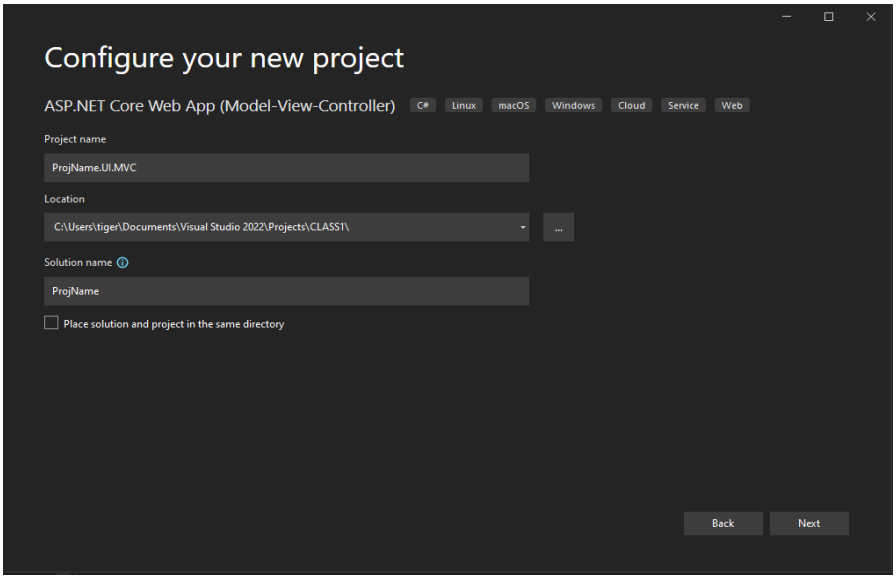
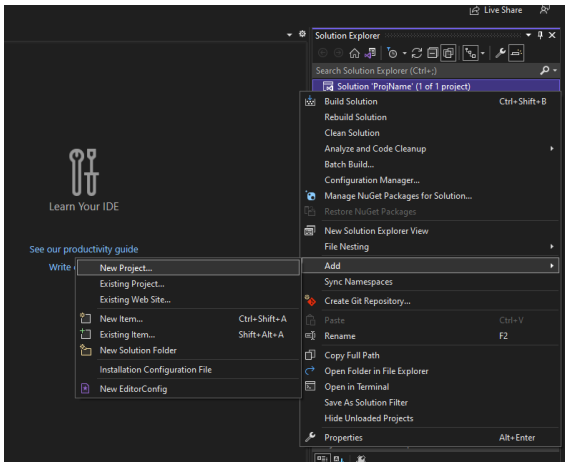
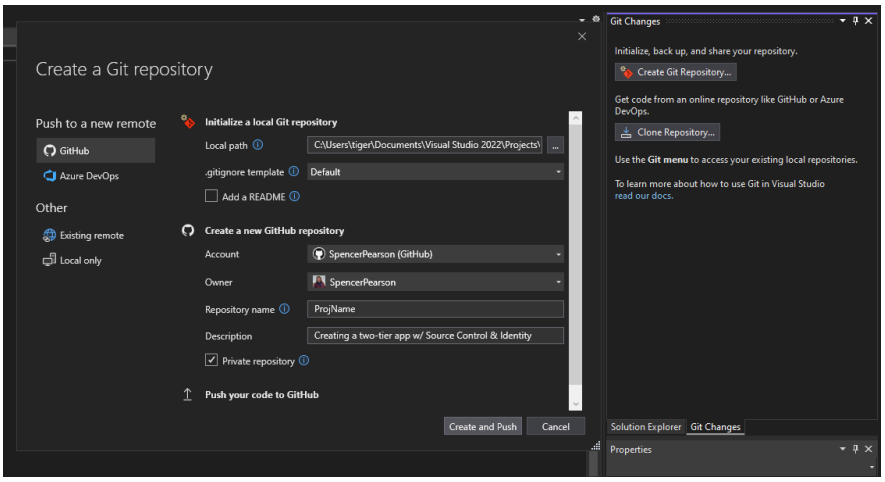
BUILDING A TWO-TIER APPLICATION

With Source Control and Identity

Starting an MVC Project with Source Control

1.		Create the UI Project 1. Click 'Create a new project'. 2. Select ASP.NET Core Web App (Model-View-Controller) → **Next** 3. Name your Project (ProjName.UI.MVC) 4. Browse to the local path. 5. Remove the .UI.MVC from the Solution name → **Next** 6. Framework: .NET 6.0 7. Authentication Type: Individual Accounts 8. Check Configure for HTTPS 9. Click Create.
2.		Create the DATA Project 1. In Solution Explorer, right click the Solution → Add New Project 2. Select Class Library (C#) → Next 3. Name your class library (ProjName.DATA.EF) 4. Click Next 5. Framework: .NET 6.0 → Create

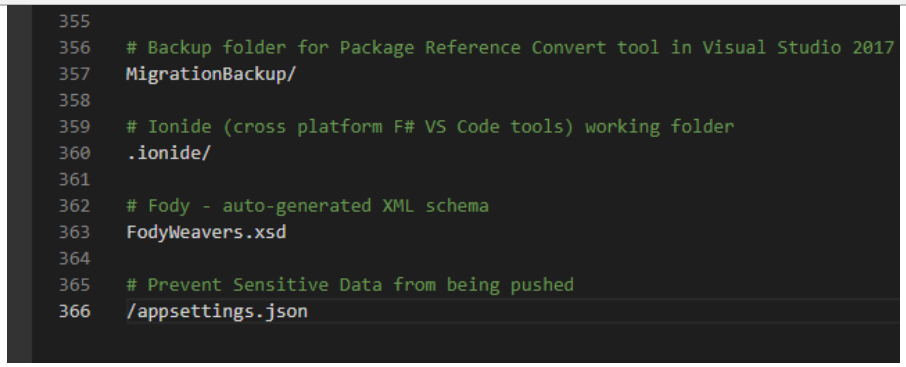
3.



Create the Repository

1. Expand Git Change.
2. Click 'Create Git Repository'.
3. Enter the Repository name.
4. Enter a Description.
5. Private repository: check if you don't want anyone else to have access to the repo, uncheck if you plan on sharing this project.
6. Click Create and Push.

4.



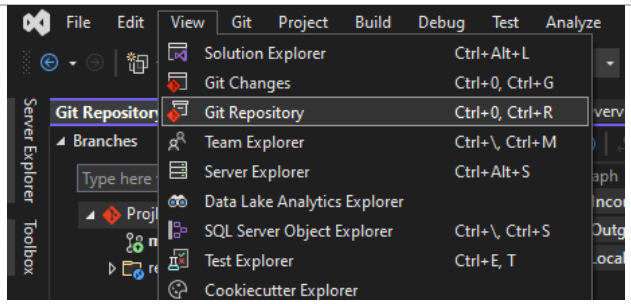
```
# Backup folder for Package Reference Convert tool in Visual Studio 2017
MigrationBackup/

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

# Fody - auto-generated XML schema
FodyWeavers.xsd

# Prevent Sensitive Data from being pushed
/appsettings.json
```

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

1. Open File Explorer and navigate to your solution folder and open .gitignore
2. Add **/appsettings.json** to the bottom of the .gitignore file and save.

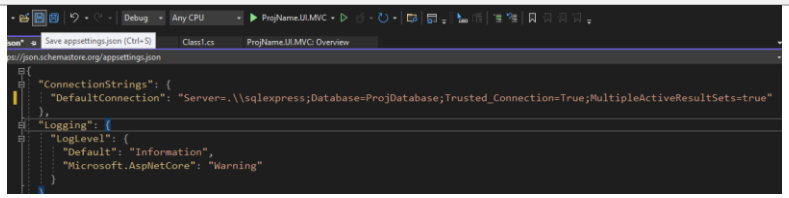
Creating a New Branch

1.



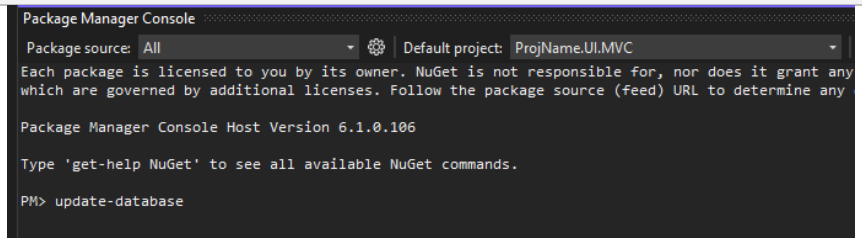
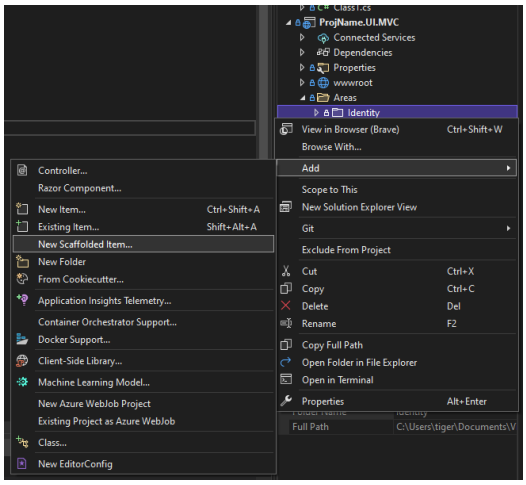
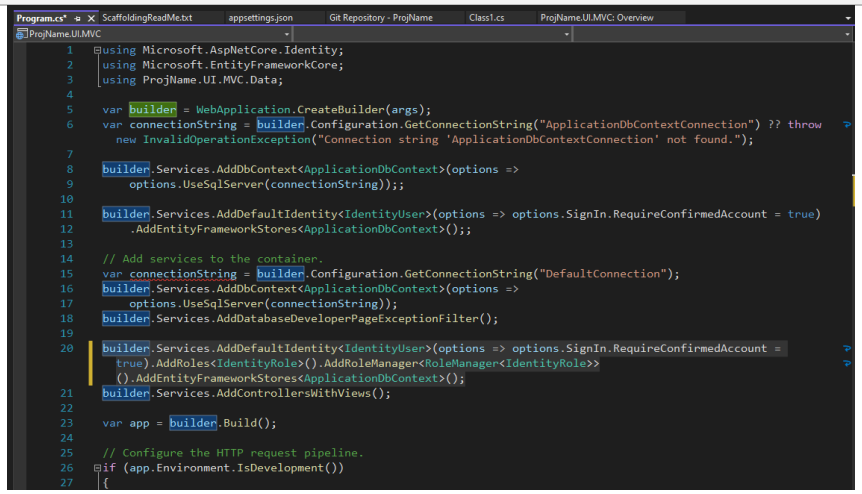
1. In VS, expand 'Git Repository'
2. Under 'Branches', right click on main → **New local branch from**
3. Name the new branch IdentitySamples
4. Select Create

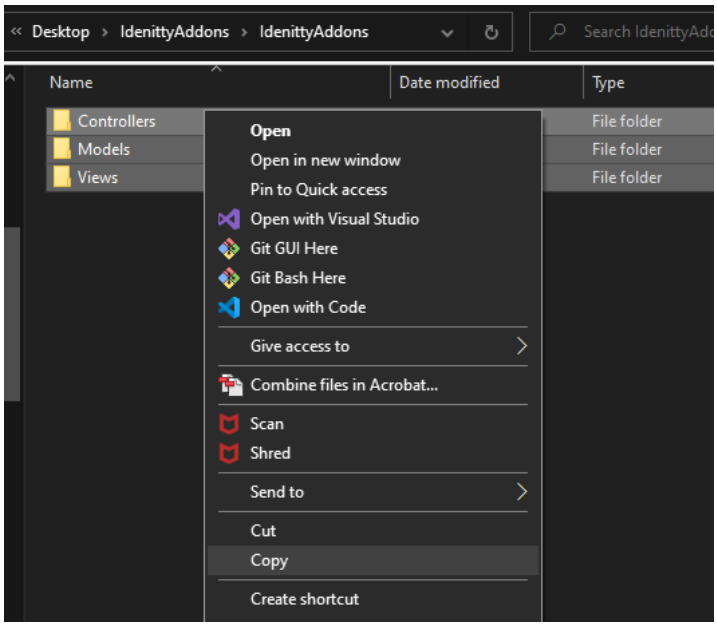
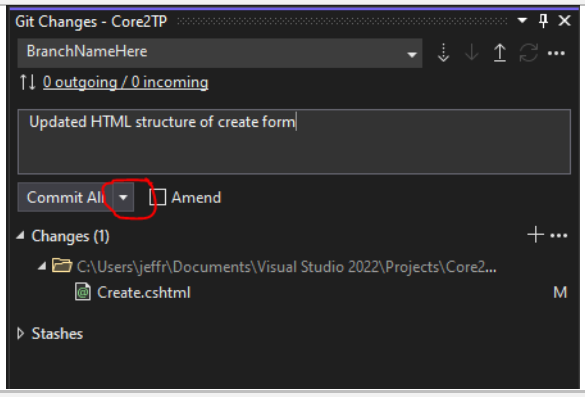
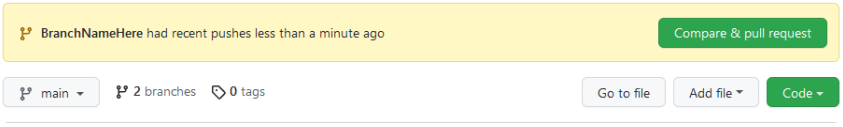
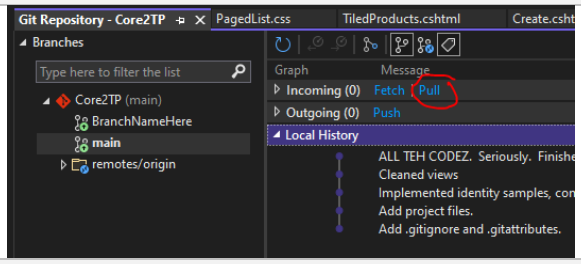
2.



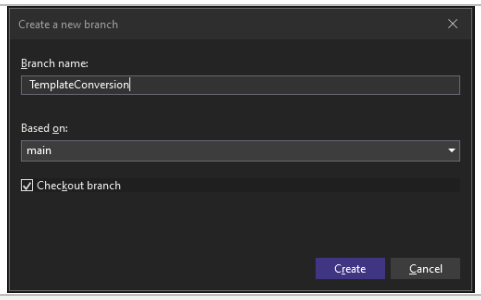
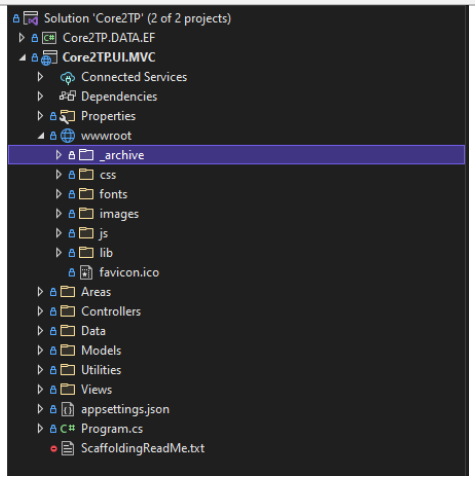
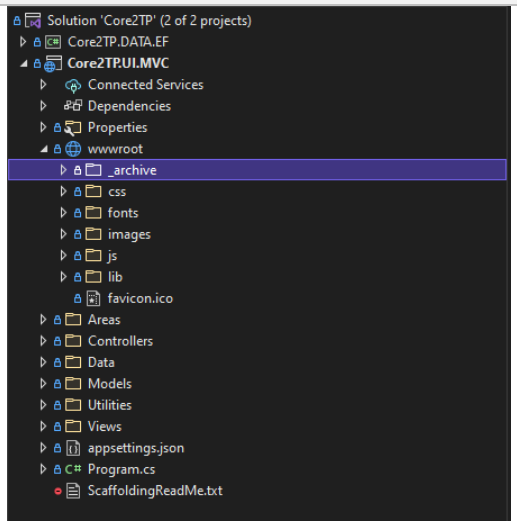
Update connection strings

1. Open appsettings.json from the root of your project
2. Update the connection string
   a. Server= .\\sqlexpress
   b. Database=[DbNameHere]

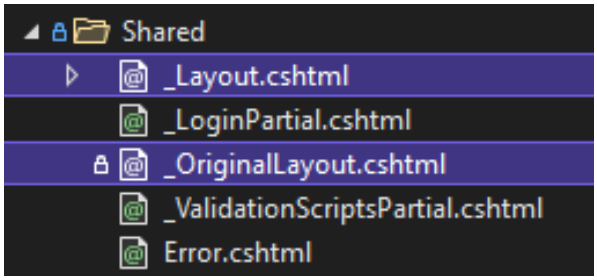
		c. Leave Trusted_Connection and MultipleActiveResultSets as default (true) 3. Save
3.		In NuGet Package Manager Console: 1. Ensure that the UI project is listed in the default project dropdown at the top of the console. 2. Type **update-database** 3. Press ENTER to run the code
4.		Add Identity Samples views… 1. In Solution Explorer, expand Areas → right click Identity → Add New Scaffolded Item 2. Select 'Identity' from the menu on the left → Add 3. In the 'Add Identity' window, check the box for 'Override All Files' 4. Data Context Class: ApplicationDbContext 5. Click Add
5.		Register Identity Samples 1. In Program.cs, locate the line **builder.Services.AddDefaultIdentity…** and replace the line with this code **builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true).AddRoles<IdentityRole>().AddRoleManager<RoleManager<IdentityRole>>().AddEntityFrameworkStores<ApplicationDbContext>();**

6.		Implement Identity Addons 1. Locate/Download the IdentityAddons.zip and extract the contents 2. Copy the folders **Controllers, Models, and Views** 3. Paste into the root of your UI project 4. Open AdministrationController.cs 5. Update the namespace to **YourProjectName.UI.MVC.Controllers** 6. Update the using statement **using YourProjectName.UI.MVC.Models**
7.		Commit and push 1. Open Git Changes 2. Type a Message describing the changes made 3. Click the arrow on the 'Commit All' button and select **Commit All and Push**
8.		Merge your pull request 1. In a browser, navigate to your repository in GitHub.com 2. Click the green 'Create Pull Request' button 3. Click Create Pull Request 4. Click Merge Automatically
9.		Update your local main repo 1. Expand Git Repository 2. Double click 'main' so it is highlighted 3. Next to Incoming(), click Pull
10.	PROCEED TO Converting a Template.	

Converting a Template

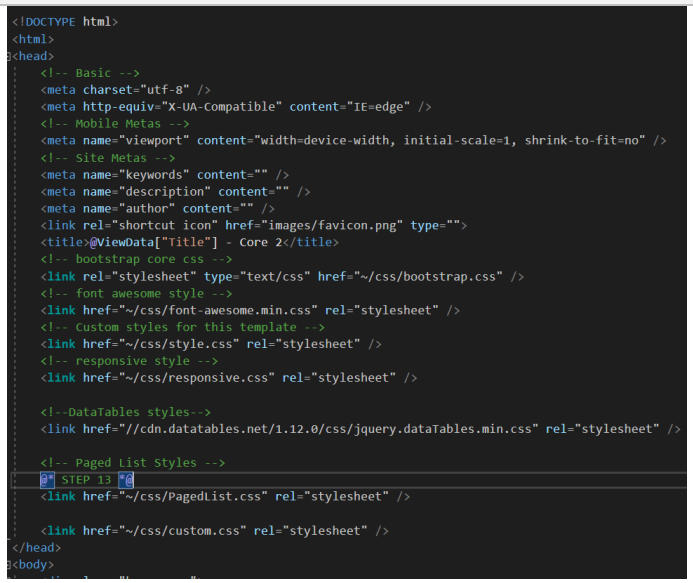
1.		1. Choose a template. 2. Create a new branch.
2.		1. Create an _Archive folder in wwwroot 2. Copy the extracted template files into the _Archive folder.
3.	1. Delete all of the folders from wwwroot EXCEPT _Archive (Note: You can optionally create a folder for these in the _Archive folder and move them there if you want to preserve them)	
4.		1. Paste all images, css, fonts, and js files/folders in the wwwroot folder. Some templates come with a vendor folder, that can be put into the Content folder as well.

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.

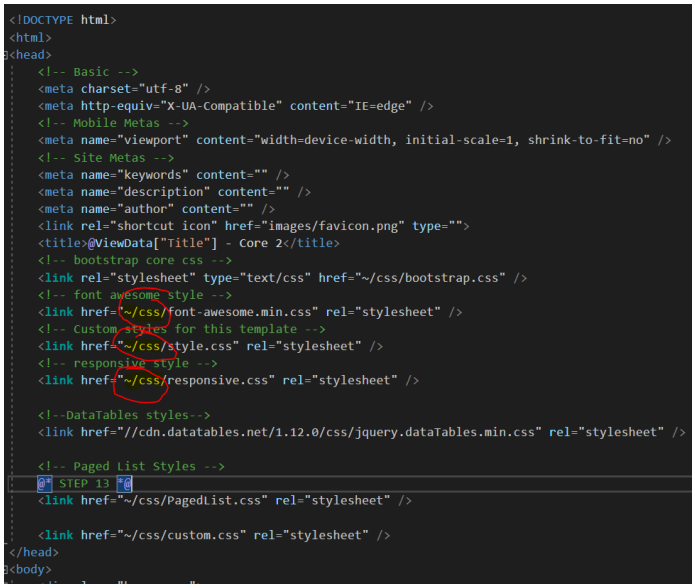
6.



```
<!DOCTYPE html>
<html>
<head>
    <!-- Basic -->
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <!-- Mobile Metas -->
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <!-- Site Metas -->
    <meta name="keywords" content="" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <link rel="shortcut icon" href="images/favicon.png" type="">
    <title>@ViewData["Title"] - Core 2</title>
    <!-- bootstrap core css -->
    <link rel="stylesheet" type="text/css" href="~/css/bootstrap.css" />
    <!-- font awesome style -->
    <link href="~/css/font-awesome.min.css" rel="stylesheet" />
    <!-- Custom styles for this template -->
    <link href="~/css/style.css" rel="stylesheet" />
    <!-- responsive style -->
    <link href="~/css/responsive.css" rel="stylesheet" />

    <!--DataTables styles-->
    <link href="//cdn.datatables.net/1.12.0/css/jquery.dataTables.min.css" rel="stylesheet" />

    <!-- Paged List Styles -->
    @" STEP 13 "@
    <link href="~/css/PagedList.css" rel="stylesheet" />

    <link href="~/css/custom.css" rel="stylesheet" />
</head>
<body>
```

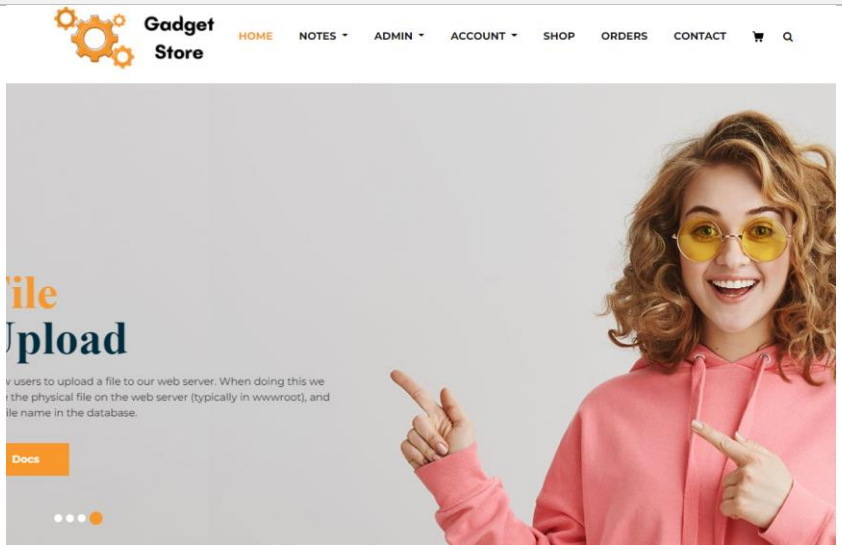
1. Open the template HTML file that you want to use as your Layout for the project
2. Copy all of the HTML in that file
3. Paste over ALL of the HTML in the _Layout file.

7.



```
<!DOCTYPE html>
<html>
<head>
    <!-- Basic -->
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <!-- Mobile Metas -->
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <!-- Site Metas -->
    <meta name="keywords" content="" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <link rel="shortcut icon" href="images/favicon.png" type="">
    <title>@ViewData["Title"] - Core 2</title>
    <!-- bootstrap core css -->
    <link rel="stylesheet" type="text/css" href="~/css/bootstrap.css" />
    <!-- font awesome style -->
    <link href="~/css/font-awesome.min.css" rel="stylesheet" />
    <!-- Custom styles for this template -->
    <link href="~/css/style.css" rel="stylesheet" />
    <!-- responsive style -->
    <link href="~/css/responsive.css" rel="stylesheet" />

    <!--DataTables styles-->
    <link href="//cdn.datatables.net/1.12.0/css/jquery.dataTables.min.css" rel="stylesheet" />

    <!-- Paged List Styles -->
    @" STEP 13 "@
    <link href="~/css/PagedList.css" rel="stylesheet" />

    <link href="~/css/custom.css" rel="stylesheet" />
</head>
<body>
```

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(), Html.ActionLink() helpers, or asp- attributes.
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 @await RenderSectionAsync(“scripts”, required: false) in the appropriate places in the Layout
8.	 `@{` `  ViewData["Title"] = "Home Page";` `}` `@section Slider{` `  <!-- slider section -->` `  <section class="slider_section ">` `  <div class="slider_bg_box">` `  <img src="~/images/slider-bg.jpg" alt="">` `  </div>` `  <div id="customCarousel1" class="carousel slide" data-ride="carousel">` `  <div class="carousel-inner">` `  <div class="carousel-item active">` `  <div class="container ">` `  <div class="row">` `  <div class="col-md-7 col-lg-6 ">` `  <div class="detail-box">` `  <h1>` `  <span>` `  Core 2` `  </span>` `   ` `  Intro to EF` `  </h1>` `  <p>` `  Entity Framework (EF) is an Object Relational Mapper (ORM) for .NET.  It he` `  </p>` `  <div class="btn-box">` `  <a href="https://docs.microsoft.com/en-us/ef/" class="btn1">` `  Docs` `  </a>` `  </div>` `  </div>` `  </div>` `  </div>` `  </div>` `  </div>`	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/push changes to your branch. 3. In GitHub in the browser, create a pull request and merge your branch into the main branch. 4. In VS, expand Git Repository, switch to main and pull

10.

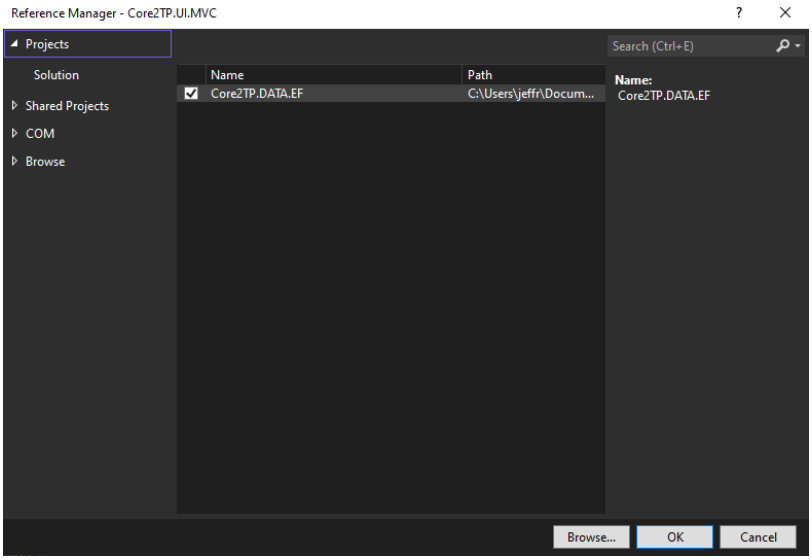
```
public IActionResult Index()
{
    return View();
}

#region Contact Form
//GET action --> here for the initial request.  Sends the blank form to the client.
0 references
public IActionResult Contact()
{
    return View();
}

//POST action --> here to handle form submission.  Accepts the filled out form as a param (cvm
[HttpPost]
0 references
public IActionResult Contact(ContactViewModel cvm)
{
    //Need credentials from appsettings.json

    //Need to build out the mailmessage with creds
    //need to send email via SMTP
    if (!ModelState.IsValid)
    {
        return View(cvm);
    }
    else
    {
        //process sending email
        #region Email Setup Steps & Email Info

        //1. Go to Tools > NuGet Package Manager > Manage NuGet Packages for Solution
        //2. Go to the Browse tab and search for NETCore.MailKit
        //3. Click NETCore.MailKit
        //4. On the right, check the box next to the CORE1 project, then click "Install"
        //5. Once installed, return here
        //6. Add the following using statements & comments:
```

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, go to GitHub in the browser, and create the PR and merge your branch into main
7. In VS, expand Git Repository, switch to main and pull.

Implementing Entity Framework

1.

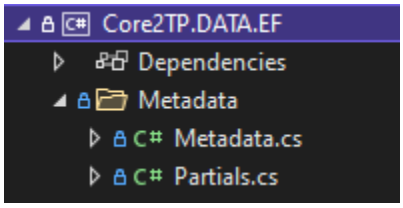
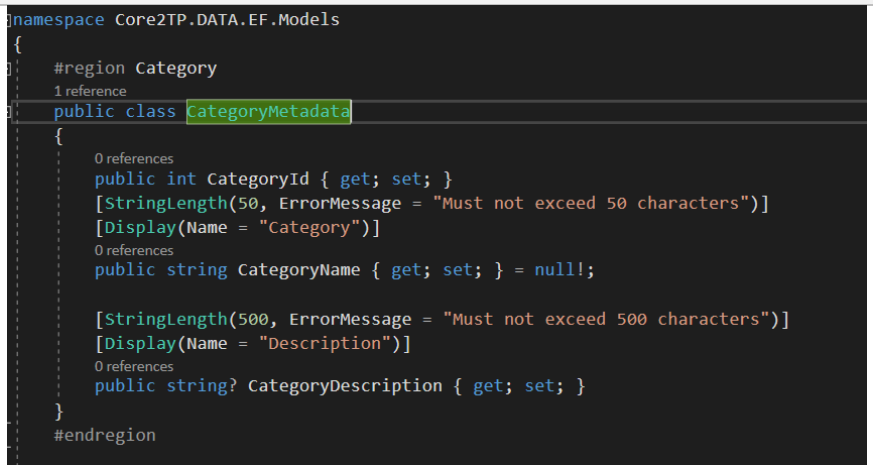


Create a new branch

1. Under the UI.MVC project, right click Dependencies → **Add a Project Reference**
2. Check the box for the DATA.EF project → **OK**
3. Open the Package Manager Console
4. Select DATA.EF in the Default Project dropdown
5. Enter the following commands (press enter after each command)
   6. Install-Package Microsoft.EntityFrameworkCore.SqlServer
   7. Scaffold-dbcontext "Server=.\sqlexpress;database=[dbnamehere];trusted_connection=true;multipleactiveresultsets=true;" Microsoft.EntityFrameworkC

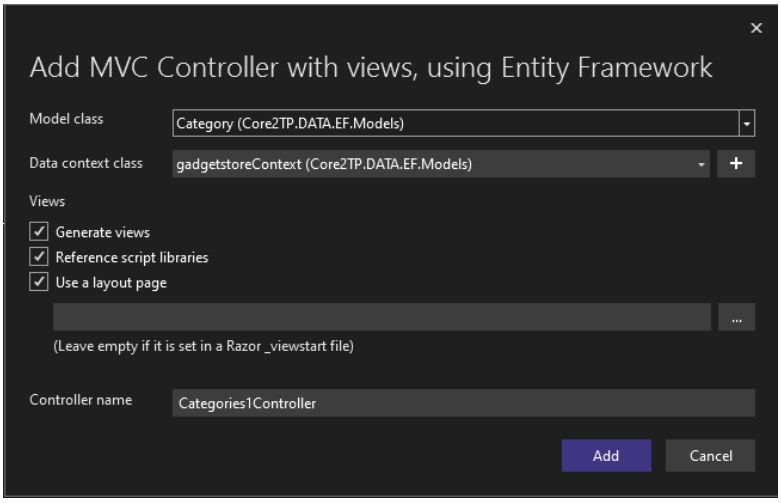
		ore.SqlServer -OutputDir Models
2.		Register the Service 1. Open Program.cs 2. Under the line **builder.Services.AddDbContext**, <u>add a new line of code</u>, **builder.Services.AddDbContext<[contextclassname]>(options => options.UseSqlServer(connectionString));** 3. Commit changes, go to GitHub in the browser, and create the PR and merge your branch into main 4. In VS, expand Git Repository, switch to main and pull

Adding Metadata

1.	Create a new branch. Remove class1.cs from the DATA.EF layer.	
2.		Add a metadata folder to the DATA.EF layer. 1. Add a new class named Metadata 2. Add a new class named Partials 3. Comment out the .Metadata at the end of the namespace in both classes
3.		For each class that needs metadata to add validation: 1. Locate and open the .cs file in the Models. 2. In Metadata.cs, create a public class named TableNameMetadata. 3. In Partials.cs, create a public partial class to create the helper/buddy class for the metadata to link up with the

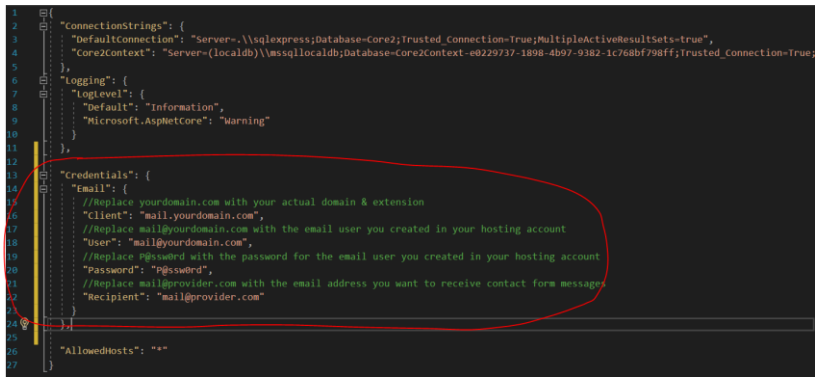
	Some questions to consider: 1. Is the field going to be **viewable**? (AuthorID) – no metadata 2. Is the field going to be **required**? Yes:(Add Required validation) 3. Is the field nullable? Yes: DisplayFormat(NullDisplayText=”x”) 4. Is the field a string value? Yes: StringLength() validation 5. No 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.	EntityModel class it’s associated with. 4. Add the appropriate fields in the metadata class and add appropriate data validation to the fields. 5. Complete this process for each class that needs metadata validation. 6. Commit changes, go to GitHub in the browser, and create the PR and merge your branch into main 7. In VS, expand Git Repository, switch to main and pull

Scaffolding Controllers/Views: The UI Layer

1.		Create a new branch and then complete the following: **1.** Right click the controllers folder and select Add Controller. **2.** Select MVC Controller with views, using Entity Framework. **3.** Model Class: Select the class for which you are creating the controller/views (Not Metadata) **4.** DataContext: Select the [DbNameContext] option in the dropdown. **5.** Leave all checkboxes checked **6.** Controller Name: should be the TableName+Controller (as shown) **7.** Click Add

		8. Add a link to 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 deta ils 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	

Protecting Sensitive Data

1.		Any credentials used in the site need to be referenced from the appsettings.json file 1. Open appsettings.json 2. Add the "Credentials" json object as shown **3.** Add **using Microsoft.Extensions.Configuration** to the Controller where the credentials are needed **4.** 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).**