Homework #5: OrderTracker

You have a perfectly functional Web application with a data connection to the new database you built in class. You also have a new RegisterViewModel class that you have modified to contain properties of BOTH the MVC default registration (email and password) and your Customer database table. Now it’s time to start registering users.

Task 1) Generate a new Register view from your RegisterViewModel.

Task 2) Modify the Register controller to add a new data record in the Customer table. The controller already registers a user. Register the user with email AND username, then create a new table entry for the Customer.

Task 3) Build new view models as follows:

* ProductViewModels
  + ProductViewModel
    - string Name
    - string Description
    - decimal UnitPrice
    - int Quantity
  + ShoppingCartViewModel
    - List<ProductViewModel> CartItems
    - Decimal CartTotal
* OrderViewModels
  + OrderViewModel
    - int OrderId
    - string CustomerName
    - List<OrderItemViewModel> OrderItems
    - bool Paid
    - decimal OrderTotal
  + OrderItemViewModel
    - int OrderId
    - string ProductName
    - int Quantity
  + ShippingViewModel
    - string CustomerName
    - string Address
    - string City
    - string State
    - string Zip
    - string Phone
  + BillingViewModel
    - string CustomerName
    - string Address
    - string City
    - string State
    - string Zip
    - string Phone
    - string NameOnCard
    - string CardNumber
  + BillingShippingViewModel
    - ShippingViewModel Shipping
    - BillingViewModel Billing
    - int OrderId

Now that you can register users, let’s work on adding products to your inventory and allowing users to shop your site and add products to a shopping cart.

Task 1) Scaffold the **Product** class from your imported data entities to generate controllers and views.

Task 2) Make a copy of the new Products\Index.cshtml and name it Shopping.cshtml

Task 3) Add two menus to your menu bar: “Shopping” and “Product Administration”. These should only be available to users that are logged in. Later in the class when we talk about roles, we will make these available to only certain users. Set the Shopping menu to open the new Products\Shopping view. Set the Product Administration menu to open the \Products\Index page.

Task 4) Run your application, go to your Product Administration link, and add several products to the database.

Task 5) Modify your Shopping view to accept the ProductViewModel class you created in your homework rather than the Product class it currently uses.

VIEWS – Shopping and ShoppingCart

@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>

}

@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>

}

Controllers

// GET: Products

public ActionResult Shopping()

{

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

var temp = db.Products.ToList();

foreach (var item in temp)

{

model.Add(new ProductViewModel {

Id = item.Id,

Description = item.Description,

Name = item.Name,

UnitPrice = item.UnitPrice,

Quantity = 0 });

}

return View(model);

}

// POST: Shopping

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Shopping(List<ProductViewModel> model)

{

// build a new view model with the user's selected products

// AND store the cart in the database for later retrieval

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

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

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);

}

}

db.SaveChanges();

TempData["cart"] = cart;

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

}

// GET: ShoppingCart

public ActionResult ShoppingCart()

{

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

ShoppingCartViewModel model = new ShoppingCartViewModel();

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

foreach(var item in temp)

{

model.CartItems.Add(item);

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

}

return View(model);

}

// POST: ShoppingCart

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ShoppingCart(ShoppingCartViewModel model)

{

// Next Lesson: Build a new order from the cart and log to DB

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

Order order = new Order();

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

order.CustId = customer.Id;

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

db.SaveChanges();

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

foreach (var item in cartItems)

{

OrderItem orderItem = new OrderItem { OrderId = newOrder.Id, ProductId = item.ProductId, Quantity = item.Quantity };

order.OrderItems.Add(orderItem);

}

db.ShoppingCarts.RemoveRange(cartItems);

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

db.SaveChanges();

TempData["OrderId"] = newOrder.Id;

// show verification of the order

return RedirectToAction("OrderConfirmation", "Orders");

}

Next Task:

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 needed
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 needed
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 needed
4. Add a My Orders menu item
5. Add a MyOrders list view based on the OrderViewModel
   1. Modify this view as needed
6. Add corresponding constructors for these views. The only one that needs a POST controller is BillingShipping