INFO3067 Week 3 Class 2

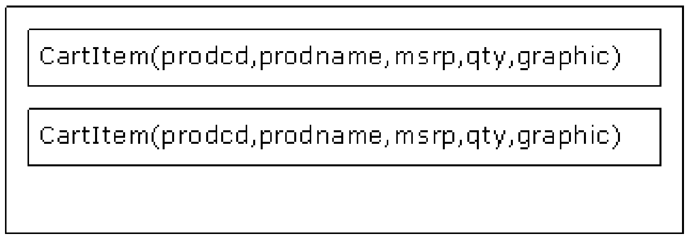
Review

- Login/Logout
- ProductViewModel, ProductModel

Case Study cont’d - Shop

Before discussing the details of the catalogue page, there is a new object that we'll need to help us maintain what products and quantities the user has selected; I’ve called this object the **CartItem** object. Its layout is seen below. We'll actually be using an **array** of CartItem objects in the catalogue and refer to the entire array as the **"Cart"** variable. We'll also need this array on multiple pages, so we'll end up putting it into and taking it out of a Session variable. This will occur as we leave and return to any of the pages in the shopping process. Note, CartItem belongs in the ViewModelObjects project.

Cart — array of CartItems

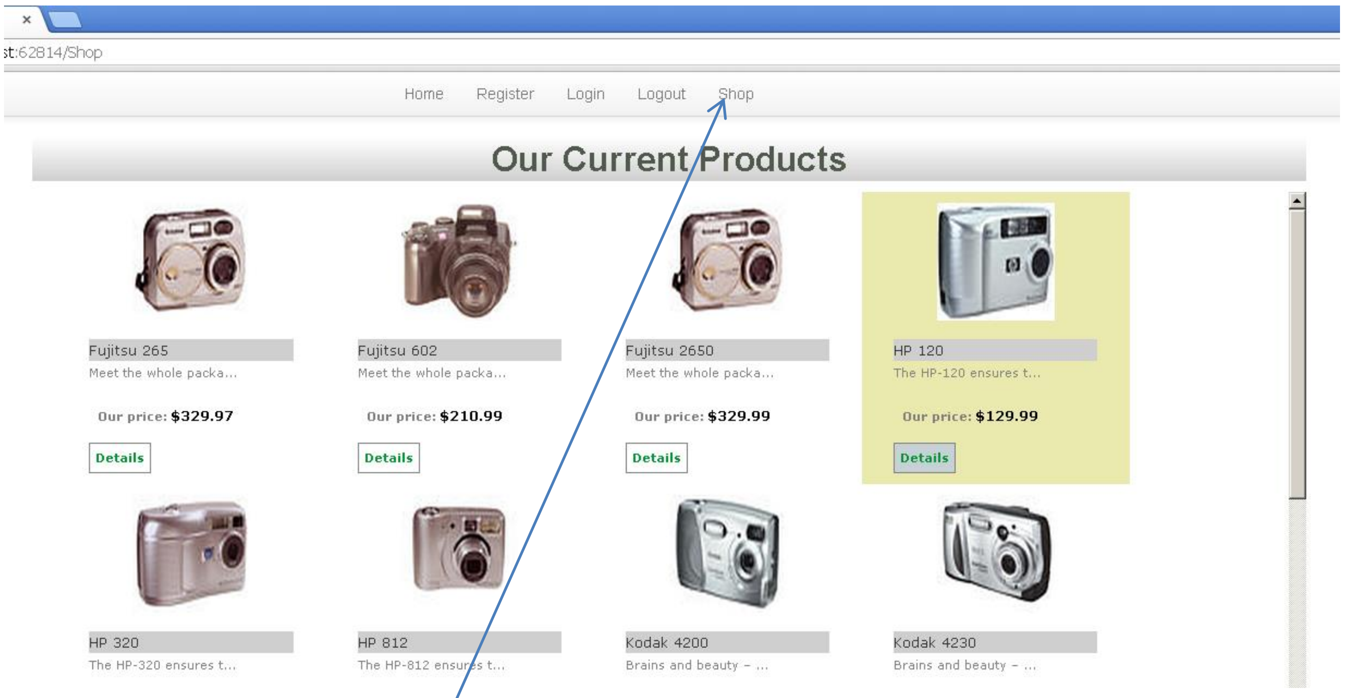

CartItem(prodcd,prodname,msrp,qty,graphic)
CartItem(prodcd,prodname,msrp,qty,graphic)

Session[“Cart”] ---------------- ----------------→Local Page--------------------------------------→Session[“Cart”]

```
/// <summary>
///  CartItem - simple container object to hold 1 item of the catalogue
///
/// Revisions      DD/MM/YY  Description
/// ----------     --------  -----------
/// E. Lauersen    06/05/13  Initial Code
/// </summary>

public class CartItem
{
    //
    //  Properties
    //
    public int Qty { get; set; }
    public string ProdCd { get; set; }
    public string ProdName { get; set; }
    public string Description { get; set; }
    public string Graphic { get; set; }
    public decimal Msrp { get; set; }
}
```

Catalogue

A catalogue can have many different looks, I encourage you do some googling to come up with your own style. A small sample is shown below (note, you will need a series of thumbnails for the products your site will be selling and **nobody is allowed to sell cameras**):



Notice that there is a new **Shop** entry in the main menu. When clicked we present a series of products. If the user so chooses they may get more details on the product by pressing the details button (which is not really a button but an anchor, as we will soon see).

There is a fair amount going on behind the scenes with this page. Starting with Shop menu click, we direct the user to a new **ShopController**. This controller's job is to set up the Cart Session variable and provide the data to its corresponding View. The Index method for this Controller would be something like what is shown below. Notice that we do not go to the database each time the page is shown, rather we **only go to the database once** and then on subsequent passes we just transfer control over to the View:

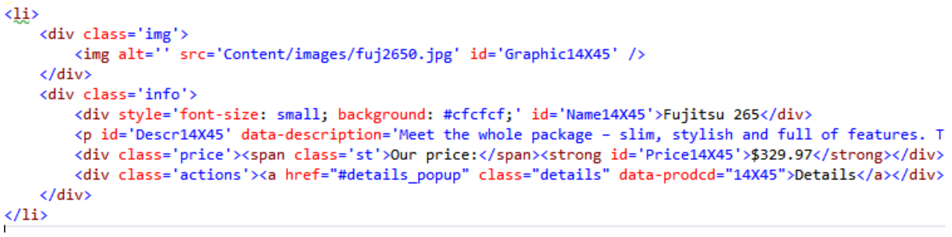
```
public ActionResult Index()
{
    if (Session["Cart"] == null) // haven't been to db yet
    {
        try
        {
            ProductViewModel prod = new ProductViewModel();
            List<ProductViewModel> Prods = prod.GetAll();
            if (Prods.Count() > 0)
            {
                CartItem[] myCart = new CartItem[Prods.Count]; // array
                int ctr = 0;

                // build CartItem array from List contents
                foreach (ProductViewModel p in Prods)
                {
                    CartItem item = new CartItem();
                    item.ProdCd = p.ProdCode;
                    item.ProdName = p.ProdName;
                    item.Graphic = p.Graphic;
                    item.Msrp = p.Msrp;
                    item.Description = p.Description;
                    item.Qty = 0;
                    myCart[ctr++] = item;
                }
                Session["Cart"] = myCart;  // load to session
            }
        }
        catch (Exception ex)
        {
            ViewBag.Message = "Catalogue Problem - " + ex.Message;
        }
    }
    return View();
}
```

A Custom HTML Helper

As we have seen, there are a number of built-in HTML Helpers available to us to generate HTML (see week 2 class 1 notes). There may be times when we want to re-use or encapsulate some dynamically generated html. To facilitate this, we can create **our own HTML Helper**. For our catalogue it would be nice to just place the catalogue in our view as simple tag like this:

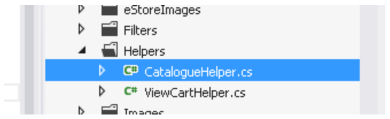
```
@using eStoreWebsite.Helpers
@{
    ViewBag.Title = "Shop/Index";
}
<div id="wrap" style="overflow: auto;">
    <div style="position: absolute; top: 50px; left: 160px;" id="main-content" data-mnuitem="shpMnuItm">
        <div class="span12">
            <h2 id="cathead">Our Current Products</h2>
            <div id="catalogue">
                @Html.Catalogue("mycat")
            </div>
        </div>
    </div>
</div>
```

In my case, I used this helper is to generate the catalogue contents in the form of an unordered list (**<ul>**). Then, I included the html for each product as a line item (**<li>**) of that unordered list. If we looked at the page source for one of these line items from above, we'd see something like:

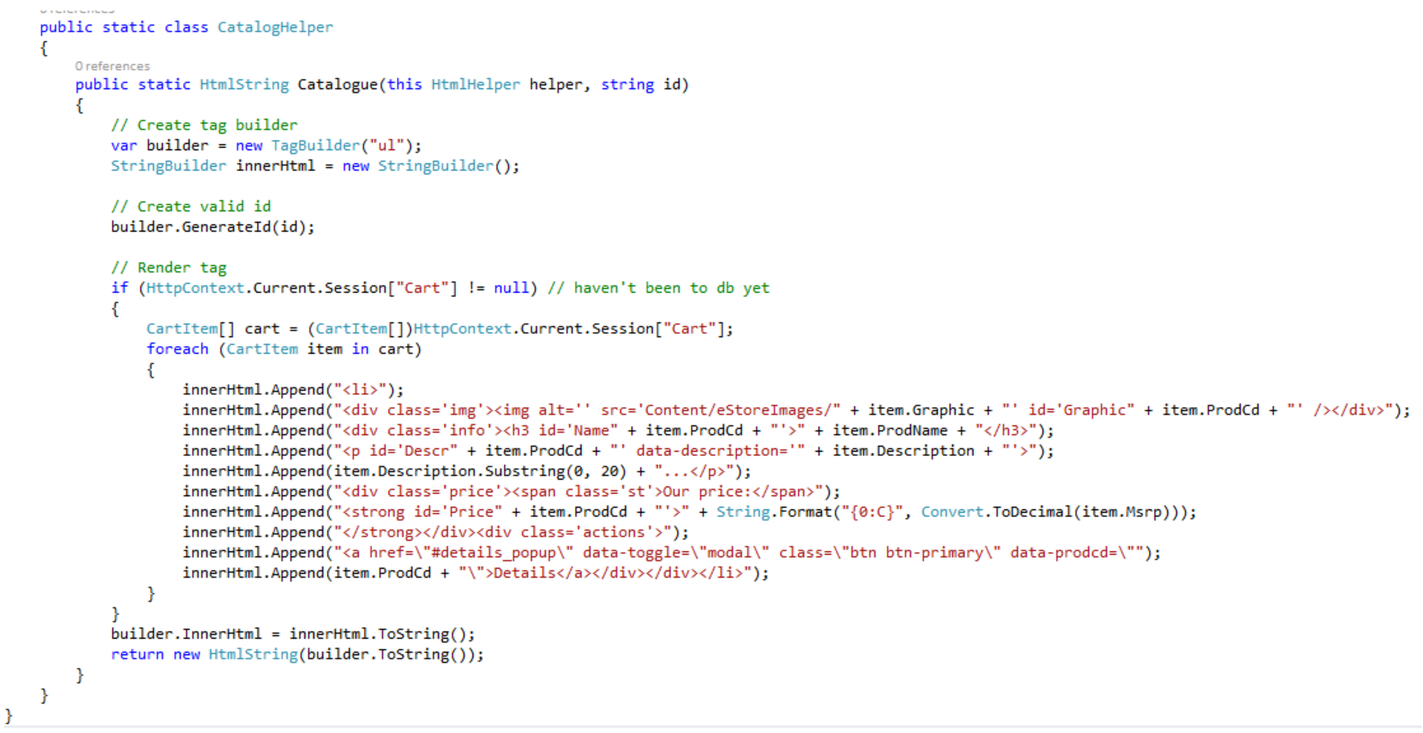


```
<li>
    <div class='img'>
        <img alt='' src='Content/images/fuj2650.jpg' id='Graphic14X45' />
    </div>
    <div class='info'>
        <div style='font-size: small; background: #cfcfcf;' id='Name14X45'>Fujitsu 265</div>
        <p id='Descr14X45' data-description='Meet the whole package - slim, stylish and full of features. T
        <div class='price'><span class='st'>Our price:</span><strong id='Price14X45'>$329.97</strong></div>
        <div class='actions'><a href="#details_popup" class="details" data-prodcd="14X45">Details</a></div>
    </div>
</li>
```

How you decide to code your catalogue is entirely up to you, do some googling to see how other catalogues on the web look, then work on creating your own. To code an html helper of our own design, create a new Folder in your project called **Helpers**. To this folder add a class called **CatalogueHelper.cs**



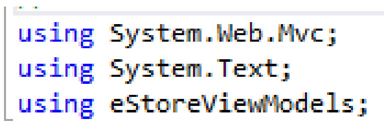
The code I used for my helper class is shown below, you will have to tailor it to fit your own design and style:



```
    public static class CatalogHelper
    {
        0 references
        public static HtmlString Catalogue(this HtmlHelper helper, string id)
        {
            // Create tag builder
            var builder = new TagBuilder("ul");
            StringBuilder innerHtml = new StringBuilder();

            // Create valid id
            builder.GenerateId(id);

            // Render tag
            if (HttpContext.Current.Session["Cart"] != null) // haven't been to db yet
            {
                CartItem[] cart = (CartItem[])HttpContext.Current.Session["Cart"];
                foreach (CartItem item in cart)
                {
                    innerHtml.Append("<li>");
                    innerHtml.Append("<div class='img'><img alt='' src='Content/eStoreImages/" + item.Graphic + "' id='Graphic" + item.ProdCd + "' /></div>");
                    innerHtml.Append("<div class='info'><h3 id='Name" + item.ProdCd + "'>" + item.ProdName + "</h3>");
                    innerHtml.Append("<p id='Descr" + item.ProdCd + "' data-description='" + item.Description + "'>");
                    innerHtml.Append(item.Description.Substring(0, 20) + "...</p>");
                    innerHtml.Append("<div class='price'><span class='st'>Our price:</span>");
                    innerHtml.Append("<strong id='Price" + item.ProdCd + "'>" + String.Format("{0:C}", Convert.ToDecimal(item.Msrp)));
                    innerHtml.Append("</strong></div><div class='actions'>");
                    innerHtml.Append("<a href=\"#details_popup\" data-toggle=\"modal\" class=\"btn btn-primary\" data-prodcd=\"");
                    innerHtml.Append(item.ProdCd + "\">Details</a></div></div></li>");
                }
            }
            builder.InnerHtml = innerHtml.ToString();
            return new HtmlString(builder.ToString());
        }
    }
}
```

It starts out creating a <ul> and then performs a loop through the cart contents to dynamically create the the individual <li> items.

Remember the cart contains all of the information we need to build this unordered list. Note that this code returns an instance of **HtmlString** to the View. The HtmlString class is used because we want the resulting string of html to be unencoded. Notice how the html and product data are interspersed so that each line item in the list is unique with its own id based on the product's ProductCode property. Also notice that the generated html uses a number of classes and id's that you have not created any css for. You will need to modify the code above and to synchronize with the design you came up with for your own catalogue. You will need to add the following usings:

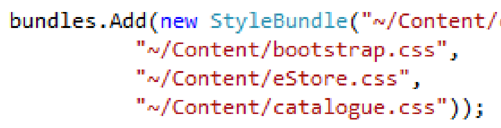


```
using System.Web.Mvc;
using System.Text;
using eStoreViewModels;
```

Bundles

Before we start creating a mass of stylesheets, we'll look at one more utility available in the ASP.Net world called **Bundling**. Bundling allows us to create a single alias for any number of stylesheets (or scripts) and instead of listing out all of the stylesheets, we can just use the alias. To create a bundle, do the following:

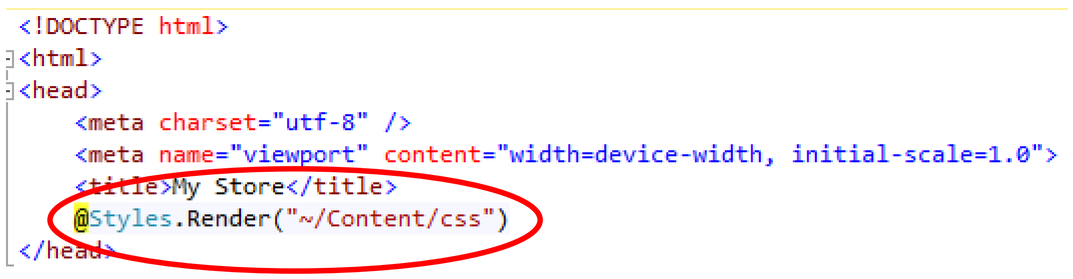
- Locate a file called **BundleConfig.cs** in your App_Start folder. Edit the last style bundle to incorporate our eStore.css and a new catalogue.css like so:



```
bundles.Add(new StyleBundle("~/Content/css").Include(
            "~/Content/bootstrap.css",
            "~/Content/eStore.css",
            "~/Content/catalogue.css"));
```

- This bundle will house 3 stylesheets bootstrap, our existing eStore.css, and whatever stylesheet you create for the new catalogue page

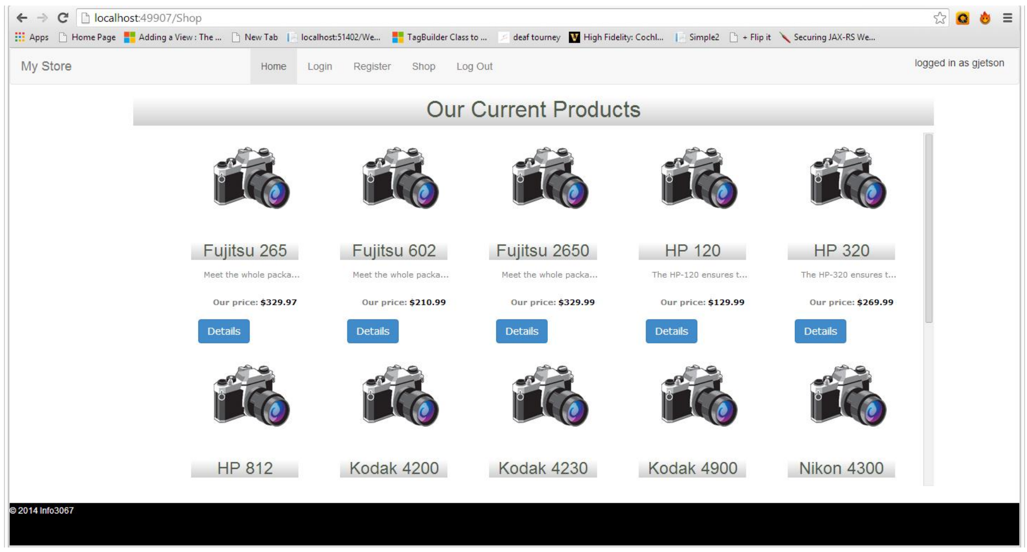
Return to the **Shared_layout.cshtml** view and just include the bundle alias. Don't forget to remove all of the <link.. lines that pointed to the individual style sheets. The <head section should look like:



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Store</title>
    @Styles.Render("~/Content/css")
</head>
```

LAB 6

- Build
  - ViewModels.CartItem.cs
  - Controllers\ShopController.cs
  - Helpers\CatalogueHelper.cs
  - Views\Shop\Index.cshtml (view that uses CatalogueHelper)
  - Content\catalogue.css (style sheet for catalogue)
- Update
  - BundleConfig.cs – modify last bundle
  - _Layout.cshtml – use bundle instead of links
- Submit 3 screen shots in the same Word doc
  - The Rendered Catalogue:



Our Current Products

Fujitsu 265
Meet the whole packa...
Our price: **$329.97**
Details

Fujitsu 602
Meet the whole packa...
Our price: **$210.99**
Details

Fujitsu 2650
Meet the whole packa...
Our price: **$329.99**
Details

HP 120
The HP-120 ensures t...
Our price: **$129.99**
Details

HP 320
The HP-320 ensures t...
Our price: **$269.99**
Details

HP 812

Kodak 4200

Kodak 4230

Kodak 4900

Nikon 4300

© 2014 Info3067

  - The **View's** source code (index.cshtml) that utilizes the CatalogueHelper (see above)
  - The <head>…</head> of _Layout.cshtml (see above)