Lab 10 Part 7
Order App

Step 1: Database Models

We will create a new app to create an order for our purchase on the website.

Create a new app called **order**

---

python manage.py startapp order

Open **settings.py** and register this app:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'accounts',
    'shop',
    'search_app',
    'cart',
    'stripe',
    'order',
]
```

Open **models.py** file and copy in the following code:

```
from django.db import models
class Order(models.Model):
    token = models.CharField(max_length=250, blank=True)
    total = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name='Euro Order Total')
    emailAddress = models.EmailField(max_length=250, blank=True,
verbose_name='Email Address')
    created = models.DateTimeField(auto_now_add=True)
    billingName = models.CharField(max_length=250, blank=True)
    billingAddress1 = models.CharField(max_length=250, blank=True)
    billingCity = models.CharField(max_length=250, blank=True)
    billingPostcode = models.CharField(max_length=10, blank=True)
    billingCountry = models.CharField(max_length=200, blank=True)
    shippingName = models.CharField(max_length=250, blank=True)
    shippingAddress1 = models.CharField(max_length=250, blank=True)
    shippingCity = models.CharField(max_length=250, blank=True)
    shippingPostcode = models.CharField(max_length=10, blank=True)
    shippingCountry = models.CharField(max_length=200, blank=True)
```

```
    class Meta:
        db_table = 'Order'
        ordering = ['-created']

    def __str__(self):
        return str(self.id)
```

Copy in the following code into **models.py** for the **OrderItem** class just below the **Order** class:

```
class OrderItem(models.Model):
    product = models.CharField(max_length=250)
    quantity = models.IntegerField()
    price = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name='Euro Price')
    order = models.ForeignKey(Order, on_delete=models.CASCADE)

    class Meta:
        db_table = 'OrderItem'

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

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

Now run the database migrations using the commands below:

---

python manage.py makemigrations order

python manage.py migrate

Step 2: Update the View

Open **cart/views.py** and add in the following import:

```
from order.models import Order, OrderItem
```

Inside the cart_detail function copy-paste the following code to the location shown below:

```
            billingName = request.POST['stripeBillingName']
            billingAddress1 = request.POST['stripeBillingAddressLine1']
            billingcity = request.POST['stripeBillingAddressCity']
            billingCountry = request.POST['stripeBillingAddressCountryCode']
            shippingName = request.POST['stripeShippingName']
            shippingAddress1 = request.POST['stripeShippingAddressLine1']
            shippingcity = request.POST['stripeShippingAddressCity']
            shippingCountry = request.POST['stripeShippingAddressCountryCode']
```

```
46      if request.method=='POST':
47          print(request.POST)
48          try:
49              token = request.POST['stripeToken']
50              email = request.POST['stripeEmail']
51              billingName = request.POST['stripeBillingName']
52              billingAddress1 = request.POST['stripeBillingAddressLine1']
53              billingcity = request.POST['stripeBillingAddressCity']
54              billingCountry = request.POST['stripeBillingAddressCountryCode']
55              shippingName = request.POST['stripeShippingName']
56              shippingAddress1 = request.POST['stripeShippingAddressLine1']
57              shippingcity = request.POST['stripeShippingAddressCity']
58              shippingCountry = request.POST['stripeShippingAddressCountryCode']
59
60              customer = stripe.Customer.create(email=email,
61                                                source=token)
```

Lines 51-58: The billing and shipping details we can get from the Stripe payment form using the request.POST[].

Now we need to create the order. Inside the cart_detail function copy-paste the following code to the location shown below:

```
'''Creating the order'''
try:
    order_details = Order.objects.create(
            token = token,
            total = total,
            emailAddress = email,
            billingName = billingName,
            billingAddress1 = billingAddress1,
            billingCity = billingcity,
            billingCountry = billingCountry,
            shippingName = shippingName,
            shippingAddress1 = shippingAddres
            shippingCity = shippingcity,
            shippingCountry = shippingCountry
        )
    order_details.save()
except ObjectDoesNotExist:
    pass
```

```
62          stripe.Charge.create(amount=stripe_total,
63                               currency="eur",
64                               description=description,
65                               customer=customer.id)
66
67          '''Creating the order'''
68          try:
69              order_details = Order.objects.create(
70                      token = token,
71                      total = total,
72                      emailAddress = email,
73                      billingName = billingName,
74                      billingAddress1 = billingAddress1,
75                      billingCity = billingcity,
76                      billingCountry = billingCountry,
77                      shippingName = shippingName,
78                      shippingAddress1 = shippingAddress1,
79                      shippingCity = shippingcity,
80                      shippingCountry = shippingCountry
81                  )
82              order_details.save()
83          except ObjectDoesNotExist:
84              pass
85
86      except stripe.error.CardError as e:
87          return e
```

Next, we need to iterate through the collection of cart items and assign each cart item to an order item. Copy-paste the following code to the cart_detail function in **cart/views.py** at the location shown below:

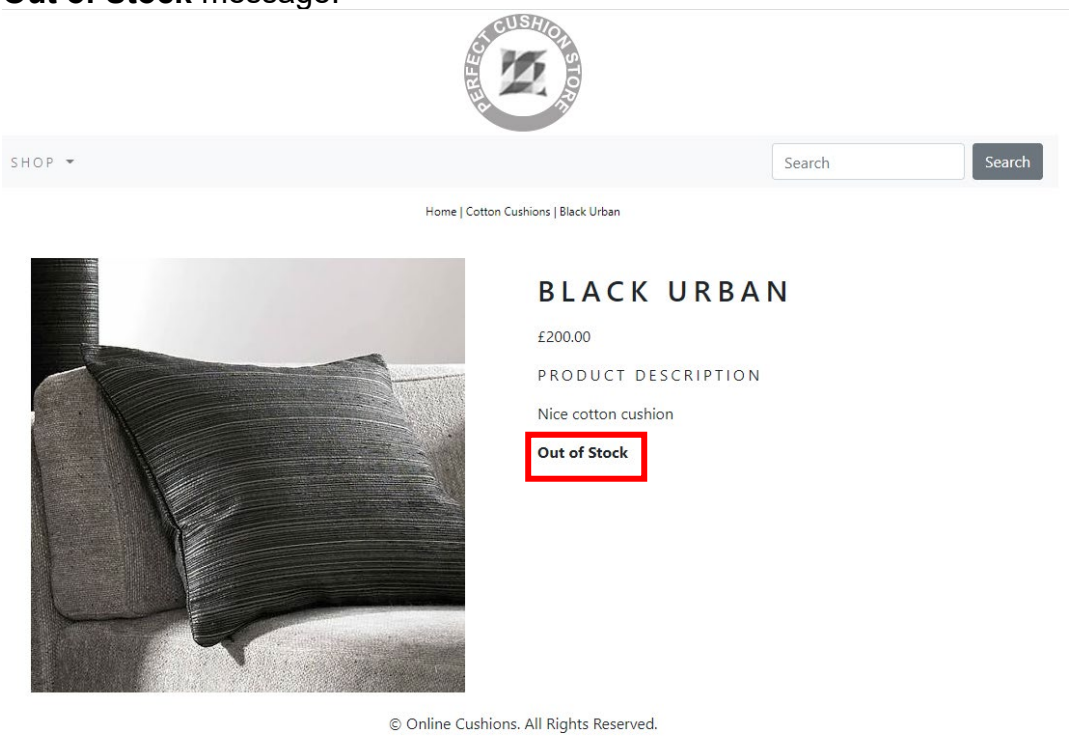
```
                for order_item in cart_items:
                    oi = OrderItem.objects.create(
                        product = order_item.product.name,
                        quantity = order_item.quantity,
                        price = order_item.product.price,
                        order = order_details)
                    oi.save
                    '''Reduce stock when order is placed or saved'''
                    products = Product.objects.get(id=order_item.product.id)
                    products.stock = int(order_item.product.stock -
order_item.quantity)
                    products.save()
                    order_item.delete()
                    '''The terminal will print this message when the order is
saved'''
                    print('The order has been created')
                return redirect ('shop:all_products')
```

```
82                  order_details.save()
83                  for order_item in cart_items:
84                      oi = OrderItem.objects.create(
85                          product = order_item.product.name,
86                          quantity = order_item.quantity,
87                          price = order_item.product.price,
88                          order = order_details)
89                      oi.save
90                      '''Reduce stock when order is placed or saved'''
91                      products = Product.objects.get(id=order_item.product.id)
92                      products.stock = int(order_item.product.stock - order_item.quantity)
93                      products.save()
94                      order_item.delete()
95                      '''The terminal will print this message when the order is saved'''
96                      print('Ther order has been created')
97                  return redirect('shop:all_products')
98              except ObjectDoesNotExist:
99                  pass
```

Lines 83-89: Iterate through the collection of items in the cart and for each cart item we create an order item object and save it to the database table
Lines 91-94: We find the product in the database matching the order item to reduce this quantity in the database. We save the product with the updated stock amount and then delete the item from the cart.
Line 96: We print a message to the terminal when the order has been saved
Line 97: We redirect the user back to the home page

Step 3: Try it out
Save the project and run the server & make a purchase. When the Stripe payment is complete you will see that the customer is redirected back to the home page.

You should also test out when a product is out of stock. For example, in the screen shot below all the Black Urban cushions have been purchased and we now see the **Out of Stock** message:



Step 4: Django Admin

To view the orders in Django admin we need to register the models. Copy-paste the following code into **order/admin.py**:

```
from .models import Order, OrderItem

class OrderItemAdmin(admin.TabularInline):
    model = OrderItem
    fieldsets = [
    ('Product',{'fields':['product'],}),
    ('Quantity',{'fields':['quantity'],}),
    ('Price',{'fields':['price'],}),
    ]
    readonly_fields = ['product','quantity','price']

class OrderAdmin(admin.ModelAdmin):
    list_display = ['id','billingName','emailAddress','created']
    list_display_links = ('id','billingName')
    search_fields = ['id','billingName','emailAddress']
    readonly_fields =
['id','token','total','emailAddress','created','billingName','billingAddress1'
,'billingCity',
                     'billingPostcode','billingCountry','shippingName','shippin
gAddress1','shippingCity','shippingPostcode','shippingCountry']
    fieldsets = [
    ('ORDER INFORMATION',{'fields': ['id','token','total','created']}),
    ('BILLING INFORMATION', {'fields':
['billingName','billingAddress1','billingCity','billingPostcode','billingCount
ry','emailAddress']}),
    ('SHIPPING INFORMATION', {'fields':
['shippingName','shippingAddress1','shippingCity','shippingPostcode','shipping
Country']}),
    ]

    inlines = [
        OrderItemAdmin
    ]

    def has_delete_permission(self, request, obj=None):
        return False

    def has_add_permission(self, request):
        return False


admin.site.register(Order, OrderAdmin)
```

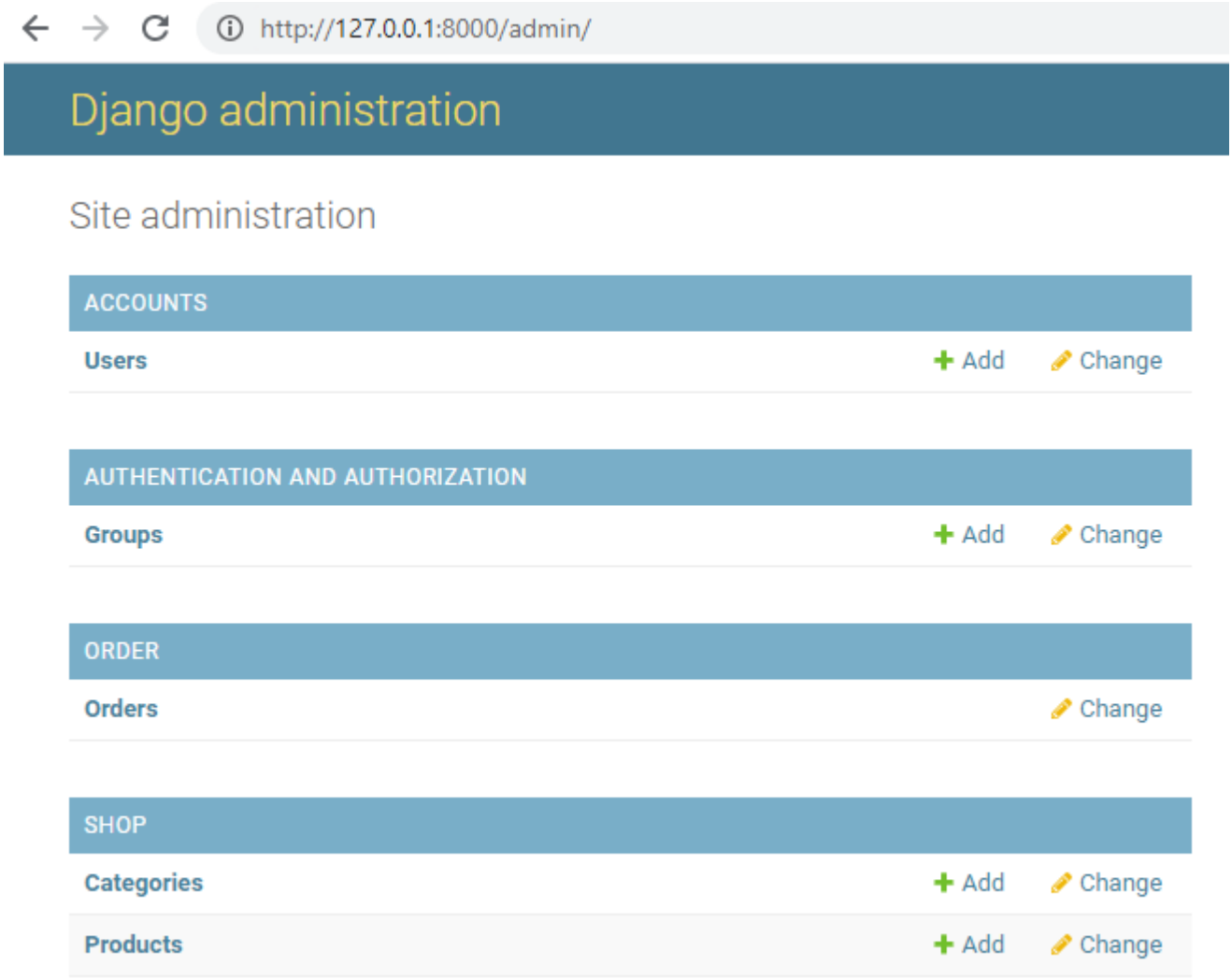
We make sure that these fields cannot be changed by making them read only

To see the order items off the order record in Django admin we need to include the OrderItemAdmin as part of the OrderAdmin class using inlines

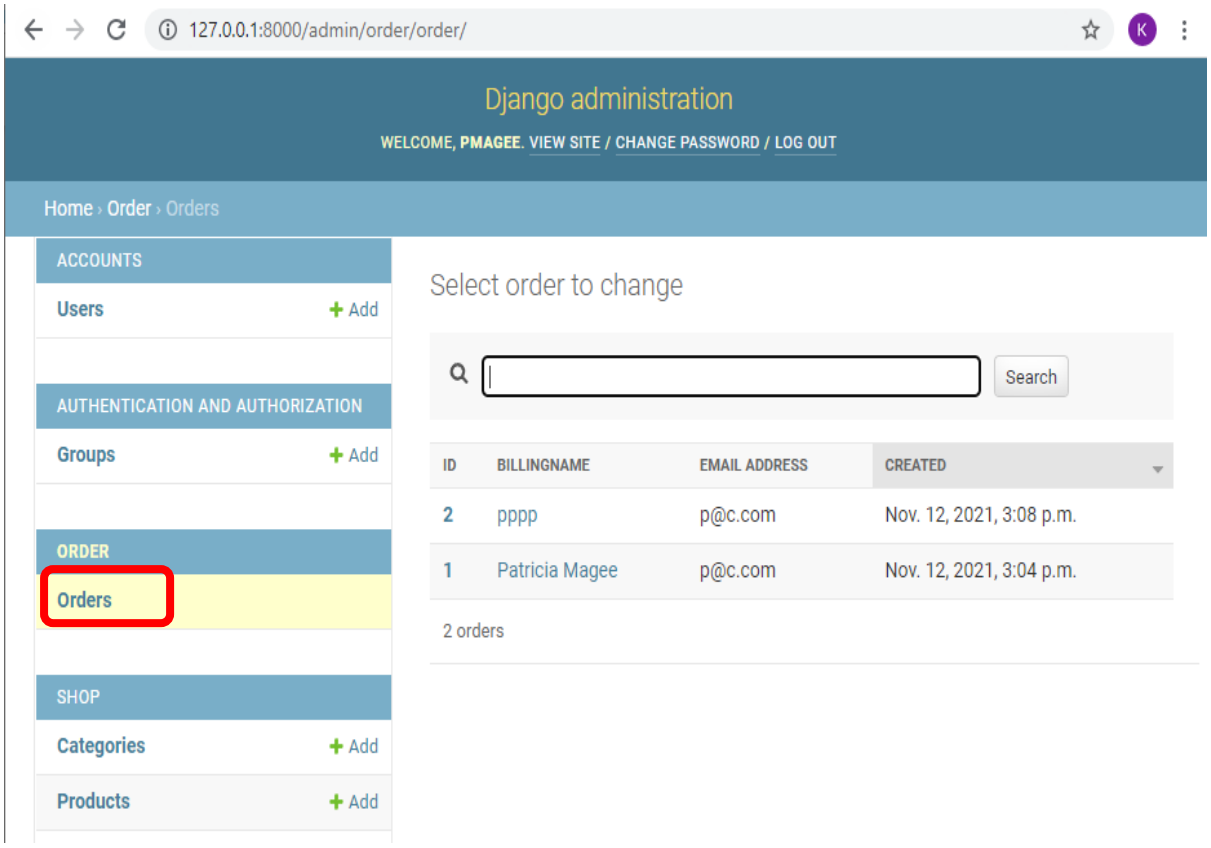
This code will disable the Delete functionality in Django Admin to prevent deletion of an order item on the back end

This code will disable the Add functionality in Django Admin to prevent deletion of an order item on the back end

Save the project and restart the server and log into **Django Admin**, you will now see that the Orders model is visible but there is no Add option:



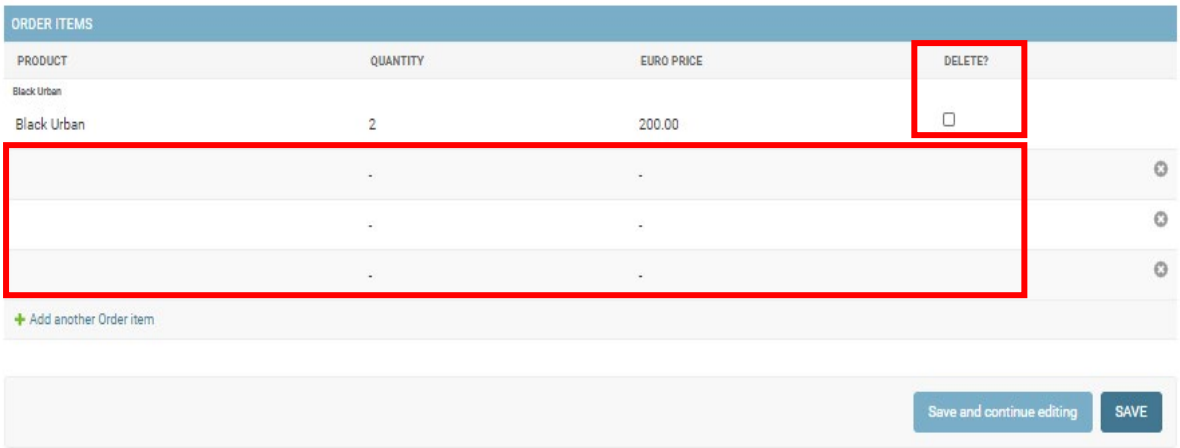
Click on the Orders and you will see the current orders in the database with 4 columns as per the list_display specified on line 14 of **admin.py**. Note the addition of the search bar which allows us to search by id, billingName or emailAddress (line 16 of admin.py)



Note also that the option to delete an order is not available.

If you click on an order to view it, you will see the entire order details are displayed including the order items at the bottom of the page.

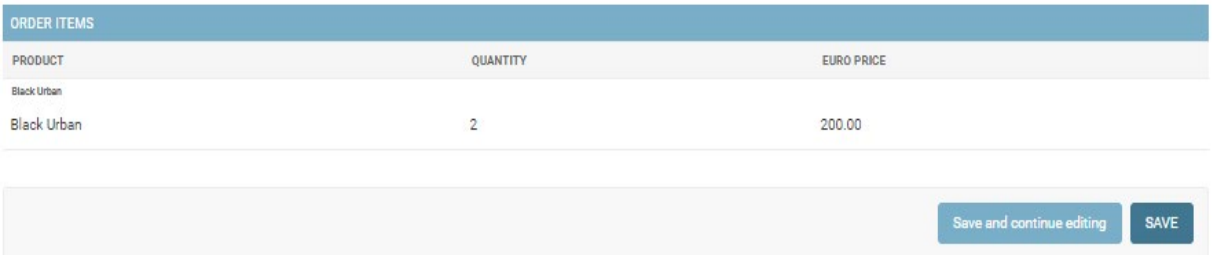
The order items at the bottom of the page include an option to delete an order item which we will remove. There are also additional empty rows displayed and there is an option to add an order item which we will also remove:



Add the following code to **admin.py** to disable the delete option and to get rid of the additional rows:

```
class OrderItemAdmin(admin.TabularInline):
    model = OrderItem
    fieldsets = [
    ('Product',{'fields':['product'],}),
    ('Quantity',{'fields':['quantity'],}),
    ('Price',{'fields':['price'],}),
    ]
    readonly_fields = ['product','quantity','price']
    can_delete= False
    max_num = 0
```

Refresh the browser and you should see the following:



Step 5: Commit your changes and push your code to the lab 10 repo on GitHub
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 7 commit"
git push -u origin main