**Регистрация заказов**

При извлечении товаров из корзины для покупок необходимо сохранить заказ в базе данных. Заказы будут содержать информацию о клиентах и продуктах, которые они покупают.

Создайте новое приложение для управления заказами клиентов, используя следующую команду:

python manage.py startapp orders

Отредактируйте **settings.py** вашего проекта и добавьте приложение **'orders'** в параметры **INSTALLED\_APPS:**

INSTALLED\_APPS = (

# ...

'orders',

)

Вы активировали новое приложение.

## Создание модели order

Потребуется модель для хранения сведений о заказе и второй модели для хранения купленных товаров, включая их количество и цену. Измените файл **models.py** приложения **orders** и добавьте в него следующий код:

from django.db import models

from shop.models import Product

class Order(models.Model):

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

email = models.EmailField()

address = models.CharField(max\_length=250)

postal\_code = models.CharField(max\_length=20)

city = models.CharField(max\_length=100)

created = models.DateTimeField(auto\_now\_add=True)

updated = models.DateTimeField(auto\_now=True)

paid = models.BooleanField(default=False)

class Meta:

ordering = ('-created',)

verbose\_name = 'Заказ'

verbose\_name\_plural = 'Заказы'

def \_\_str\_\_(self):

return 'Order {}'.format(self.id)

def get\_total\_cost(self):

return sum(item.get\_cost() for item in self.items.all())

class OrderItem(models.Model):

order = models.ForeignKey(Order, related\_name='items')

product = models.ForeignKey(Product, related\_name='order\_items')

price = models.DecimalField(max\_digits=10, decimal\_places=2)

quantity = models.PositiveIntegerField(default=1)

def \_\_str\_\_(self):

return '{}'.format(self.id)

def get\_cost(self):

return self.price \* self.quantity

Модель **Order** содержит несколько полей для сведений о клиенте и поле **paid**, которое по умолчанию имеет значение **False**. Позже мы будем использовать это поле для различения оплаченных и неоплаченных заказов. Мы также определяем метод **get\_total\_cost()**, чтобы получить общую стоимость товаров, купленных в этом заказе.

Модель **OrderItem** позволяет хранить продукт, количество и цену, уплаченную за каждый товар. Мы включаем **get\_cost()** для возврата стоимости товара.

Выполните следующую команду, чтобы создать начальные миграции для приложения **orders**:

python manage.py makemigrations

Будут выведены следующие данные:

Migrations for 'orders':

0001\_initial.py:

- Create model Order

- Create model OrderItem

Для применения новой миграции выполните следующую команду:

python manage.py migrate

Теперь модели заказа синхронизируются с базой данных.

## Включение моделей заказов на сайте администрирования

Давайте добавим модели заказов на сайт администрирования. Измените файл **admin.py** приложения **orders**:

from django.contrib import admin

from .models import Order, OrderItem

class OrderItemInline(admin.TabularInline):

model = OrderItem

raw\_id\_fields = ['product']

class OrderAdmin(admin.ModelAdmin):

list\_display = ['id', 'first\_name', 'last\_name', 'email',

'address', 'postal\_code', 'city', 'paid',

'created', 'updated']

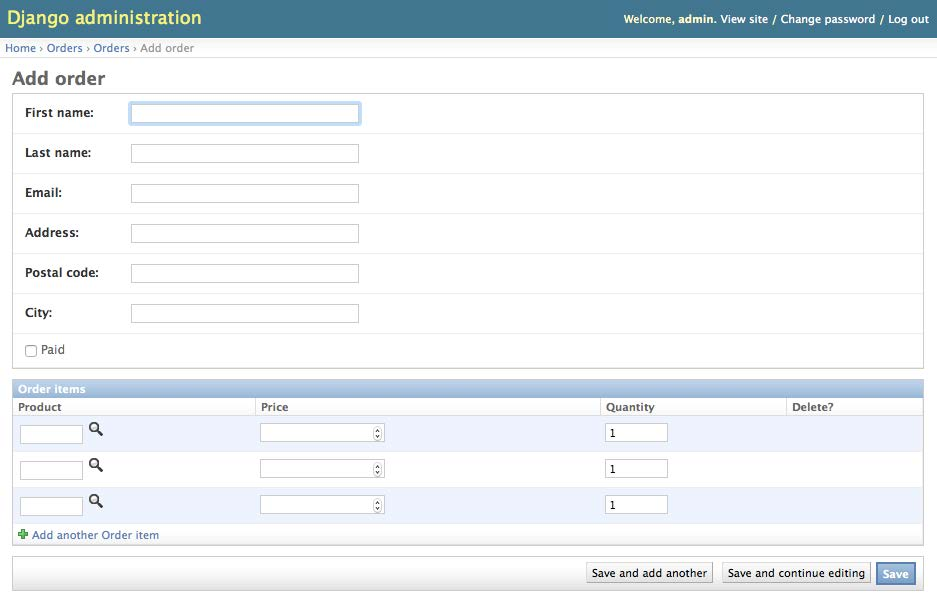
list\_filter = ['paid', 'created', 'updated']

inlines = [OrderItemInline]

admin.site.register(Order, OrderAdmin)

Мы используем **ModelInline** для модели **OrderItem**, чтобы включить ее в качестве inline встроенного в класс OrderAdmin. Inline режим позволяет включить модель для отображения на той же странице редактирования, что и родительская модель.

Запустите сервер разработки командой python manage.py runserver, а затем откройте в браузере <http://127.0.0.1:8000/admin/orders/order/add/> . Появится следующая страница:



**Создание заказов клиентов**

Нам нужно использовать только что созданные модели заказов для сохранения товаров, содержащихся в корзине для покупок, когда пользователь наконец пожелает разместить заказ. Функции создания нового заказа будут работать следующим образом:

1. Мы предоставляем форму заказа для заполнения пользовательских данных.
2. Создается новый экземпляр заказа с данными, введенными пользователями, а затем создается связанный экземпляр **OrderItem** для каждого товара в корзине.
3. Очистищаем все содержимое корзины и перенаправляем пользователей на страницу success

Во-первых, нам нужна форма для ввода сведений о заказе. Создайте новый файл в каталоге приложения **orders** и назовите его **forms.py**. Добавьте в него следующий код:

from django import forms

from .models import Order

class OrderCreateForm(forms.ModelForm):

class Meta:

model = Order

fields = ['first\_name', 'last\_name', 'email', 'address', 'postal\_code', 'city']

Это форма, которую мы собираемся использовать для создания новых объектов **Order**. Теперь нам нужно представление, чтобы обработать форму и создать новый заказ. Измените файл **views.py** приложения **orders** и добавьте в него следующий код:

from django.shortcuts import render

from .models import OrderItem

from .forms import OrderCreateForm

from cart.cart import Cart

def order\_create(request):

cart = Cart(request)

if request.method == 'POST':

form = OrderCreateForm(request.POST)

if form.is\_valid():

order = form.save()

for item in cart:

OrderItem.objects.create(order=order,

product=item['product'],

price=item['price'],

quantity=item['quantity'])

# очистка корзины

cart.clear()

return render(request, 'orders/order/created.html',

{'order': order})

else:

form = OrderCreateForm

return render(request, 'orders/order/create.html',

{'cart': cart, 'form': form})

В представлении **order\_create** мы получаем текущую корзину из сесссии с **cart = Cart(request)**. В зависимости от метода запроса мы будем выполнять следующие задачи:

* **GET request** : Создается экземпляр формы **OrderCreateForm** и отображается шаблон **orders/order/create.html**
* **POST request** : Проверяет валидность введенных данных. Если данные являются допустимыми, то для создания нового экземпляра заказа будет использоваться **order = form.save()**. Затем мы сохраняем его в базу данных, а затем храним в переменной **order**. После создания заказа мы перейдем по товарам корзины и создадим **OrderItem** для каждого из них. Наконец, мы очищаем содержимое корзины

Теперь создайте новый файл в каталоге приложения **orders** и назовите его **urls.py**. Добавьте в него следующий код:

from django.conf.urls import url

from . import views

urlpatterns = [

url(r'^create/$', views.order\_create, name='order\_create'),

]

Это шаблон URL-адреса для представления **order\_create**. Отредактируйте файл **urls.py** **myshop** и включите следующий шаблон. Не забывайте, его следует разместить перед паттерном **shop.urls**:

url(r'^orders/', include('orders.urls', namespace='orders')),

Отредактируйте шаблон cart/detail.html приложения **cart** и замените эту строку:

<a href="#" class="button">Checkout</a>

На следующий код:

<a href="{% url "orders:order\_create" %}" class="button">

Checkout

</a>

Теперь пользователи могут перейти от страницы cart detail к странице order form. Нам по-прежнему необходимо определить шаблоны для размещения заказов. Создайте следующую структуру файла в каталоге приложения **orders**:

templates/

orders/

order/

create.html

created.html

Отредактируйте шаблон **orders/order/create.html**:

{% extends "shop/base.html" %}

{% block title %}Checkout{% endblock %}

{% block content %}

<h1>Checkout</h1>

<div class="order-info">

<h3>Your order</h3>

<ul>

{% for item in cart %}

<li>

{{ item.quantity }}x {{ item.product.name }}

<span>${{ item.total\_price }}</span>

</li>

{% endfor %}

</ul>

<p>Total: ${{ cart.get\_total\_price }}</p>

</div>

<form action="." method="post" class="order-form">

{{ form.as\_p }}

<p><input type="submit" value="Place order"></p>

{% csrf\_token %}

</form>

{% endblock %}

Этот шаблон отображает товары корзины, включая итоговую сумму, и форму для размещения заказа.

Отредактируйте шаблон **orders/order/created.html**:

{% extends "shop/base.html" %}

{% block title %}Thank you{% endblock %}

{% block content %}

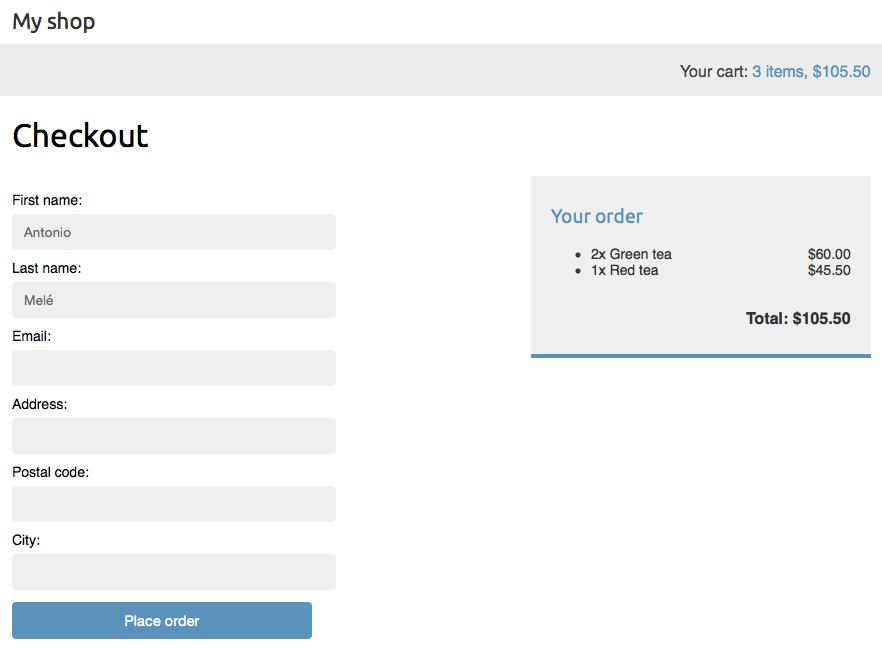
<h1>Thank you</h1>

<p>Your order has been successfully completed. Your order number is

<strong>{{ order.id }}</strong>.</p>

{% endblock %}

Это шаблон, который мы показываем при успешном создании заказа. Запустите сервер веб-разработки. Откройте / в браузере [http://127.0.0.1:8000](http://127.0.0.1:8000/), добавьте в корзину несколько товаров и нажмите на ссылку **checkout**. Появится страница, подобная следующей:



Заполните форму валидными данными и нажмите кнопку **Place order**. Заказ будет создан, и появится страница сообщающая об успешном выполнении действия:

