Homework #6: OrderTracker

At this point you should have the following View Models, Views, and Controllers:

**Product View Models:**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace OrderTracker.Models

{

public class ProductViewModel

{

[Required]

public int Id { get; set; }

[Required]

[Display(Name="Product Name")]

public string Name { get; set; }

[Required]

[Display(Name = "Product Description")]

public string Description { get; set; }

[Required]

[Display(Name = "Unit Price")]

public decimal UnitPrice { get; set; }

[Display(Name = "Quantity")]

public int Quantity { get; set; }

}

public class ShoppingCartViewModel

{

public List<ProductViewModel> CartItems;

public decimal CartTotal;

}

}

**Shopping.cshtml:**

@model IList<OrderTracker.Models.ProductViewModel>

@{

ViewBag.Title = "Index";

}

<h2>Let's Go Shopping</h2>

@using (Html.BeginForm("Shopping", "Products", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<table class="table">

<tr>

<th width="80">

Quantity

</th>

<th>

Product Name

</th>

<th>

Description

</th>

<th>

Unit Price

</th>

<th></th>

</tr>

@for (var i = 0; i < Model.Count; i++ )

{

<tr>

@Html.HiddenFor(m => m[i].Id)

<td>

@Html.EditorFor(m => m[i].Quantity, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(m => m[i].Quantity, "", new { @class = "text-danger" })

</td>

<td>

@Html.HiddenFor(m => m[i].Name)

@Html.DisplayFor(m => m[i].Name)

</td>

<td>

@Html.HiddenFor(m => m[i].Description)

@Html.DisplayFor(m => m[i].Description)

</td>

<td>

@Html.HiddenFor(m => m[i].UnitPrice)

@Html.DisplayFor(m => m[i].UnitPrice)

</td>

</tr>

}

</table>

<div class="form-group">

<input type="submit" value="Add to cart" class="btn btn-lg" />

</div>

}

**ShoppingCart.cshtml**

@model OrderTracker.Models.ShoppingCartViewModel

@{

ViewBag.Title = "Index";

}

<h2>Shopping Cart</h2>

@using (Html.BeginForm("ShoppingCart", "Products", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<table class="table">

<tr>

<th width="80">

Quantity

</th>

<th>

Product Name

</th>

<th>

Description

</th>

<th>

Unit Price

</th>

<th>

Subtotals

</th>

</tr>

@for (var i = 0; i < Model.CartItems.Count; i++)

{

<tr>

@Html.HiddenFor(m => m.CartItems[i].Id)

<td>

@Html.DisplayFor(m => m.CartItems[i].Quantity)

@Html.HiddenFor(m => m.CartItems[i].Quantity)

</td>

<td>

@Html.HiddenFor(m => m.CartItems[i].Name)

@Html.DisplayFor(m => m.CartItems[i].Name)

</td>

<td>

@Html.HiddenFor(m => m.CartItems[i].Description)

@Html.DisplayFor(m => m.CartItems[i].Description)

</td>

<td>

@Html.HiddenFor(m => m.CartItems[i].UnitPrice)

@Html.DisplayFor(m => m.CartItems[i].UnitPrice)

</td>

<td>

@(Model.CartItems[i].UnitPrice \* Model.CartItems[i].Quantity)

</td>

</tr>

}

</table>

<div class="form-group">

<p class="text-primary">Total Cost: $@Model.CartTotal</p>

</div>

<div class="form-group">

<input type="submit" value="Place Order" class="btn btn-lg" />

</div>

}

**Shopping and Shopping Cart Controllers:**

// GET: Shopping

public ActionResult Shopping()

{

// create a new list of ProductViewModel items called "model"

// this will be our view model for the Shopping view

List<ProductViewModel> model = new List<ProductViewModel>();

// get the complete list of Products from the database

var temp = db.Products.ToList();

// loop through the product list, creating a new ProductViewModel

// object for each product in the list, and add it to the model

foreach (var item in temp)

{

model.Add(new ProductViewModel {

Id = item.Id,

Description = item.Description,

Name = item.Name,

UnitPrice = item.UnitPrice,

Quantity = 0 });

}

// send the model to the view

return View(model);

}

// POST: Shopping

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Shopping(List<ProductViewModel> model)

{

// create a new list of ProductViewModel items called "cart"

// this represents the contents of our shopping cart

List<ProductViewModel> cart = new List<ProductViewModel>();

// get the Customer record for the current user from the database

var customer = db.Customers.FirstOrDefault(c => c.UserName == User.Identity.Name);

// loop through each item in the model we received back from the view, and if the item's

// Quantity value is greater than zero, build a new ShoppingCart object from the item

// (and using the customer ID) and add the object to the ShoppingCart table in the   
 // database, then add the item from the model to the new cart list you created above

foreach (var item in model)

{

if (item.Quantity > 0)

{

db.ShoppingCarts.Add(new ShoppingCart { ProductId = item.Id, Quantity = item.Quantity, UserId = customer.Id });

cart.Add(item);

}

}

// save the changes to the database

db.SaveChanges();

// store the cart object you created in MVC's TempData object - you can use a string index

// to reference it (i.e. TempData["cart"])

TempData["cart"] = cart;

// return a redirect to you ShoppingCart view

return RedirectToAction("ShoppingCart", "Products");

}

// GET: ShoppingCart

public ActionResult ShoppingCart()

{

// create a new list of ProductViewModel objects called "temp" and set it equal

// to the contents of the TempData object as you declared in the previous controller

List<ProductViewModel> temp = (List<ProductViewModel>)TempData["cart"];

// create a new ShoppingCartViewModel object called "model"

// this will be the view model for the ShoppingCart view

ShoppingCartViewModel model = new ShoppingCartViewModel();

// instantiate the CartItems attribute of the model as a new list of ProductViewModel   
 // objects

model.CartItems = new List<ProductViewModel>();

// loop through each item in the shopping cart (this is your "temp" list above)

// add each item to the CartItems list attribute of the model and update

// the CartTotal attribute of the model (add each item's cost to the total)

foreach(var item in temp)

{

model.CartItems.Add(item);

model.CartTotal = model.CartTotal + (item.Quantity \* item.UnitPrice);

}

// send the model to the view

return View(model);

}

// POST: ShoppingCart

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ShoppingCart(ShoppingCartViewModel model)

{

// get the customer record from the database for the current user

var customer = db.Customers.FirstOrDefault(c => c.UserName == User.Identity.Name);

// create a new Order object

Order order = new Order();

// instantiate the OrderItems attribute of the Order object as a new list of OrderItem   
 // objects

order.OrderItems = new List<OrderItem>();

// set the order's customer id attribute to the current user's customer id, then

// add the order to the database Orders table and save the changes.

// The Add method returns an instance of the newly created table entry. You will need

// this, so make sure you assign a variable to collect the Add method's return value

order.CustId = customer.Id;

var newOrder = db.Orders.Add(order);

db.SaveChanges();

// get a list of ShoppingCart entries from the database that match the current user's   
 // customer id

var cartItems = db.ShoppingCarts.Where(c => c.UserId == customer.Id);

// loop through the list you just received, creating a new OrderItem object using the   
 // current order idand the item information from the cart entries, then add the   
 // OrderItem object to the order's OrderItems list attribute

foreach(var item in cartItems)

{

OrderItem orderitem = new OrderItem();

orderitem.OrderId = newOrder.Id;

orderitem.ProductId = item.ProductId;

orderitem.Quantity = item.Quantity;

newOrder.OrderItems.Add(orderitem);

}

// remove the list of shopping cart items from the database, then save the changes for the   
 // current order entry

db.ShoppingCarts.RemoveRange(cartItems);

db.Entry(newOrder).State = EntityState.Modified;

db.SaveChanges();

// show verification of the order by redirecting to the OrderConfirmation view

return RedirectToAction("OrderConfirmation", "Orders", new { id = newOrder.Id });

}

Next: You need to build the Order View Models. In the interest of time, here they are:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace OrderTracker.Models

{

public class OrderViewModel

{

[Required]

[Display(Name = "Order Number")]

public int OrderId { get; set; }

[Required]

[Display(Name = "Customer Name")]

public string CustomerName { get; set; }

[Display(Name = "Order Items")]

public List<OrderItemViewModel> OrderItems { get; set; }

[Display(Name = "Paid")]

public bool Paid { get; set; }

[Display(Name = "Order Total")]

public decimal OrderTotal { get; set; }

}

public class OrderItemViewModel

{

[Required]

public int OrderId { get; set; }

[Required]

[Display(Name = "Product")]

public string ProductName { get; set; }

[Required]

[Display(Name = "Quantity")]

public int Quantity { get; set; }

}

public class BillingViewModel

{

[Required]

[Display(Name = "Customer Name")]

public string CustomerName { get; set; }

[Required]

[Display(Name = "Billing Address")]

public string Address { get; set; }

[Required]

[Display(Name = "City")]

public string City { get; set; }

[Required]

[Display(Name = "State")]

public string State { get; set; }

[Required]

[Display(Name = "Zip")]

public string Zip { get; set; }

[Required]

[Display(Name = "Phone")]

public string Phone { get; set; }

[Required]

[Display(Name = "Name on card")]

public string NameOnCard { get; set; }

[Required]

[Display(Name = "Card number")]

public string CardNumber { get; set; }

}

public class ShippingViewModel

{

[Required]

[Display(Name = "Customer Name")]

public string CustomerName { get; set; }

[Required]

[Display(Name = "Shipping Address")]

public string Address { get; set; }

[Required]

[Display(Name = "City")]

public string City { get; set; }

[Required]

[Display(Name = "State")]

public string State { get; set; }

[Required]

[Display(Name = "Zip")]

public string Zip { get; set; }

[Required]

[Display(Name = "Phone")]

public string Phone { get; set; }

}

public class BillingShippingViewModel

{

public BillingViewModel Billing { get; set; }

public ShippingViewModel Shipping { get; set; }

public int OrderId { get; set; }

}

}

Now for the Homework!! Do for Monday…

Have this program in as complete and working state as possible so you can demo it to Bobby.

1. Add an OrderConfirmation View to the Orders view folder. It will be a Details view based on the OrderViewModel and will NOT have a database context.
   1. Modify this view as you see fit…
2. Add a BillingShipping view to the Orders view folder. It will be a Create view based on the BillingShippingViewModel and will NOT have a database context
   1. Modify this view as you see fit…
3. Add a PaymentConfirmation view to the Orders folder. It will be a Details view based on the OrderViewModel (the Paid attribute should be updated and set to true) and should NOT have a database context
   1. Modify this view as you see fit…
4. Add a My Orders menu item
5. Add a MyOrders list view based on the OrderViewModel
   1. Modify this view as you see fit…
6. Add corresponding controllers for these views. The only one that needs a POST controller is BillingShipping. We’ll walk through these controllers on Monday. On Tuesday, be prepared to start working on the final project.