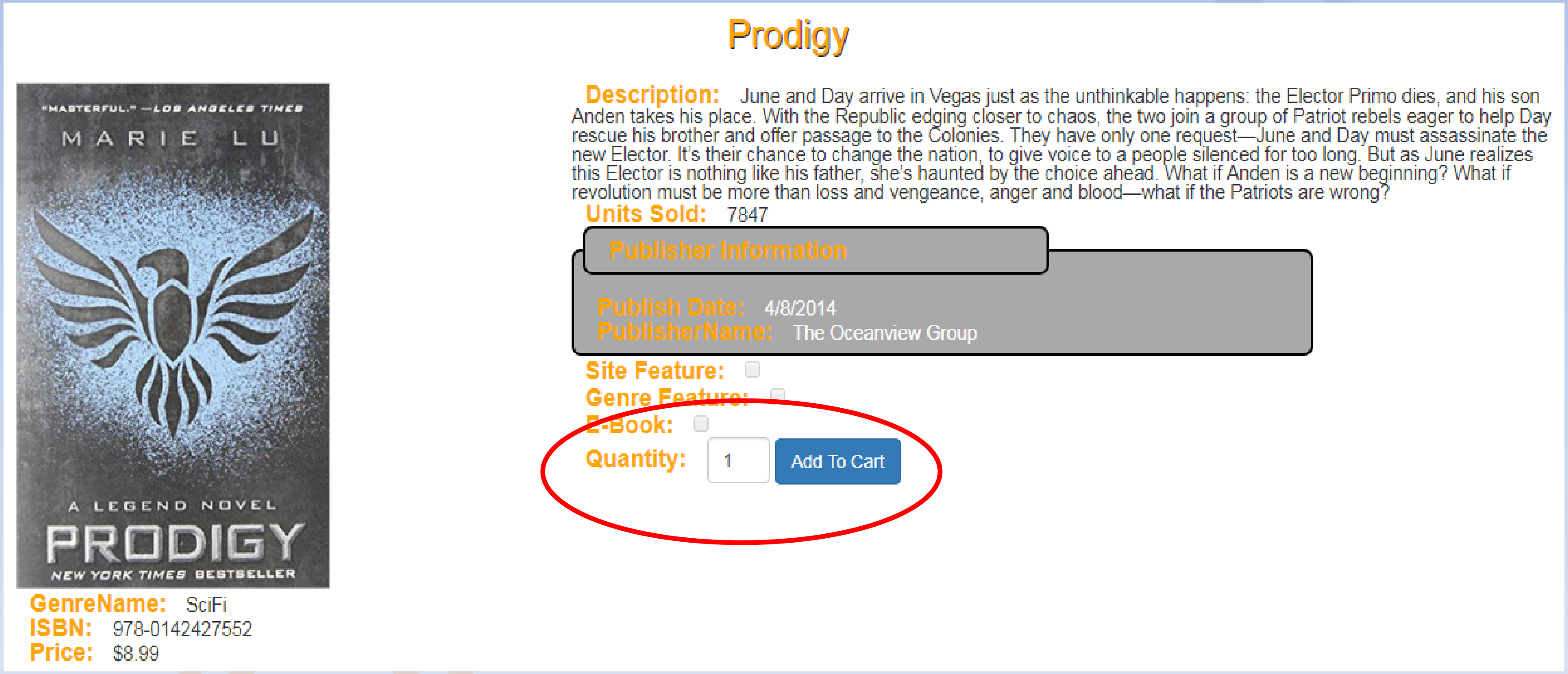
In the beginning…

Set up the UI so that an end user can select an item from the details view and add it to their cart.



Prodigy

GenreName: SciFi
ISBN: 978-0142427552
Price: $8.99

Description: June and Day arrive in Vegas just as the unthinkable happens: the Elector Primo dies, and his son Anden takes his place. With the Republic edging closer to chaos, the two join a group of Patriot rebels eager to help Day rescue his brother and offer passage to the Colonies. They have only one request—June and Day must assassinate the new Elector. It's their chance to change the nation, to give voice to a people silenced for too long. But as June realizes this Elector is nothing like his father, she's haunted by the choice ahead. What if Anden is a new beginning? What if revolution must be more than loss and vengeance, anger and blood—what if the Patriots are wrong?

Units Sold: 7847

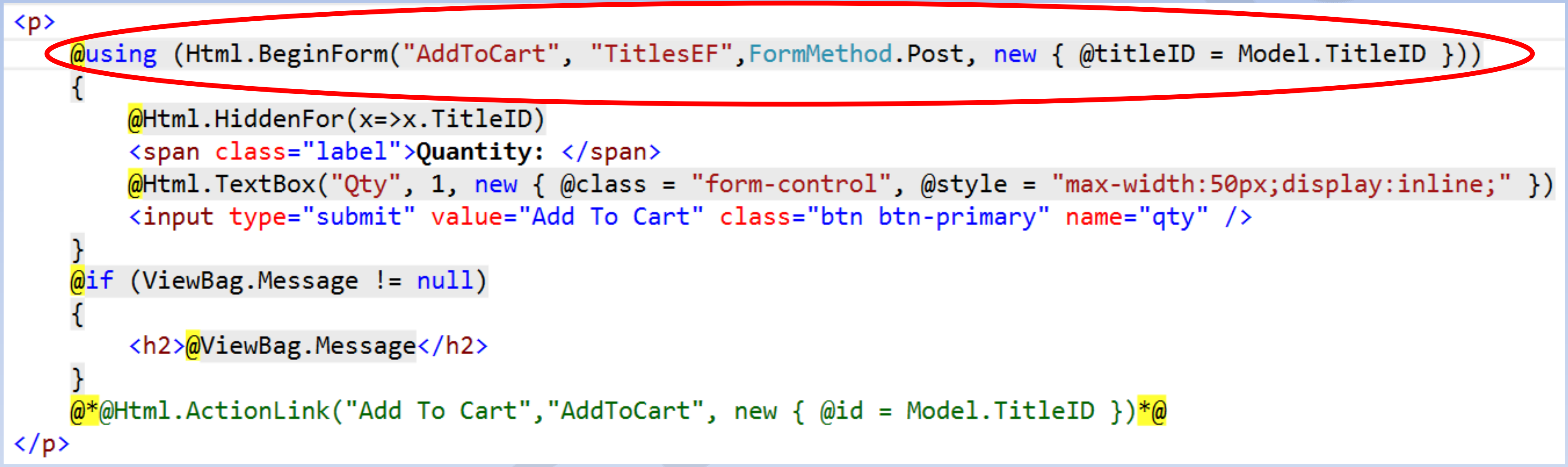
Publisher Information

Publish Date: 4/8/2014
PublisherName: The Oceanview Group

Site Feature: ☐
Genre Feature: ☐
E-Book: ☐
Quantity: 1 Add To Cart

In the beginning, Code

Set up the UI so that an end user can select an item from the details view and add it to their cart.

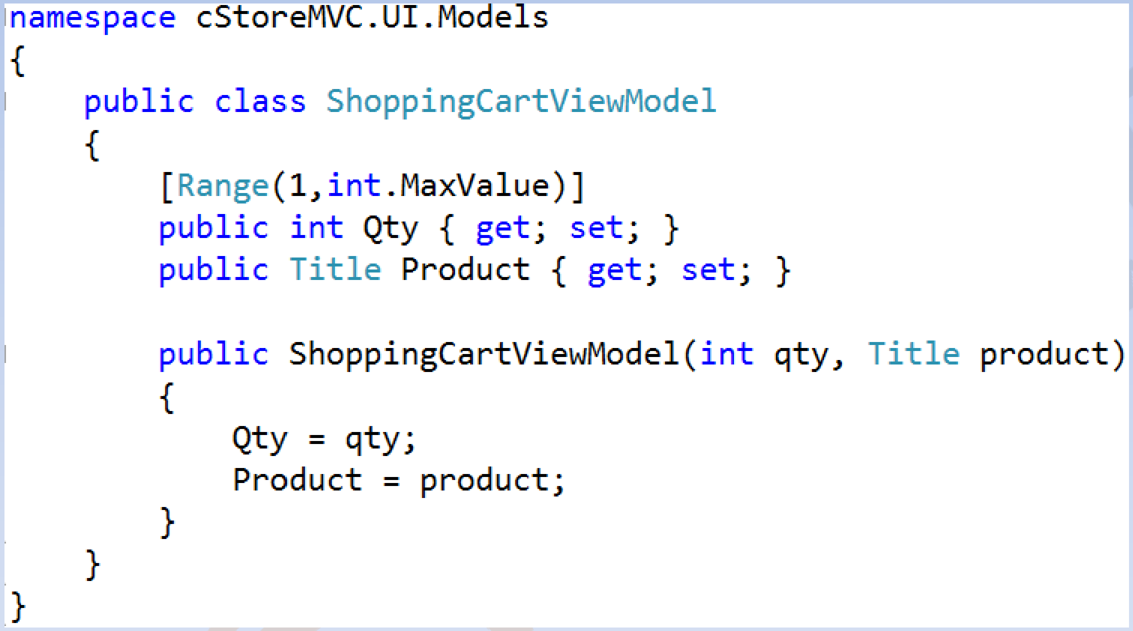


```
<p>
    @using (Html.BeginForm("AddToCart", "TitlesEF",FormMethod.Post, new { @titleID = Model.TitleID }))
    {
        @Html.HiddenFor(x=>x.TitleID)
        <span class="label">Quantity: </span>
        @Html.TextBox("Qty", 1, new { @class = "form-control", @style = "max-width:50px;display:inline;" })
        <input type="submit" value="Add To Cart" class="btn btn-primary" name="qty" />
    }
    @if (ViewBag.Message != null)
    {
        <h2>@ViewBag.Message</h2>
    }
    @*@Html.ActionLink("Add To Cart","AddToCart", new { @id = Model.TitleID })*@
</p>
```

Because this is in the details view and we are going to submit to the server, we need to specify the Controller Name and Action that are going to handle this form. Note that an input type of submit is used here as opposed to a link for adding to cart.

Creating your ShoppingCart Class

In order to place an item in the cart, we need to know which product (title) will be added and how many the end user would like to add.   To accomplish this we will create a ViewModel that will be specific to the use of the UI/Shopping Cart, it will not have meaning anywhere else in our application. The code will go in our Models folder in the UI Layer.

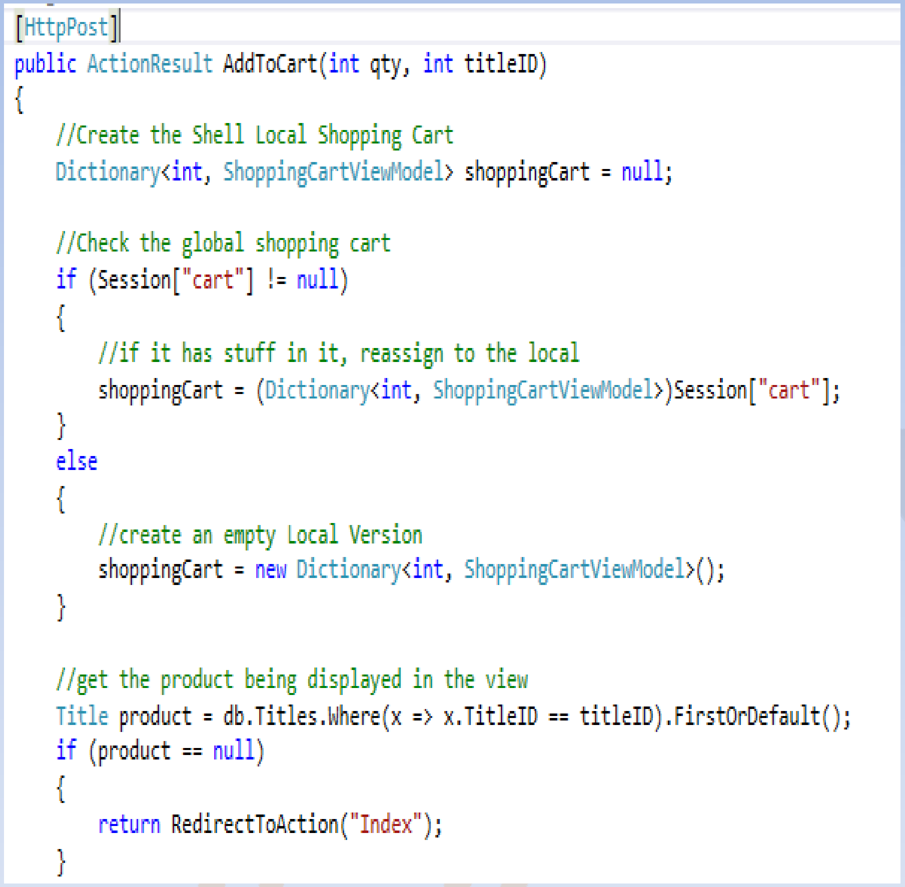


```
namespace cStoreMVC.UI.Models
{
    public class ShoppingCartViewModel
    {
        [Range(1,int.MaxValue)]
        public int Qty { get; set; }
        public Title Product { get; set; }

        public ShoppingCartViewModel(int qty, Title product)
        {
            Qty = qty;
            Product = product;
        }
    }
}
```

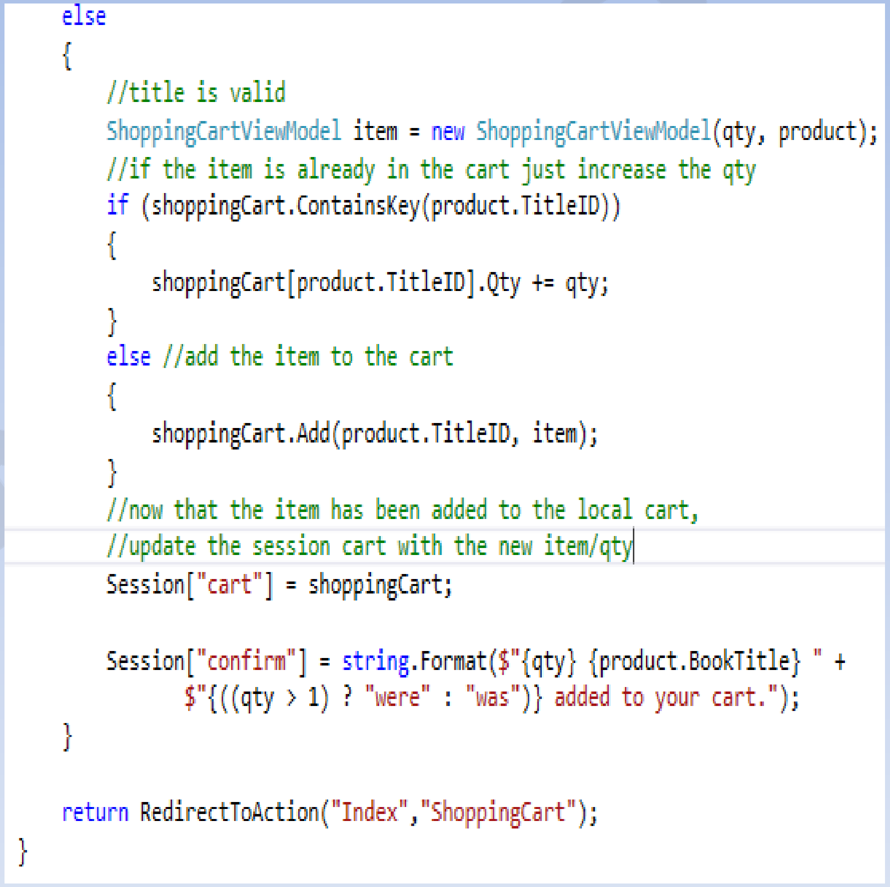
We have 2 properties, and have validation so the minimum quantity to submit is 1. A fully qualified constructor provides us our requirements to build a ShoppingCart object.

Handling the Form Submission/Add To Cart

Per our BeginForm() we are sending submitting to an Action Named AddToCart in the TitlesEF controller, that action does not exist so we will create it.



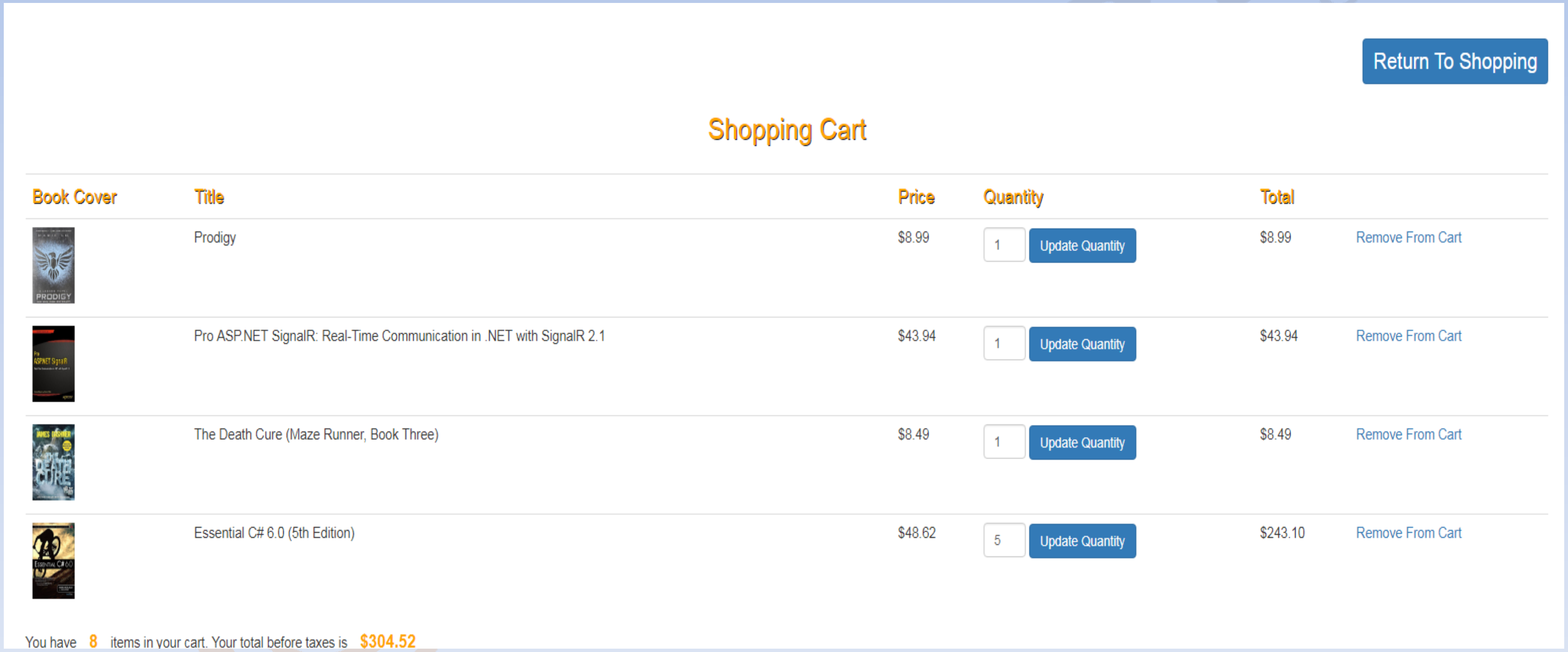
```
[HttpPost]
public ActionResult AddToCart(int qty, int titleID)
{
    //Create the Shell Local Shopping Cart
    Dictionary<int, ShoppingCartViewModel> shoppingCart = null;

    //Check the global shopping cart
    if (Session["cart"] != null)
    {
        //if it has stuff in it, reassign to the local
        shoppingCart = (Dictionary<int, ShoppingCartViewModel>)Session["cart"];
    }
    else
    {
        //create an empty Local Version
        shoppingCart = new Dictionary<int, ShoppingCartViewModel>();
    }

    //get the product being displayed in the view
    Title product = db.Titles.Where(x => x.TitleID == titleID).FirstOrDefault();
    if (product == null)
    {
        return RedirectToAction("Index");
    }
```



```
    else
    {
        //title is valid
        ShoppingCartViewModel item = new ShoppingCartViewModel(qty, product);
        //if the item is already in the cart just increase the qty
        if (shoppingCart.ContainsKey(product.TitleID))
        {
            shoppingCart[product.TitleID].Qty += qty;
        }
        else //add the item to the cart
        {
            shoppingCart.Add(product.TitleID, item);
        }
        //now that the item has been added to the local cart,
        //update the session cart with the new item/qty
        Session["cart"] = shoppingCart;

        Session["confirm"] = string.Format($"{qty} {product.BookTitle} " +
            $"{((qty > 1) ? "were" : "was")} added to your cart.");
    }

    return RedirectToAction("Index","ShoppingCart");
}
```

Transitioning to ShoppingCart/Index

If the add to cart activity was successful, we transition the end user to view their cart. To do this we will need to create a new Controller for Shopping Cart. The only "View" that will be rendered is the Index View and it needs to be strongly typed to the ShoppingCartViewModel class. We will allow for the manipulation of the cart (Edit quantity of items or remove items) in this view.



Return To Shopping

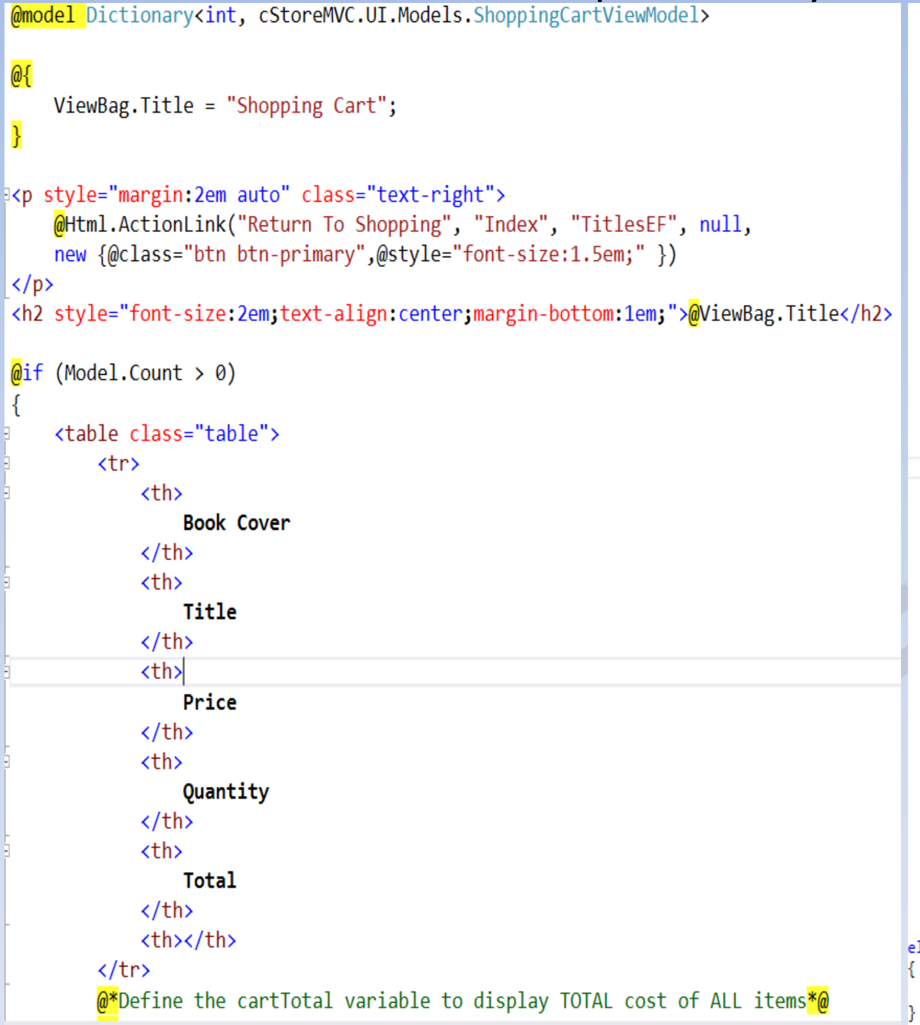
Shopping Cart

Book Cover	Title	Price	Quantity		Total	
	Prodigy	$8.99	1	Update Quantity	$8.99	Remove From Cart
	Pro ASP.NET SignalR: Real-Time Communication in .NET with SignalR 2.1	$43.94	1	Update Quantity	$43.94	Remove From Cart
	The Death Cure (Maze Runner, Book Three)	$8.49	1	Update Quantity	$8.49	Remove From Cart
	Essential C# 6.0 (5th Edition)	$48.62	5	Update Quantity	$243.10	Remove From Cart

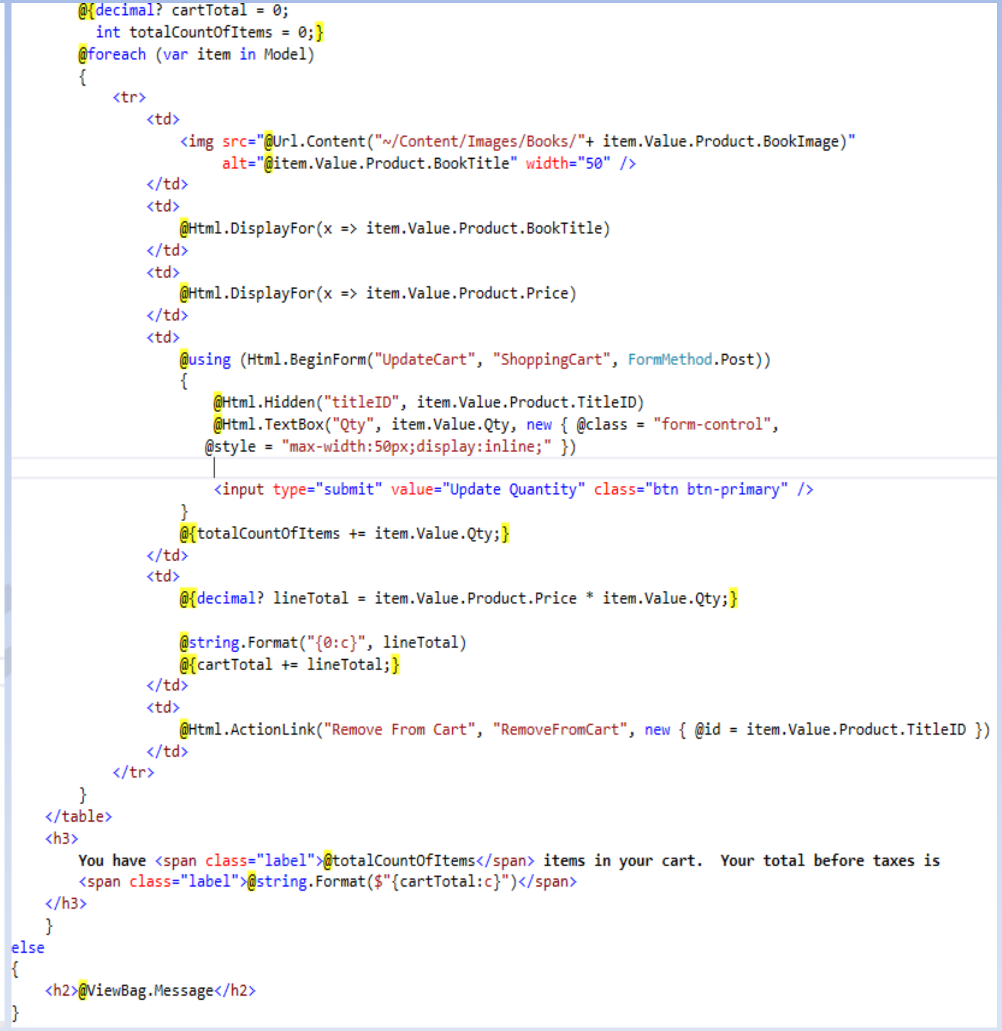
You have 8 items in your cart. Your total before taxes is $304.52

ShoppingCart/Index View, Code

Here is the code to accomplish the layout of this view:



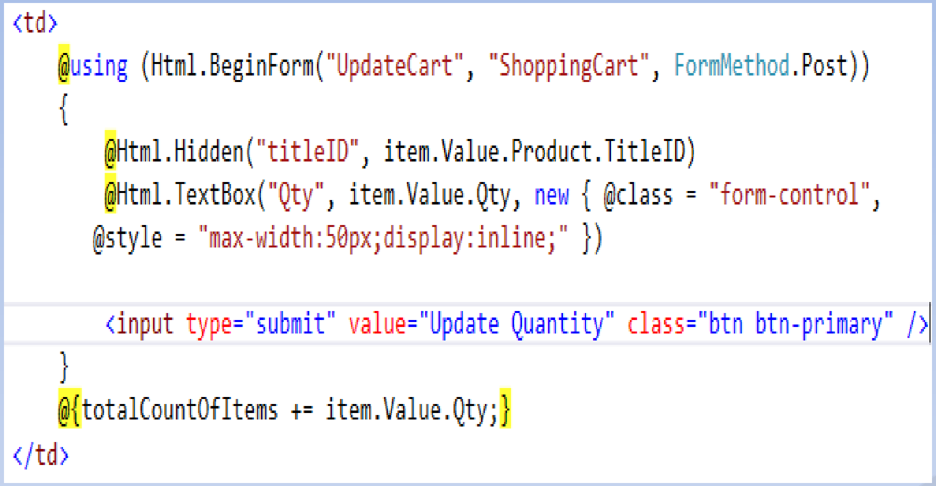
```
@model Dictionary<int, cStoreMVC.UI.Models.ShoppingCartViewModel>

@{
    ViewBag.Title = "Shopping Cart";
}

<p style="margin:2em auto" class="text-right">
    @Html.ActionLink("Return To Shopping", "Index", "TitlesEF", null,
    new {@class="btn btn-primary",@style="font-size:1.5em;" })
</p>
<h2 style="font-size:2em;text-align:center;margin-bottom:1em;">@ViewBag.Title</h2>

@if (Model.Count > 0)
{
    <table class="table">
        <tr>
            <th>
                Book Cover
            </th>
            <th>
                Title
            </th>
            <th>
                Price
            </th>
            <th>
                Quantity
            </th>
            <th>
                Total
            </th>
            <th></th>
        </tr>
        @*Define the cartTotal variable to display TOTAL cost of ALL items*@
```



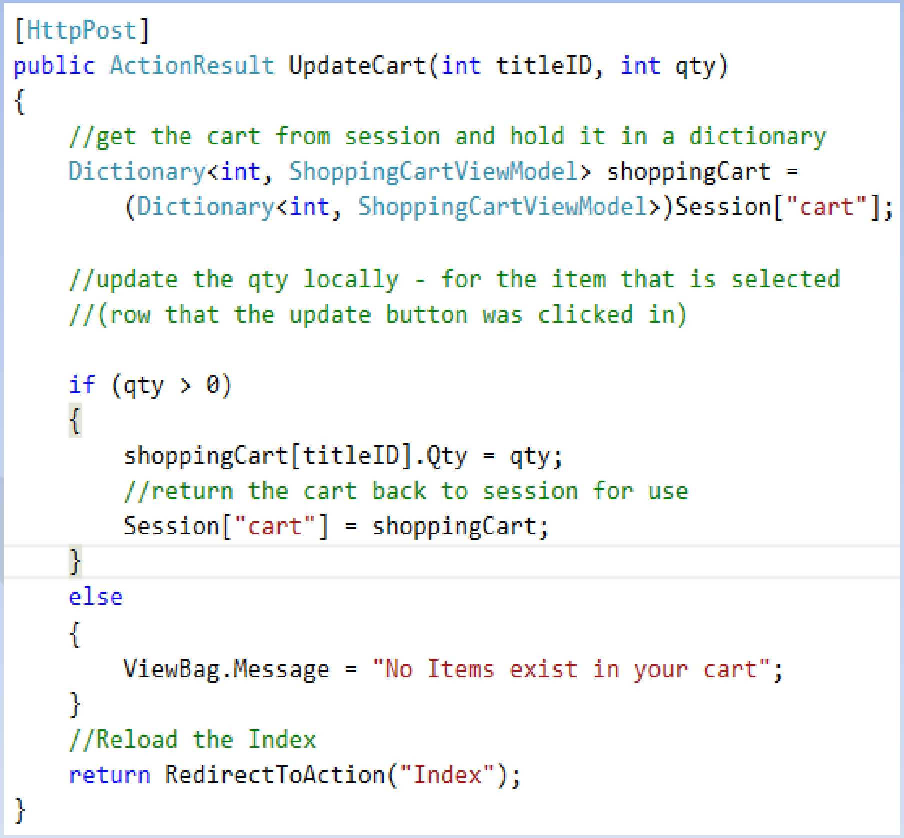
```
        @{decimal? cartTotal = 0;
          int totalCountOfItems = 0;}
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    <img src="@Url.Content("~/Content/Images/Books/"+ item.Value.Product.BookImage)"
                         alt="@item.Value.Product.BookTitle" width="50" />
                </td>
                <td>
                    @Html.DisplayFor(x => item.Value.Product.BookTitle)
                </td>
                <td>
                    @Html.DisplayFor(x => item.Value.Product.Price)
                </td>
                <td>
                    @using (Html.BeginForm("UpdateCart", "ShoppingCart", FormMethod.Post))
                    {
                        @Html.Hidden("titleID", item.Value.Product.TitleID)
                        @Html.TextBox("Qty", item.Value.Qty, new { @class = "form-control",
                       @style = "max-width:50px;display:inline;" })

                        <input type="submit" value="Update Quantity" class="btn btn-primary" />
                    }
                    @{totalCountOfItems += item.Value.Qty;}
                </td>
                <td>
                    @{decimal? lineTotal = item.Value.Product.Price * item.Value.Qty;}

                    @string.Format("{0:c}", lineTotal)
                    @{cartTotal += lineTotal;}
                </td>
                <td>
                    @Html.ActionLink("Remove From Cart", "RemoveFromCart", new { @id = item.Value.Product.TitleID })
                </td>
            </tr>
        }
    </table>
    <h3>
        You have <span class="label">@totalCountOfItems</span> items in your cart.  Your total before taxes is
        <span class="label">@string.Format($"{cartTotal:c}")</span>
    </h3>
    }
else
{
    <h2>@ViewBag.Message</h2>
}
```

Changing the Quantity

Here is the code to accomplish the layout of this view: (UI – Left, Controller – Right)



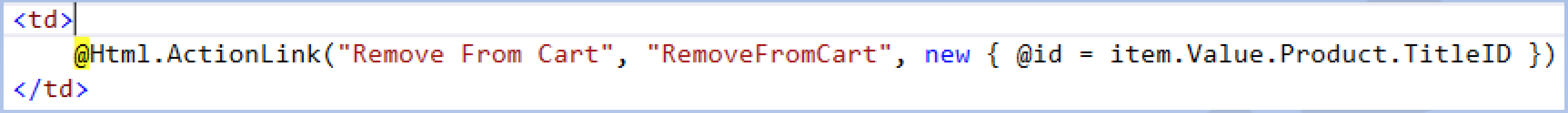
```
<td>
    @using (Html.BeginForm("UpdateCart", "ShoppingCart", FormMethod.Post))
    {
        @Html.Hidden("titleID", item.Value.Product.TitleID)
        @Html.TextBox("Qty", item.Value.Qty, new { @class = "form-control",
       @style = "max-width:50px;display:inline;" })

        <input type="submit" value="Update Quantity" class="btn btn-primary" />
    }
    @{totalCountOfItems += item.Value.Qty;}
</td>
```



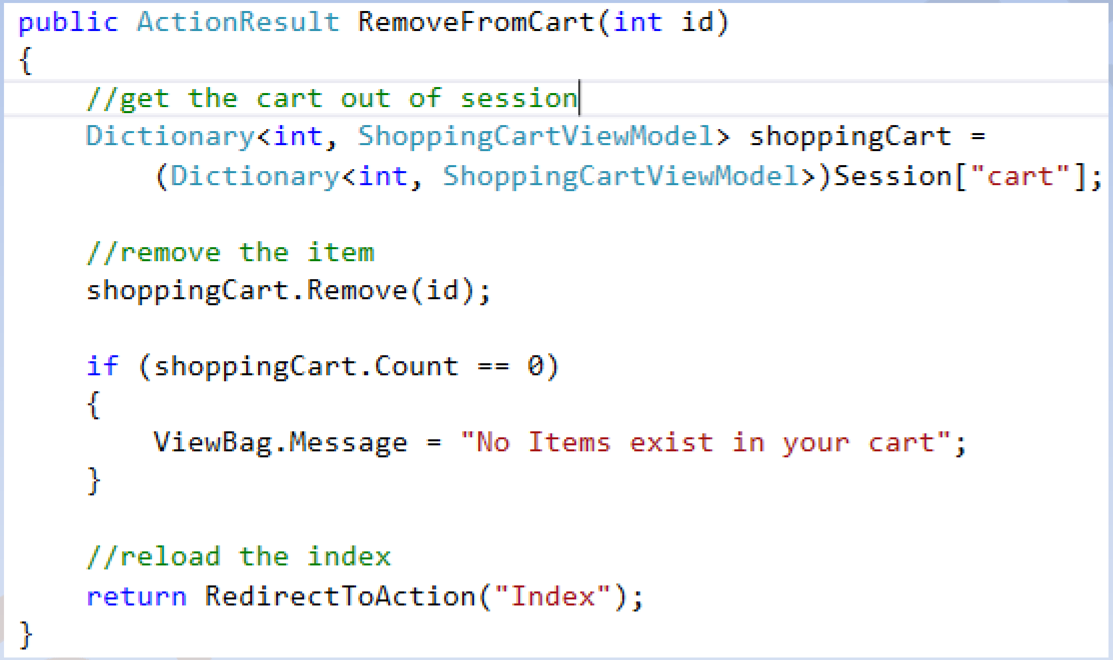
```
[HttpPost]
public ActionResult UpdateCart(int titleID, int qty)
{
    //get the cart from session and hold it in a dictionary
    Dictionary<int, ShoppingCartViewModel> shoppingCart =
        (Dictionary<int, ShoppingCartViewModel>)Session["cart"];

    //update the qty locally - for the item that is selected
    //(row that the update button was clicked in)

    if (qty > 0)
    {
        shoppingCart[titleID].Qty = qty;
        //return the cart back to session for use
        Session["cart"] = shoppingCart;
    }
    else
    {
        ViewBag.Message = "No Items exist in your cart";
    }
    //Reload the Index
    return RedirectToAction("Index");
}
```

Removing the Line Item from the Cart

Here is the code to accomplish the layout of this view: (UI – Top, Controller – Bottom)



```
<td>
    @Html.ActionLink("Remove From Cart", "RemoveFromCart", new { @id = item.Value.Product.TitleID })
</td>
```



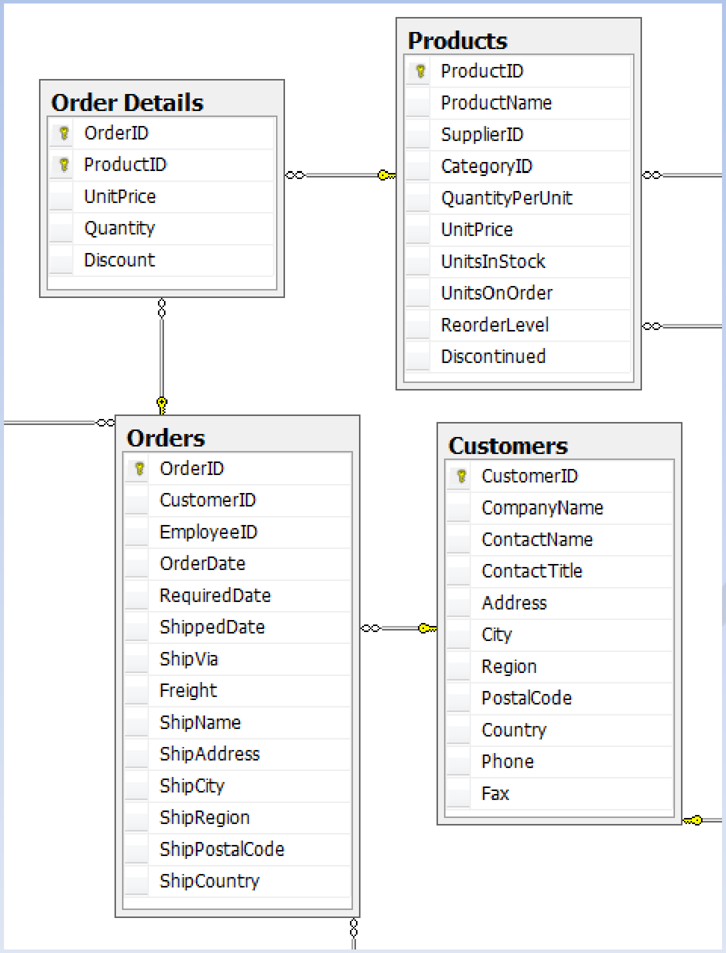
```
public ActionResult RemoveFromCart(int id)
{
    //get the cart out of session
    Dictionary<int, ShoppingCartViewModel> shoppingCart =
        (Dictionary<int, ShoppingCartViewModel>)Session["cart"];

    //remove the item
    shoppingCart.Remove(id);

    if (shoppingCart.Count == 0)
    {
        ViewBag.Message = "No Items exist in your cart";
    }

    //reload the index
    return RedirectToAction("Index");
}
```

Next Steps

This code gives you a functional cart that is updatable and is unique user to user without requiring the use of Identity (as session is scoped to a browser (user) session). All that remains is to create an order and process the payment. Usually processing of payment is done through some third party API (if you have a corporate bank account, check with your bank for processing or you can use a service such as paypal). Once the payment has been received or the order has been shipped, you can update the quantity in your data structure (if you are using your structure to maintain your inventory numbers)

Northwind provides this example structure

For this example, I have only provided the relevant tables, but other tables (per Northwind) that would need referencing are: Categories, Suppliers, Employees, Shippers and any other extra lookup tables.



THE END