Building a 2 tier application with Identity

Set-up Source Control

1. Create a new repository on BitBucket
    a. Repositories > Create Repository
    b. Name the repository and leave Git as the Repository Type > Create repository
    c. Click the Clone link and copy the content of the HTTPS box
2. Clone in SourceTree
    a. Click the Clone/New button
    b. Paste the contents of the HTTPS box from BitBucket in the Source Path/URL box
        i. If necessary, delete everything up to the https://…. (Git Command)
    c. Click the ellipses (…) next to Destination Path box
        i. Navigate to the local folder where you want to store your project
        ii. Create a new folder at this location *(*name this folder the same as the repo)*
        iii. Inside that newly created folder, add another folder titled "src"
    d. Click Clone

Build the simple Structure

3. New Project: Other Project Types, Blank Solution
    a. Name it the Name of your project (TwoTierReview)
4. Add a project to the solution
    a. Class Library (Data Layer) – Name it with an abbreviation of the solution with Data and the type of data being used (TTR.DATA.EF)
5. Add a project to the solution
    a. Web Application (UI Layer) - Name it with an abbreviation of the solution with UI and the type of UI being used (TTR.UI.MVC)
6. Add Reference to the UI.MVC Layer for the EF layer
    a. Right Click References in the UI.MVC Layer
    b. Add Reference
    c. From Projects select your DATA.EF layer

The UI Layer - Identity

7. Run IdentitySample – Nuget Package Manager Console
    a. If running from scratch
        i. Open a html page (Views/Home/About)
        ii. Retrieve the IdentitySample Run command from the toolbox (double click)
            1. `Install-Package Microsoft.AspNet.Identity.Samples -Pre`
        iii. Copy it from the about page and paste it into the pm> in the console
        iv. Check the default project dropdown MUST be the UI.MVC Layer
        v. Press Enter

vi. Select A for all and press enter

b. If running on an existing project
   i. Open a html page (Views/Home/About)
   ii. Retrieve the IdentitySample Run command from the toolbox (double click)
      1. `Install-Package Microsoft.AspNet.Identity.Samples -Pre`
   iii. Copy it from the about page and paste it into the pm> in the console
   iv. Check the default project dropdown MUST be the UI.MVC Layer
   v. Press Enter
   vi. Select Each file to be overwritten
      1. All files should be answered Y EXCEPT FOR
         a. Views/Shared/_layout
         b. Views/Home/Index
         c. Views/Home/Contact
         d. Global.asax.cs
         e. Global.asax
         f. Controllers/HomeController
         g. App_Start/RouteConfig.cs
         h. App_Start/BundleConfig.cs

c. All Projects
   i. **Find (In this Project UI.MVC) IdentitySample**
   ii. **Replace with your project name (TTR.UI.MVC)**
   iii. Root Web.config
      1. Remove the New Connection added by IdentitySample
   iv. **Update the Original DefaultConnection with the values of your database connection string.**
   v. Login with default credentials
      1. Admin@example.com
      2. Admin@123456
   vi. Go To UsersAdmin
      1. Add yourself as a user and choose a password **AND Select the Admin Role**
      2. Delete the Admin@example.com (security purposes)

The UI Layer - Convert Template to Layout

1. Choose a Template
2. Unzip the template into an _Archive Folder in your UI layer (if it does not exist, create it)
3. Copy Images, Styles, Js, into the _Archive Folder
4. Open the Template's index.html and the Views/Shared/_Layout
5. Delete all from the BODY of the _layout.cs **EXCEPT**
   a. Navigation (including the if statement for the identity management views)
   b. Html.Partial("_LoginPartial")
   c. RenderBody()
   d. All Render Scripts at the bottom

6. Paste in the contents of the Body of your template
7. Make it your own – **Do these in SMALL steps so you can ctrl+Z if necessary**
    a. Change Header and titles
    b. Change Navigation (copy from saved items)
    c. Add Html.Partial to nav or sidebar
    d. Locate the area for content, remove the existing content and replace with RenderBody()
    e. Remove all unnecessary links
    f. Update the footer
    g. Remove the site.css from the Bundles.config in the appStart folder add the template stylesheet(s) to the config
8. Set the UI layer as Start-Up Project in Solution Explorer (if it is not already)
    a. Test the UI

Data Layer

1. Remove the Class1.cs
2. Add new folder called Metadata
3. Add New item
4. Under C# templates choose Data
5. Choose ADO.NET Entity Data Model
    a. EF Designer From Database
6. New Connection
    a. ServerName: .\sqlexpress
    b. Select DataBase Name (your DB) from the dropdown
    c. OK
    d. Next
    e. Entity Framework 6.x
    f. Select tables
    g. All Checkboxes checked (Pluralize, include foreign key, import selected stored procs
    h. Finish
    i. BUILD – **(Ctrl+Shift+B)**
    j. Change any Mislabeled Table Names
    k. BUILD – **(Ctrl+Shift+B)**
    l. Close the tab
7. Open the App.Config
    a. Copy the YourDBEntities ConnectionString
    b. Paste into the ROOT web.config
8. Metadata Folder – You can Do all metadata at once or work in the patter of doing (Model, Controller, and then views for Each Db Table). The pattern is recommended for organization.
    a. Add Class called **DBNameMetadata** or an Individual file for each metadata item
    b. Add **properties from the .tt files into the Metadata Class**
    c. Add **new Class declaration for public partial** matching the .tt file
    d. Use the **MetadataType** attribute over the public partial class **typeof(MetadataClassName)**
    e. Add metadata attributes to the properties in the metadata class
        i. Is the field going to be **viewable**? (Product ID) - no metadata
        ii. Is the field **required** in the table? Yes: Add Required Validation
        iii. Is the field **nullable**? Yes: DisplayFormat(NullDisplayText="X")
        iv. Is the field a **string value**? Yes: StringLength() validation
        v. **No validation on Boolean items**.
        vi. Is field name suitable for user consumption? No: Add **Display(Name="X")**
        vii. Is the field a **BIG text** value? Yes: [UIHint("MultilineText")]
        viii. Does field require **special formatting**? Yes: [RegularExpression("pattern", ErrorMessage="X")] - **Email, phone, SSN, etc.**
        ix. Does the field have a specific **RANGE**? Yes: Range(min,max,ErrorMessage="X")

Code Example:

```
namespace TTR.DATA.EF//.Metadata - Namespace MUST match .tt file
{
    public class CategoryMetadata
    {
        //retrieved from the Category.cs from NorthwindModel.tt
        public int CategoryID { get; set; }
        [Display(Name ="Category")]
        [Required(ErrorMessage ="* Name is required")]
        [StringLength(15,ErrorMessage ="* Must be 15 chars or less.")]
        public string CategoryName { get; set; }

        [UIHint("MultilineText")]
        public string Description { get; set; }

        //public byte[] Picture { get; set; } - bmp picture not used

    }
    [MetadataType(typeof(CategoryMetadata))]
    public partial class Category { }
}
```

The UI Layer – Scaffolding Controllers/Views

1. Right click the controllers folder
    a. **Add Controller**
    b. Select **MVC5 Controller with views, using Entity Framework**
    c. **Model Class** – Select the class for which you are creating the views/controller **(NOT METADATA)**
    d. **DataContext** – Select the **Entities** Option in the dropdown
    e. **Mark the checkbox** for the following Options
        i. Generate Views
        ii. Reference Script Libraries
        iii. Use a Layout Page
    f. **Controller name**
        i. If it is just the table name – Leave as the default
        ii. If it is the table name prepended with the DB name, remove the DB Name prepending
            1. ProductsController (stays ProductsController)
            2. (from the Northwind Database) **NWProductsController** – **change to ProductsController**
    g. Select **Add**
2. **Add a link** to the index view of the new controller **to the layout**
3. 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**
    d. In the table/**Index view remove fields from the table that are unnecessary** that can be shown in the details

e. 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 details button from the view**.
f. **Test** each view before moving to the next

4. 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**
5. **Basic functionality is complete at this point**
    a. If you desire more views, additional functionality, you may do so at this point if time allows. You may also begin a list of desired functionality to complete at the end.
    b. **Ensure that ALL base requirements are met BEFORE adding additional functionality** outside of the scope of your project requirements.

Return to Data Layer, Number 8 and continue through until completing number 5 of this section. Repeat until all tables are represented with controllers and views.

TO DO:

- Soft Delete vs Hard Delete
- Checking information for soft delete
- Synching up views for soft delete (index and delete)