**Exercise 1**

1. Create a new MVC project.
2. Start the application to make sure it runs.
3. Change the content of the Home view.  Don't worry if you haven't worked with HTML before.  We'll cover HTML in a future lecture.  For now, just look at the HTML and see if you can figure out how each of the HTML tags affect the display in the browser.
4. Remove Contact action and view.
5. Find the common layout and so the following:
   1. Change the application name text.
   2. Remove the link to Contact in the navigation bar.
   3. Modify the text in the copyright footer.
   4. What dynamically generates the copyright year?
      1. Razor syntax c# function DateTime.Now.Year
   5. Where does the year get generated, browser or Web server?
      1. On the Server
6. Start the application to make sure you can see the changes you made in step 5.

**Exercise 2**

1. Create a folder in your project and name it "Data".
2. Create an in-memory database using a static class under the "Data" folder.
3. Create a sub-folder under "Data" and call it "Entities".
4. Create a model for an application entity named “User” under the “Entities” folder.  For the new entity's properties,  
   Include the following:
   1. first name
   2. middle name
   3. last name
   4. email address
   5. years in school (make an int)
   6. anything else you want
5. Create an empty controller with an action named "Create" for rendering a create “User” view. Remember that this is an HTTP GET endpoint.  
   Insert [HttpGet] (with the brackets) just above the new action.
6. Create a view that renders an HTML form for entering a new User.  Use the layout that was created in Exercise 1 (defined in the \_ViewStart.cshtml file.

What are the items that start with "@Html." called?

Razor syntax  
How does the view know what model it is supposed to use for rendering input fields for model properties?

It is included at the top of the file as:

@model IEnumerable<cset356Lab2.Data.Entities.Users>

1. Create a link in the top navigation bar that allows a user to navigate from the starting page to the create User page.
2. Start the application to make sure navigation to the create User page works and that the HTML form displays as expected (all properties of the model are present).  
   Is the text displayed in the footer the same as what is displayed in the footer on the home page?  Why is it the same?
3. Click on the "Back to List" link.  What happens?  Why?

It throughs a 404 error because the redirect has not been changed to go to List instead of the default Index. Once I changed that it redirected to the list view

**Exercise 3**

1. Add a static property to the in-memory database that represents a collection of "Users".
2. Create an action named "Create" that takes a User as input and stores it in the “Users” collection.  Remember that this is an HTTP POST endpoint.  
   Insert [HttpPost] (with the brackets) just above the new action.
3. Add an action and view combination that renders a list of existing Users.  
   Where is the "Users" data defined in the view?
4. In the "list" action, get the current collection of Users from the in-memory database and pass the collection to the view.
5. Add code to redirect the user to the list of Users after they create a new User.
6. Change the link in the navigation bar to allow the user to navigate to the list User page rather than the create User page.
7. Start the application to make sure it is functioning as expected (you can add a User and see a list of users that have been added).

**Exercise 4**

1. Run the application and try adding a User with only the "Years in School" entered.  
   What is displayed when the application returns you to the list of Users?
   1. I had done things slightly out of order so I didn’t get to see this but I assume that it would have displayed an entry in the table with blanks for all fields except the years in school field which was filled out. Without the [required] I would assume that it would let you save a user with fields missing and you may even be able to have a user id that had no other information because there would be no data verification.
2. Add attributes to the User model to force all of the input fields to be required except middle name.  
   Try adding a "User" with one or more of the required fields missing.  What happens?
3. Add logic in the create action to check the validity of the model.  If valid redirect the user to the list User page.  
   Otherwise, send them back to the User create page with the User data they entered.
4. Add attributes to the User model to make the labels for the input fields more “user friendly”.
5. Start the application to make sure it is functioning as expected.

**Exercise 5 (Skip this step as we will cover databases in the Week 4 lecture)**

1. Replace the in-memory database with a real database.  
     
   * Keep the in-memory database static class for next week’s lab.
   * Add EntityFramework NuGet package.
   * Add an int UserId property to the User model with the [Key] attribute;
   * Recreate the UserController using the EntityFramework template in Visual Studio.
   * Comment out the actions that do not yet have views.
   * Start the application to make sure it is functioning as expected.

**Extra Credit**

1. On the User list page you notice that each User listed has links at the end of the listing for editing, displaying details and deleting a user.  Implement the "details" function.