

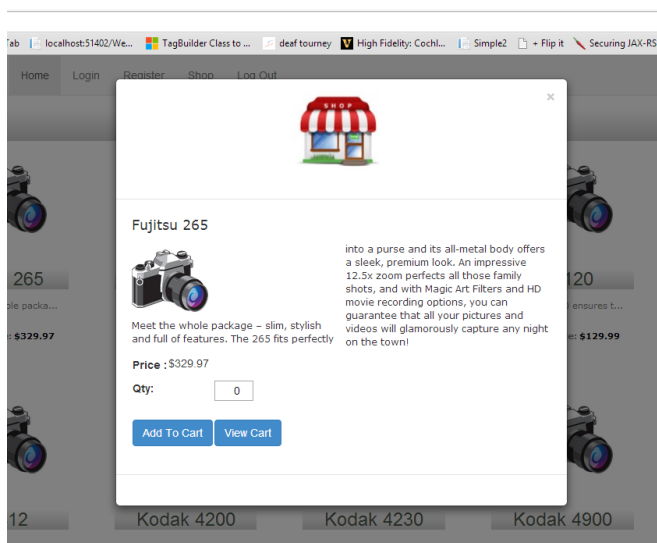
INFO3067 Week 4 Class 1

Review

- Helpers
- Catalogue

Case Study cont'd – Shop

- Now that we have the Catalogue built and displayed, we're ready to allow the user to choose the product(s) they're interested in. The [Details](#) anchor on each item will invoke a modal popup that will not only provide additional information, but also allow them to place the item in a shopping cart:



- The styling and layout of this popup is up to you. Minimally I'd like to see a picture, an in depth description consisting of at least 2 sentences (**in two columns using css3 column count**), the price and the ability to choose a quantity of the product.
- To be able to display information we need to determine the product the user is interested and then make the various data available. How we make the data available is via a small JQuery routine that basically copies the catalogue info that we built with the catalogue helper from last class, into the modal fields. The contents of this routine is as follows and can be added into the the .js file we copied over from the exercise project to handle the tab click, notice how the code sets the text property of one element based on the contents of the other, the key here is that we've coded unique product codes, that get passed to this routine via the details anchor click (**a.btn-primary** below):

```
// details anchor click - to load popup from catalogue
$("a.btn-primary").on("click", function (e) {
    pcd = $(this).attr("data-procd");
    $("#qty").val("0");
    $("#messg2").text("");
    $("#detailsGraphic").attr("src", $("#Graphic" + pcd).attr("src"));
    $("#detailsProdName").text($("#Name" + pcd).text());
    $("#detailsDescr").text($("#Descr" + pcd).data('description'));
    $("#detailsProcd").val(pcd);
    $("#detailsPrice").text($("#Price" + pcd).text());
}); //details
```

The markup for this popup should present an Ajax based form similar to the login form that calls an **AddToCart** method in the **ShopController**, so your markup should include:

```
@using (
    Ajax.BeginForm("AddToCart", "Shop", new AjaxOptions
    {
        LoadingElementId = "ajaxSplash",
        InsertionMode = InsertionMode.Replace,
        HttpMethod = "GET",
        UpdateTargetId = "messg2"
    })
{
    ...
}
```

Also, notice the parameter **LoadingElementId="ajaxSplash"** this element is used because the ajax call to the host is done asynchronously and may take a few seconds. What you should see is the element flash and then go away when the call is complete. The element itself is just a div with the following markup:

```
<div id="ajaxSplash" style="display: none; z-index: 60;">wait...</div>
```

This should make the form complete. Next we need to pass the desired quantity and product code back to the ShopController to update our cart variable. The AJAX form is set to run this new method in the ShopController and looks something like this:

```

public ActionResult AddToCart(int qty, String detailsProdcd)
{
    CartItem[] Cart;

    Cart = (CartItem[])Session["Cart"];

    foreach (CartItem item in Cart)
    {
        if (item.ProdCd == detailsProdcd)
        {
            if (qty >= 0) // update only selected item
                item.Qty = qty;
            break;
        }
    }

    Session["Cart"] = Cart; // store updates in session
    ViewBag.Message = qty + " - item(s) Added!";
    return PartialView("PopupMessage");
}

```

The only two pieces of information we need from the Ajax form is the quantity and the selected product code. The quantity will be available as a standard dropdown value from the form, and you can provide the product code as a hidden field on the form. You can use an html helper to code a hidden field this like:

```
@Html.Hidden("detailsProdcd")
```

The remainder of the method is straight forward. We put the session variable contents into a local CartItem array, then search the array looking for the product code passed in the hidden field and update the correct CartItem quantity value when found. We then place the Cart back into the Session variable and tell the user the cart was updated.

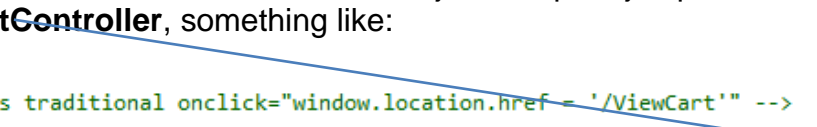
Viewing The Cart's Contents

Notice that we have a 2nd button on the modal that lets the user view what they are interested in purchasing. This second button will use a bit of java script to jump to a new controller called **ViewCartController**, something like:

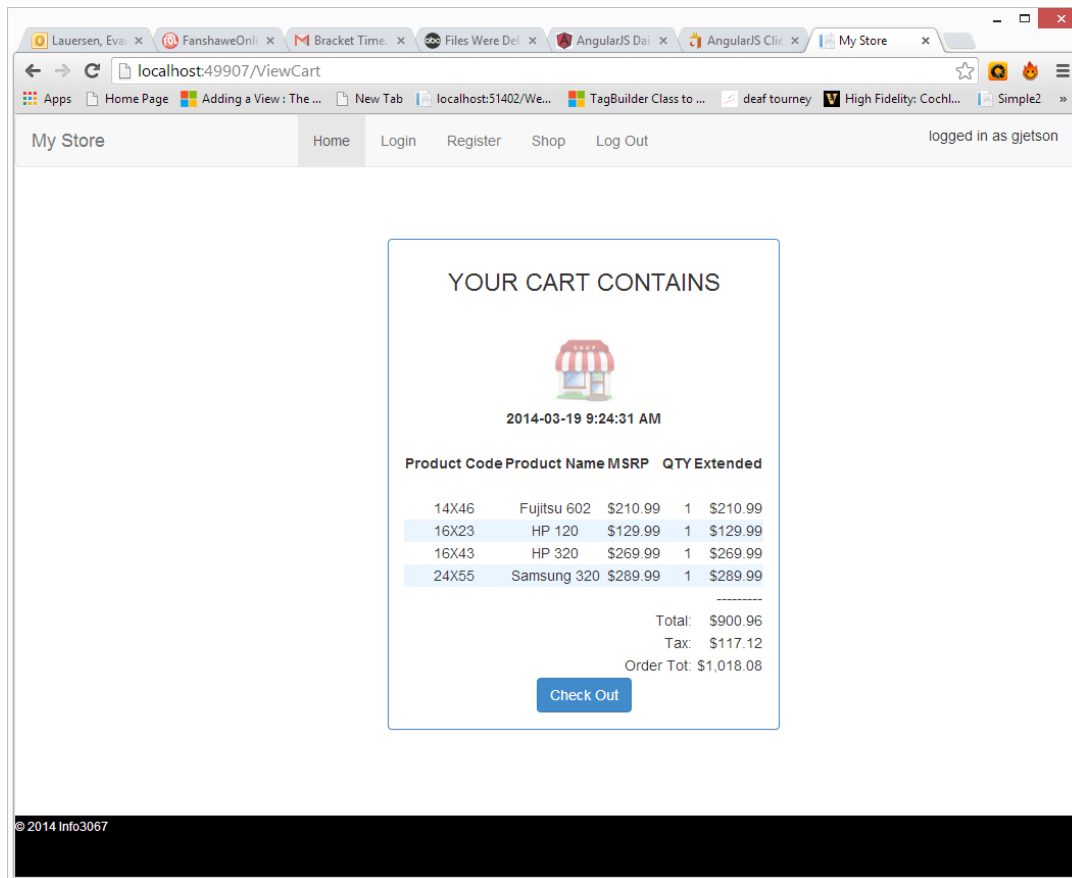
```

<!--jQuery onclick is the same as traditional onclick="window.location.href = '/ViewCart'" -->
<input type="submit" value="    View Cart    " class="btn-custom" onclick="$(location).attr('href','ViewCart')" />

```



The ViewCart controller is just a standard controller returning the Index View. The Index view in return will present the user with their current selection(s):



Here we see the results of the user choosing 4 different items. To build this view, you can use a helper like we did with the catalogue screen. All of the information for the view comes from the cart item array or is calculated. The extended column is simply quantity x MSRP. Assume a 13% tax rate when calculating the totals.

LAB 7

- Build a new modal for the Catalogue's View Details button
 - Ajax form to execute AddToCart Method in ShopController
- Build the HTML Helper ViewCartHelper
- Build the new ViewCartController
- Build the new View for ViewCart (it will use the helper)
- Submit 3 screen shots in the same Word doc
 - Popup Rendered showing chosen item
 - ViewCartHelper source code
 - Rendered View that displays cart contents
- Note, We will finish off the first half coding requirements in the next class.