МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

КАФЕДРА ИНФОРМАТИКА

КУРСОВАЯ РАБОТА

СОЗДАНИЕ ПРОЕКТА ИНТЕРНЕТ-МАГАЗИН

Выполнил:студент группы БИБ2103

Тагоев Т.А.

Проверил:

Москва 2022

Цель работы:

Вам необходимо создать интернет-магазин для продажи любых товаров (отличных от образца). Задание включает в себя как обязательную часть, так и дополнительное задание, выполнение которого положительно скажется на оценках за КР и экзамен. Сразу стоит отметить, что версии фреймворков указаны в качестве примера и вам необязательно использовать именно эти версии (так как возможно, что они уже устарели), поэтому перед выполнением, ознакомьтесь с актуальными версиями и используйте их. Создайте новый проект Джанго, чтобы разработать Интернет-Магазин. Пользователи смогут просматривать каталог продуктов и добавлять продукты в корзину для покупок. Будут охвачены следующие функциональные возможности Интернет-магазина: • Создание моделей каталога продуктов, добавление их на сайт администрирования и создание основных представлений для отображения каталога. • Создание системы корзины для покупок с помощью сессий Джанго, чтобы пользователи могли сохранять выбранные продукты при просмотре сайта. • Создание форм и функциональных возможностей для размещения заказов. • Отправка по электронной почте подтверждение пользователям при размещении заказа.

Выполнение:

Сначала создайте виртуальную среду для нового проекта и активируйте ее следующим образом:

mkdir env virtualenv

env/myshop

env/myshop/Scripst/activate

Установка Django в окружение:

pip install Django==2.0.5

Создаем новый проект myshop c приложением shop:

django-admin startproject myshop

cd myshop/

django-admin startapp shop

Редактируем файл настроек settings.py, добавляем приложение shop в список ранее установленных:

INSTALLED\_APPS = [

# ...

'shop.apps.ShopConfig',

]

Добавление моделей каталога товаров

Отредактируем файл models.py приложения shop

from django.db import models

class Category(models.Model):

name = models.CharField(max\_length=200, db\_index=True)

slug = models.SlugField(max\_length=200, unique=True)

class Meta:

ordering = ('name',)

verbose\_name = 'category'

verbose\_name\_plural = 'categories'

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

return self.name

class Product(models.Model):

category = models.ForeignKey(Category,

related\_name='products',

on\_delete=models.CASCADE)

name = models.CharField(max\_length=200, db\_index=True)

slug = models.SlugField(max\_length=200, db\_index=True)

image = models.ImageField(upload\_to='products/%Y/%m/%d', blank=True)

description = models.TextField(blank=True)

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

available = models.BooleanField(default=True)

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

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

class Meta:

ordering = ('name',)

index\_together = (('id', 'slug'),)

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

return self.name

Этот фрагмент описывает две модели: для категории, Category, и товара, Product. Модель Category состоит из двух полей: наименования, name, и уникального поля слага, slug.

Поскольку наша модель содержит поле для изображений, необходимо установить пакет Pillow с помощью команды:

pip install Pillow==5.1.0

Запустим команду создания миграций и выполним синхронизацию миграций и базы данных:

Python manage.py makemigrations

Python manage.py migrate

Регистрация моделей каталога на сайте администрирования

Добавим модели на сайт администрирования, чтобы управлять категориями и настраивать отображение товаров. Отредактируем файл admin.py приложения shop:

from django.contrib import admin

from .models import Category, Product

@admin.register(Category)

class CategoryAdmin(admin.ModelAdmin):

list\_display = ['name', 'slug']

prepopulated\_fields = {'slug': ('name',)}

@admin.register(Product)

class ProductAdmin(admin.ModelAdmin):

list\_display = ['name', 'slug', 'price', 'available', 'created', 'updated']

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

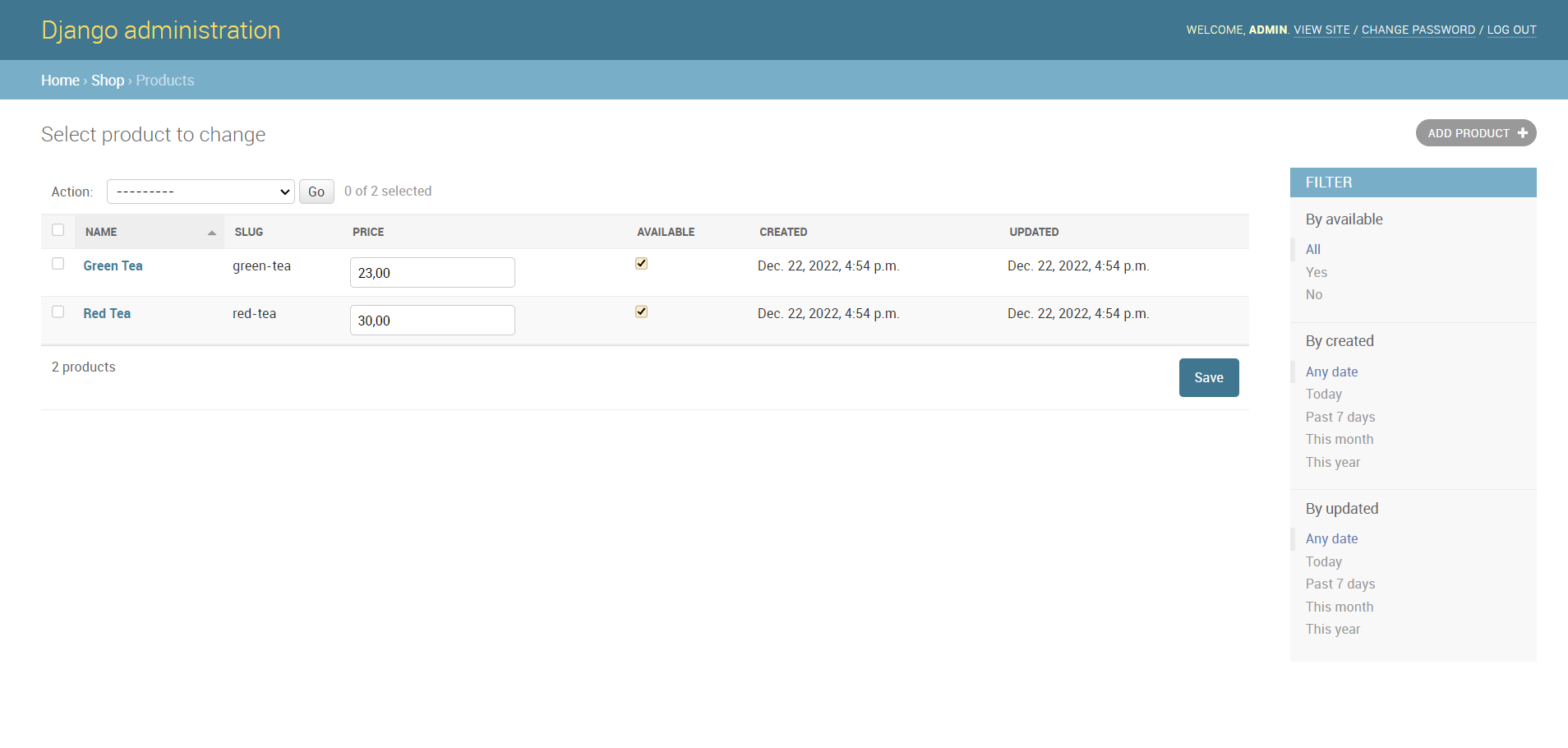
list\_editable = ['price', 'available']

prepopulated\_fields = {'slug': ('name',)}

Создаем суперпользователя:

Python manage.py createsuperuser

Запустим сервер python manage.py runserver и перейдем на сайт администрирования по адресу <http://127.0.0.1:8000/admin>, добавим категорию и товар



Реализация обработчиков для каталога

Отредактируем файл views.py:

from django.shortcuts import render, get\_object\_or\_404

from .models import Category, Product

def product\_list(request, category\_slug=None):

category = None

categories = Category.objects.all()

products = Product.objects.filter(available=True)

if category\_slug:

category = get\_object\_or\_404(Category, slug=category\_slug)

products = products.filter(category=category)

return render(request, 'shop/product/list.html',

{'category': category,

'categories': categories,

'products': products}

Кроме отображения списка, в магазине нужно отображать страницу каждого товара с его подробным описанием:

def product\_detail(request, id, slug):

product = get\_object\_or\_404(Product, id=id, slug=slug, available=True)

return render(request, 'shop/product/detail.html', {'product': product})

Cоздаем новый файл urls.py в папке приложения shop и добавим:

from django.urls import path

from . import views

app\_name = 'shop'

urlpatterns = [

path('', views.product\_list, name='product\_list'),

path('<slug:category\_slug>/', views.product\_list,

name='product\_list\_by\_category'),

path('<int:id>/<slug:slug>/', views.product\_detail, name='product\_detail'),

]

Редактирум файл urls.py проекта myshop:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('shop.urls', namespace='shop')),

]

Добавляем в модели метод получения канонического URL’а конкретного объекта. Отредактируем файл models.py приложения shop:

from django.urls import reverse

# ...

class Category(models.Model):

# ...

def get\_absolute\_url(self):

return reverse('shop:product\_list\_by\_category', args=[self.slug])

class Product(models.Model):

# ...

def get\_absolute\_url(self):

return reverse('shop:product\_detail', args=[self.id, self.slug])

Добавление шаблонов для отображения страниц каталога

Описываем шаблоны для отображения товаров. Создаем структуру каталогов и файлов в папке приложения shop:

Templates/

Shop/

Base.html

Product/

List.html

Detail.html

Определяем базовый шаблон, который будет задавать общую структуру страницы сайта.Отредактируем файл shop/base.html:

{% load static %}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>{% block title %}My shop{% endblock %}</title>

<link href="{% static "css/base.css" %}" rel="stylesheet">

</head>

<body>

<div id="header">

<a href="/" class="logo">My shop</a>

</div>

<div id="subheader">

<div class="cart">Your cart is empty.</div>

</div>

<div id="content">

{% block content %}{% endblock %}

</div>

</body>

</html>

Редактируем файл list.html:

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

{% load static %}

{% block title %}

{% if category %}{{ category.name }}{% else %}Products{% endif %}

{% endblock %}

{% block content %}

<div id="sidebar">

<h3>Categories</h3>

<ul>

<li {% if not category %}class="selected"{% endif %}>

<a href="{% url "shop:product\_list" %}">All</a>

</li>

{% for c in categories %}

<li {% if category.slug == c.slug %}class="selected" {% endif %}>

<a href="{{ c.get\_absolute\_url }}">{{ c.name }}</a>

</li>

{% endfor %}

</ul>

</div>

<div id="main" class="product-list">

<h1>{% if category %}{{ category.name }}{% else %}Products{% endif %}</h1>

{% for product in products %}

<div class="item">

<a href="{{ product.get\_absolute\_url }}">

<img src="{% if product.image %}{{ product.image.url }}{%

else %}{% static "img/no\_image.png" %}{% endif %}">

</a>

<a href="{{ product.get\_absolute\_url }}">{{ product.name }}</a>

<br>

${{ product.price }}

</div>

{% endfor %}

</div>

{% endblock %}

Так как для сохранения изображений используется поле ImageField, необходимо настроить сервер для разработки, чтобы он находил загруженные пользователями картинки.

Отредактируем файл settings.py проекта myshop и добавим:

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media/')

Чтобы сервер Django мог обращаться к файлам пользователей, отредактируем файл urls.py проекта myshop и добавим:

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

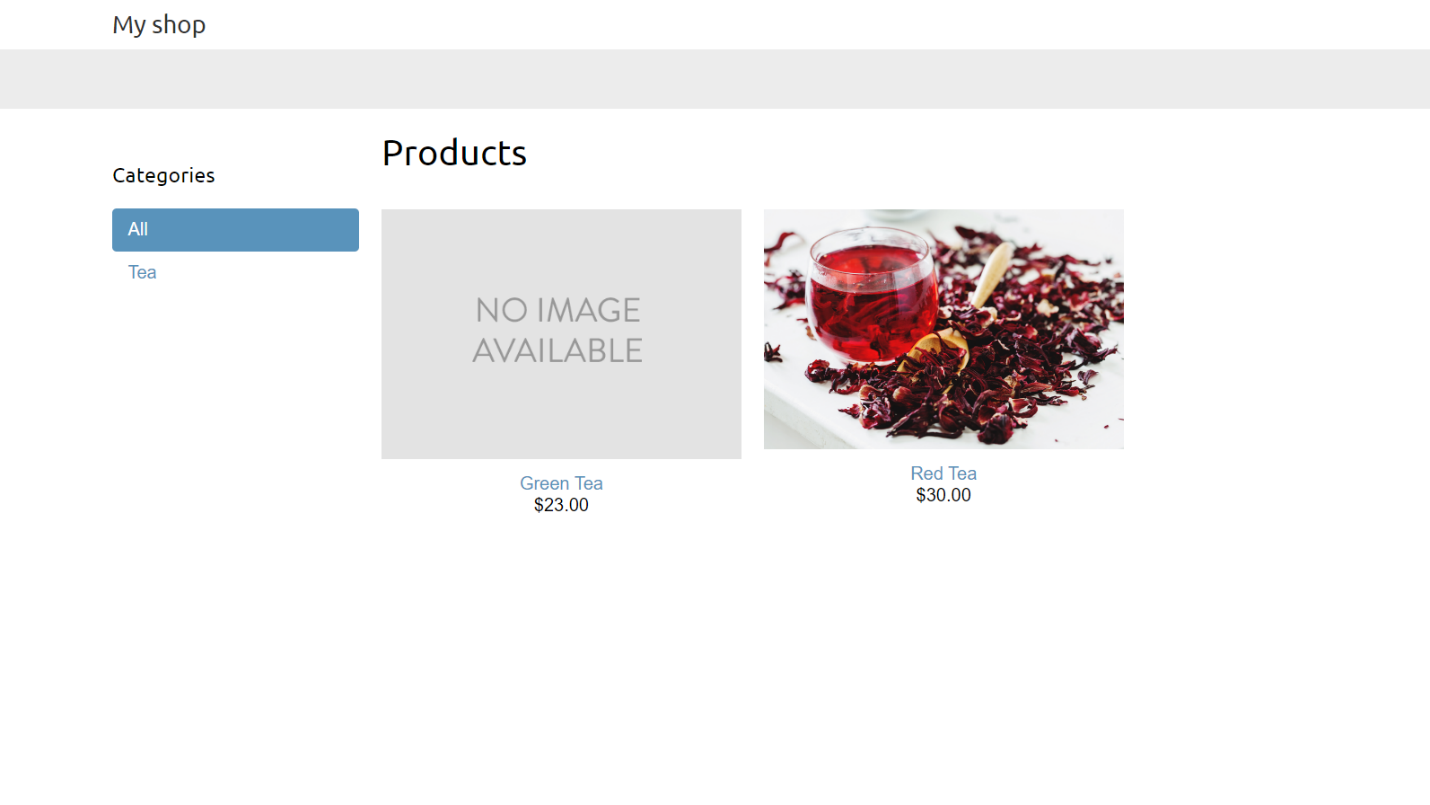
# ...

]

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Добавим несколько файлов через сайт администрирования и перейдем на главную страницу на <http://127.0.0.1:8000/>.



Добавим страницу товара. Редактируем файл detail.html:

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

{% load static %}

{% block title %}

{{ product.name }}

{% endblock %}

{% block content %}

<div class="product-detail">

<img src="{% if product.image %}{{ product.image.url }}{% else %}

{% static "img/no\_image.png" %}{% endif %}">

<h1>{{ product.name }}</h1>

<h2>

<a href="{{ product.category.get\_absolute\_url }}">{{product.category }}</a>

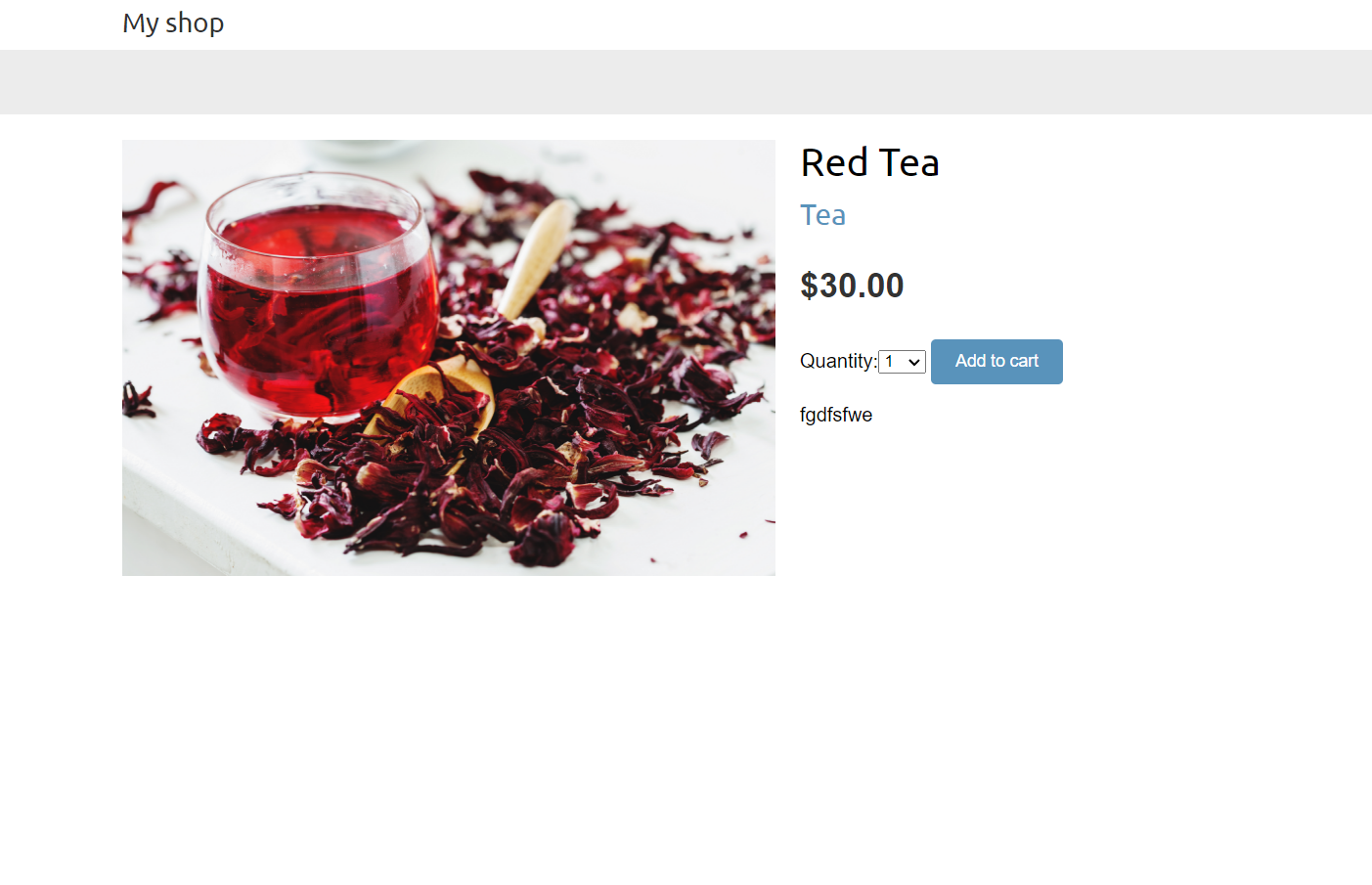
</h2>

<p class="price">${{ product.price }}</p>

{{ product.description|linebreaks }}

</div>

{% endblock %}



Создание корзины

Создаем приложение для работы с корзиной покупок:

Python manage.py startapp cart

Затем добавим его в список активных приложений проекта:

INSTALLED\_APPS = [

# ...

'shop.apps.ShopConfig',

'cart.apps.CartConfig',

]

Создаем в каталоге приложения cart файл cart.py

from decimal import Decimal

from django.conf import settings

from shop.models import Product

class Cart(object):

def \_\_init\_\_(self, request):

"""Инициализация объекта корзины."""

self.session = request.session

cart = self.session.get(settings.CART\_SESSION\_ID)

if not cart:

# Сохраняем в сессии пустую корзину.

cart = self.session[settings.CART\_SESSION\_ID] = {}

self.cart = cart

from shop.models import Product

class Cart(object):

def \_\_init\_\_(self, request):

"""Инициализация объекта корзины."""

self.session = request.session

cart = self.session.get(settings.CART\_SESSION\_ID)

if not cart:

# Сохраняем в сессии пустую корзину.

cart = self.session[settings.CART\_SESSION\_ID] = {}

self.cart = cart

class Cart(object):

# ...

def remove(self, product):

"""Удаление товара из корзины."""

product\_id = str(product.id)

if product\_id in self.cart:

del self.cart[product\_id]

self.save()

class Cart(object):

# ...

def \_\_iter\_\_(self):

"""Проходим по товарам корзины и получаем соответствующие объекты Product."""

product\_ids = self.cart.keys()

# Получаем объекты модели Product и передаем их в корзину.

products = Product.objects.filter(id\_\_in=product\_ids)

cart = self.cart.copy()

for product in products:

cart[str(product.id)]['product'] = product

for item in cart.values():

item['price'] = Decimal(item['price'])

item['total\_price'] = item['price'] \* item['quantity']

yield item

class Cart(object):

# ...

def \_\_len\_\_(self):

"""Возвращает общее количество товаров в корзине."""

return sum(item['quantity'] for item in self.cart.values())

class Cart(object):

# ...

def get\_total\_price(self):

return sum(

Decimal(item['price']) \* item['quantity']

for item in self.cart.values()

)

class Cart(object):

# ...

def clear(self):

# Очистка корзины.

del self.session[settings.CART\_SESSION\_ID]

self.save()

Обработка действий с корзиной покупок

Чтобы пользователь мог добавить товар в корзину, необходимо создать форму, в которой он мог бы выбрать количество единиц. Добавим файл forms.py в каталог приложения cart со следующим содержимым:

from django import forms

PRODUCT\_QUANTITY\_CHOICES = [(i, str(i)) for i in range(1, 21)]

class CartAddProductForm(forms.Form):

quantity = forms.TypedChoiceField(choices=PRODUCT\_QUANTITY\_CHOICES, coerce=int)

update = forms.BooleanField(required=False,

initial=False,

widget=forms.HiddenInput)

Создание обработчика для добавления товаров в корзину.Откроем файл views.py приложения cart и вставляем такое содержимое:

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.views.decorators.http import require\_POST

from shop.models import Product

from .cart import Cart

from .forms import CartAddProductForm

@require\_POST

def cart\_add(request, product\_id):

cart = Cart(request)

product = get\_object\_or\_404(Product, id=product\_id)

form = CartAddProductForm(request.POST)

if form.is\_valid():

cd = form.cleaned\_data

cart.add(product=product,

quantity=cd['quantity'],

update\_quantity=cd['update'])

return redirect('cart:cart\_detail')

Для удаления товаров из корзины создаем обработчик cart\_remove. Дописываем его код в файл views.py :

def cart\_remove(request, product\_id):

cart = Cart(request)

product = get\_object\_or\_404(Product, id=product\_id)

cart.remove(product)

return redirect('cart:cart\_detail')

Созадаем обработчик для страницы списка товаров, добавленных в корзину:

def cart\_detail(request):

cart = Cart(request)

return render(request, 'cart/detail.html', {'cart': cart})

Описываем шаблоны URL’ов для них. Создаем новый файл, urls.py, в папке приложения cart и добавьте в него такой код:

from django.urls import path

from . import views

app\_name = 'cart'

urlpatterns = [

path('', views.cart\_detail, name='cart\_detail'),

path('add/<int:product\_id>/', views.cart\_add, name='cart\_add'),

path('remove/<int:product\_id>/', views.cart\_remove, name='cart\_remove'),

]

Подключаем созданный файл в основном urls.py проекта myshop:

urlpatterns = [

path('admin/', admin.site.urls),

path('cart/', include('cart.urls', namespace='cart')),

path('', include('shop.urls', namespace='shop')),

]

Шаблон для отображения содержимого корзины

Создаем шаблон для отображения содержимого корзины. Внутри каталога приложения cart создаем такую структуру папок:

Templates/

Cart/

Detail.html

Редактируем файл detail.html:

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

{% load static %}

{% block title %}Your shopping cart{% endblock %}

{% block content %}

<h1>Your shopping cart</h1>

<table class="cart">

<thead>

<tr>

<th>Image</th>

<th>Product</th>

<th>Quantity</th>

<th>Remove</th>

<th>Unit price</th>

<th>Price</th>

</tr>

</thead>

<tbody>

{% for item in cart %}

{% with product=item.product %}

<tr>

<td>

<a href="{{ product.get\_absolute\_url }}">

<img src="{% if product.image %}{{ product.image.url }}

{% else %}{% static "img/no\_image.png" %}{% endif %}">

</a>

</td>

<td>{{ product.name }}</td>

<td>{{ item.quantity }}</td>

<td><a href="{% url "cart:cart\_remove" product.id%}">Remove</a></td>

<td class="num">${{ item.price }}</td>

<td class="num">${{ item.total\_price }}</td>

</tr>

{% endwith %}

{% endfor %}

<tr class="total">

<td>Total</td>

<td colspan="4"></td>

<td class="num">${{ cart.get\_total\_price }}</td>

</tr>

</tbody>

</table>

<p class="text-right">

<a href="{% url "shop:product\_list" %}" class="button light">Continue shopping</a>

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

</p>

{% endblock %}

Добавление товаров в шаблонах

Теперь нужно добавить кнопку Add to cart на страницу товара. Отредактируем файл views.py приложения shop и добавим форму CartAddProductForm

from cart.forms import CartAddProductForm

def product\_detail(request, id, slug):

product = get\_object\_or\_404(Product, id=id, slug=slug, available=True)

cart\_product\_form = CartAddProductForm()

return render(request, 'shop/product/detail.html',

{'product': product,

'cart\_product\_form': cart\_product\_form})

Редактируем шаблон detail.html

<p class="price">${{ product.price }}</p>

<form action="{% url "cart:cart\_add" product.id %}" method="post">

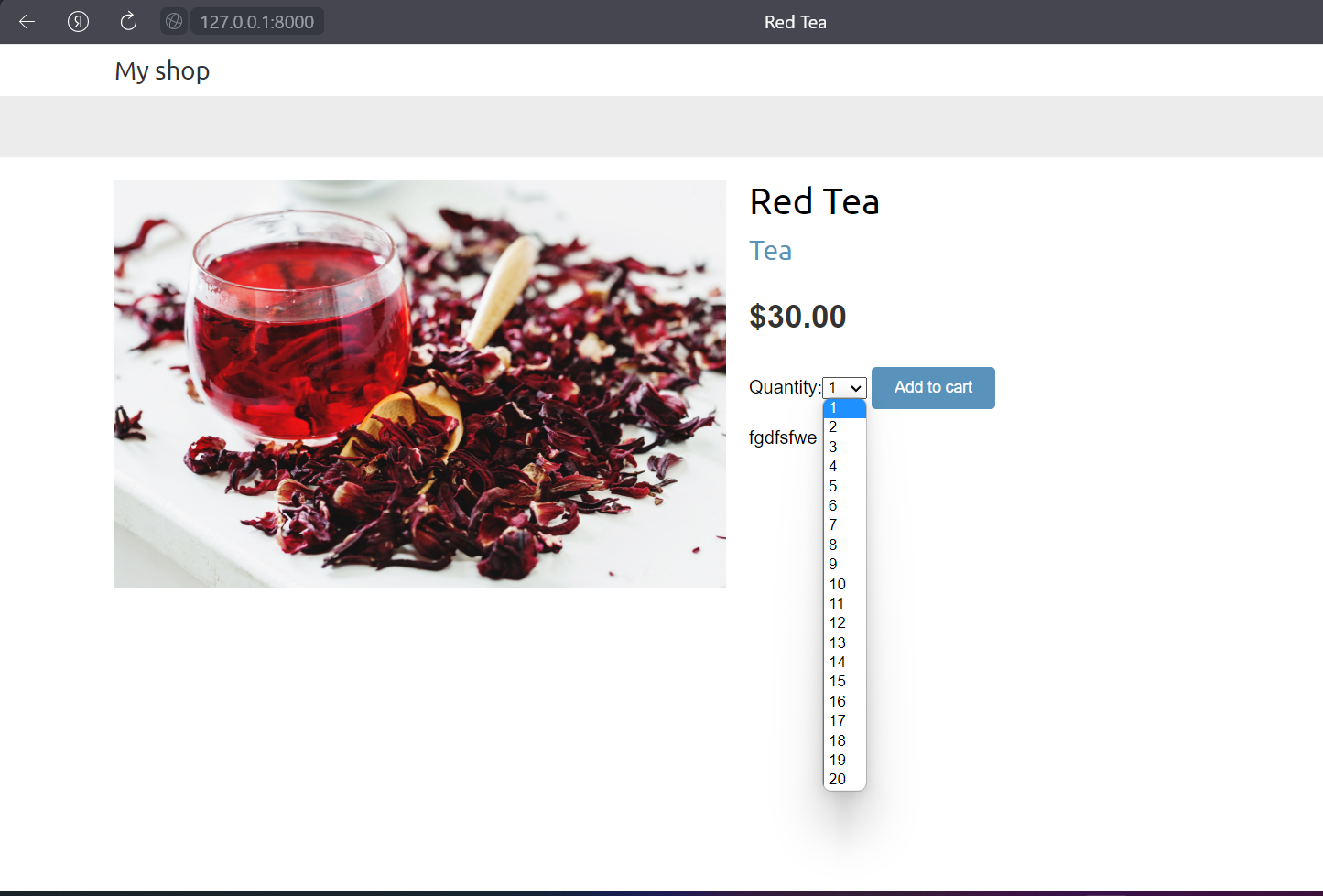
{{ cart\_product\_form }}

{% csrf\_token %}

<input type="submit" value="Add to cart">

</form>

{{ product.description|linebreaks }}



Изменение количества товаров в корзине

Редактируем файл views.py приложения cart:

def cart\_detail(request):

cart = Cart(request)

for item in cart:

item['update\_quantity\_form'] = CartAddProductForm(

initial={'quantity': item['quantity'],

'update': True})

return render(request, 'cart/detail.html', {'cart': cart})

Добавим отображение формы на странице корзины. Откроем файл cart/detail.html приложения cart:

<td>

<form action="{% url "cart:cart\_add" product.id %}" method="post">

{{ item.update\_quantity\_form.quantity }}

{{ item.update\_quantity\_form.update }}

<input type="submit" value="Update">

{% csrf\_token %}

</form>

</td>

Изображение выглядит как текст

Автоматически созданное описание

Создание контекстного процессора для корзины

Создаем новый файл context\_processors.py в папке приложения cart

from .cart import Cart

def cart(request):

return {'cart': Cart(request)}

Откроем файл settings.py проекта и допишем cart.context\_processors.cart в раздел context\_processors настройки TEMPLATES:

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

# ...

'cart.context\_processors.cart',

],

},

},

]

Откроем шаблон shop/base.html приложения shop и отредактируем:

<div class="cart">

{% with total\_items=cart|length %}

{% if cart|length > 0 %}

Your cart:

<a href="{% url "cart:cart\_detail" %}">

{{ total\_items }} item{{ total\_items|pluralize }},

${{ cart.get\_total\_price }}

</a>

{% else %}

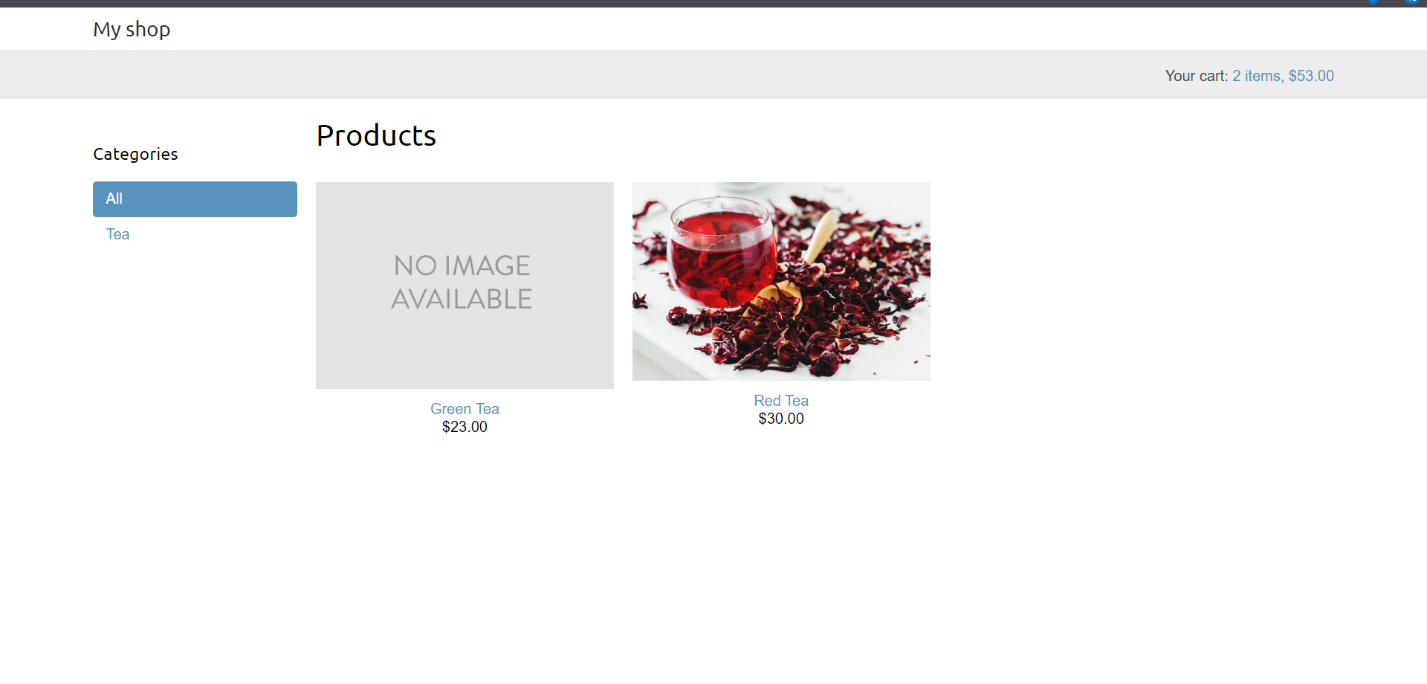
Your cart is empty.

{% endif %}

{% endwith %}

</div>

Перезапустив сервер разработки, добавим несколько товаров в корзину. В верхней части страницы мы увидим количество товаров и общую стоимость покупок корзины:



Регистрация заказов

Нам необходимо создать и сохранить заказ в базе данных. Объект заказа будет содержать сведения о покупателе и товарах из корзины.

Создаем приложение для управления заказами с помощью команды:

python manage.py startapp orders

Добавим его в список установленных приложений в файле settings.py

INSTALLED\_APPS = [

...

'orders.apps.OrdersConfig',

]

Создание моделей заказа

Отредактируем файл models.py приложения orders:

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',)

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',

on\_delete=models.CASCADE)

product = models.ForeignKey(Product,

related\_name='order\_items',

on\_delete=models.CASCADE)

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

Выполним команду для создания начальной миграции в приложении orders:

python manage.py makemigrations

И запустим синхронизацию моделей и базы данных командой:

python manage.py migrate

Добавление моделей на сайт администрирования

Редактируем 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']

@admin.register(Order)

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]

Изображение выглядит как стол

Автоматически созданное описание

Обработка заказов покупателей

Добавляем новый файл forms.py в папку приложения orders:

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']

Редактируем файл views.py приложения orders:

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'])

# Очищаем корзину.

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})

Создаем новый файл в директории приложения orders и назовем его urls.py

from django.urls import path

from . import views

app\_name = 'orders'

urlpatterns = [

path('create/', views.order\_create, name='order\_create'),

]

Это шаблон URL’а для обработчика order\_create. Нужно добавить его в общий файл urls.py проекта shop

path('orders/', include('orders.urls', namespace='orders'))

Редактируем шаблон detail.html приложения cart:

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

Checkout

</a

Создаем структуру папок в каталоге приложения orders:

Templates/

Orders/

Order/

Create.html

Created.html

Откройте файл orders/order/create.html и разместим в нем следующий код:

{% 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 %}

Отредактиктируем файл created.html:  
{% extends "shop/base.html" %}

{% block title %}

Thank you

{% endblock %}

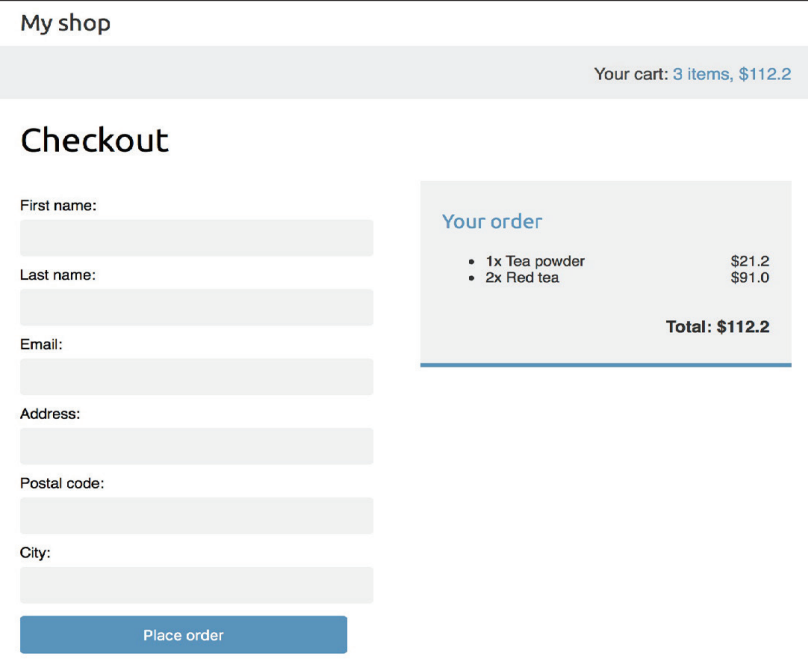
{% block content %}

<h1>Thank you</h1>

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

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

{% endblock %}



Если заполнить форму и нажать кнопку Place order, то заказ будет сохранен в базу данных, а покупатель увидит страницу об успешном создании закааза.

**Задание на защиту: Добавляем черный чай**.

Редактируем файл views.py приложения cart:

def cart\_detail(request):

cart = Cart(request)

for item in cart:

item['update\_quantity\_form'] = CartAddProductForm(

initial={'quantity': item['quantity'],

'update': True})

return render(request, 'cart/detail.html', {'cart': cart})

**Редактируем шаблон detail.html**

<p class="price">${{ product.price }}</p>

<form action="{% url "cart:cart\_add" product.id %}" method="post">

{{ cart\_product\_form }}

{% csrf\_token %}

<input type="submit" value="Add to cart">

</form>

{{ product.description|linebreaks }}

