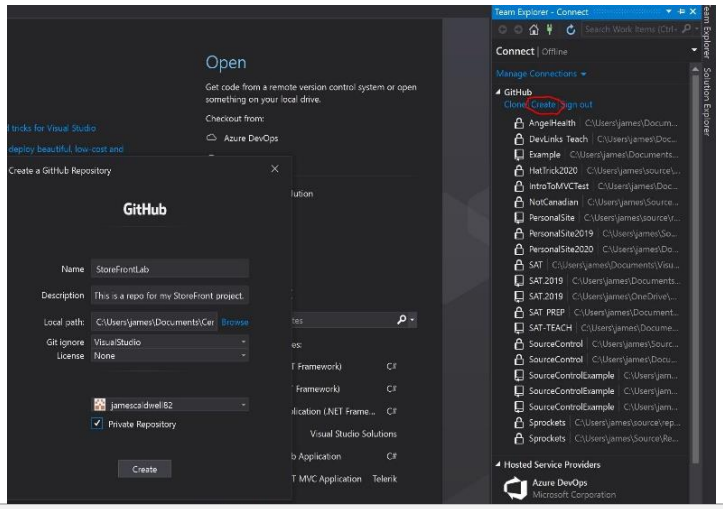
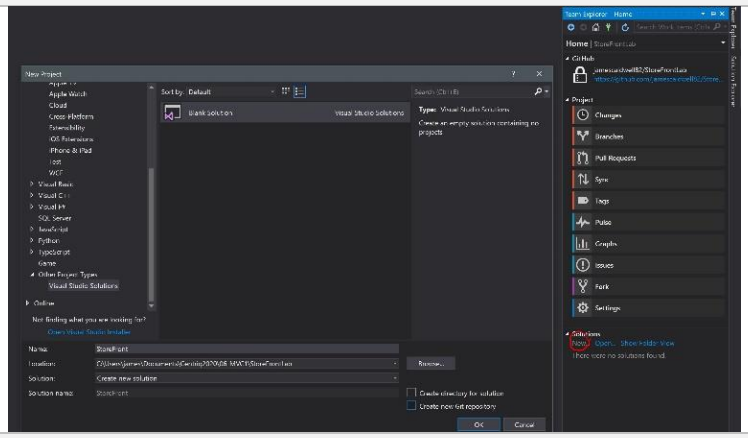
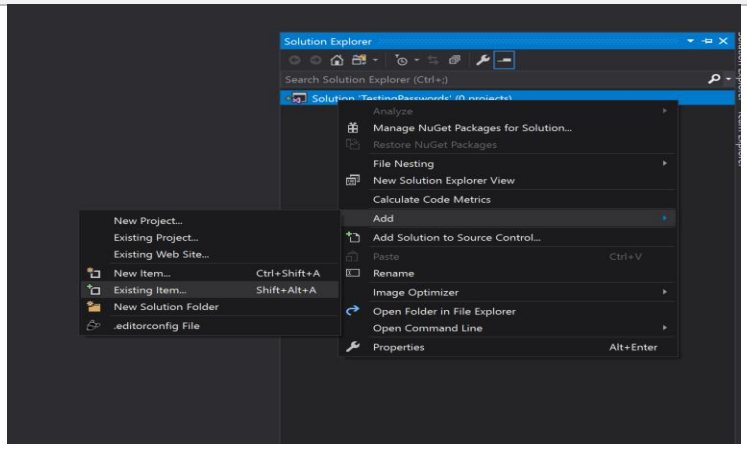
With Source Control and Identity

Starting an MVC Project w/ Source Control

1.		Launch Visual Studio 1. Go to Team Explorer 2. Select Create under the GitHub repos 3. Name your repo (i.e. StoreFrontLab) 4. Type in description 5. Browse to the local path 6. Select VisualStudio for the Git Ignore field 7. Check private repo This creates a folder in the selected location you browsed.
2.		Create new solution 1. In the bottom of Team Explorer, in Solutions, click on New 2. Select Blank Solution from the Other Project Types 3. Name your solution (i.e. StoreFrontLab) 4. Uncheck Create Directory and Create repo selections
3.		Add the .gitignore file to Visual Studio 1. Right click on the solution and select Add > Existing Item. 2. Select the .gitignore file and click Add. 3. This will open the .gitignore file and switch the Solution Explorer to folder view.

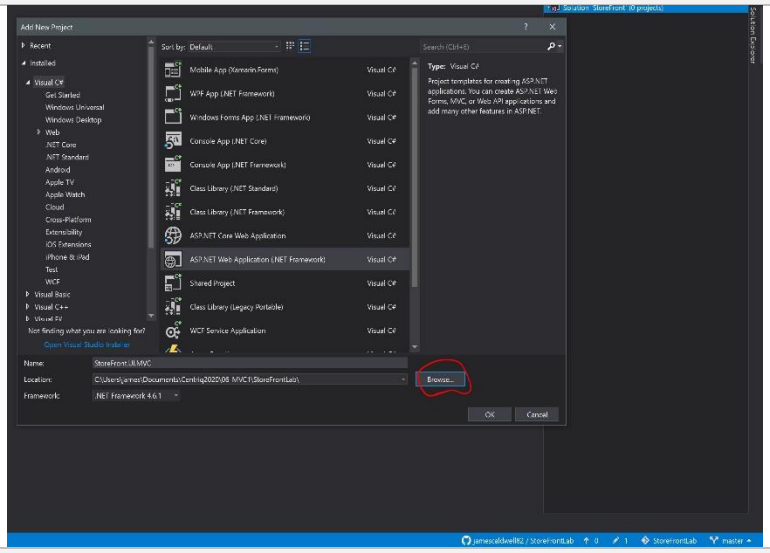
4.

```
# NVidia Nsight GPU debugger configuration file
*.nvuser

# MFractors (Xamarin productivity tool) working folder
.mfractor/

# Local History for Visual Studio
.localhistory/

# BeatPulse healthcheck temp database
healthchecksdb

# Backup folder for Package Reference Convert tool in Visual Studio 2017
MigrationBackup/

# Ionide (cross platform F# VS Code tools) working folder
.ionide/

# Prevent Sensitive data from being submitted
/ProjectName.UI.MVC/configs
```

Edit the .gitignore file to prevent sensitive data from being pushed to the repo.

1. Add the code shown in the screenshot to the bottom of the .gitignore file and save changes. (Make sure to replace ProjectName with your actual project name)
2. In the solution explorer, click the icon directly to the right of the home icon to return to solution view.

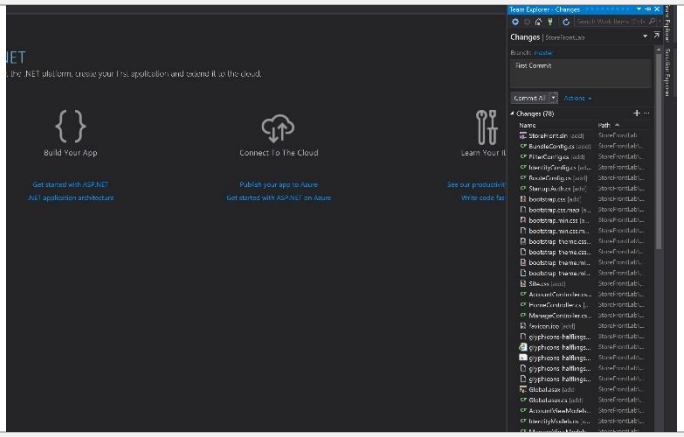
5.



In Solution Explorer:

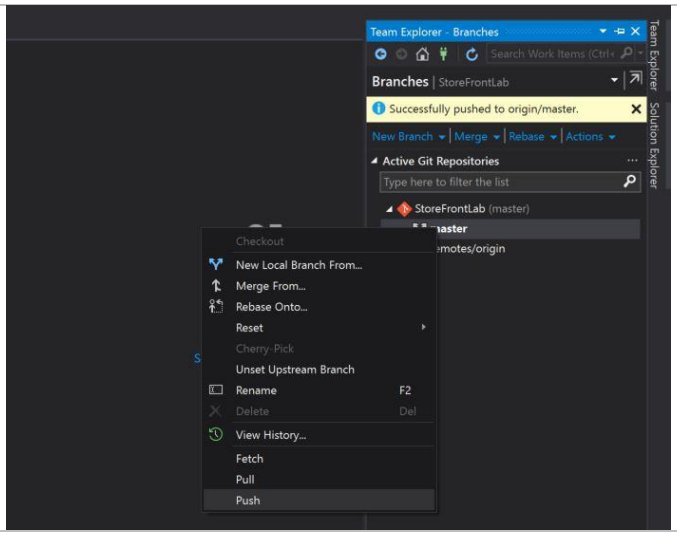
1. Right click on the solution
2. Select Add Project
3. Select ASP.NET Web Application (.NET Framework)
4. Name the project (i.e. StoreFrontLab.UI.MVC)
5. Browse to the solution folder (the folder that has the .git folder inside of it).
6. Click OK.

6.

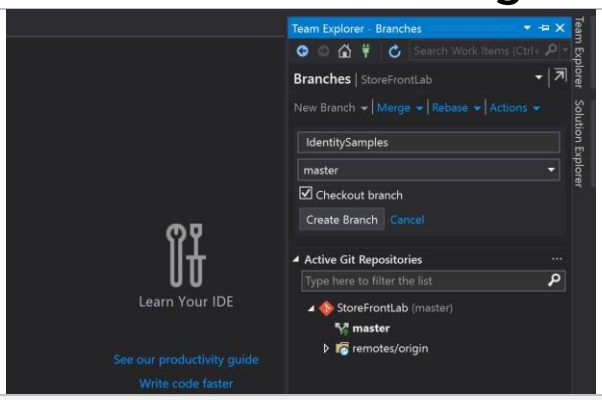
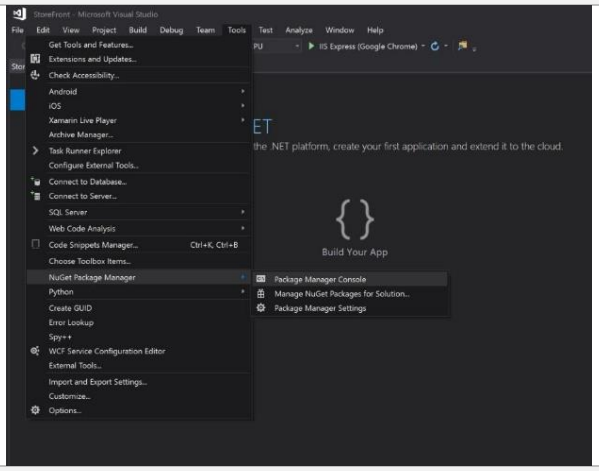


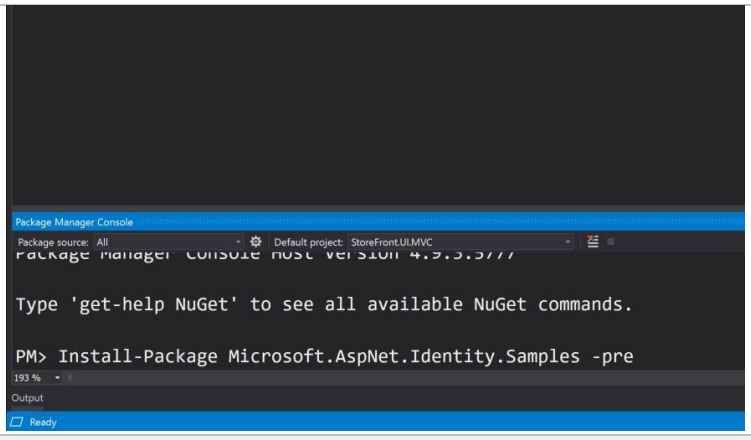
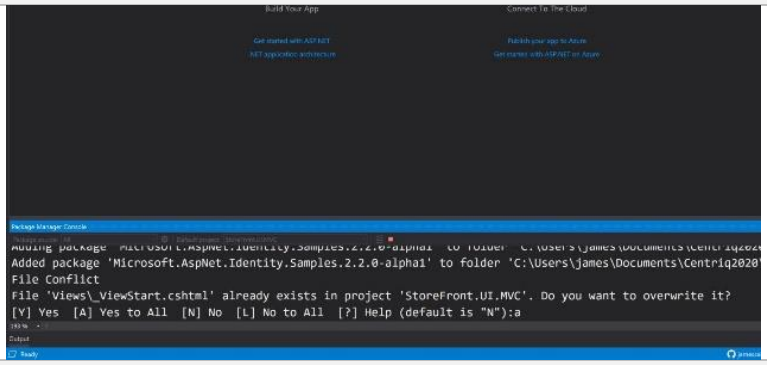
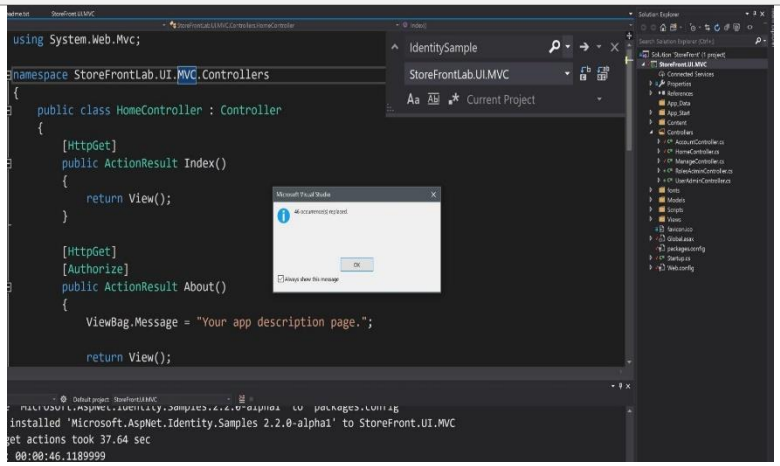
In Team Explorer:

1. Click on the Home button
2. Click on Changes
3. You should see a list of all of the UI layer files (i.e bin files and other MVC files.
4. Type in a commit message
5. Select Commit All

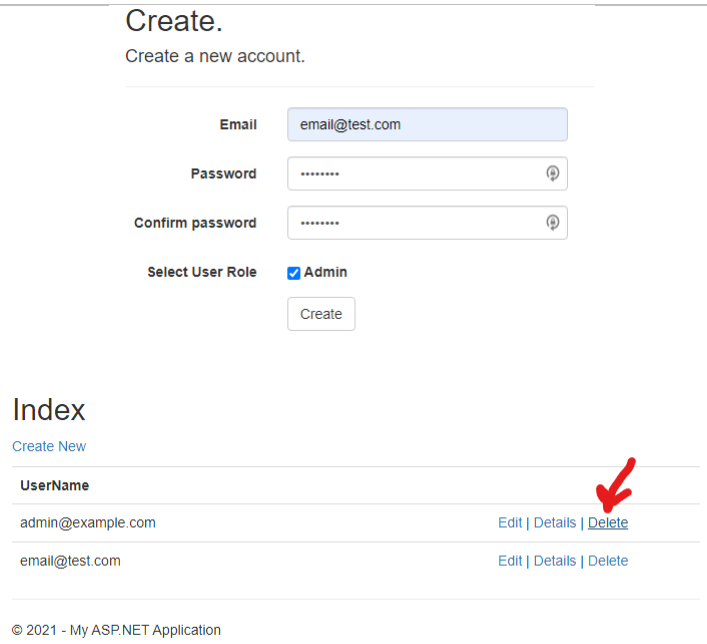
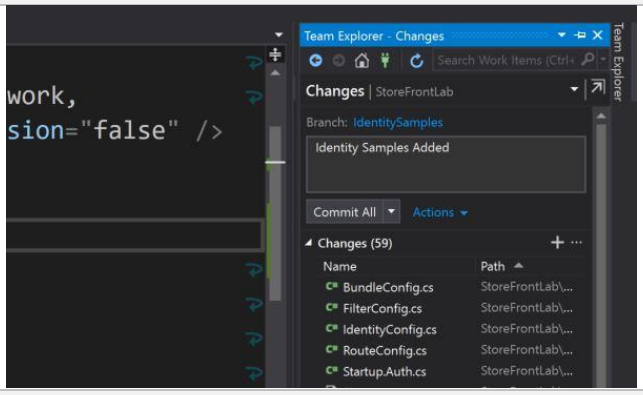
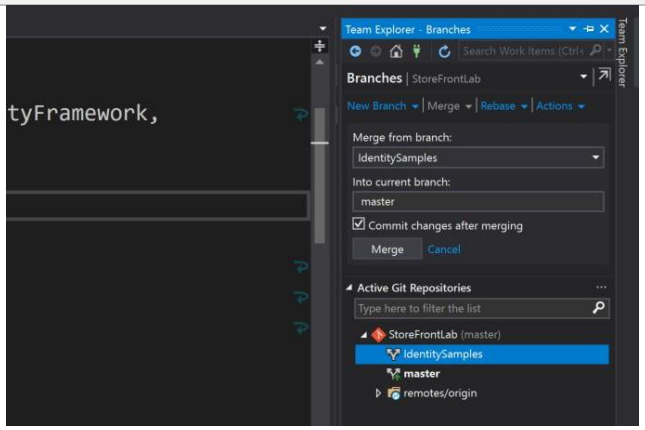
7.			In Team Explorer: 1. Click on the Home button 2. Click on Branches 3. Right click on main 4. Select Push Once this process completes, you should be able to navigate to your Github repo in the browser and see all of the UI project files.

Creating a New Branch

1.			In Team Explorer: 1. Click on the Home button 2. Click on Branches 3. Click on New Branch 4. Name the new branch IdentitySamples 5. Select Create Branch
2.			1. Select Tools < NuGet Package Manager < Package Manager Console 2. This gives us a command line interface to install Identity Samples

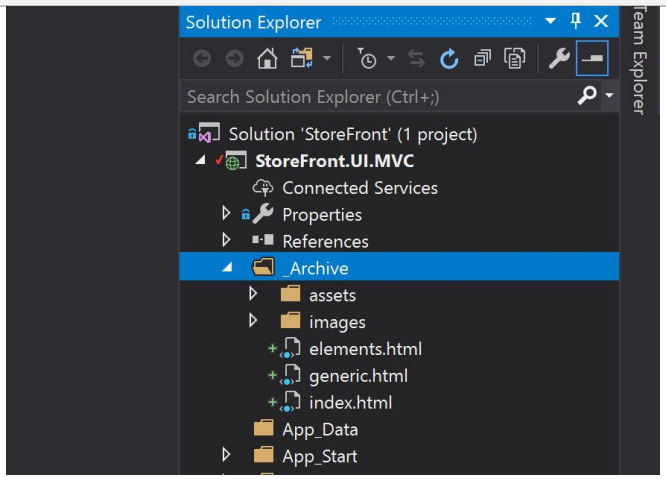
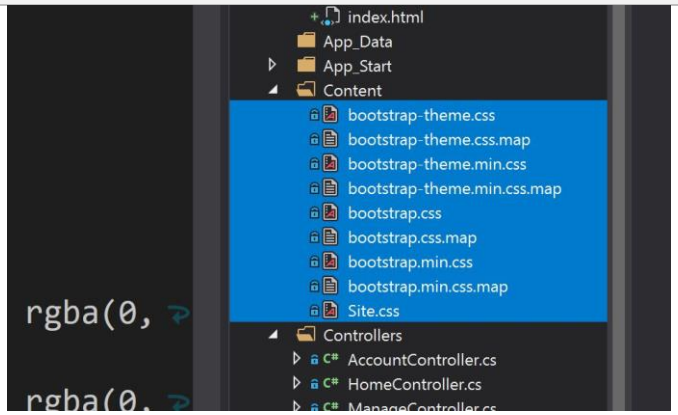
3.			In NuGet Package Manager Console: 1. Ensure that the UI project is listed in the default project dropdown at the top of the console. 2. Type **Install-Package Microsoft.AspNet.Identity.Samples -pre** WITH exact casing and spacing as shown on the left. 3. Press ENTER to run the code
4.			Still in NuGet Package Manager Console: 1. When prompted to replace all files, select A for all. 2. If you get a popup that says changes have occurred outside of the editor, select YES TO ALL
5.			In Solution Explorer, navigate to the Home Controller: 1. Double click on IdentitySamples in the Namespace 2. CTRL + F to find all instances of the name IdentitySample 3. Change the dropdown in the find window to find and replace. (Click the **V** to the left of the search box) 4. Change selection to Current Project 5. Set the replace value (bottom text box) to the name of the UI layer (i.e. StoreFrontLab.UI.MVC) 6. Click ok and you should receive the confirmation that 46 replacements were made.

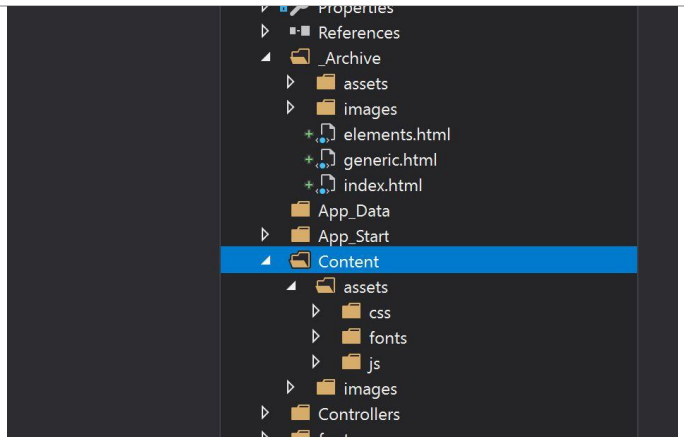
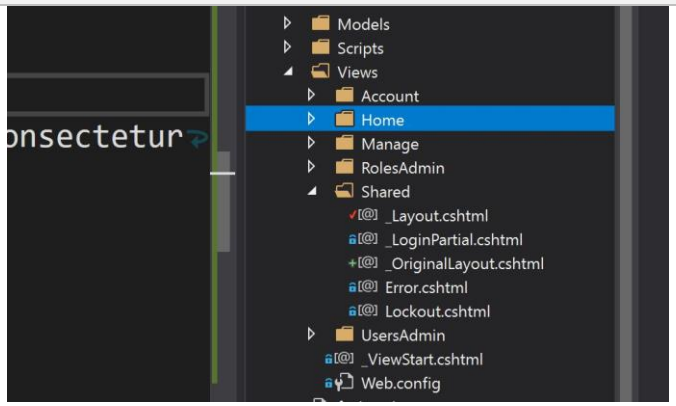
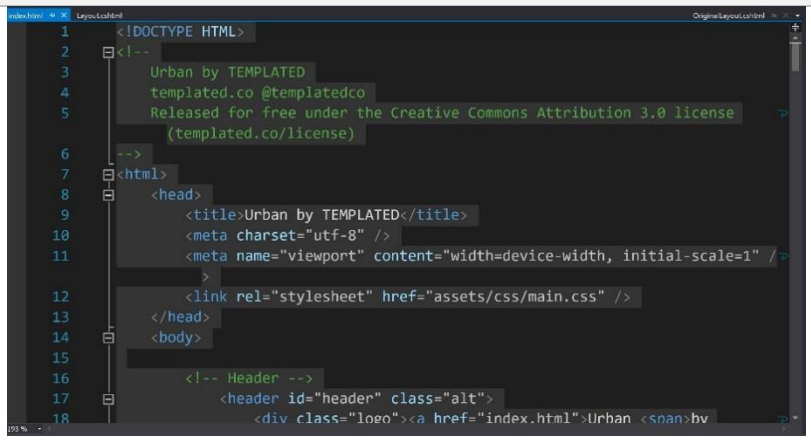
6.	<connectionStrings> <!--<add name="DefaultConnection" connectionString="Data Source=(LocalDb) \MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet- TestMVC-20210310093524.mdf;Initial Catalog=aspnet- TestMVC-20210310093524;Integrated Security=True" providerName="System.Data.SqlClient" />-->> <add name="DefaultConnection" connectionString="Data Source=.\sqlexpress;Initial Catalog=BookStorePlus;Integrated Security=true" providerName="System.Data.SqlClier Value(s): </connectionStrings>	Create a folder to hold application configurations. These configurations are kept in a separate location to avoid them being committed to Source Control. 1. Right click on the solution and select Add > New Folder. Name the folder configs 2. Right click on the configs folder and select Add > XML File. Name the file connections.config. Delete any code that was scaffolded into the file. 3. Go to the web.config file in the root of the UI layer and copy the entire <connectionStrings> section. 4. Paste the content into the connections.config file. 5. Comment out or remove one of the <add/> tags with the name of DefaultConnection. 6. Update the other <add/> with the values for the project's database. The final version of the connections.config file is shown in the screenshot.
7.	<configuration> <configSections> <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=2374 -->> <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" > </configSections> <connectionStrings configSource="configs\connections.config"></connectionStrings>	*Complete the In the ROOT web.config file: 1. Delete all content inside of the <connectionStrings> tags. 2. Add the configSource attribute onto the opening <connectionStrings>. The value for this attribute should be the relative file path to the connections.config file.

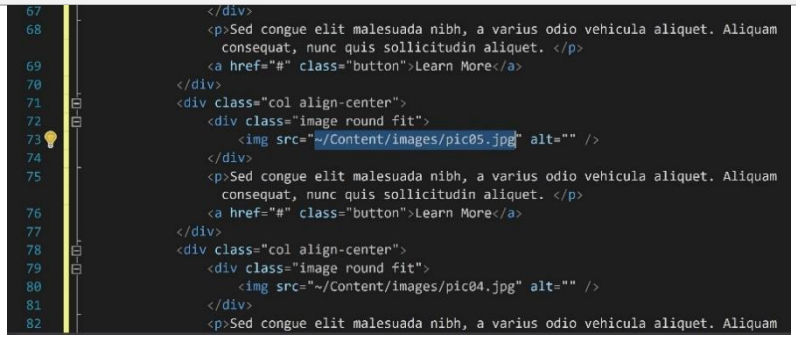
8.		Launch the project: 1. Click login and use the default credentials: Username=admin@example.com and Pass=Admin@123456 2. Click UsersAdmin in the main navbar and then click Create New 3. Enter the info for a new user and click the box to beside admin. Click Create. 4. Logout and log back in with the new user credentials. 5. Click UsersAdmin and click delete beside the default user (admin@example.com) 6. Confirm the delete on the next screen.
9.		In Team Explorer: 1. Select the Home button 2. Click on Changes 3. You should see several changes that need to be committed. 4. Type in your commit message. 5. Commit All
10.		In Team Explorer: 1. Click on Home 2. Click on Branches 3. Double Click on the Main branch to make it active (it will become **bold**) 4. Click Merge. Select the IdentitySamples branch for the Merge from branch and ensure the main branch is shown in the Into current branch box. 5. Click Merge. 6. Right click on the main branch and select push. 7. After it is complete, you should see the updates

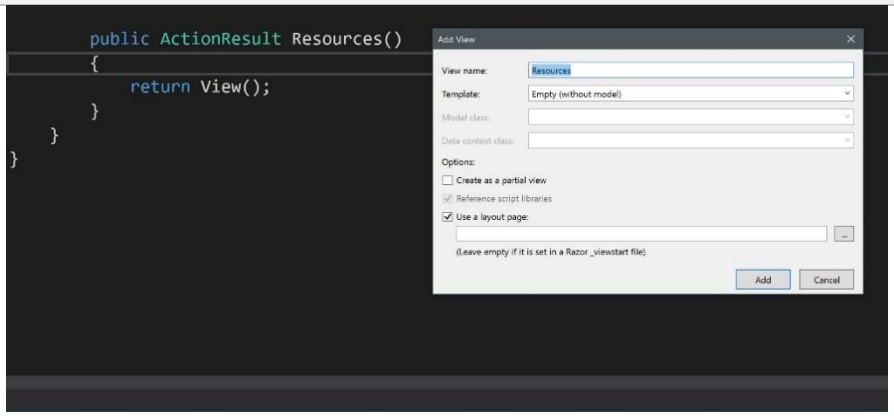
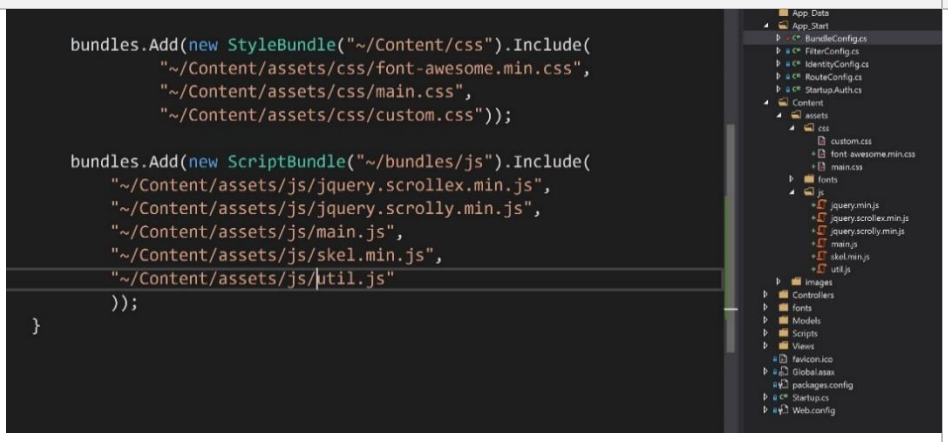
		reflected in the repo on GitHub.
11.		PROCEED TO Converting a Template

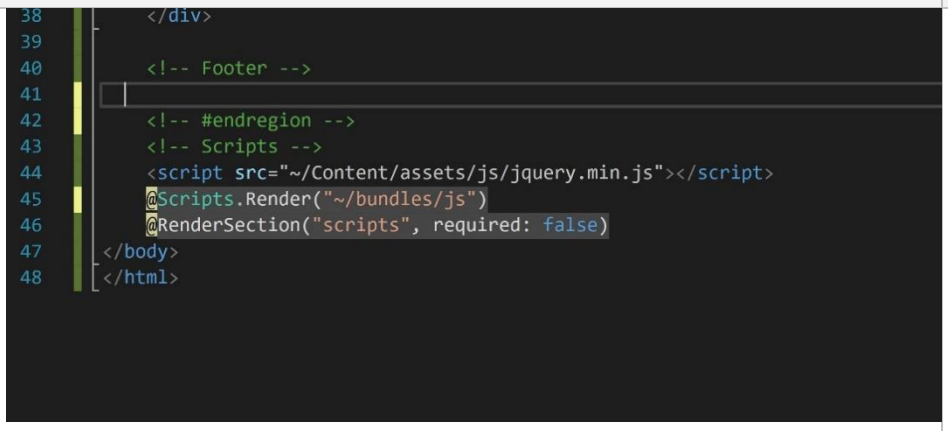
Converting a Template

1.		1. Choose a template. 2. Create a new branch
2.		1. Create an _Archive folder. 2. Copy the extracted template files into the _Archive folder.
3.		Delete all of the resources from the Content folder. (Note: You can optionally create a folder for these in the _Archive folder and move them there if you want to preserve them)

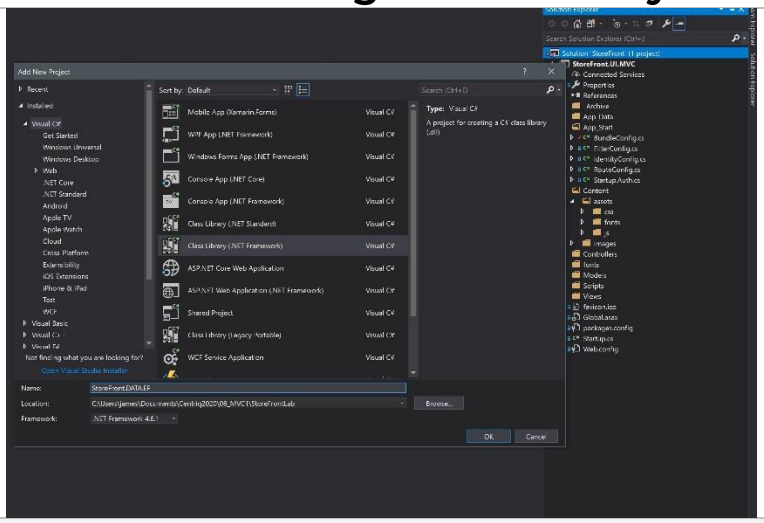
4.		1. Paste all images and css files in the content folder. Some templates come with a vendor folder, that can be put into the Content folder as well. 2. Paste all JavaScript files in the Scripts folder. 3. Paste all font files in the Fonts folder.
5.		1. Copy the _Layout in the Views > Shared folder. 2. Right Click Shared folder and paste 3. Rename _Layout-Copy to OriginalLayout 4. This is simply there to preserve the original version should you need to retrieve code snippets from it.
6.		1. Open the template HTML file that you want to use as your Layout for the project 2. Copy all of the HTML in that file 3. Paste over ALL of the HTML in the _Layout file.

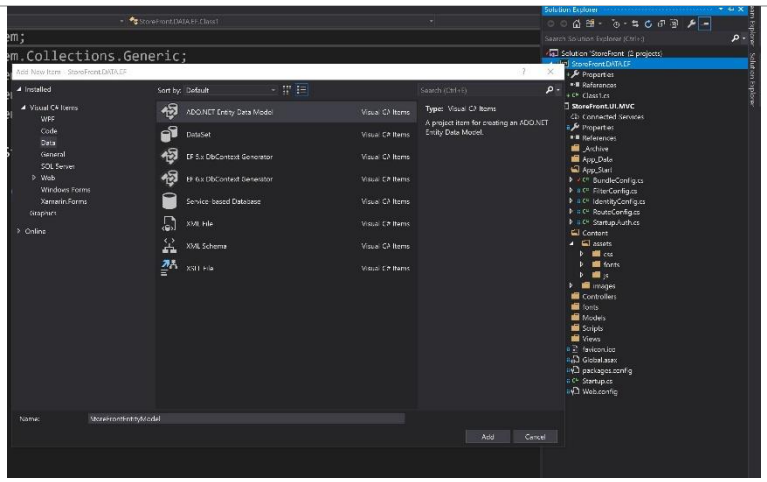
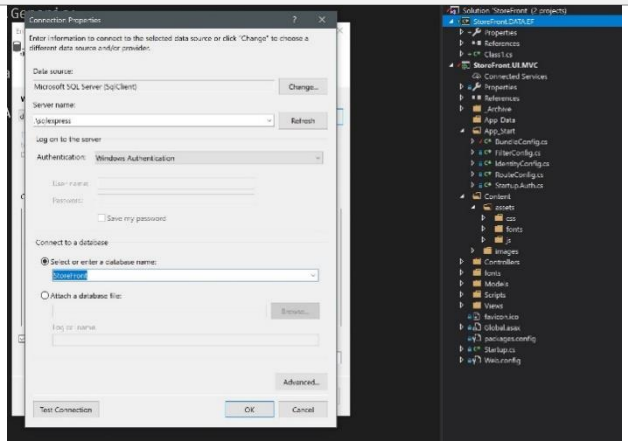
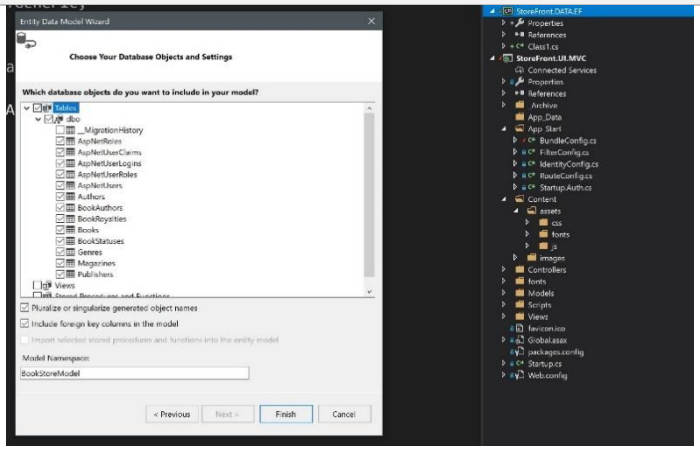
7.	 	1. Work top to bottom, modifying the necessary html 2. Pay attention to structures that contain href or src attributes 3. For Hyperlinks, you can use the Url.Action() or Html.ActionLink() helpers. 4. Links in the head of the document and scripts at the bottom of the body need to be modified to point to the correct folder. 5. Keep all of the structures that will remain the same across all pages as content in the _Layout. 6. Content that will change across all pages should be moved to the Home > Index in order to keep the "home" page intact on your site. 7. Remember to place the @RenderBody() and @RenderSection("scripts", required: false) in the appropriate places in the Layout
8.	 	1. On the Index.cshtml in the Home folder, work from the top to bottom, modifying the structures as necessary 2. Pay attention to structures with href or src attributes
9.		1. Run the Application to ensure that the product you have converted looks just like the template file you converted from.

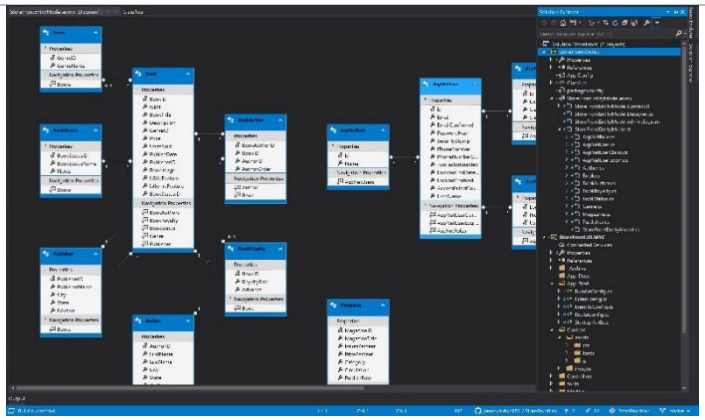
		2. Commit changes to your branch. 3. Merge your branch into the main branch. 4. Push the main branch to the remote repo.
10.		For any additional pages in the project: 1. Create a new branch 2. Create a new Action in the Home Controller. 3. Right click in the Action and Select Add View 4. Copy all unique content from the desired template HTML file and paste in the created view. 5. Modify the content as appropriate. 6. Commit changes, merge branch into main, and then push main to the remote repo.
	<u>OPTIONAL STEPS BELOW</u>	
11.		Create a branch OPTION 1: Add files to an existing bundle 1. Navigate to the App_Start folder > BundleConfig.cs file 2. In the Include() method for the bundle named "~/Content/css", edit the references to the appropriate css files from your content folder. OPTION 2: Create new bundle for JS 1. Add a new bundle for all of your JS files. In the Include() method, add a

		comma separated list of all the necessary files. Remember that the string in the ScriptBundle() or StyleBundle() (example: **"~/Content/css")** is the name of the bundle which you will reference in the _Layout. All of the strings in the Include() reference a file path to an individual resource.
12.		Reference the Bundles in the Layout 1. To reference CSS bundles, in the <head>, use @Styles.Render(**"bundleName"**) 2. To reference JS bundles, at the bottom of the body (before the @RenderSection("scripts", required: false), use @Scripts.Render(**"bundleName"**) 3. Commit changes, merge branch into Main, and push Main to the remote repo.

Adding the Data Layer

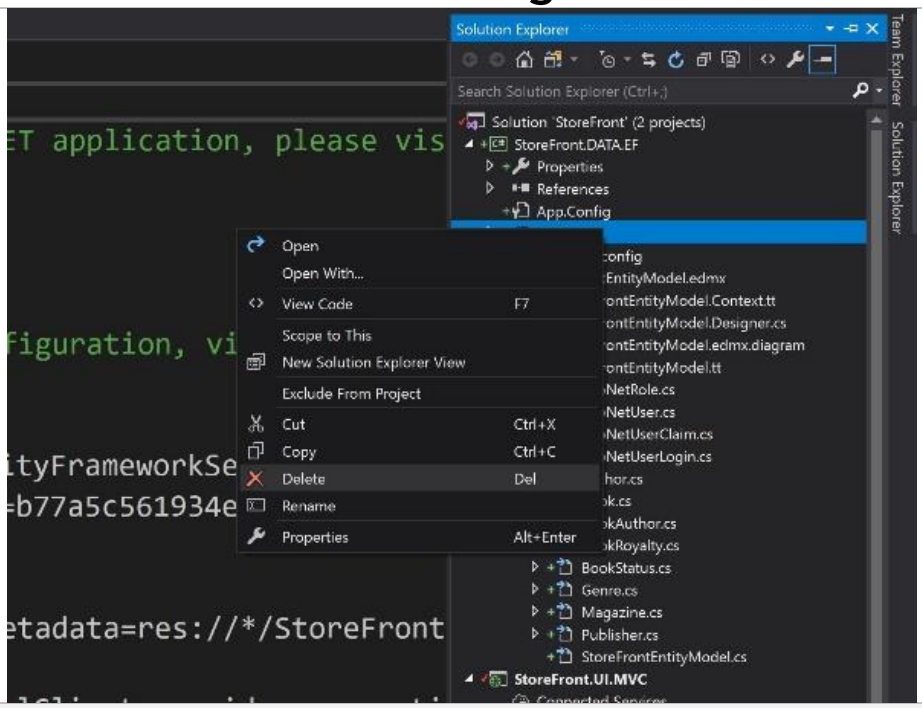
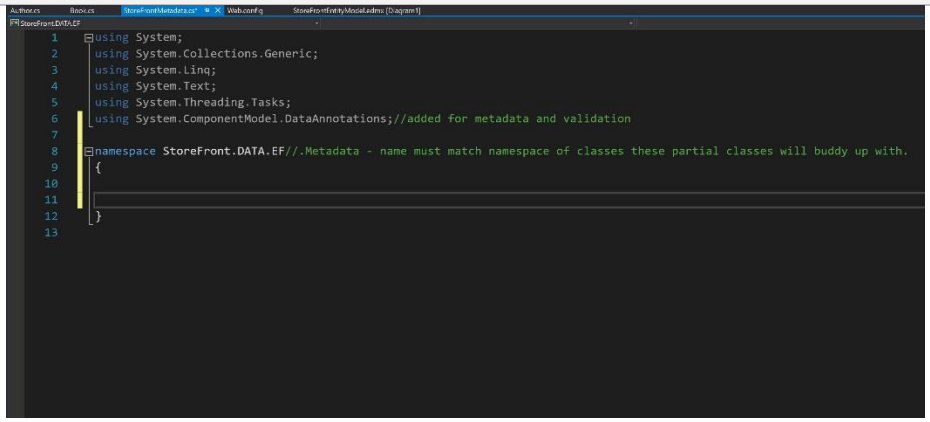
1.			Create a new branch Add a new project to the solution. 1. Project Type – Class Library (.NET Framework) Name: ProjectName.DATA.EF Make sure that you are placing the new project in the same folder as the .git (i.e. the solution folder).

2.		Right Click on the DATA.EF project and add a new item. 1. Choose Data from the templates on the left side of the menu. 2. Then select ADO.NET Entity Data Model from the list in the middle. 3. Provide a name for the data model (normally the same name as the database).
3.		Follow thru the following steps in the EF Designer UI: 1. Select New Connection 2. Type in .\sqlexpress in Server Name field. 3. Select the appropriate database in the Select or Enter a database name field. 4. Click OK
4.		5. Select Next when you return to the Data Connection screen 6. Select Entity Framework 6.x and Select Next 7. Select the tables that you will be generating controllers/views for. 8. Leave all checkboxes checked (Pluralize, include foreign key, import selected stored procs)

5.		CTRL + SHIFT + B to build 1. Change any mislabeled names in the tables CTRL + SHIFT + B to build after modifications

6.	```<connectionStrings>
	  <add name="BookStorePlusEntities" connectionString="metadata=res://*/
	  BookStorePlusModel.csdl|res://*/BookStorePlusModel.ssdl|res://*/
	  BookStorePlusModel.msl;provider=System.Data.SqlClient;provider
	  connection string="data source=.\sqlexpress;initial
	  catalog=BookStorePlus;integrated
	  security=True;MultipleActiveResultSets=True;App=EntityFramework""
	  providerName="System.Data.EntityClient" />
	  <add name="DefaultConnection" connectionString="Data Source=.
	  \sqlexpress;Initial Catalog=BookStorePlus;Integrated Security=true"
	  providerName="System.Data.SqlClient" />
	</connectionStrings>```
		1. Open the App.Config file in the DATA.EF layer.
		2. Look for the EntityModel Connection in the App.Config and copy the connection string
		3. Open the configs/connections.config file in the MVC.UI layer.
		4. Remember to hit CTRL + K + D to format the file prior to any edits.
		5. Paste the copied connection string into the connectionStrings object.
		6. Commit changes, merge branch into main, and push main to the remote repo.

Adding Metadata

1.		Create a new branch Remove class1.cs from the DATA.EF layer
2.		Add a metadata folder named DBNameMetadata (i.e. StoreFrontMetadata) to the DATA.EF layer. 1. Add a new class named DBNameMetadata 2. Comment out the .Metadata at the end of the namespace

3.

```
6   using System.ComponentModel.DataAnnotations;//added for metadata and validation
7
8   namespace StoreFront.DATA.EF//.Metadata - name must match namespace of classes these partial classes will buddy up with.
9   {
10
11      public class AuthorMetaData
12      {
13          [Required]
14          [Display(Name = "First Name")]
15          [StringLength(15, ErrorMessage = "Max 15 characters")]
16          public string FirstName { get; set; }
17          [Required]
18          [Display(Name = "Last Name")]
19          [StringLength(15, ErrorMessage = "Max 15 characters")]
20          public string LastName { get; set; }
21          [DisplayFormat(NullDisplayText = "City not provided")]
22          public string City { get; set; }
23          [DisplayFormat(NullDisplayText = "State not provided")]
24          public string State { get; set; }
25          [Display(Name = "Zip Code")]
26          [DisplayFormat(NullDisplayText = "Zip Code not provided")]
27          public string ZipCode { get; set; }
28          [DisplayFormat(NullDisplayText = "Country not provided")]
29          public string Country { get; set; }
30      }
31
32
33      [MetadataType(typeof(AuthorMetaData))]
34      public partial class Author
35      {
36          //custom property
37          public string FullName
38          {
39              get { return FirstName + " " + LastName; }
40          }
41      }
42  }//end namespace
```

Some questions to consider:

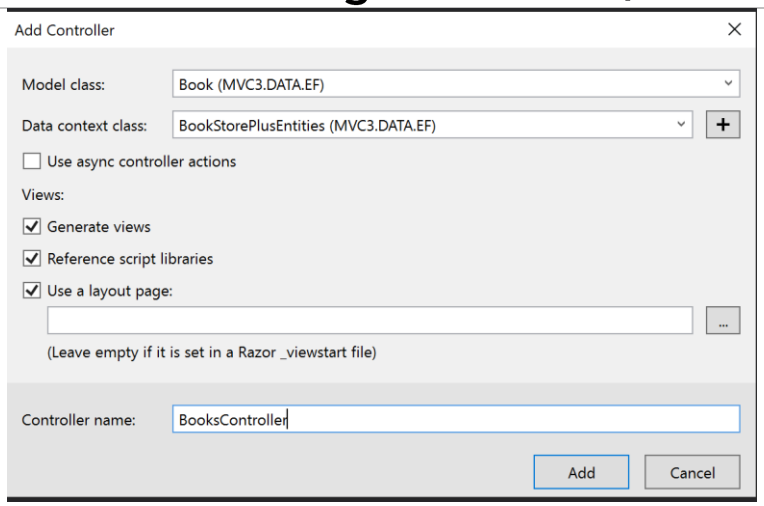
1. Is the field going to be **viewable**? (AuthorID) – no metadata
2. Is the field going to be **required**? Yes:(Add Required validation)
3. Is the field nullable? Yes: DisplayFormat(NullDisplayText="x")
4. Is the field a string value? Yes: StringLength() validation
5. No validation on Boolean items
6. Is the field name suitable for the UI? No: Display(Name="x")
7. Is the field a BIG text value? Yes: UIHint("MultilineText")
8. Does field require special formatting? Yes: [RegularExpression("pattern", ErrorMessage= "x")]
9. Does the field have a specific range? Yes: Range(min,max,ErrorMessage="X")
10. Is the field a date? (Do you want to remove the time) [DisplayFormat(DatatFormatString="{0:d}")] if you want that to carry over to the edit fields as well, add [DisplayFormat(DatatFormatString="{0:d}, ApplyFormatInEditMode=true")]
11. If you wish to add custom classes, this is done in the partial class.

For each class that needs metadata to add validation:

1. Locate the .tt file in the ADO Entity Model.
2. Create a public class named TableNameMetadata.
3. Create a public partial class to create the helper/buddy class for the metadata to link up with the EntityModel class it's associated with.
4. Remember to add the validation seen on line 33 in the image to the left.
5. Add the appropriate fields in the metadata class and add appropriate data validation to the fields.
6. Complete this process for each class that needs metadata validation.
7. Commit changes, merge branch into main, and push main to the remote repo.

Scaffolding Controllers/Views: The UI Layer

1.



Create a new branch and then complete the following:

1. Copy all content from inside the <connectionStrings> on the Connections.config folder and paste it inside the <connectionStrings> tag in the Web.config file in the Root of the UI.MVC project.
2. Temporarily remove or comment out the configSource attribute on the opening <connectionStrings>. **Note: DO NOT push to source control while the connection string information is the root Web.config file.*
3. Right click the controllers folder and select Add > Controller.
4. Select MVC5 Controller with views, using Entity Framework.
5. Model Class: Select the class for which you are creating the controller/views (Not Metadata)
6. DataContext: Select the Entities option in the dropdown.
7. Mark all checkboxes (as shown)
8. Controller Name: should be the TableName+Controller (as shown)
9. Add a link to the Index.cshtml view in the navbar on the _Layout.

2. Modify Views – preferred order is (Index, Details, Create, Edit, Delete, any additional views)
   a. UI Layer, Views Folder, [ControllerYouJustCreatedName] Folder and expand
   b. Determine if you will need to separate views (Active/Discontinued – Table/Tile Layout)

	c. Draw out/Structure/Wire Frame i. Make a plan for each of your views ii. Execute the plan. iii. This will help with structuring your HTML and CSS a. In the table/Index view remove fields from the table that are unnecessary that can be shown in the details b. If you are structuring the view for a small lookup table you may display all information on the index and remove (comment out) the details action in the controller as well as remove the det ails button from the view. c. Test each view before moving to the next
3.	Once all views have been modified/structured a. Determine Access (may be done prior to any application building with a Use/Case Diagram) b. Secure each action (or the entire controller) as needed. [Authorize] or [Authorize(Roles="X")] c. If you secure at the controller level you may not have to secure buttons in the views i. If you only have an admin role this is true ii. If access varies by role and you have multiple roles, each view's buttons will need to be secured accordingly. (Advanced) d. TEST
4.	Once all scaffolding has been completed a. Delete all information inside of the <connectionString> in the root Web.config file in the UI.MVC project. b. Uncomment or add the configSource attribute back to the opening <connectionStrings> tag.

Protecting Sensitive Data

1.

```xml
<?xml version="1.0" encoding="utf-8"?>
<appSettings>
  <!--Added to protect sensitive data-->

  <!--Replace yourdomain.com with your actual domain and extension-->
  <add key="EmailClient" value="mail.yourdomain.com"/>
  <!--Email user on your host-->
  <add key="EmailUser" value="email@yourdomain.com"/>
  <!--Password for the email user on your host-->
  <add key="EmailPass" value="Password"/>
  <!--Email Address to send contact form entries to-->
  <add key="EmailTo" value="email@domain.com"/>
</appSettings>
```

```xml
<appSettings file="configs\AppSecretKeys.config">
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  <add key="owin:AppStartup" value="IdentitySample.Startup,TestingProject" />
</appSettings>
```

Create a custom config file to hold sensitive username and password information.

1. Right click on the configs folder in the UI.MVC layer. Select Add > XML File. Name the file AppSecretKeys.config.
2. Add a <appSettings> section with multiple <add> to define different pieces of sensitive data as shown in the screenshot.
3. Open the web.config file in the root of the UI.MVC layer and navigate to the <appSettings> section.
4. Add the file attribute to the opening <appSettings> as shown in the second screenshot.

```
public ActionResult Contact(ContactViewModel cvm)
{
    if (!ModelState.IsValid)
    {
        return View(cvm);
    }
    string message = $"You have received an email from {cvm.Name} with a subject " +
        $"{cvm.Subject}.  Please respond to {cvm.Email} with your response to " +
        $"the following message: <br />{cvm.Message}";
    MailMessage mm = new MailMessage(
        //FROM
        ConfigurationManager.AppSettings["EmailUser"].ToString(),
        //TO
        ConfigurationManager.AppSettings["EmailTo"].ToString(),
        cvm.Subject,
        message);

    mm.IsBodyHtml = true;
    mm.Priority = MailPriority.High;
    mm.ReplyToList.Add(cvm.Email);
    SmtpClient client = new SmtpClient(ConfigurationManager.AppSettings["EmailClient"].ToString());

    client.Credentials = new NetworkCredential(ConfigurationManager.AppSettings["EmailUser"].ToString(),
        ConfigurationManager.AppSettings["EmailPass"].ToString());
    try
    {
        client.Send(mm);
    }
    catch (Exception ex)
    {
        ViewBag.CustomerMessage =
            $"We're sorry your request could not be completed at this time." +
            $"  Please try again later.  Error Message: <br /> {ex.StackTrace}";
        return View(cvm);
    }
    return View("EmailConfirmation", cvm);
}
```

5. Utilize the configured app settings in the Controller where email functionality is being configured.

**Make sure to utilize this process whenever you need to provide sensitive data (usernames and passwords).