Toc38851242)

[SubscriptionProduct Table/Entity 17](#_Toc38851243)

[UserSubscription Table/Entity 18](#_Toc38851244)

[Code First Approach 19](#_Toc38851245)

[Update Database Command 19](#_Toc38851246)

[Modifying AspNetUser Table 20](#_Toc38851247)

[Areas 21](#_Toc38851248)

[Add Admin Area 21](#_Toc38851249)

[Copying the \_Layout to the Areas Shared Folder 22](#_Toc38851250)

[V 1.0 Initial 22](#_Toc38851251)

[Creating Buttons – will be used on all views 23](#_Toc38851252)

[CreateButton Partial View 23](#_Toc38851253)

[EditButton Partial View 23](#_Toc38851254)

[EditButtonDetail Partial View 24](#_Toc38851255)

[Model 24](#_Toc38851256)

[Partial View 24](#_Toc38851257)

[BackToListButton Partial View 25](#_Toc38851258)

[SmallButton Partial View 25](#_Toc38851259)

[Model 25](#_Toc38851260)

[Partial View 26](#_Toc38851261)

[TableButton Partial 26](#_Toc38851262)

[CSS 27](#_Toc38851263)

[BundleConfig 27](#_Toc38851264)

[Section Entity : Creating Admin User Interface 28](#_Toc38851265)

[Scaffolding the Section Entity 28](#_Toc38851266)

[SectionController 29](#_Toc38851267)

[Index View 29](#_Toc38851268)

[Edit View 29](#_Toc38851269)

[Details View 30](#_Toc38851270)

[Delete View 31](#_Toc38851271)

[Update \_SiteAdminMenuPartial Section Link using @UrlAction 31](#_Toc38851272)

[Playing with Data 32](#_Toc38851273)

[Adding Controller – Actions - Views for Part / ItemType / ProductType / ProductLinkText Entities 32](#_Toc38851274)

[Item Entity : Scaffolding 32](#_Toc38851275)

[Controllers - Actions - Views 32](#_Toc38851276)

[Create Updates 32](#_Toc38851277)

[Controller Action – Create HTTP GET 32](#_Toc38851278)

[Controller Action – Create HTTP POST 33](#_Toc38851279)

[View 33](#_Toc38851280)

[Edit Updates 35](#_Toc38851281)

# Resources placed in folder

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

# Created the project with

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

# App\_Start Folder



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



Default route is specified inside the RouteConfig.cs file.

## BundleConfig.cs

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

### V1.0 Initial File

using System.Web;

using System.Web.Optimization;

namespace Web.Memberships

{

public class BundleConfig

{

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

public static void RegisterBundles(BundleCollection bundles)

{

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

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

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

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

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

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

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

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

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

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

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

"~/Content/bootstrap.css",

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

}

}

}

### V1.1 Adding Sitejs Bundle

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



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

# Content Folder

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

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

# Helper Extensions

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

## Reflection Extensions

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

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Web.Memberships.Extensions

{

public static class ReflectionExtensions

{

}

}

### IsPropertyExists Extension

/// <summary>

/// Checks if the property exists in type T

/// </summary>

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

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

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

/// <returns>bool</returns>

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

{

//must have the basics

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

//property name exists

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

if (property == null) return false;

return true;

}

### GetPropertyValue Extension

/// <summary>

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

/// </summary>

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

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

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

/// <returns>string</returns>

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

{

//must have the basics

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

var value = item.GetType()

.GetProperty(propertyName)

.GetValue(item, null)

.ToString();

return value;

}

## ICollection Extensions

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

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Web.Memberships.Extensions

{

public static class ICollectionExtensions

{

}

}

### ToSelectListItem Extension

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

/// <summary>

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

/// </summary>

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

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

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

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

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

{

if (items == null) return null;

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

{

Text = x.GetPropertyValue("Title"),

Value = x.GetPropertyValue("Id"),

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

});

return selectListItems;

}

## Convert Extension Method

For this extension [ProductModel](#_Creating_ProductModel) is required. Therefore, the [extension](#_Convert_Extension_Method) has been created with the Product scaffolding below.

# Adding the Admin Menu

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

## \_Layout.cshtml

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

### V1.0 Initial File

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

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

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

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

</head>

<body>

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

<div class="container">

<div class="navbar-header">

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

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

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

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

</button>

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

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

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

</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

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

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

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

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

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

</footer>

</div>

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

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

@RenderSection("scripts", required: false)

</body>

</html>

### V1.1 Adding Admin Menu

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



### V1.2 Adding SiteJs bundle

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

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

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



## \_SiteAdminMenuPartial.cshtml

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

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

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

### V1.0 Admin Menu Initial

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

<a href="#">

Admin

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

</a>

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

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

<li class="divider" />

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

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

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

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

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

<li class="divider" />

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

<li class="divider" />

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

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

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

<li class="divider" />

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

<li class="divider" />

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

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

<li class="divider" />

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

<li class="divider" />

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

</ul>

</li>

## 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.



# 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 Area\Admin\Views\Sgared 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 Models folder in Areas\Admin folder and the create a class with name “EditButtonModel”. Add the following to it.

using System.Text;

namespace Web.Memberships.Areas.Admin.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 Admin\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”.

## 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>

# 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.

## 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 : Scaffolding

## 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

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.

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())

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)

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

}

}