========== GUIDE 11 PART 2

ADD TO CART BUTTON, INCREASE, DECREASE, REMOVE

Step by step

Goal

1. Add an Add to Cart button on the product list page.
2. When clicked, it adds the product to the logged in user's cart.
3. On the cart page, users can increase quantity, decrease quantity, or remove an item.

STEP 1

Update Home Index view to add the Add to Cart button

Where you are working

Home Index view that lists products.

What you are adding

A form that posts the productId to CartController.AddProductToCart.

Code to add inside each product card

```
<form method="post"
      asp-controller="Cart"
      asp-action="AddProductToCart">
    <input type="hidden" name="productId" value="@product.Id" />
    <button type="submit" class="btn btn-warning btn-sm mb-2">
        Add to Cart <i class="bi bi-cart4"></i>
    </button>
</form>
```

step by step

1. The user clicks Add to Cart.
2. The browser sends a POST request to /Cart/AddProductToCart.
3. The hidden input sends the selected product's Id as productId.
4. The controller receives productId and adds the product to the cart.

Purpose of each key part

method="post" This changes data, so it should be POST.

asp-controller="Cart" Sends request to CartController.

asp-action="AddProductToCart" Calls the AddProductToCart method.

hidden input name="productId" Matches the controller parameter name, so model binding works.

Checkpoint

Run the app. On the product list page, you should see an Add to Cart button for each product. Clicking it should redirect back to Home/Index.

STEP 2

AddProductToCart action in CartController

What this action does

It either increases quantity if the product is already in the cart, or creates a new cart item if it is not.

Code

```
public async Task<IActionResult> AddProductToCart(int productId)
{
    //Get logged-in user's cart
    var userCart = await GetUserCartAsync();

    //Check if product is already existing in the cart
    var existingCartItem = userCart.CartItems.FirstOrDefault(c => c.ProductId == productId);

    // The Product is already in the cart, therefore increase quantity by 1
    if (existingCartItem != null)
    {
        existingCartItem.Quantity += 1;
    }
    else
    {
        //The Product is already in the cart therefore create a new cart item for the logged in user
        var newCartItem = new CartItem
        {
            ProductId = productId,
            Quantity = 1,
            CartId = userCart.Id
        };

        userCart.CartItems.Add(newCartItem);
    }

    await context.SaveChangesAsync();
    return RedirectToAction("Index", "Home");
}
```

step by step

1. Get the current user's cart from the database.
2. Search the cart items for the productId.
3. If found, increase Quantity by 1.
4. If not found, create a new CartItem with Quantity 1 and attach it to the cart.
5. Save changes.
6. Redirect back to Home Index.

Purpose of each key part

GetUserCartAsync Guarantees the user has a cart.

FirstOrDefault Checks if the cart already has this product.

existingCartItem.Quantity += 1 Ensures one row per product, quantity changes instead of duplicates.

new CartItem Creates the cart line item for this product.

SaveChangesAsync Commits insert or update to the database.

RedirectToAction Sends the user back to the product page.

Checkpoint

Click Add to Cart for the same product multiple times. Then open the cart page. You should see quantity increase.

STEP 3

Update DisplayCart view to include action buttons

What you are doing

You add forms for Increase, Decrease, and Remove.

DisplayCart view code

```
@model Cart

<h2>Your Cart Items</h2>

@if (Model.CartItems == null || Model.CartItems.Count == 0)
{
    <p>Your cart is empty.</p>
}
else
{
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Product</th>
                <th>Quantity</th>
                <th>Price</th>
                <th>Total</th>
                <th>Actions</th>
            </tr>
        </thead>

        <tbody>
            @foreach (var item in Model.CartItems)
            {
                <tr>
                    <td>@item.Product.Name</td>
                    <td>@item.Quantity</td>
                    <td>@item.Product.Price £</td>
                    <td>£@(item.Product.Price * item.Quantity)</td>

                    <td>
                        <form asp-controller="Cart" asp-action="IncreaseQuantity" method="post" class="d-inline">
                            <input type="hidden" name="CartItemId" value="@item.Id" />
                            <button type="submit" class="btn btn-sm btn-success">+</button>
                        </form>

                        <form asp-controller="Cart" asp-action="DecreaseQuantity" method="post" class="d-inline">
                            <input type="hidden" name="CartItemId" value="@item.Id" />
                            <button type="submit" class="btn btn-sm btn-warning">-</button>
                        </form>

                        <form asp-controller="Cart" asp-action="RemoveItem" method="post" class="d-inline">
                            <input type="hidden" name="CartItemId" value="@item.Id" />
                            <button type="submit" class="btn btn-sm btn-danger">Remove</button>
                        </form>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
```

step by step

1. The page receives a Cart model from the controller.
2. It loops through CartItems.
3. For each item, it shows product name, quantity, price, line total.
4. Each action button submits a POST request with the cart item's Id.
5. The controller updates the database then redirects back to DisplayCart.

Purpose of each key part

Hidden input CartItemId This is how the controller knows which cart item to update.

asp-controller and asp-action Routes each form to the correct method in CartController.

method="post" These actions change data, so POST is correct.

Checkpoint

Open cart.

Click +, quantity should increase.

Click −, quantity should decrease.

Click Remove, item disappears.

STEP 4

Create IncreaseQuantity action

What this does

Find one cart item by id and add 1 to Quantity.

Code

```
public async Task<IActionResult> IncreaseQuantity(int cartItemId)
{
    var cartItem = await context.CartItems.FirstOrDefaultAsync(ci => ci.Id == cartItemId);

    if (cartItem != null)
    {
        cartItem.Quantity++;
        await context.SaveChangesAsync();
    }

    return RedirectToAction("DisplayCart", "Cart");
}
```

step by step

1. Find the cart item row using cartItemId.
2. If it exists, increase Quantity by 1.
3. Save changes.
4. Redirect back to DisplayCart.

Purpose of each key part

FirstOrDefaultAsync Fetches the cart item from the database.

cartItem.Quantity++ Updates the quantity field.

RedirectToAction DisplayCart Refreshes the cart page so the new quantity displays.

Checkpoint

Click the + button. Quantity should update.

STEP 5

Create DecreaseQuantity action

What this does

Decrease quantity by 1. If quantity reaches 0, delete the cart item.

Code

```
public async Task<IActionResult> DecreaseQuantity(int cartItemId)
{
    var cartItem = await context.CartItems.FirstOrDefaultAsync(ci => ci.Id == cartItemId);

    if (cartItem != null)
    {
        cartItem.Quantity--;

        if (cartItem.Quantity == 0)
        {
            context.CartItems.Remove(cartItem);
        }

        await context.SaveChangesAsync();
    }

    return RedirectToAction("DisplayCart", "Cart");
}
```

step by step

1. Find the cart item by id.
2. Reduce Quantity by 1.
3. If it becomes 0, remove the row from the database.
4. Save changes.
5. Redirect back to DisplayCart.

Purpose of each key part

Quantity-- Reduces the quantity.

Remove when 0 Stops cart items staying in the cart with zero quantity.

Checkpoint

Click − until it hits 0. Item should disappear.

STEP 6

Create RemoveItem action

What this does

Deletes the cart item immediately.

Code

```
public async Task<IActionResult> RemoveItem(int cartItemId)
{
    var cartItem = await context.CartItems.FirstOrDefaultAsync(ci => ci.Id == cartItemId);

    if (cartItem != null)
    {
        context.CartItems.Remove(cartItem);
        await context.SaveChangesAsync();
    }

    return RedirectToAction("DisplayCart", "Cart");
}
```

step by step

1. Find the cart item by id.
2. Remove it from the database.
3. Save changes.
4. Redirect back to DisplayCart.

Purpose of each key part

Remove Deletes the row.

SaveChangesAsync Commits the delete.

Checkpoint

Click Remove. Item should vanish from the table.

STEP 7

Final wiring checklist

1. CartController has these actions

   DisplayCart

   AddProductToCart

   IncreaseQuantity

   DecreaseQuantity

   RemoveItem
2. DisplayCart view forms match parameter names
3. Your form uses CartItemId in the input name.
4. Your controller parameter is cartItemId.

To keep it consistent, use the same spelling and casing. Best option is to change the input name to cartItemId in all forms.

Example

```
<input type="hidden" name="cartItemId" value="@item.Id" />
```

Then it binds cleanly every time.

3. Your Layout dropdown includes Cart link

   CartController

   DisplayCart action

Code

```
<li>
    <a class="dropdown-item" asp-controller="Cart" asp-action="DisplayCart">Cart</a>
</li>
```