Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Технический Университет Связи и Информатики»

(МТУСИ)

КУРСОВАЯ РАБОТА

по дисциплине

Технологии и методы программирования

на тему

Создание проекта Интернет-Магазина

Выполнил: студент группы БИБ2101 факультета КиИБ Щипакин М.А.

Проверила: Алексанян Д.А.

Москва, 2022г.

Содержание

Задание для курсовой работы…………………………………………………….3

Код курсовой работы……………………………………………………………..4

Ход работы……………………………………………………………………….27

Вывод……………………………………………………………………………..34

Задание для курсовой работы

Вам необходимо создать интернет-магазин для продажи любых товаров (отличных от образца). Задание включает в себя как обязательную часть, так и дополнительное задание.

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.

Код курсовой работы

Иерархия проекта (рис. 1, рис. 2, рис. 3, рис. 4 и рис. 5):

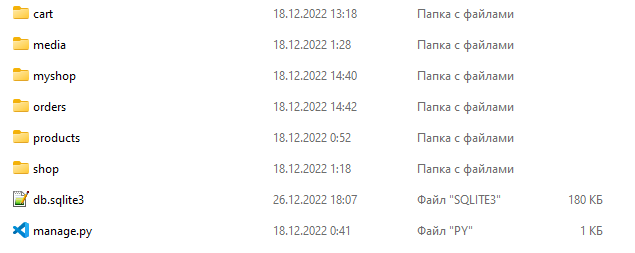


Рис. 1 – Директория myshop

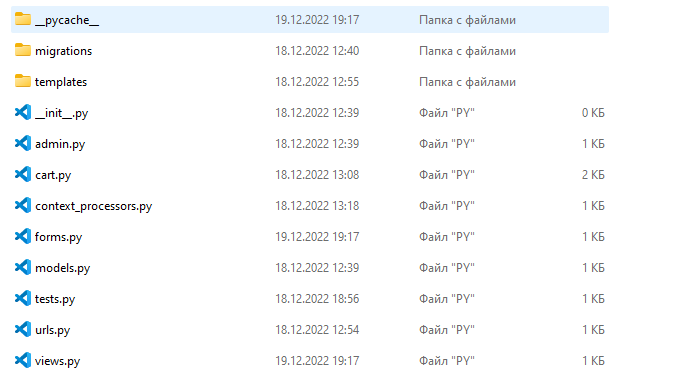


Рис. 2 – Директория myshop/cart

Код файла cart.py:

from decimal import Decimal

from django.conf import settings

from shop.models import Product

class Cart(object):

    def \_\_init\_\_(self, request):

        self.session = request.session

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

        if not cart:

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

        self.cart = cart

    def add(self, product, quantity=1, update\_quantity=False):

        product\_id = str(product.id)

        if product\_id not in self.cart:

            self.cart[product\_id] = {'quantity': 0,'price': str(product.price)}

        if update\_quantity:

            self.cart[product\_id]['quantity'] = quantity

        else:

            self.cart[product\_id]['quantity'] += quantity

        self.save()

    def save(self):

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

        self.session.modified = True

    def remove(self, product):

        product\_id = str(product.id)

        if product\_id in self.cart:

            del self.cart[product\_id]

            self.save()

    def \_\_iter\_\_(self):

        product\_ids = self.cart.keys()

 # получение объектов product и добавление их в корзину

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

        for product in products:

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

        for item in self.cart.values():

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

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

            yield item

    def \_\_len\_\_(self):

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

    def get\_total\_price(self):

        return sum(Decimal(item['price']) \* item['quantity'] for item in self.cart.values())

    def clear(self):

 # удаление корзины из сессии

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

        self.session.modified = True

Код файла context-processors:

from .cart import Cart

def cart(request):

    return {'cart': Cart(request)}

Код файла forms.py

from django import forms

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

class CartAddProductForm(forms.Form):

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

    update = forms.BooleanField(required=False, initial=False,widget=forms.HiddenInput)

Код файла urls.py:

from django.conf.urls import url

from . import views

app\_name='cart'

urlpatterns = [

    url(r'^$', views.cart\_detail, name='cart\_detail'),

    url(r'^add/(?P<product\_id>\d+)/$', views.cart\_add, name='cart\_add'),

    url(r'^remove/(?P<product\_id>\d+)/$', views.cart\_remove, name='cart\_remove'),

]

Код файла views.py:

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

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

from shop.models import Product

from .forms import CartAddProductForm

from .cart import Cart

# Create your views here.

@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['Количество'],update\_quantity=cd['update'])

    return redirect('cart:cart\_detail')

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')

def cart\_detail(request):

    cart = Cart(request)

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

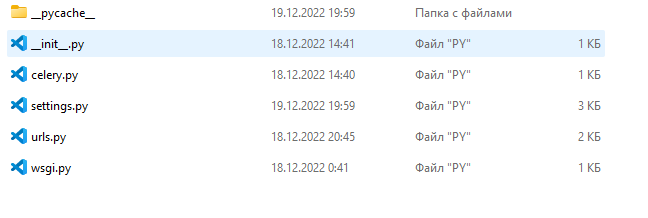


Рис. 3 – Директория myshop/myshop

Код файла setting.py:

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

import os

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/1.8/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = ')#9#1fg9m=y^7)fb2ad%#1frjgbz-#7fwup6t0jax+^e6lrw^)'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = (

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'shop',

    'cart',

    'orders',

)

MIDDLEWARE\_CLASSES = (

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'django.middleware.security.SecurityMiddleware',

)

ROOT\_URLCONF = 'myshop.urls'

TEMPLATES = [

    {

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

        'DIRS': [],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

                'cart.context\_processors.cart',

            ],

        },

    },

]

WSGI\_APPLICATION = 'myshop.wsgi.application'

# Database

# https://docs.djangoproject.com/en/1.8/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    }

}

# Internationalization

# https://docs.djangoproject.com/en/1.8/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'Europe/Moscow'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

EMAIL\_BACKEND = 'django.core.mail.backends.console.EmailBackend'

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/1.8/howto/static-files/

STATIC\_URL = '/static/'

MEDIA\_URL = '/media/'

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

CART\_SESSION\_ID = 'cart'

Код файла urls.py:

from django.conf.urls import include, url

from django.contrib import admin

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

    url(r'^admin/', include(admin.site.urls)),

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

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

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

]

if settings.DEBUG:

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

Код файла wsgi.py:

import os

from django.core.wsgi import get\_wsgi\_application

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "myshop.settings")

application = get\_wsgi\_application()

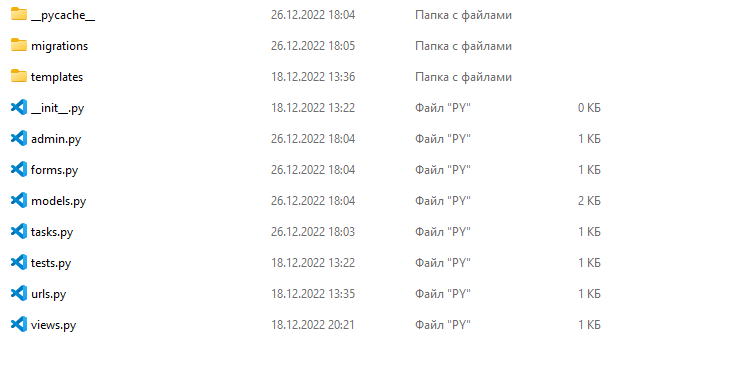


Рис. 4 – Директория myshop/orders

Код файла admin.py:

from django.contrib import admin

from .models import Order, OrderItem

# Register your models here.

class OrderItemInline(admin.TabularInline):

    model = OrderItem

    raw\_id\_fields = ['product']

class OrderAdmin(admin.ModelAdmin):

    list\_display = ['id', 'Имя', 'Фамилия', 'Почта','Адрес', 'Почтовый\_индекс', 'Город', 'paid','created', 'updated']

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

    inlines = [OrderItemInline]

admin.site.register(Order, OrderAdmin)

Код файла forms.py:

from django import forms

from .models import Order

class OrderCreateForm(forms.ModelForm):

    class Meta:

        model = Order

        fields = ['Имя', 'Фамилия', 'Почта', 'Адрес', 'Почтовый\_индекс','Город']

Код файла models.py:

from django.db import models

from shop.models import Product

# Create your models here.

class Order(models.Model):

    Имя = models.CharField(max\_length=50)

    Фамилия = models.CharField(max\_length=50)

    Почта = models.EmailField()

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

    Почтовый\_индекс = models.CharField(max\_length=20)

    Город = 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

Код файла urls.py

from django.conf.urls import url

from . import views

urlpatterns = [

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

]

]

Код файла views.py:

from django.shortcuts import render

from .models import OrderItem

from .forms import OrderCreateForm

from cart.cart import Cart

from .tasks import order\_created

# Create your views here.

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'])

            cart.clear()

            # очистка корзины

            order\_created.delay(order.id)

            return render(request, 'orders/order/created.html',{'order': order})

    else:

        form = OrderCreateForm

    return render(request, 'orders/order/create.html',{'cart': cart, 'form': form})



Рис. 5 – Директория myshop/shop

Код файла admin.py:

from django.contrib import admin

from .models import Category, Product

# Register your models here.

class CategoryAdmin(admin.ModelAdmin):

    list\_display = ['name', 'slug']

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

admin.site.register(Category, CategoryAdmin)

class ProductAdmin(admin.ModelAdmin):

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

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

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

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

admin.site.register(Product, ProductAdmin)

Код файла models.py:

from django.db import models

from django.core.urlresolvers import reverse

# Create your models here.

class Category(models.Model):

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

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

    class Meta:

        ordering = ('name',)

        verbose\_name = 'Категория'

        verbose\_name\_plural = 'Категории'

    def \_\_str\_\_(self):

        return self.name

    def get\_absolute\_url(self):

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

class Product(models.Model):

    category = models.ForeignKey(Category, related\_name='products')

    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)

    stock = models.PositiveIntegerField()

    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

    def get\_absolute\_url(self):

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

Код файла urls.py:

from django.conf.urls import url

from . import views

app\_name='shop'

urlpatterns = [

    url(r'^$', views.product\_list, name='product\_list'),

    url(r'^(?P<category\_slug>[-\w]+)/$',views.product\_list,name='product\_list\_by\_category'),

    url(r'^(?P<id>\d+)/(?P<slug>[-\w]+)/$',views.product\_detail,name='product\_detail'),

    url(r'^/rules',views.product\_carrying\_rules, name='product\_carrying\_rules'),

]

Код файла views.py:

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

from .models import Category, Product

from cart.forms import CartAddProductForm

# Create your views here.

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})

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})

def product\_carrying\_rules(request):

    return render(request,'shop/product/rules.html')

Ход работы:

Сначала создадим для нашего проекта виртуальную среду с помощью следующих команд:

mkdir env – создание директории под названием env

virtualenv env/myshop – создания виртуального окружения в директории env/myshop

source env/myshop/bin/activate – активирование виртуальной среды

В данном проекте я использовал Python версии 3.5.4 и Django 1.8.6.

Устанавливаем в эту виртуальную среду Django и начинаем новый проект myshop с приложением, называемым shop, с помощью команд:

django-admin startproject myshop

cd myshop/

django-admin startapp shop

Чтобы подключить приложение shop к проекту myshop, необходимо в файл settings.py добавить название приложения в INSTALLED\_APPS(рис 6)

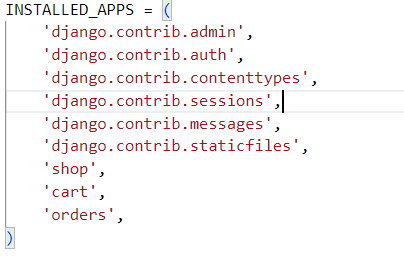


Рис. 6 – Подключение приложений к проекту myshop

Создание модели каталога товаров

Теперь создадим моделей каталогов продуктов. В папке shop в файл models.py добавить следующий код:

from django.db import models

class Category(models.Model):

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

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

class Meta:

ordering = ('name',)

verbose\_name = 'Категория'

verbose\_name\_plural = 'Категории'

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

return self.name

class Product(models.Model):

category = models.ForeignKey(Category, related\_name='products')

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)

stock = models.PositiveIntegerField()

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

В начале создаем класс Category. Категория имеет всего два поля: name – название товара и slug - Алиас категории(её URL). Db\_index = True – для этого поля будет создан индекс в базе данных. Ускоряет поиск нужных элементов в бд. Также класс Category содержит внутренний класс Meta. Если нам нужно добавить данные о самой модели, мы используем класс Meta. В классе Meta используются следующие опции:

Ordering - Эта опция принимает список строковых значений, которые являются полями модели. Он используется для определения порядка объектов модели. Когда объекты этой модели будут извлечены, они будут представлены в этом порядке.

Verbose\_name - Эта опция используется для определения удобочитаемого единственного имени модели и заменяет стандартное соглашение об именах Django. Это имя также отобразится в панели администратора (/admin/).

Verbose\_name\_plural - Этот параметр используется для определения удобочитаемого имени во множественном числе для модели, которое снова перезапишет стандартное соглашение об именах Django. Это имя также отобразится в панели администратора (/admin/).

Также есть метод в классе Category. стандартный метод класса для Python \_\_str \_\_ (), чтобы вернуть удобочитаемую строку для каждого объекта.

Следующий класс, который мы создаем, это класс Product. Product содержит следующие поля:

category : Это ForeignKey модели Category. Это отношение "многие к одному": продукт относится к одной категории, а категория содержит несколько продуктов. Атрибут related\_name задает имя обратного отношения от модели Category обратно к модели Product. Из одного товара с помощью related\_name, можно запросить связанные сущности (все товары данной категории).

name : Название продукта. – максимальная длинна поля 200 символов.

slug : Алиас продукта(его URL). – максимальная длинна поля 200 символов.

image : Изображение продукта. Изображения загружаются в папку products/(Текущая дата). Blank – True – поле может быть пустым. То есть товар без изображения

description : Необязательное описание для продукта.

price : Это поле DecimalField. В нем используется десятичное число Python. Десятичный тип для хранения десятичного числа с фиксированной точностью. Максимальное число цифр (включая десятичные разряды) задается с помощью атрибута max\_digits(максимальное кол-во цифр) и десятичных знаков с атрибутом decimal\_places(знаков после запятой)

stock : Это поле PositiveIntegerField для хранения остатков данного продукта.

available : Это булево значение, указывающее, доступен ли продукт или нет. Позволяет включить/отключить продукт в каталоге. По умолчанию True, то есть товар доступен

created : Это поле хранит дату когда был создан объект.

updated : В этом поле хранится время последнего обновления объекта.

Также класс Product содержит внутренний класс Meta. Если нам нужно добавить данные о самой модели, мы используем класс Meta. В классе Meta используются следующие опции:

Ordering - Эта опция принимает список строковых значений, которые являются полями модели. Он используется для определения порядка объектов модели. Когда объекты этой модели будут извлечены, они будут представлены в этом порядке.

Index\_together - Наборы имен полей, которые, взятые вместе, индексируются. Этот список полей будет проиндексирован вместе.

Также есть метод в классе Product. стандартный метод класса для Python \_\_str \_\_ (), чтобы вернуть удобочитаемую строку для каждого объекта.

Поскольку мы собираемся использовать изображения в наших моделях, откроем терминал и установим Pillow. -pip install Pillow

Регистрация моделей catalog на сайте администрирования

Давайте добавим наши модели на сайт администрирования, чтобы мы могли легко управлять категориями и продуктами. Измените файл admin.py приложения shop и добавьте в него следующий код (рис. 7):



Рис. 7 – Код файла admin.py

Создаем класс CategoryAdmin, в котором есть два поля. List\_display показывает какие поля будут показываться в админке, при создании категории. Мы используем атрибут prepopulated\_fields, чтобы указать поля, в которых значение автоматически задается с использованием значения других полей.

Затем добавляем данный класс в поле созданий и изменений категорий.

Аналогично с классом ProductAdmin, и также добавляем в админку нашего сайта.

Теперь панель администратора выглядит следующим образов (рис. 8):

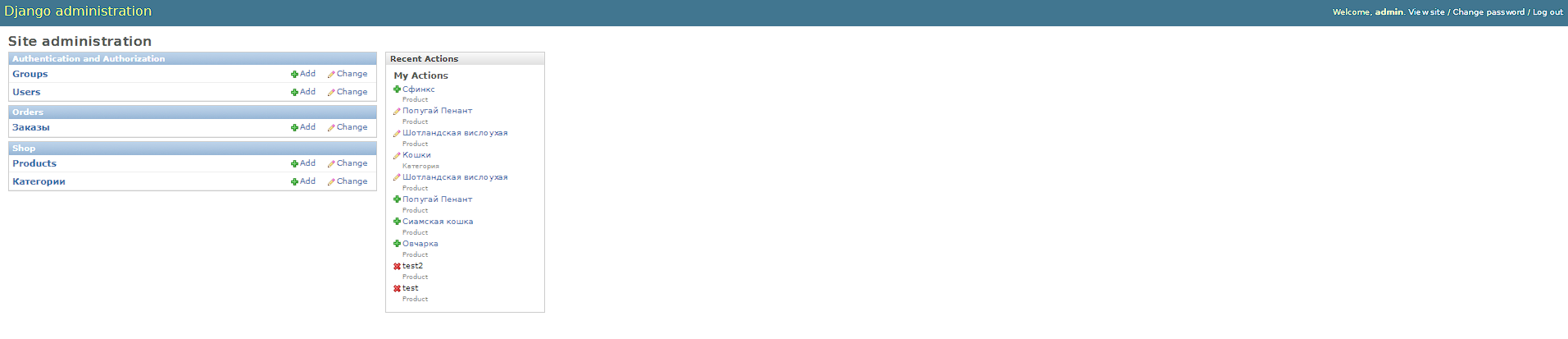


Рис. 8 – Панель администрирования

Добавим несколько категорий и различных товаров на наш сайт. Теперь сайт выглядит следующим образом (рис. 9):

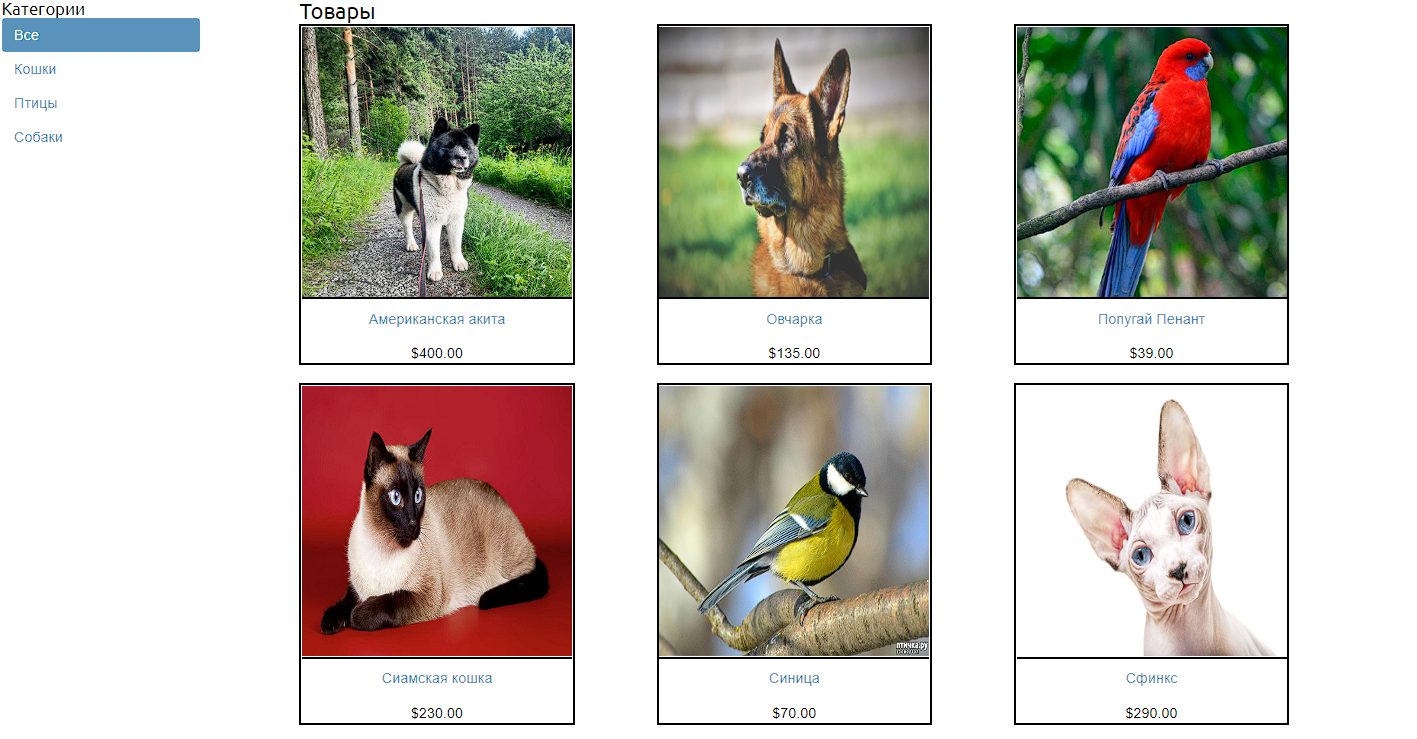


Рис. 9 – Отображение товаров на сайте

Создание корзины

После создания каталога продуктов следующим шагом является создание корзины покупок, которая позволит пользователям выбирать продукты, которые они хотят приобрести. Корзина позволяет пользователям выбирать нужные продукты и временно хранить их во время просмотра сайта до тех пор, пока не будет размещен заказ. Корзина должна быть сохранена в сессии, чтобы элементы корзины хранились во время визита пользователя.

Создайте новый файл в каталоге приложения cart и назовите его cart.py. Добавьте в него следующий код :

from decimal import Decimal

from django.conf import settings

from shop.models import Product

class Cart(object):

    def \_\_init\_\_(self, request):

        self.session = request.session

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

        if not cart:

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

        self.cart = cart

    def add(self, product, quantity=1, update\_quantity=False):

        product\_id = str(product.id)

        if product\_id not in self.cart:

            self.cart[product\_id] = {'quantity': 0,'price': str(product.price)}

        if update\_quantity:

            self.cart[product\_id]['quantity'] = quantity

        else:

            self.cart[product\_id]['quantity'] += quantity

        self.save()

    def save(self):

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

        self.session.modified = True

    def remove(self, product):

        product\_id = str(product.id)

        if product\_id in self.cart:

            del self.cart[product\_id]

            self.save()

    def \_\_iter\_\_(self):

        product\_ids = self.cart.keys()

 # получение объектов product и добавление их в корзину

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

        for product in products:

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

        for item in self.cart.values():

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

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

            yield item

    def \_\_len\_\_(self):

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

    def get\_total\_price(self):

        return sum(Decimal(item['price']) \* item['quantity'] for item in self.cart.values())

    def clear(self):

 # удаление корзины из сессии

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

        self.session.modified = True

Создаем необходимые функции для работы корзины: добавить товар в корзину, обновить состояние корзины, удаление товара из корзины, перебор корзины и получение товаров из базы данных, подсчет количества товаров в корзине, подсчет общей суммы товаров в корзине и удаление корзины из сессии.

Мы храним текущую сессию с помощью self.session = request.session, чтобы сделать его доступным для других методов класса Cart. Во-первых, мы пытаемся получить корзину с текущей сессии с помощью self.session.get(settings.CART\_SESSION\_ID). Если в сессии отсутствует корзина, то мы создадим сессию с пустой корзиной, установив пустой словарь в сессии. Мы ожидаем, что наш словарь корзины будет использовать коды продуктов в качестве ключей и словарь с количеством и ценой в качестве значения для каждого ключа. Таким образом, мы можем гарантировать, что продукт не будет добавлен в корзину более одного раза; можно также упростить доступ к данным элементов корзины.

Метод save() сохраняет все изменения в корзине в сессии и помечает сессию как modified с помощью session.modified = True. Это говорит о том, что сессия modified и должна быть сохранена.

методе \_\_iter\_\_() мы извлекаем экземпляры продукта, присутствующие в корзине, чтобы включить их в номенклатуры корзины. Наконец, мы проходим по элементам корзины, преобразуя цену номенклатуры обратно в десятичное число и добавляя атрибут total\_price к каждому элементу. Теперь можно легко выполнить итерацию по товарам в корзине.

Мы собираемся определить пользовательский метод \_\_len\_\_(), чтобы вернуть общее количество элементов, хранящихся в корзине.

Теперь создадим представления корзины. Создадим файл forms.py в каталоге приложения cart и добавим в него следующий код :

from django import forms

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

class CartAddProductForm(forms.Form):

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

    update = forms.BooleanField(required=False, initial=False,widget=forms.HiddenInput)

Эта форма будет использоваться для добавления продуктов в корзину. Класс CartAddProductForm содержит следующие поля:

quantity : позволяет пользователю выбрать количество между 1-20. Мы используем поле TypedChoiceField с coerce=int для преобразования ввода в целое число.

update : позволяет указать, следует ли добавлять сумму к любому существующему значению в корзине для данного продукта (False) или если существующее значение должно быть обновлено с заданным значением (True).

Отредактируем файл views.py приложения cart и добавьте в него следующий код (рис. 10):



Рис. 10 – Код файла views.py

Это представление для добавления продуктов в корзину или обновления количества для существующих товаров.

Мы используем декоратор require\_POST, чтобы разрешить только POST запросы, поскольку это представление изменит данные. Представление получает ID продукта в качестве параметра. Мы извлекаем экземпляр продукта с заданным ID и проверяем CartAddProductForm. Если форма валидна, мы либо добавляем, либо обновляем продукт в корзине. Представление перенаправляет по URLадресу cart\_detail, который будет отображать содержимое корзины. Мы собираемся создать cart\_detail представление в ближайшее время.

Представление cart\_remove получает id продукта в качестве параметра. Мы извлекаем экземпляр продукта с заданным id и удаляем продукт из корзины. Затем мы перенаправляем пользователя на URL-адрес cart\_detail.

Теперь добавляем шаблон для отображения и работы корзины на сайте в html и проверяем результат (рис. 11).

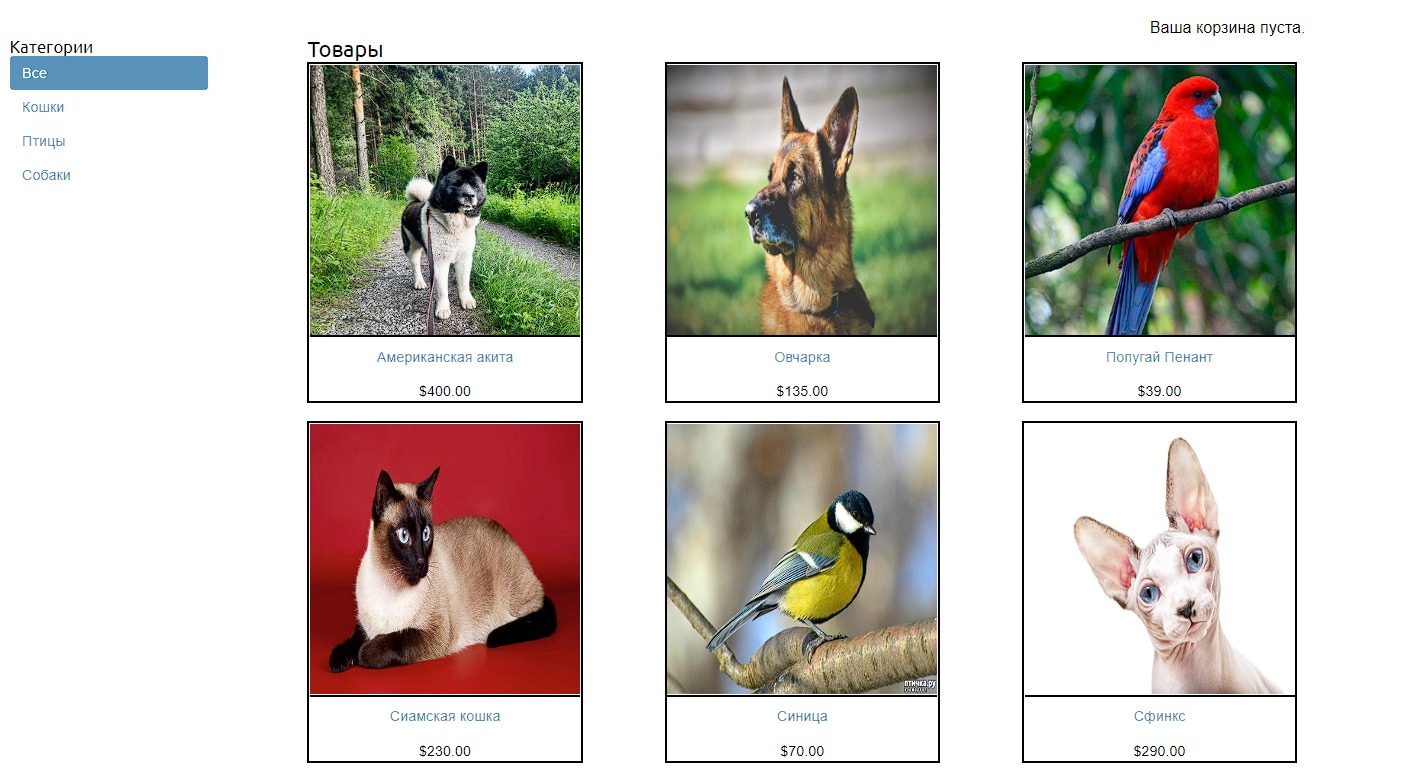


Рис. 11 – отображение сайта с корзиной товаров

Регистрация заказов

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 (рис. 12):



Рис. 12 – Код файла models.py

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.

Включение моделей заказов на сайте администрирования. Добавим модели заказов на сайт администрирования. Измените файл admin.py приложения orders (рис. 13):

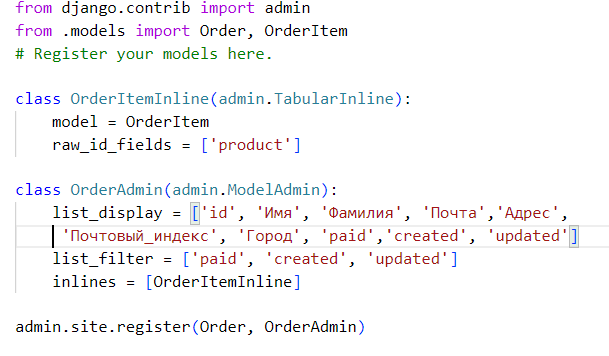


Рис. 13 – Код файла admin.py в директории orders

Создаем класс OrderAdmin, поля которого будут отображаться при оформлении заказа в панели администратора (рис. 14):

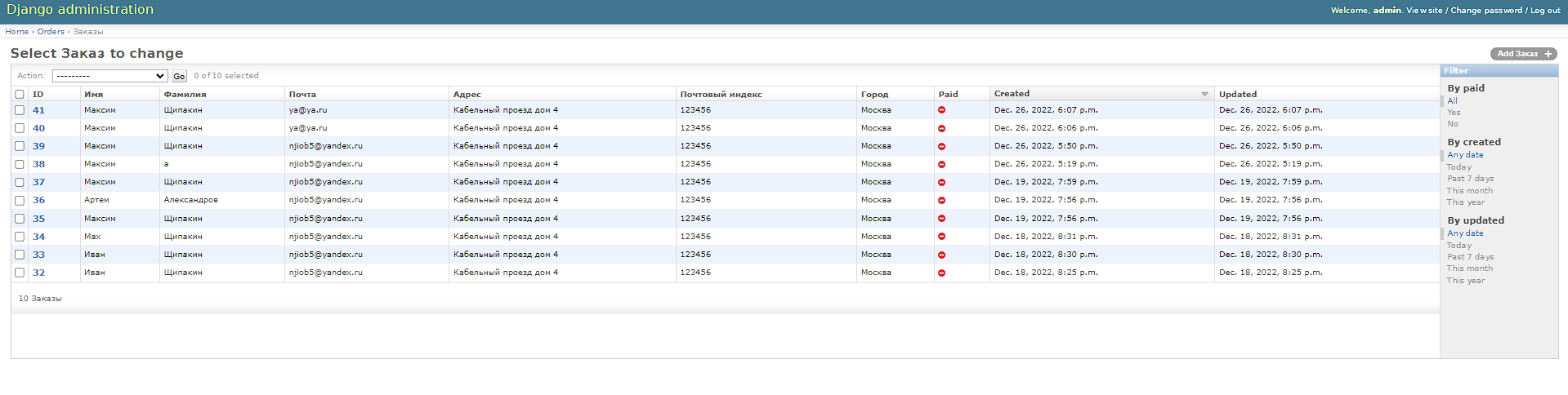


Рис. 14 Оформление заказа в панели администратора

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 = ['Имя', 'Фамилия', 'Почта', 'Адрес', 'Почтовый\_индекс','Город']

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

from .tasks import order\_created

# Create your views here.

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'])

            cart.clear()

            # очистка корзины

            order\_created.delay(order.id)

            return render(request, 'orders/order/created.html',{'order': order})

    else:

        form = OrderCreateForm

    return render(request, 'orders/order/create.html',{'cart': cart, 'form': form})

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.

Добавляем шаблон html для отображения страницы заказа и проверяем работоспособность (рис. 15 и рис. 16):

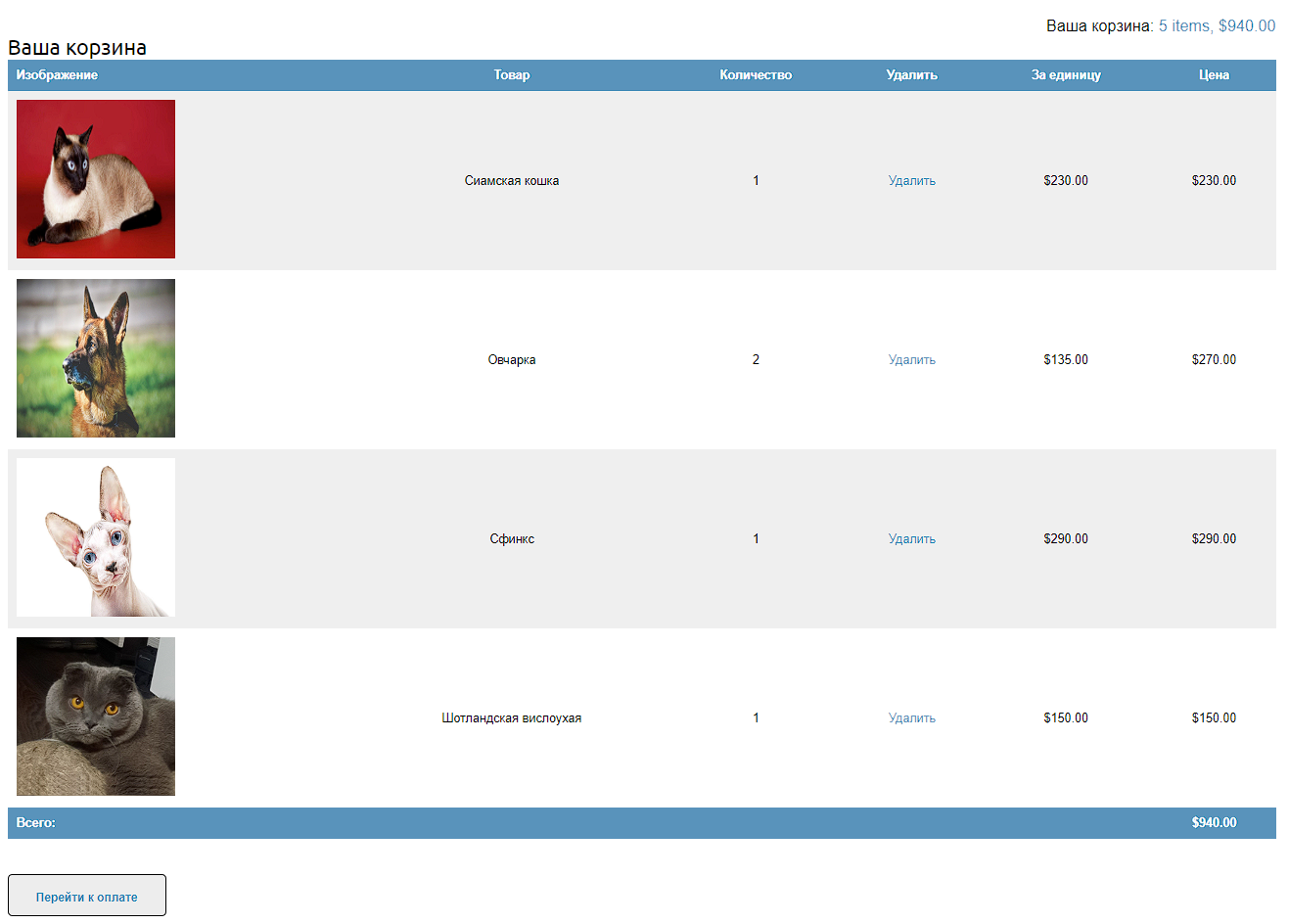


Рис. 20 – Наполнение корзины

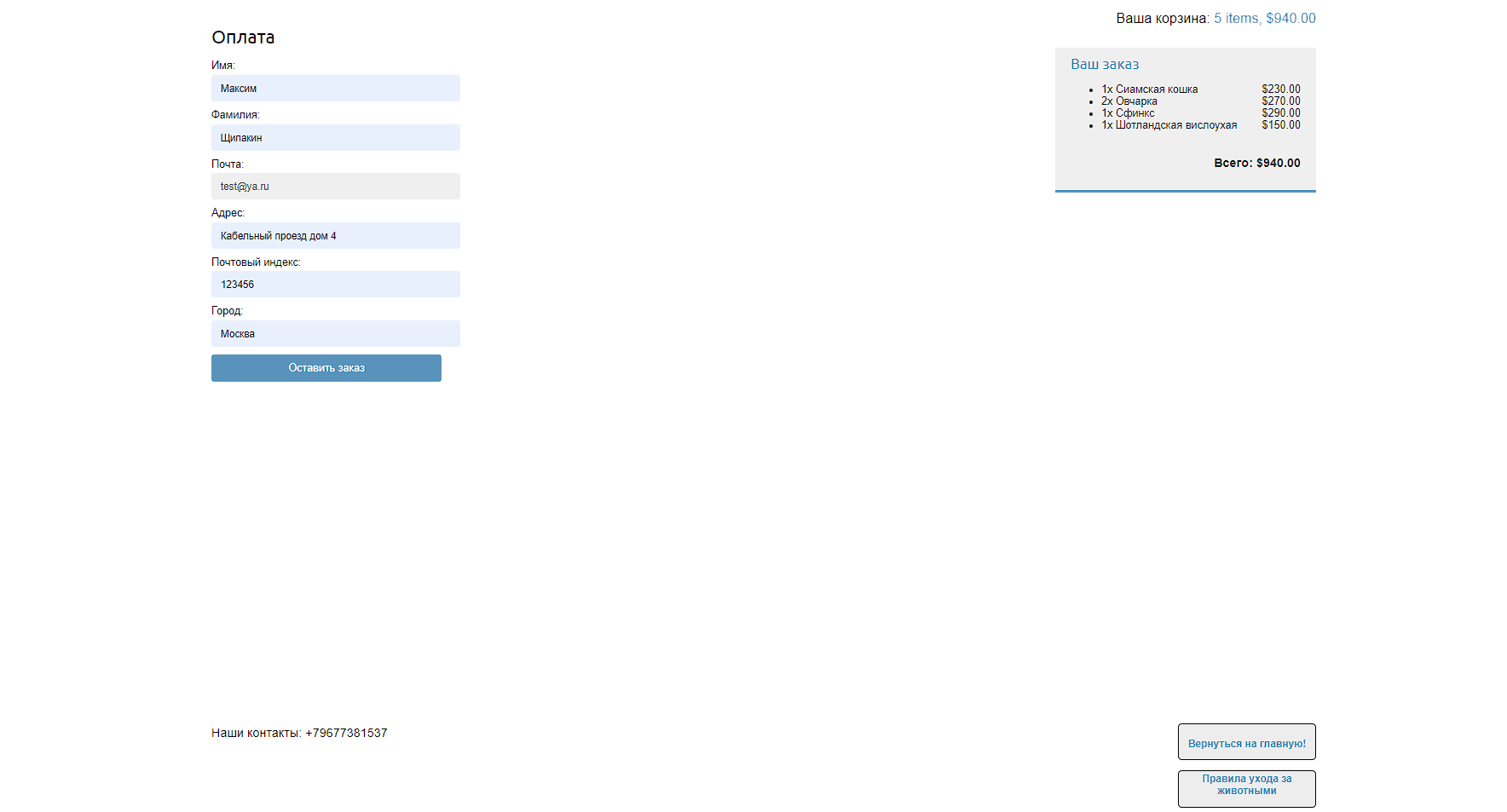


Рис. 21 – Оформление заказа

Celery — это распределенная очередь задач, которая может обрабатывать большие объемы сообщений. Она выполняет обработку в реальном времени, но также поддерживает планирование задач. С помощью Celery можно не только легко создавать асинхронные задачи, но и выполнять их как можно быстрее, но можно также планировать их запуск в определенное время. Пример работы Celery в проекте (рис. 22 и рис. 23):

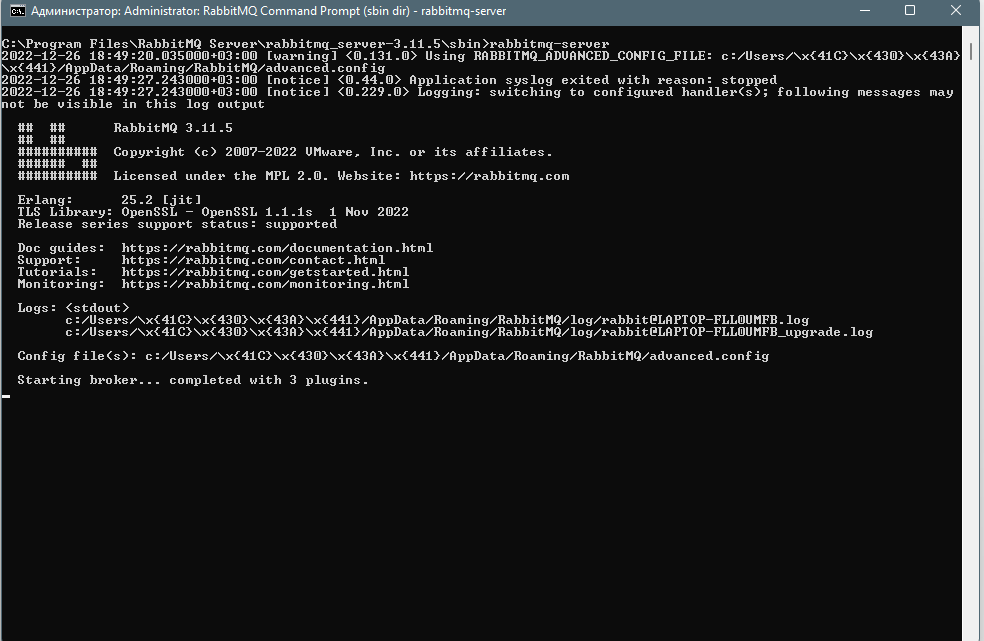


Рис. 22 – Запуск сервера rabbitmq

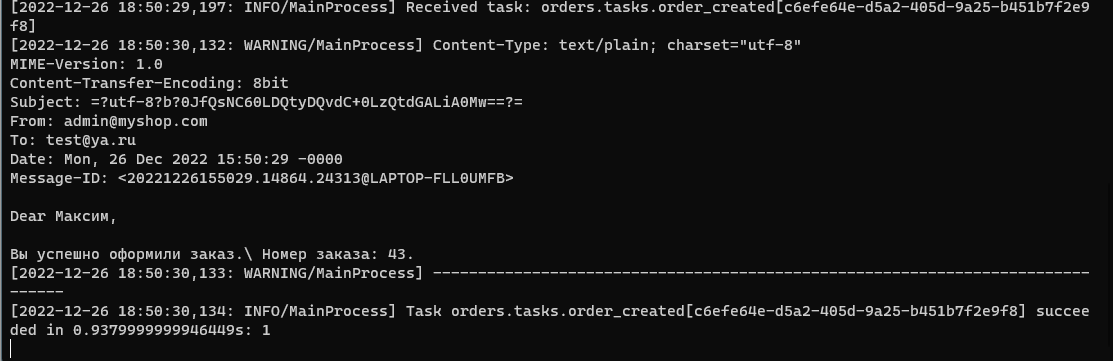


Рис. 23 – Отправка подтверждения заказа на почту с помощью celery worker

Вывод

Мне удалось создать интернет-магазин для продажи любых товаров. Удалось создать модель каталога продуктов, добавление её на сайт администрирования и создание основных представлений для отображения каталога.

Удалось создать систему корзины для покупок с помощью сессий Джанго, чтобы пользователи могли сохранять выбранные продукты при просмотре сайта.

Удалось создать форму и функциональные возможности для размещения заказов.

Отправку по электронной почте подтверждения пользователям при размещении заказа.

Полезные ссылки:

<https://github.com/Gl0b0x?tab=repositories> – ссылка на GitHub