Lab 10 Part 5

The Cart App

Step 1: Create new app & Database Models

In this exercise we will create the **cart** app which will involve the creation of the **cart** models, views & templates, and the active cart functionality of the product page.

Create a new app called **cart**

---

python manage.py startapp cart

Open **settings.py** and register this app. In VS Code right click on the **cart** app folder and create a **templates** folder:

Open **cart/models.py** file and copy-paste the following code:

```
from shop.models import Product

class Cart(models.Model):
    cart_id = models.CharField(max_length=250, blank=True)
    date_added = models.DateField(auto_now_add=True)

    class Meta:
        db_table = 'Cart'
        ordering = ['date_added']

    def __str__(self):
        return self.cart_id

class CartItem(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    cart = models.ForeignKey(Cart, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField()
    active = models.BooleanField(default=True)

    class Meta:
        db_table = 'CartItem'

    def sub_total(self):
        return self.product.price * self.quantity

    def __str__(self):
        return self.product
```

- The class **Cart** has two fields, the cart_id and the date_added. The Meta class is used here to specify the name of the database table to be “Cart” and the Cart table will be ordered by the column “date_added” in ascending order
- The class **CartItem** has 4 fields
- The first field is product and it is a foreign key. The CartItem model will have a many-to-one foreign key relationship to Product: one product can have many cart items, but not the other way around
- The second field is cart and it is a foreign key. The CartItem model will have a many-to-one foreign key relationship to Cart: one cart can have many cart items, but not the other way around
- The quantity field represents the quantity of the cart item(s)
- The active field can be used to represent whether an item is still active in the shopping cart. This could be used to remove items from the cart after a certain period
- The Meta class is used here to specify the name of the database table to be “CartItem”
- The function sub_total calculates and returns the total for each cart item

Now run the database migrations using the commands below:

---

python manage.py makemigrations cart

python manage.py migrate

Step 2: Using Django sessions

Django provides a session framework that supports anonymous and user sessions. The session framework allows you to store arbitrary data for each visitor. Session data is stored on the server side, and cookies contain the session ID unless you use the cookie-based session engine. The session middleware manages the sending and receiving of cookies. The default session engine stores session data in the database. To use sessions, you have to make sure that the MIDDLEWARE setting of your project contains 'django.contrib.sessions.middleware.SessionMiddleware'. This is added by default to the MIDDLEWARE setting when you create a new project using the **startproject** command.

The session middleware makes the current session available in the request object. You can access the current session using **request.session**, treating it like a Python dictionary to store and retrieve session data.

A **function based view**, or view for short, is a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response.

To implement some of the shopping cart functionality we will use function-based views as well as class-based views.

Open **cart/views.py**, delete the import on line 1 and add the following code:

```
cart > views.py > ...
1  def _cart_id(request):
2      cart = request.session.session_key
3      if not cart:
4          cart = request.session.create()
5      return cart
```

- This first view function called _cart_id takes a request object as a parameter
- Line 2: This line of code assigns the session key to a variable called cart. The **session_key** is the name of the column in the session table in the database
- Lines 3-4: If there is no session key then we call the **create()** method to create a session key
- Line 5: The cart variable containing the session key is returned from this function

Step 3: View for Add Item to cart

Copy-paste the following imports and code for a function called **add_cart** which takes two parameters; a request object and the id of the product that we want to add to the cart.

```
from django.shortcuts import redirect
from shop.models import Product
from .models import Cart, CartItem

def _cart_id(request):
    cart = request.session.session_key
    if not cart:
        cart = request.session.create()
    return cart

def add_cart(request, product_id):
    product = Product.objects.get(id=product_id)
    try:
        cart = Cart.objects.get(cart_id=_cart_id(request))
    except Cart.DoesNotExist:
        cart = Cart.objects.create(cart_id=_cart_id(request))
        cart.save()
    try:
        cart_item = CartItem.objects.get(product=product, cart=cart)
        if (cart_item.quantity < cart_item.product.stock):
            cart_item.quantity +=1
        cart_item.save()
    except CartItem.DoesNotExist:
        cart_item = CartItem.objects.create(product=product, quantity=1,cart=cart)
    return redirect('cart:cart_detail')
```

- Line 12: We retrieve the product using the product id supplied.
- Lines 13-14: Using exception handling, we check if a cart object exists in the Cart table with a card id that matches the session key. We call the function _cart_id that we wrote earlier to get the cart id.
- Line 15: If there is no cart object then the exception Cart.DoesNotExist is thrown.
- Line 16: We create a Cart object and supply the cart id using the _cart_id function.
- Line 17: The cart is saved to the database table.
- Lines 18-22: We need to check if the item being added to the cart is already in the cart i.e., the user may decide to add more of the same item to the cart.
- Line 19: Find the cart item that matches the product and cart parameters supplied.
- Line 20-21: If the cart item is in stock then increment the quantity of the item in the cart.
- Line 22: Save the cart item.
- Line 23: An exception is thrown if the cart item is not already in the cart.
- Lines 24-25: A new Cart item object is created with a quanitity of 1 & saved to the database.
- Line 26: We use the redirect function to redirect the user to the cart_detail URL that will display the content of the cart.

Step 4: View for Cart Details

Add the following imports at the top of **views.py**:

```
cart > views.py > add_cart
1   from django.shortcuts import redirect, render
```

```
4   from django.core.exceptions import ObjectDoesNotExist
```

Copy-paste the following function to **cart/views.py** which will query the cart for all the cart items & calculate the total amount for each cart item.

```
def cart_detail(request, total=0, counter=0, cart_items = None):
    try:
        cart = Cart.objects.get(cart_id=_cart_id(request))
        cart_items = CartItem.objects.filter(cart=cart, active=True)
        for cart_item in cart_items:
            total += (cart_item.product.price * cart_item.quantity)
            counter += cart_item.quantity
    except ObjectDoesNotExist:
        pass
    return render(request, 'cart.html', {'cart_items':cart_items,
'total':total, 'counter':counter})
```

- This view function has 4 parameters, the request object and three optional parameters representing the total, counter, and cart items.
- We use exception handling to query the Cart table for the cart object to match the cart/session id.
- We retrieve the active cart items from the CartItem table using the cart parameter.
- We use a for loop to iterate through the collect of Cart Item objects and calculate the total & also assign the quantity to the counter variable.
- An exception is thrown if there is no Cart object. The pass statement is used here as a placeholder for future code.
- This view will render a template called **cart.html** passing dictionary variables to it and will return an HTTP response with the rendered output. These variables are typically referred to as context variables.

Step 5: URLConf

Create the **cart/urls.py** file and copy-paste the following code:

```
from django.urls import path
from . import views

app_name='cart'

urlpatterns = [
    path('add/<uuid:product_id>/', views.add_cart, name='add_cart'),
    path('', views.cart_detail, name='cart_detail'),
]
```

Open **onlineshop/urls.py** and add in the following code:

```
shop_project > urls.py
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('shop.urls')),
    path('search/', include('search_app.urls')),
    path('cart/', include('cart.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Step 6: Cart Template

Next, we need to create the template for the cart. Create a new folder called **templates** inside the **cart** app and create a file **cart.html** in the **cart/templates** folder and copy-paste the following code:

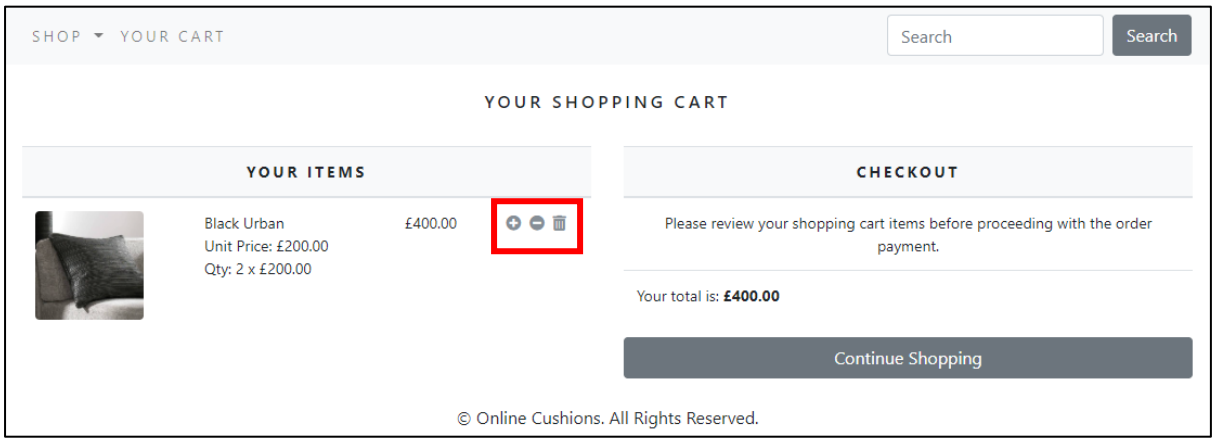
```
{% extends "base.html" %}
{% load static %}
{% block metadescription %}
    This is the shopping cart page. Proceed to review your items and place the order.
{% endblock %}
{% block title %}
    Cart - Perfect Cushion Store
{% endblock %}
{% block content %}
    {% if not cart_items %}
        <div>
            <div class="text-center">
                <br>
                <h1 class="text-center my_title">
                    Your shopping cart is empty
                </h1>
                <br>
                <p class="text-center">
                    Please click <a href="{% url 'shop:all_products' %}">here</a> to continue shopping.
                </p>
            </div>
        </div>
    {% else %}
        <div>
            <div class="text-center">
                <br>
                <h1 class="text-center my_title">
```

```
                Your shopping cart
            </h1>
            <br>
        </div>
    </div>
    <div class="row mx-auto">
        <div class="col-12 col-sm-12 col-md-12 col-lg-6 text-center">
            <table class="table my_custom_table">
                <thead class="my_custom_thead">
                    <tr>
                        <th colspan="5">
                            Your items
                        </th>
                    </tr>

                </thead>
                <tbody>
                    {% for cart_item in cart_items %}
                        <tr>
                            <td>
                                <a href="{{cart_item.product.get_url}}"><img src="{{cart_item.product.image.url}}"
                                 alt="" class="float-left rounded custom_image"></a>
                            </td>
                            <td class="text-left">
                                {{cart_item.product.name}}
                                <br>
                                Unit Price: €{{cart_item.product.price}}
                                <br>
                                Qty: {{cart_item.quantity}} x €{{cart_item.product.price}}
                            </td>
                            <td>
```

```
                        €{{cart_item.sub_total}}
                    </td>
                    {% if cart_item.quantity < cart_item.product.stock %}
                        <td>
                            <a href="" class="custom_a"><i class="fas fa-plus-circle
                               custom_icon"></i></a> 
                            <a href="" class="custom_a"><i class="fas fa-minus-circle
                               custom_icon"></i></a> 
                            <a href="" class="custom_icon"><i class="fas fa-trash-alt custom_icon"></i></a>
                        </td>
                    {% else %}
                        <td>
                             <a href="" class="custom_a"><i class="fas fa-minus-circle
                            custom_icon"></i></a> 
                            <a href="" class="custom_icon"><i class="fas fa-trash-alt custom_icon"></i></a>
                        </td>
                        <td></td>
                    {% endif %}
                </tr>
            {% endfor %}
        </tbody>
    </table>

</div>
<div class="col-12 col-sm-12 col-md-12 col-lg-6 text-center">
    <table class="table my_custom_table">
        <thead class="my_custom_thead">
            <tr>
                <th>
                    Checkout
                </th>
```

```
                        </tr>
                    </thead>
                    <tbody>
                        <tr>
                            <td>
                                Please review your shopping cart items before proceeding with the order payment.
                            </td>
                        </tr>
                        <tr>
                            <td class="text-left">
                                Your total is: <strong>€{{ total }}</strong>
                            </td>
                        </tr>
                    </tbody>
                </table>
                <div class="mx-auto">
                    <a href="{% url 'shop:all_products' %}" class="btn btn-secondary btn-block
                     my_custom_button">Continue
                    Shopping</a>
                </div>
            </div>
        </div>
        <br>
    {% endif %}
{% endblock %}
```

The **cart.html** files contain icons for the shopping cart as shown below. These icons are part of the **fontawesome** icons toolkit & for them to work, we need to include a link to the fontawesome style sheet in **base.html**.



Open **base.html** and copy in the following line of code to the location shown:

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.15.1/css/all.css"
crossorigin="anonymous">
```

```
9     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
10    rel="stylesheet"
11    integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUUVM"
12    crossorigin="anonymous"/>
13    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.15.1/css/all.css" crossorigin="anonymous">
14    <meta name="description" content="{% block metadescription %}{% endblock %}">
15    <title>{% block title %}{% endblock %}</title>
16  </head>
```

Step 7: Custom CSS

Open **custom.css** and copy-paste the following code to style the cart template:

```
/****Cart Page****/
.my_custom_table {
        min-width: 400px;
        font-size: 14px;
}
.my_custom_thead {
        font-weight: normal;
        text-transform: uppercase;
        letter-spacing: .2em;
        background-color: #f8f9fa!important;
}
.custom_image {
        width: 100px;
        height: 100px;
}
.custom_a {
        text-decoration: none;
}
.custom_icon {
        text-decoration: none;
        color: #868e96 !important;
}
.my_custom_button {
        margin-top: 5px;
}
```

Step 8: Update Template

Now we need to add the link for the **Add to Cart** button on the product page. Open **product.html** and add in the following template tag:

```
29                  <p class="text-justify my_prod_text"><b>Out of Stock</b></p>
30              {% else %}
31                  <a class="btn btn-secondary" href="{% url 'cart:add_cart' product.id %}">Add to Cart</a>
32              {% endif %}
33          </div>
```

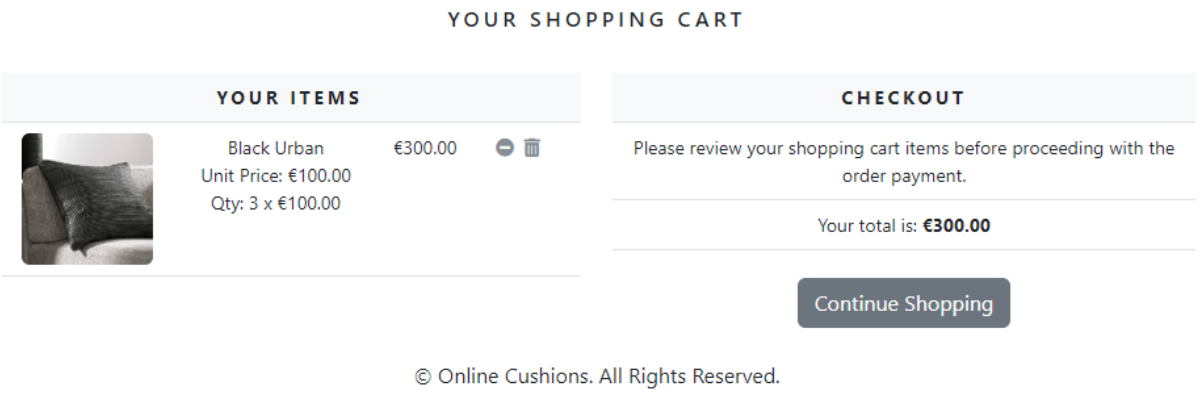
We also need to add the link for the **Add to Cart** button on the cart page. Open **cart.html** and add in the following template tag:

```
{% if cart_item.quantity < cart_item.product.stock %}
    <td>
        <a href="{% url 'cart:add_cart' cart_item.product.id %}" class="custom_a"><
        <a href="" class="custom_a"><i class="fas fa-minus-circle custom_icon"></i>
        <a href="" class="custom_icon"><i class="fas fa-trash-alt custom_icon"></i>
    </td>
{% else %}
    <td>
         <a href="" class="custom_a"><i class="fas fa-minus-circle custom_icon
        <a href="" class="custom_icon"><i class="fas fa-trash-alt custom_icon"></i>
    </td>
    <td></td>
{% endif %}
```

Run the server and try to add a product to the shopping cart.

Note: If the CSS for the web page has not been applied try clearing your browsing history. If you are using Google Chrome, go to your settings --> clear browsing data --> select clear cached images and files then click clear data

For example, in the screenshot below three Black Urban cushions have been added to the shopping cart. As there are only 3 of these products in stock the icon/link to add more of this product is no longer visible. The logic for this is provided in the if else statement shown above in **cart.html**.



Step 9: Implement a counter in the navbar

To implement a counter on the cart icon in the navbar which will show the number of items added to cart we will write a context processor. A context processor is a Python function that takes the request object as an argument and returns a dictionary that gets added to the request context. They come in handy when you need to make something available globally to all templates.

Create a new file in the **cart** app folder called **context_processors.py** and copy-paste the following code:

```
from .models import Cart, CartItem
from .views import _cart_id

def counter(request):
    item_count = 0
    if 'admin' in request.path:
        return {}
    else:
        try:
            cart = Cart.objects.filter(cart_id=_cart_id(request))
            cart_items = CartItem.objects.all().filter(cart=cart[:1])
            for cart_item in cart_items:
                item_count += cart_item.quantity
        except Cart.DoesNotExist:
            item_count = 0
    return {'item_count':item_count}
```

The function counter takes a request object as a parameter

- The if statement will check if admin is in the request path and if it is then we return an empty dictionary. This just means that this context processor is not available to the admin templates.
- Otherwise, we use exception handling to find the cart in the database and then retrieve the collection of Cart Item objects filtered by cart/session id. The [:1] is used here to ensure that only one result is returned (called slicing).
- We use a for loop to iterate through the collection of Cart Item objects & use the item_count variable to track the quanitity.
- If the cart doesn't exist, an exception is thrown and the item_count is set to 0.
- A dictionary is returned containing the variable item_count

Add the following code to register this context processor in **settings.py**:

```
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'shop.context_processors.menu_links',
                'cart.context_processors.counter',
            ],
```

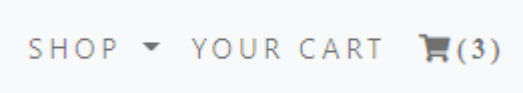
Open **navbar.html** and copy-paste the if-else block below at the location shown.

```
        <li class="nav-item">
          <a class="nav-link" href="#">Your Cart</a>
        </li>
        {% if item_count > 0 %}
          <li class="nav-item">
              <a class="nav-link" href="{% url 'cart:cart_detail' %}"><i
              class="fas fa-shopping-cart">({{item_count}})</i></a>
          </li>
          {% endif %}
      </ul>
```

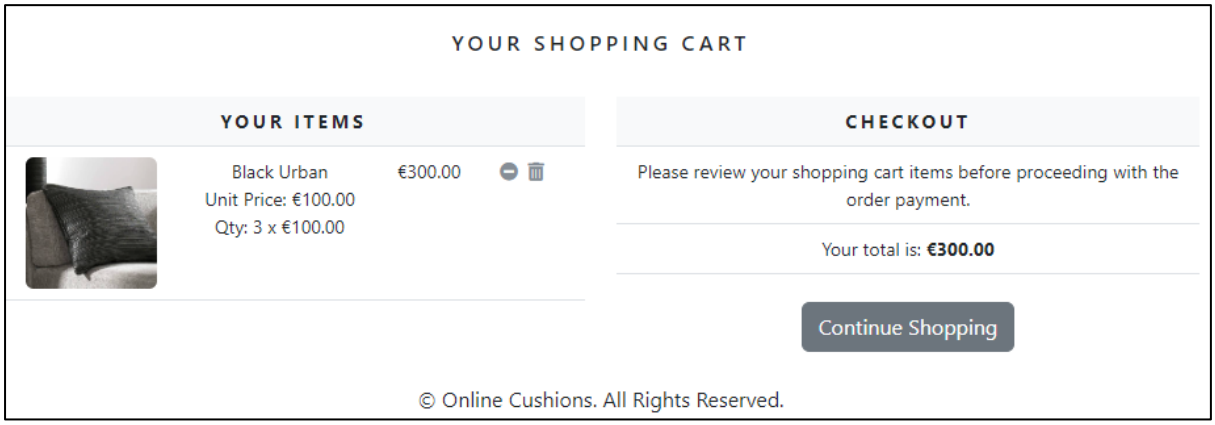
This block of code checks if the item_count is >0. If it is, then the cart is displayed with the number of items. We also add in the following:

- a hyperlink so that when the user clicks on the shopping cart icon, the cart will be displayed
- the fontawesome shopping cart item as shown
- the field item_count

When you run the server and add an item to the cart, it will display in the navbar as shown below:



If you click on the trolley icon, the cart is displayed:



If you navigate back to the home page, you will still see the cart in the navbar.

Step 10: Implement a function for the minus icon

Next, we need to add some code for the minus button which will reduce the items in the shopping cart by 1 each time it is clicked. To do this we will write a new view function in **cart/views.py**.

Add the following import into **cart/views.py**:

```
cart > views.py
from django.shortcuts import render, redirect, get_object_or_404
```

Copy-paste the following code into **cart/views.py**:

```
def cart_remove(request, product_id):
    cart= Cart.objects.get(cart_id=_cart_id(request))
    product = get_object_or_404(Product, id=product_id)
    cart_item = CartItem.objects.get(product=product, cart=cart)
    if cart_item.quantity > 1:
        cart_item.quantity -= 1
        cart_item.save()
    else:
        cart_item.delete()
    return redirect('cart:cart_detail')
```

- Find the cart using the cart id from the request object
- Find the product using the product_id parameter supplied
- Find the cart item in the CartItem table using the product and cart variables
- If the quanitity of this item is greater than zero then reduce the quantity by 1 and save the item
- Otherwise delete the item from the cart
- Redirect the user back to the **cart.html** page

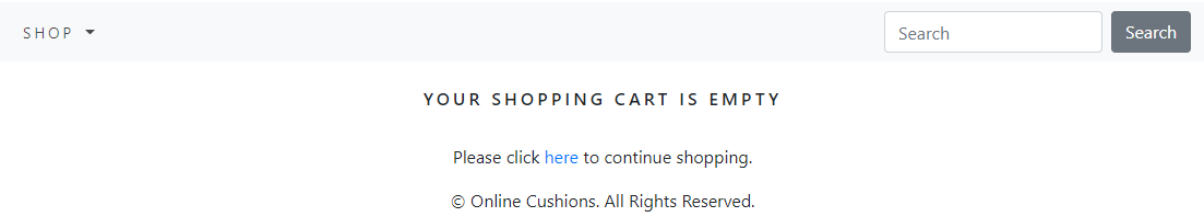
Open **cart/urls.py** and add the following line of code:

```
cart > urls.py
from django.urls import path
from . import views

app_name='cart'

urlpatterns = [
    path('add/<uuid:product_id>/', views.add_cart, name='add_cart'),
    path('', views.cart_detail, name='cart_detail'),
    path('remove/<uuid:product_id>/', views.cart_remove, name='cart_remove')
]
```

Open **cart.html** and add the url for the cart_remove into both the if and else statement as shown below:

```
{% if cart_item.quantity < cart_item.product.stock %}
    <td>
        <a href="{% url 'cart:add_cart' cart_item.product.id %}" class="custom_a"><i class="
        <a href="{% url 'cart:cart_remove' cart_item.product.id %}" class="custom_a"><i clas
        <a href="" class="custom_icon"><i class="fas fa-trash-alt custom_icon"></i></a>
    </td>
{% else %}
    <td>
         <a href="{% url 'cart:cart_remove' cart_item.product.id %}" class="custom_a"><
        <a href="" class="custom_icon"><i class="fas fa-trash-alt custom_icon"></i></a>
    </td>
    <td></td>
{% endif %}
```

Run the server and try to remove an item from the shopping cart by clicking on the minus button. It should work and if you keep clicking this button until there are no items left then you will see the following screen:



Step 11: Completely remove the product when the trash icon is pressed

The next function we want to write is **full_remove** which empties the shopping cart entirely. Copy-paste the following code into **cart/views.py**:

```
def full_remove(request, product_id):
    cart = Cart.objects.get(cart_id=_cart_id(request))
    product = get_object_or_404(Product, id=product_id)
    cart_item = CartItem.objects.get(product=product, cart=cart)
    cart_item.delete()
    return redirect('cart:cart_detail')
```

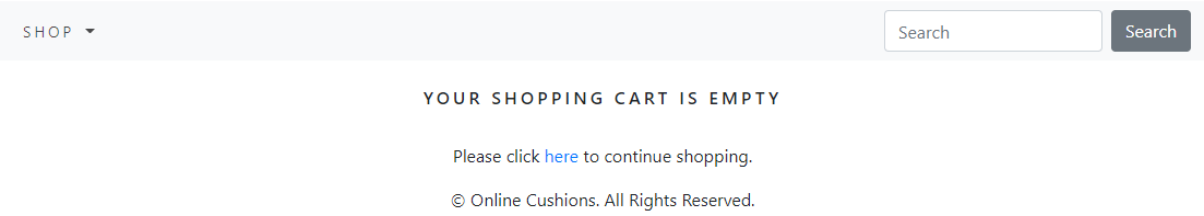
Open **cart/urls.py** and add the following line of code:

```
cart > urls.py
from django.urls import path
from . import views

app_name='cart'

urlpatterns = [
    path('add/<uuid:product_id>/', views.add_cart, name='add_cart'),
    path('', views.cart_detail, name='cart_detail'),
    path('remove/<uuid:product_id>/', views.cart_remove, name='cart_remove'),
    path('full_remove/<uuid:product_id>/', views.full_remove, name='full_remove'),
]
```

Open **cart.html** and add the url for the **full_remove** into both the if and else statement as shown below:

```
{% if cart_item.quantity < cart_item.product.stock %}
    <td>
        <a href="{% url 'cart:add_cart' cart_item.product.id %}" class="custo
        <a href="{% url 'cart:cart_remove' cart_item.product.id %}" class="cu
        <a href="{% url 'cart:full_remove' cart_item.product.id %}" class="cu
    </td>
{% else %}
    <td>
         <a href="{% url 'cart:cart_remove' cart_item.product.id %}" cla
        <a href="{% url 'cart:full_remove' cart_item.product.id %}" class="cu
    </td>
    <td></td>
{% endif %}
```

Run the server and add an item to the shopping cart. Use the plus icon to add some more of this item to the cart. Click on the trash icon to remove these items & again you should see the following indicating that the cart is empty:



Step 12: Commit the changes and push your code to the lab 10 repo.

Stop the server and run the following git commands to update the local and remote repositories:

git add -A

git commit -m “lab 10 part 5 commit”

git push -u origin main