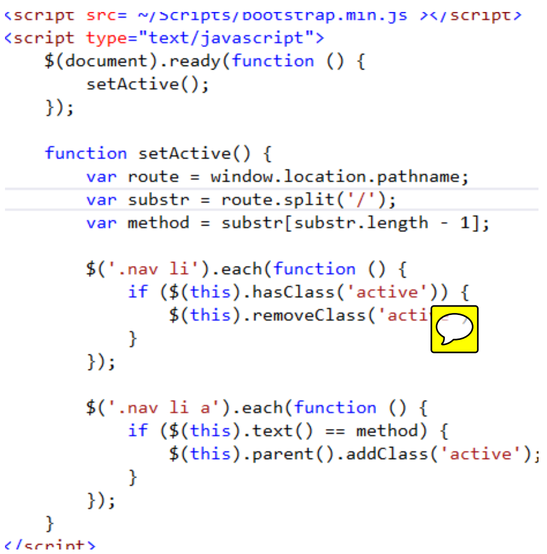
INFO3067 Week 1 Class 2

Review

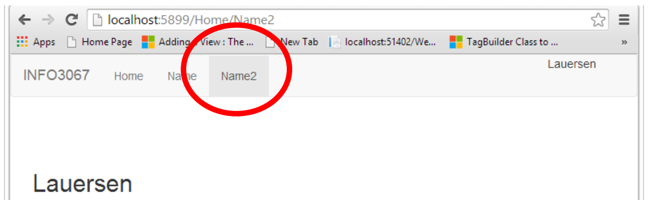
- Course Software
- ASP frameworks
- Controllers
    - ActionResult
- Views
    - Index.cshtml
- Shared_Layout.cshtml
- Bootstrap
    - Nuget utility

Bootstrap – Jquery code to set Active class

If you want, you can add the following code to the bottom of the _layout.cshtml shared style to include this JQuery code. This code should highlight the anchor that was clicked.



```
<script src="~/Scripts/bootstrap.min.js"></script>
<script type="text/javascript">
    $(document).ready(function () {
        setActive();
    });

    function setActive() {
        var route = window.location.pathname;
        var substr = route.split('/');
        var method = substr[substr.length - 1];

        $('.nav li').each(function () {
            if ($(this).hasClass('active')) {
                $(this).removeClass('acti
            }
        });

        $('.nav li a').each(function () {
            if ($(this).text() == method) {
                $(this).parent().addClass('active');
            }
        });
    }
</script>
```

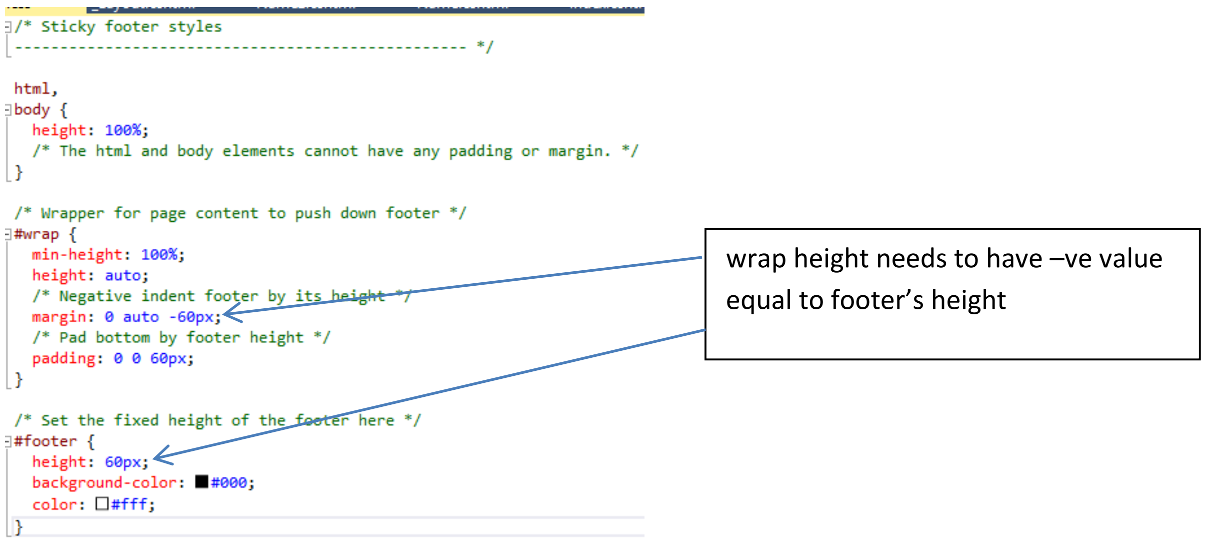


Bootstrap cont'd - Adding A Footer

We have a rudimentary menu in place from last class and today we'll add a footer, and include some other css to simulate a popup window. Then we'll leave the MVC world and set up the backend for our N-Tier infrastructure for case 1.

Continuing on with the MVCExercise1 project from last class do the following:

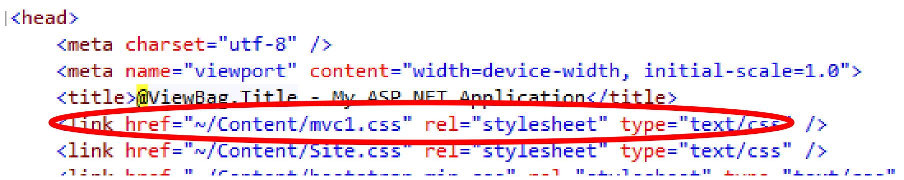
- Add a style sheet to your Content folder called **mvc1.css**



```
/* Sticky footer styles
-------------------------------------------------- */

html,
body {
  height: 100%;
  /* The html and body elements cannot have any padding or margin. */
}

/* Wrapper for page content to push down footer */
#wrap {
  min-height: 100%;
  height: auto;
  /* Negative indent footer by its height */
  margin: 0 auto -60px;
  /* Pad bottom by footer height */
  padding: 0 0 60px;
}

/* Set the fixed height of the footer here */
#footer {
  height: 60px;
  background-color: ■#000;
  color: □#fff;
}
```

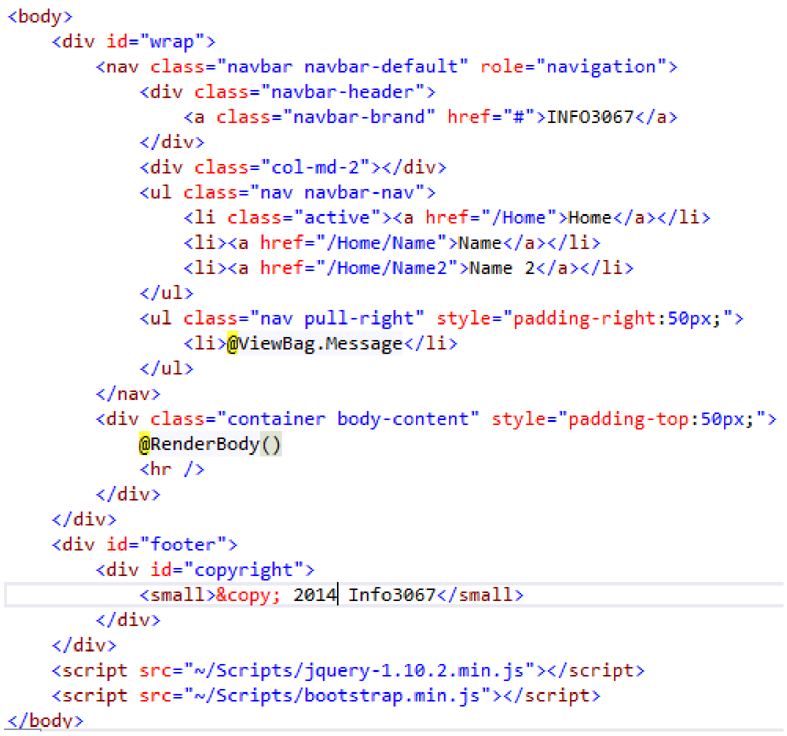
wrap height needs to have –ve value equal to footer's height

- Notice that the **margin** in the wrap section is -60px and this is the same as the footer section's **height** value.
- Add a reference to this new stylesheet in theViews\Shared\ _Layout.cshtml file:



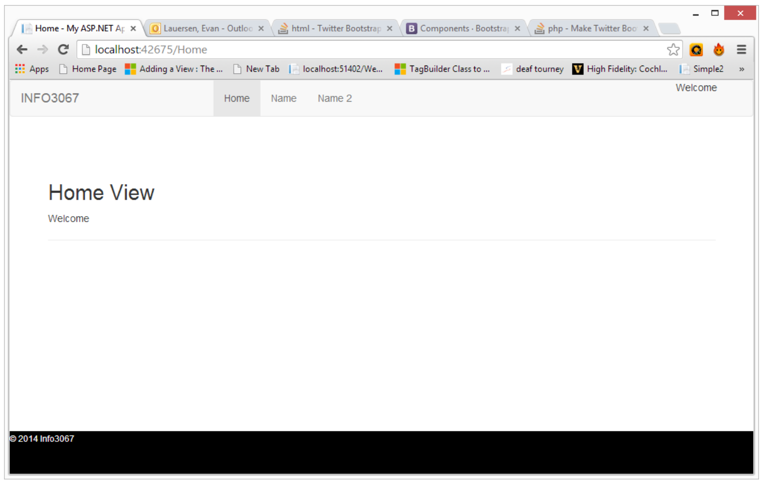
```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    <link href="~/Content/mvc1.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
```

- Modify your markup so it incorporates the new wrap and footer tags:



```
<body>
    <div id="wrap">
        <nav class="navbar navbar-default" role="navigation">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">INFO3067</a>
            </div>
            <div class="col-md-2"></div>
            <ul class="nav navbar-nav">
                <li class="active"><a href="/Home">Home</a></li>
                <li><a href="/Home/Name">Name</a></li>
                <li><a href="/Home/Name2">Name 2</a></li>
            </ul>
            <ul class="nav pull-right" style="padding-right:50px;">
                <li>@ViewBag.Message</li>
            </ul>
        </nav>
        <div class="container body-content" style="padding-top:50px;">
            @RenderBody()
            <hr />
        </div>
    </div>
    <div id="footer">
        <div id="copyright">
            <small>&copy; 2014 Info3067</small>
        </div>
    </div>
    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
    <script src="~/Scripts/bootstrap.min.js"></script>
</body>
```

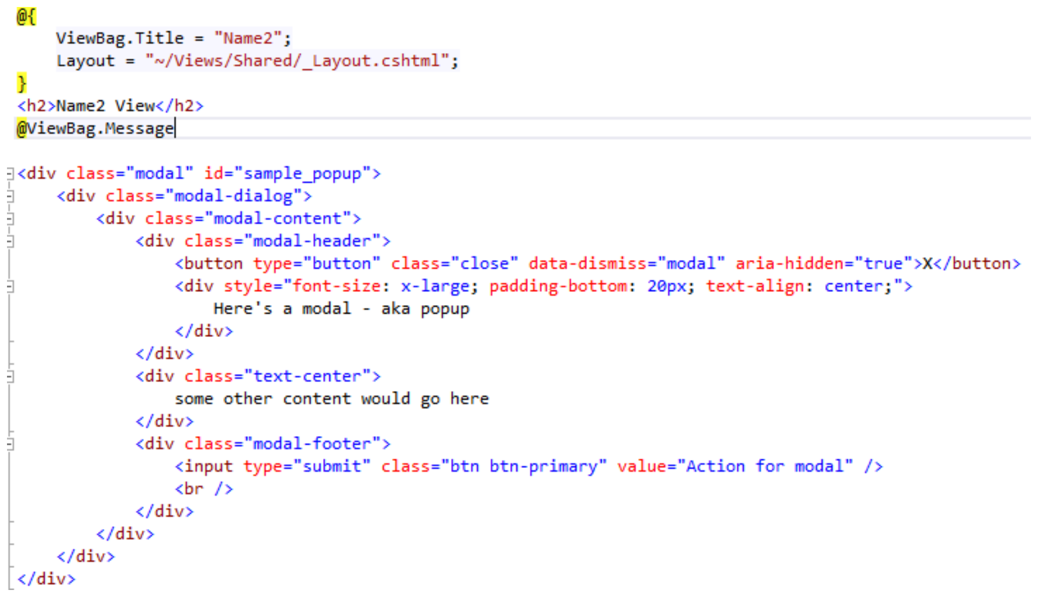
- Then try selecting the options off the menu:



Using the Modal component in bootstrap

There will be a few spots in the case study where it will be handy to use a popup window. Bootstrap has a built in modal component that will fill this requirement quite nicely. We'll walk through a basic example that you can replicate when needed.

- Add the following markup to your Name2.cshtml file:

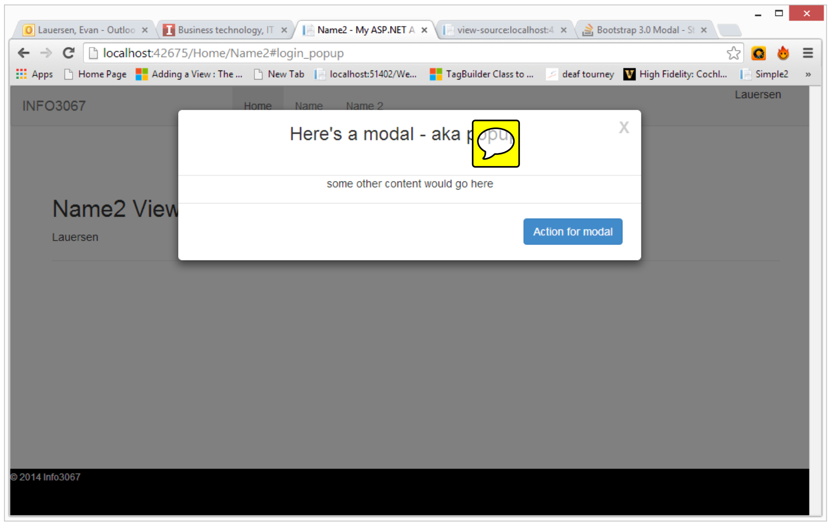


```
@{
    ViewBag.Title = "Name2";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Name2 View</h2>
@ViewBag.Message

<div class="modal" id="sample_popup">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">X</button>
                <div style="font-size: x-large; padding-bottom: 20px; text-align: center;">
                    Here's a modal - aka popup
                </div>
            </div>
            <div class="text-center">
                some other content would go here
            </div>
            <div class="modal-footer">
                <input type="submit" class="btn btn-primary" value="Action for modal" />
                <br />
            </div>
        </div>
    </div>
</div>
```

- Then add another anchor in the _Layout.cshtml to have this code:

```
<li><a href="/Home/Name2#sample_popup" data-toggle="modal">Modal</a></li>
```

- Then try the new menu item, notice that we stay on the Name2 page but the css simulates a popup window



Name2 View

Lauersen

Here's a modal - aka popup

some other content would go here

Action for modal

© 2014 Info3067

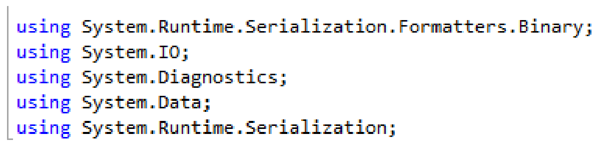
We'll leave the MVCExercises project for now, but we'll come back to it next week. Now we turn our attention to setting up the case study framework.

eStoreCase1 Setup

We'll start the case setup process by setting up the database. Run the script found in the class 2 section on FOL called **InitialStoreSQL.txt** in a Sql Server mgmt. studio query window. Don’t forget to change the location for the database **yourpathgoeshere** to an actual folder location as the script will not create a folder for you.

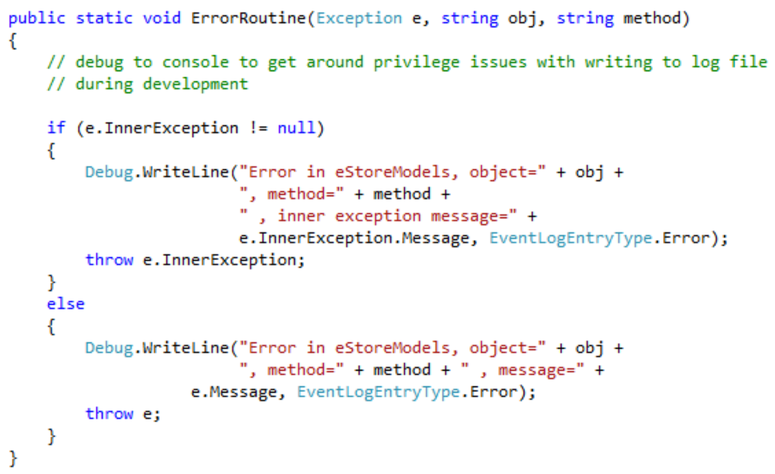
Once the database is setup, return to Visual Studio and setup the following:

- Blank solution call it **eStoreCase1**
- Add a C# class library project called **eStoreModels**
    - Delete the Class1.cs file
    - Add an ADO.NET Entity Data Model called **eStoreDBModel.edmx**
        - Generate from Database
        - Create a new connection and point it to the new db and use the name **eStoreDBEntities**
        - Choose Entity Framework 6
        - Add the 4 tables to the model (Customers, Products, Orders, OrderLineItems)
    - Add a new class called **eStoreModelConfig.cs**
        - Make the class public
        - Add these usings:



```
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Diagnostics;
using System.Data;
using System.Runtime.Serialization;
```

- Add code for the following 3 methods:



```
public static void ErrorRoutine(Exception e, string obj, string method)
{
    // debug to console to get around privilege issues with writing to log file
    // during development

    if (e.InnerException != null)
    {
        Debug.WriteLine("Error in eStoreModels, object=" + obj +
                        ", method=" + method +
                        " , inner exception message=" +
                        e.InnerException.Message, EventLogEntryType.Error);
        throw e.InnerException;
    }
    else
    {
        Debug.WriteLine("Error in eStoreModels, object=" + obj +
                        ", method=" + method + " , message=" +
                   e.Message, EventLogEntryType.Error);
        throw e;
    }
}
```

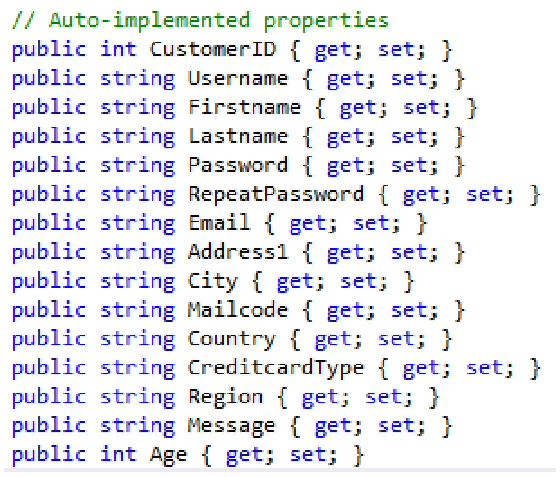
```
public static byte[] Serializer(Object inObject)
{
    BinaryFormatter frm = new BinaryFormatter();
    MemoryStream strm = new MemoryStream();
    frm.Serialize(strm, inObject);
    byte[] ByteArrayObject = strm.ToArray();
    return ByteArrayObject;
}

/// <summary>
/// Deserializer
/// </summary>
/// <param name="ByteArrayIn">Serialized Object from BusinessUser
/// <returns>Reconstructed Object</returns>
public static Object Deserializer(byte[] ByteArrayIn)
{
    BinaryFormatter frm = new BinaryFormatter();
    MemoryStream strm = new MemoryStream(ByteArrayIn);
    Object returnObject = frm.Deserialize(strm);
    return returnObject;
}
```

- Add another class called **CustomerModel.cs**
    - Have it inherit from eStoreModelConfig
    - Make it public scoped
    - Add this Register method:

```
/// <returns>int representing newly updated id for customer or -1 if error</returns>
public int Register(byte[] bytCustomer)
{
    int custId = -1;
    Customer cust = new Customer();
    eStoreDBEntities dbContext = new eStoreDBEntities();

    try
    {
        Dictionary<string, Object> dictionaryCustomer = (Dictionary<string, Object>)Deserializer(bytCustomer);
        dbContext = new eStoreDBEntities();
        String username = Convert.ToString(dictionaryCustomer["username"]);
        cust = dbContext.Customers.FirstOrDefault(c => c.Username == username);
        cust.FirstName = Convert.ToString(dictionaryCustomer["firstname"]);
        cust.LastName = Convert.ToString(dictionaryCustomer["lastname"]);
        cust.Email = Convert.ToString(dictionaryCustomer["email"]);
        cust.Age = Convert.ToInt32(dictionaryCustomer["age"]);
        cust.Address1 = Convert.ToString(dictionaryCustomer["address1"]);
        cust.City = Convert.ToString(dictionaryCustomer["city"]);
        cust.Mailcode = Convert.ToString(dictionaryCustomer["mailcode"]);
        cust.Region = Convert.ToString(dictionaryCustomer["region"]);
        cust.Country = Convert.ToString(dictionaryCustomer["country"]);
        cust.Creditcardtype = Convert.ToString(dictionaryCustomer["creditcardtype"]);
        dbContext.SaveChanges();
        custId = cust.CustomerID;
    }
    catch (Exception ex)
    {
        ErrorRoutine(ex, "CustomerModel", "Register");
    }
    return custId;
}
```

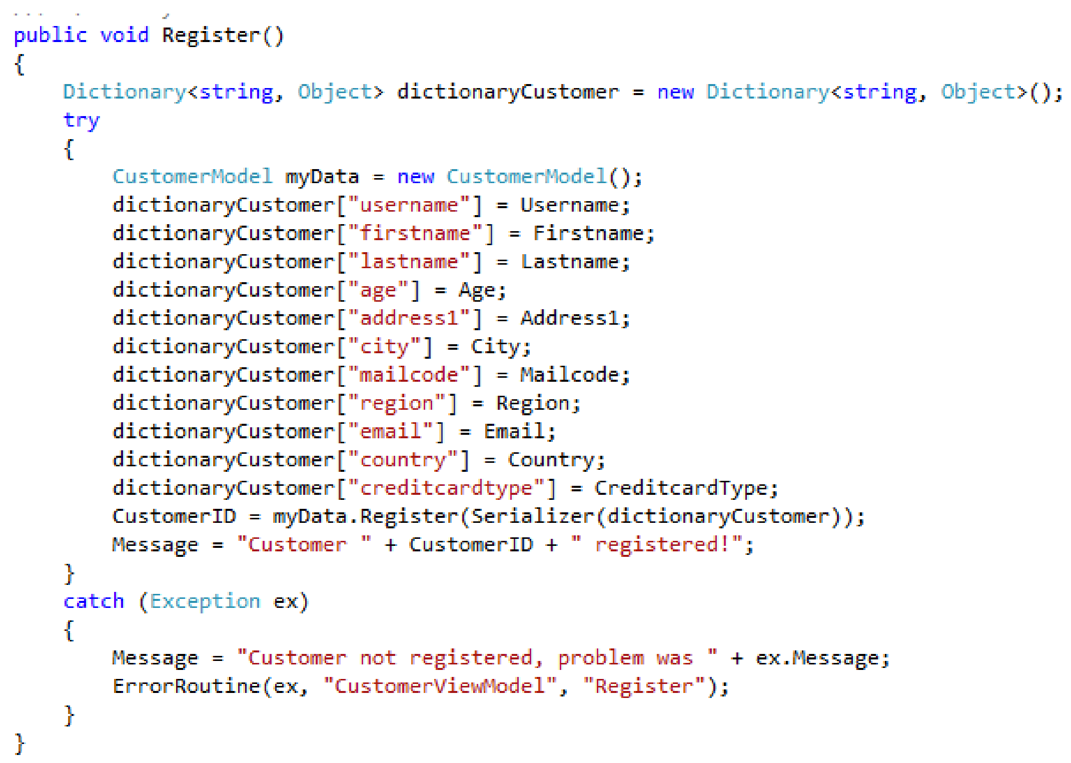
- Build the dll and insure there are no errors
- Add another class library to the solution called **eStoreViewModels**
    - Have it reference eStoreModels

- Add a class called **eStoreViewModelConfig.cs**
  - Use eStoreModelConfig as template and make necessary changes
- Add a class called **CustomerViewModel.cs**
  - Add the following auto-implemented properties:



```
// Auto-implemented properties
public int CustomerID { get; set; }
public string Username { get; set; }
public string Firstname { get; set; }
public string Lastname { get; set; }
public string Password { get; set; }
public string RepeatPassword { get; set; }
public string Email { get; set; }
public string Address1 { get; set; }
public string City { get; set; }
public string Mailcode { get; set; }
public string Country { get; set; }
public string CreditcardType { get; set; }
public string Region { get; set; }
public string Message { get; set; }
public int Age { get; set; }
```

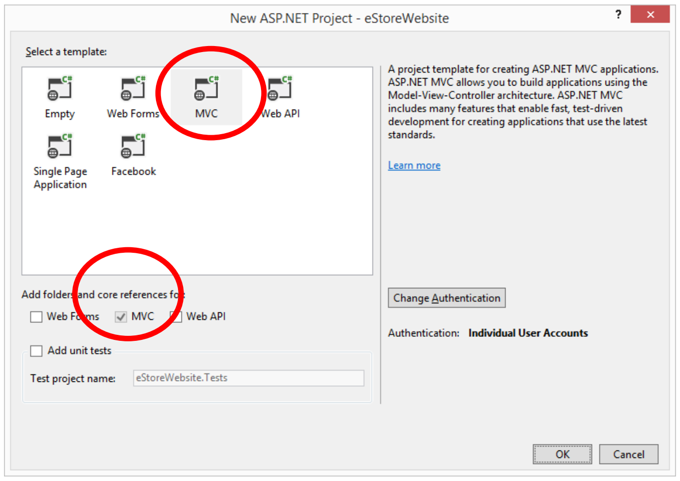
- Add the following **Register** method



```
public void Register()
{
    Dictionary<string, Object> dictionaryCustomer = new Dictionary<string, Object>();
    try
    {
        CustomerModel myData = new CustomerModel();
        dictionaryCustomer["username"] = Username;
        dictionaryCustomer["firstname"] = Firstname;
        dictionaryCustomer["lastname"] = Lastname;
        dictionaryCustomer["age"] = Age;
        dictionaryCustomer["address1"] = Address1;
        dictionaryCustomer["city"] = City;
        dictionaryCustomer["mailcode"] = Mailcode;
        dictionaryCustomer["region"] = Region;
        dictionaryCustomer["email"] = Email;
        dictionaryCustomer["country"] = Country;
        dictionaryCustomer["creditcardtype"] = CreditcardType;
        CustomerID = myData.Register(Serializer(dictionaryCustomer));
        Message = "Customer " + CustomerID + " registered!";
    }
    catch (Exception ex)
    {
        Message = "Customer not registered, problem was " + ex.Message;
        ErrorRoutine(ex, "CustomerViewModel", "Register");
    }
}
```

- Build the eStoreViewModels project and insure there are no errors
- Add a new MVC based website to the solution called **eStoreWebsite**

- Now choose the **MVC** and **MVC** options which will install a pile of extra files that we won't need but all of the references that we will.



New ASP.NET Project - eStoreWebsite

Select a template:

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Add folders and core references for:

[ ] Web Forms [✓] MVC [ ] Web API

[ ] Add unit tests

Test project name: eStoreWebsite.Tests

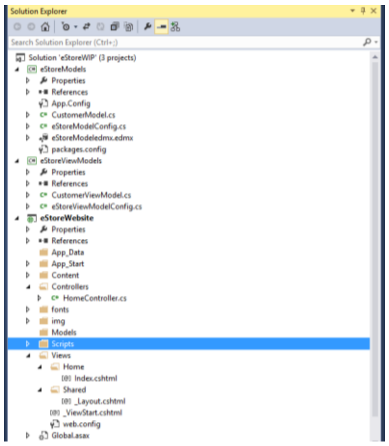
Authentication: **Individual User Accounts**

-
- Set this project as the **startup project** for the solution
- Add a reference for the eStoreViewModels project
- Add a new stylesheet called **eStore.css** to the Content folder and place the contents of the mvc1.css in it
- Modify the view in the Views\Home folder called **Index.cshtml**
  - Remove the existing contents and replace with the contents with the MVCExercises Name2.cshtml view, change the Contents to say **eStore Home** instead of Name2 View
  - Change the modal's id to register_popup
- Update the _**Layout.cshtml** in the Shared folder
  - Replace the contents with _Layout.cshtml from the MVCExercises project
  - Change the mvc1.css to eStore.css
  - Change the menu to have 2 items on it
    - Home→/Home
    - Register→/Home#register_popup (don't for get to add the data-toggle attribute to trigger the modal)
  - Add some sort of graphic(s) to your home page (that represents what kind of product your site will be setup to sell)

LAB 2

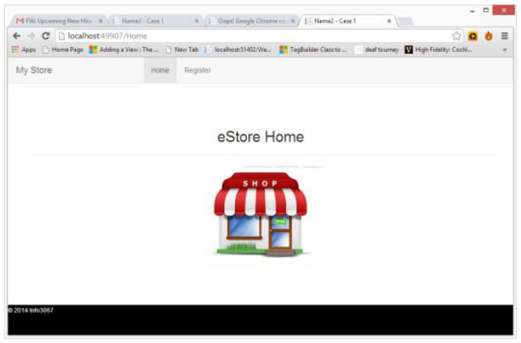
Submit the following:

- Screenshot showing the case study framework (1 solution, 3 projects), make sure I can see the classes in the .dll projects, and the controller and view classes in the web site project.

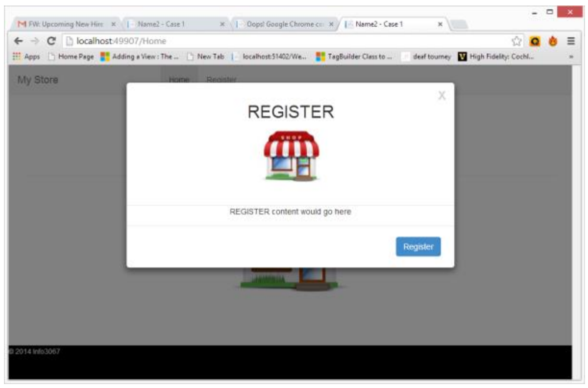


```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'eStoreWIP' (3 projects)
  eStoreModels
    Properties
    References
    App.Config
    CustomerModel.cs
    eStoreModelConfig.cs
    eStoreModeledmx.edmx
    packages.config
  eStoreViewModels
    Properties
    References
    CustomerViewModel.cs
    eStoreViewModelConfig.cs
  eStoreWebsite
    Properties
    References
    App_Data
    App_Start
    Content
    Controllers
      HomeController.cs
    fonts
    img
    Models
    Scripts
    Views
      Home
        Index.cshtml
      Shared
        _Layout.cshtml
      _ViewStart.cshtml
      web.config
    Global.asax
```

- Screenshot showing the home page with your own image of something you intend on selling, plus the bootstrap menu and footer



- Screenshot showing the register modal off of the home page



REGISTER content would go here

Summary of Key Terms

- Footer
    - Wrap div
- Modals
- Case 1 Architecture
    - Models
        - eStoreModelConfig
        - CustomerModel
    - ViewModels
        - eStoreViewModelConfig
        - CustomerViewModel
    - eStoreWebsite
        - internet application