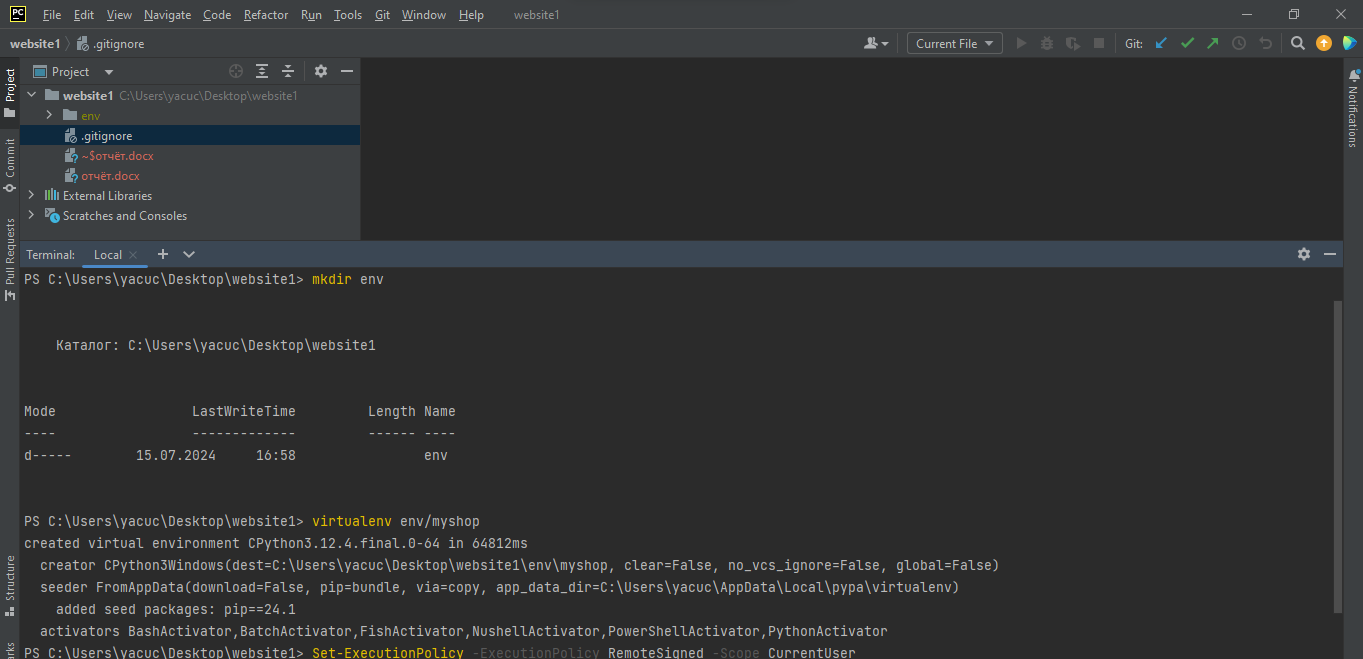
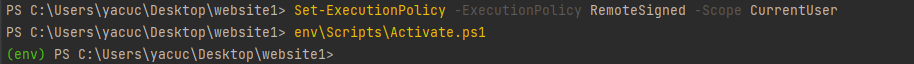
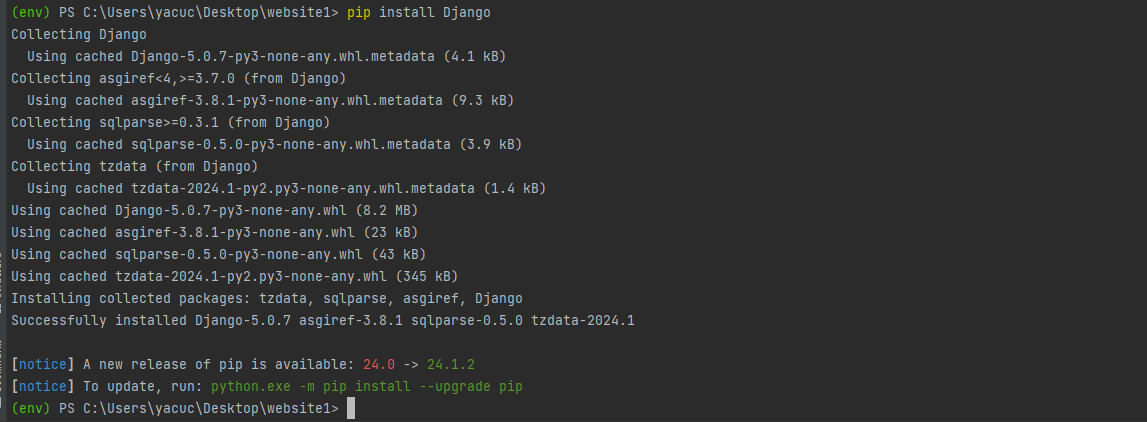
Создаём виртуальную среду для нового проекта и активируем её следующим образом:

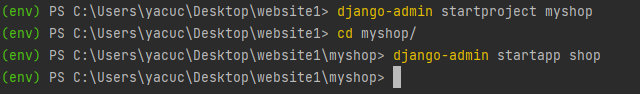




Установим Django в виртуальной среде:



Начнём новый проект под названием myshop с приложением, называемым shop:

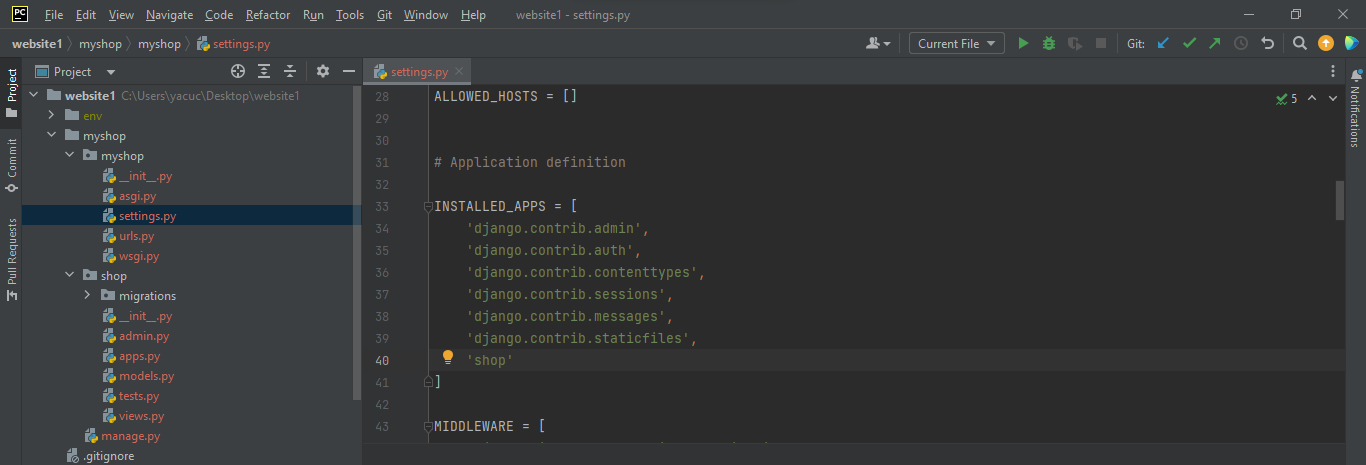


Изменим файл settings.py проекта и добавим приложение к настройкам INSTALLED\_APPS: INSTALLED\_APPS =(

# ...

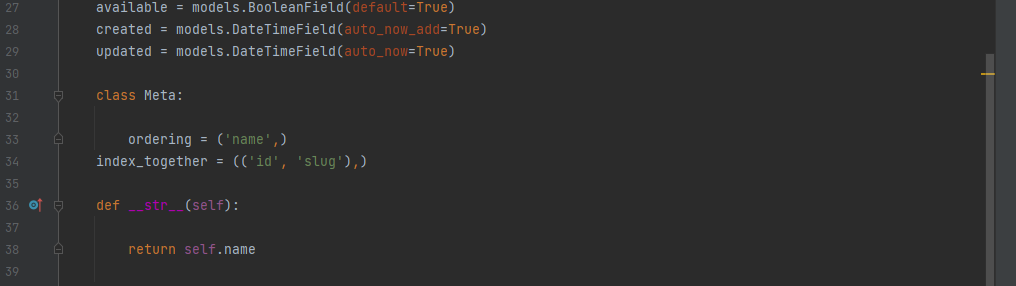
'shop',

)

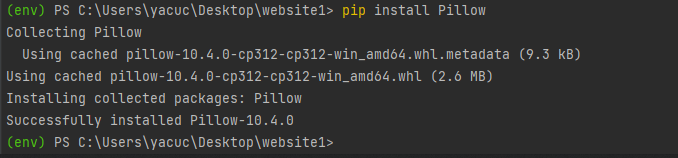


Теперь приложение активно для данного проекта. Определим модели для каталога продуктов.

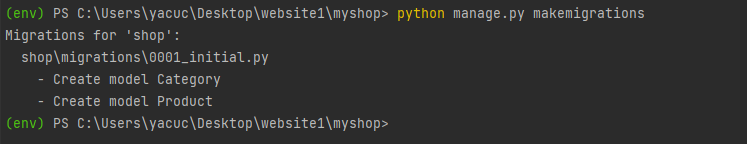
Каталог нашего магазина будет состоять из продуктов, сгруппированных по разным категориям. Каждый продукт будет иметь имя, необязательное описание, необязательное изображение, цену и доступный запас. Отредактируем файл models.py только что созданного приложения shop:

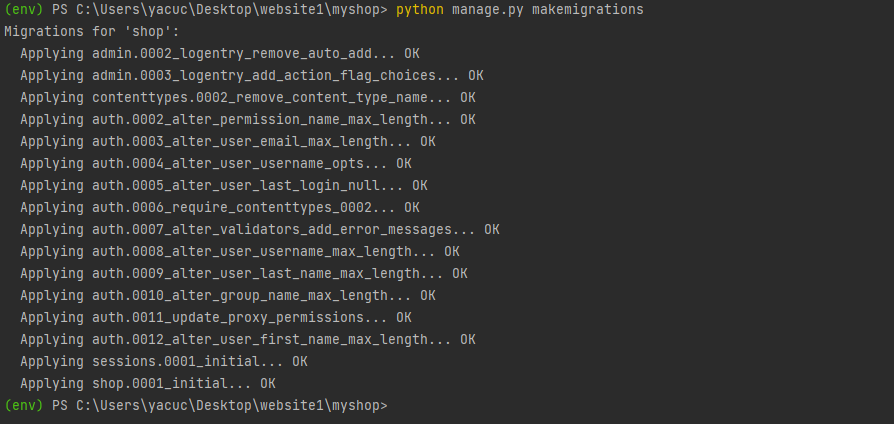
Поскольку мы собираемся использовать изображения в наших моделях, установим Pillow:



Теперь выполним следующую команду, чтобы создать начальные миграции для проекта:

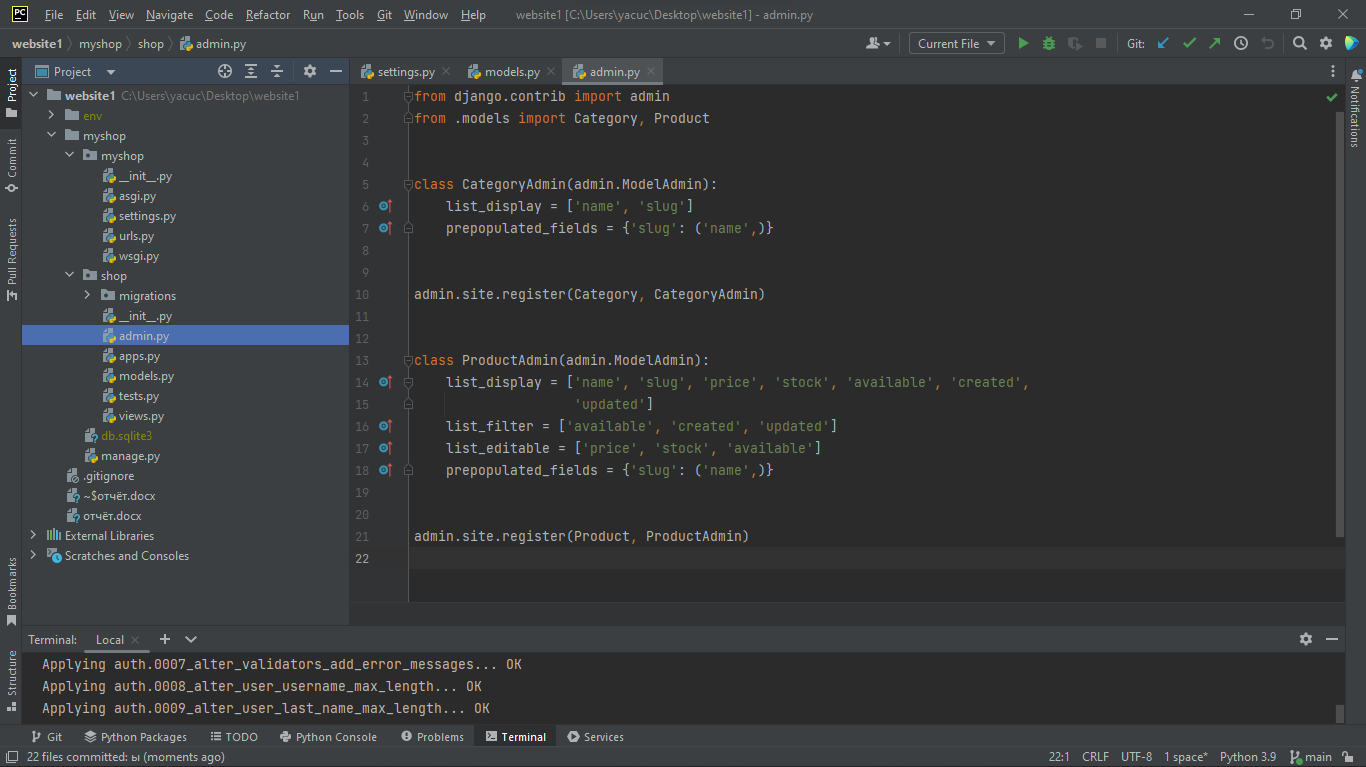


Выполним следующую команду для синхронизации базы данных:

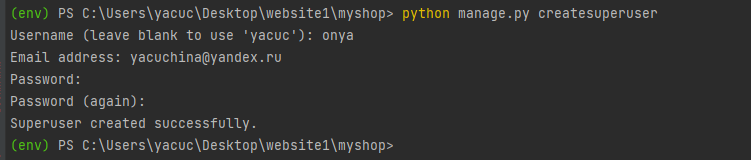


Теперь база данных синхронизирована с моделями.

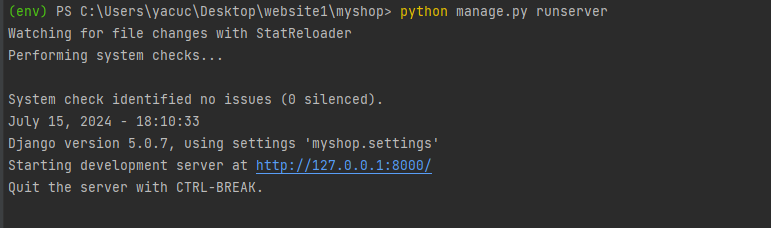
Добавим наши модели на сайт администрирования, чтобы мы могли легко управлять категориями и продуктами. Изменим файл admin.py приложения shop:

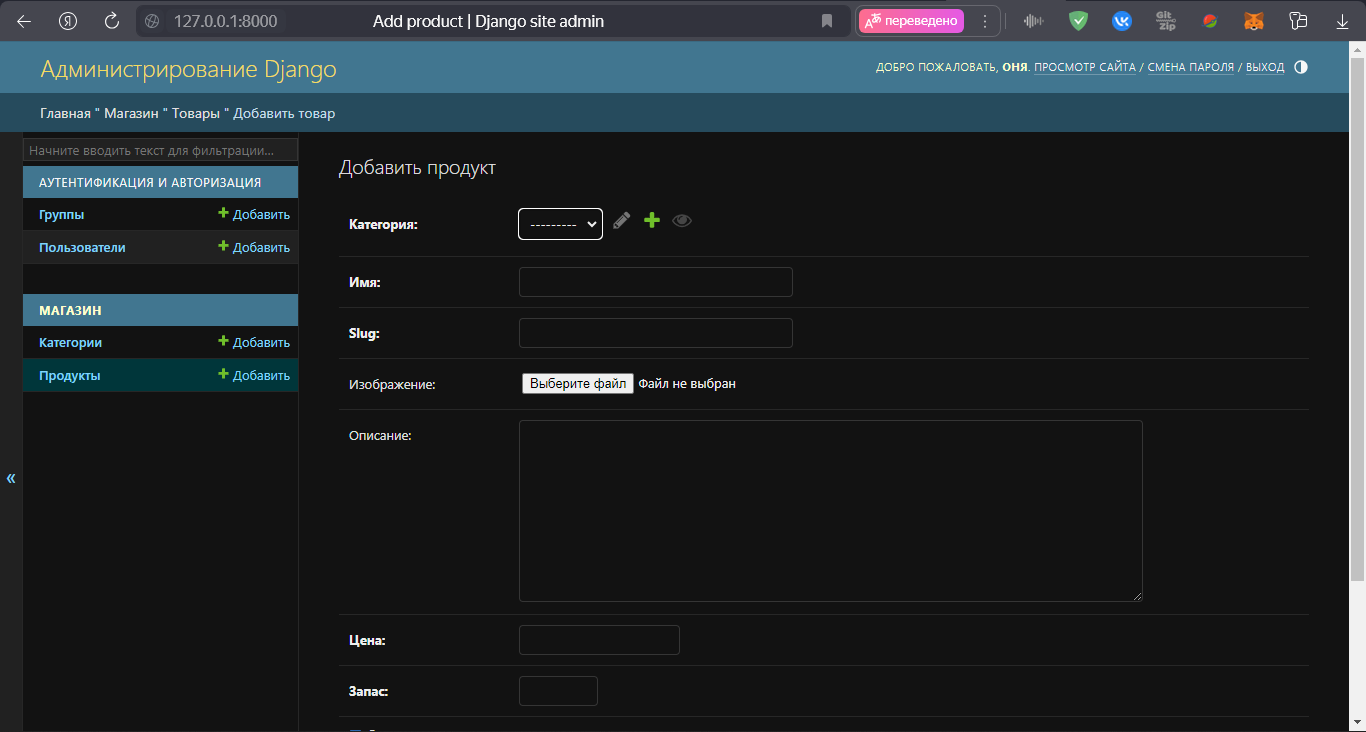


Теперь создадим суперпользователя:

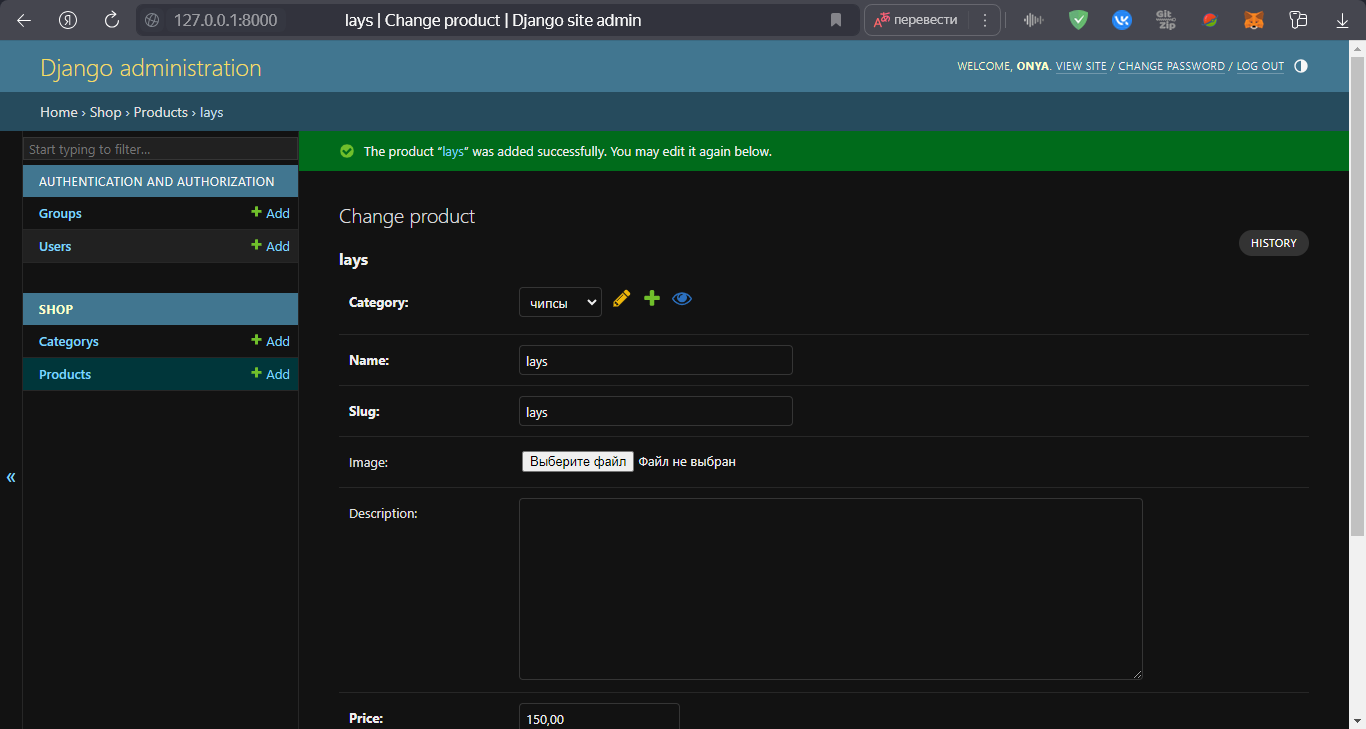
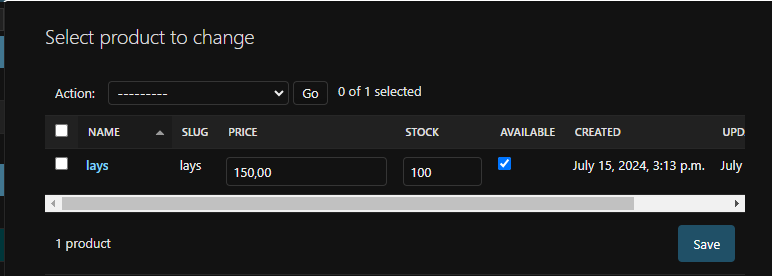


Запустим сервер разработки:





Добавим новую категорию и продукт с помощью интерфейса администрирования:

Теперь необходимо создать шаблоны для списка товаров и одного товара. Создадим следующую структуру файлов в каталоге приложения shop:

templates/

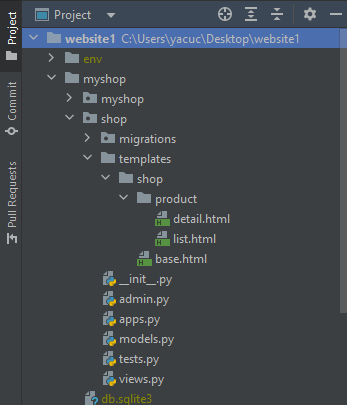
shop/

base.html

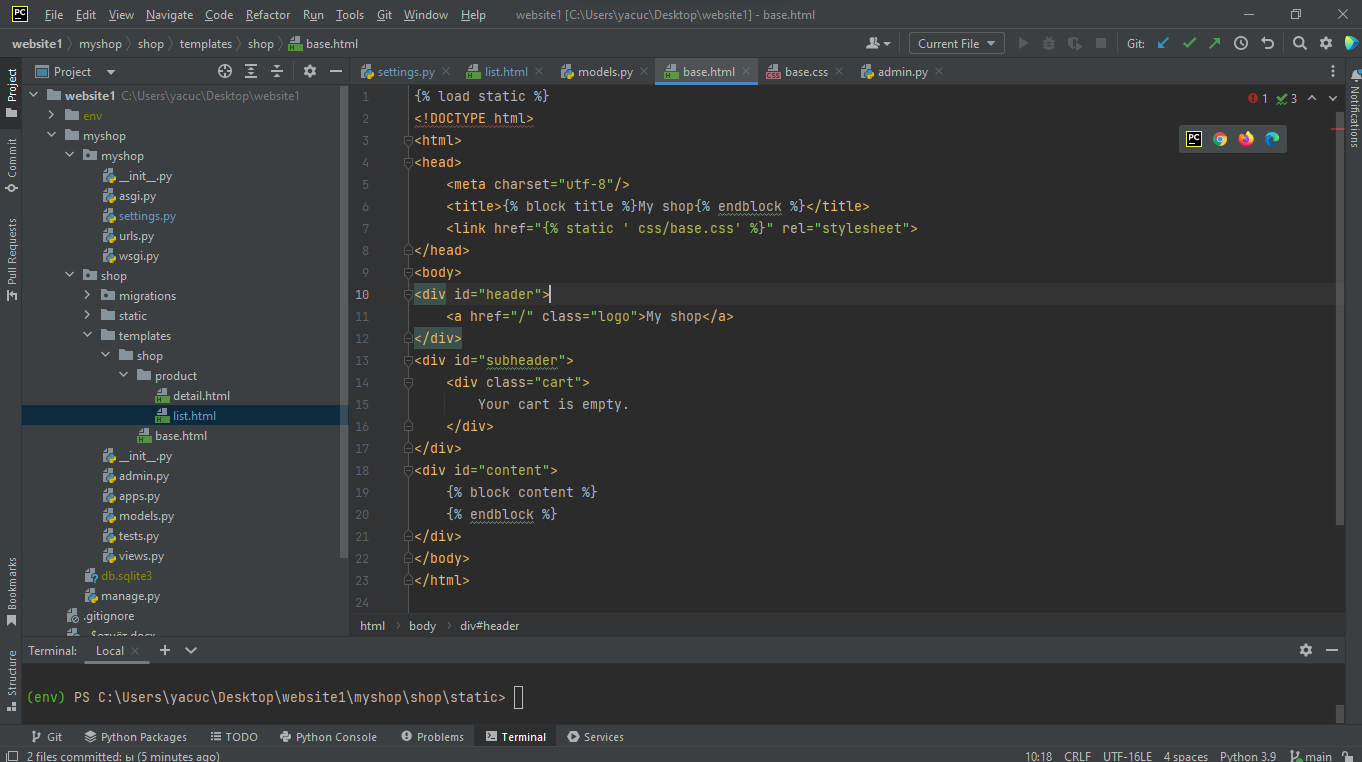
product/

list.html

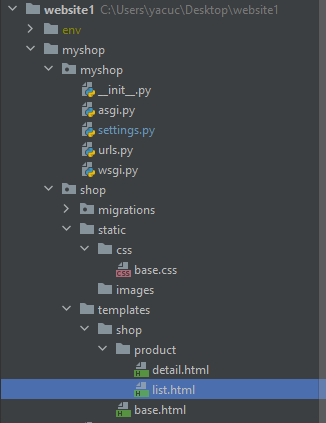
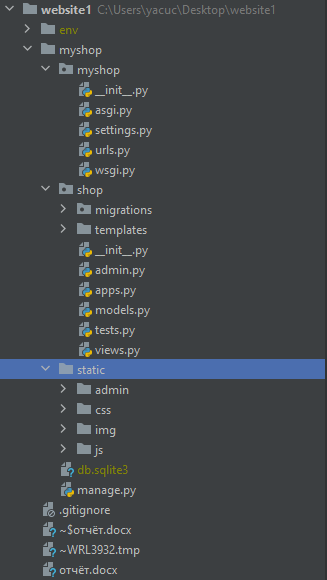
detail.html



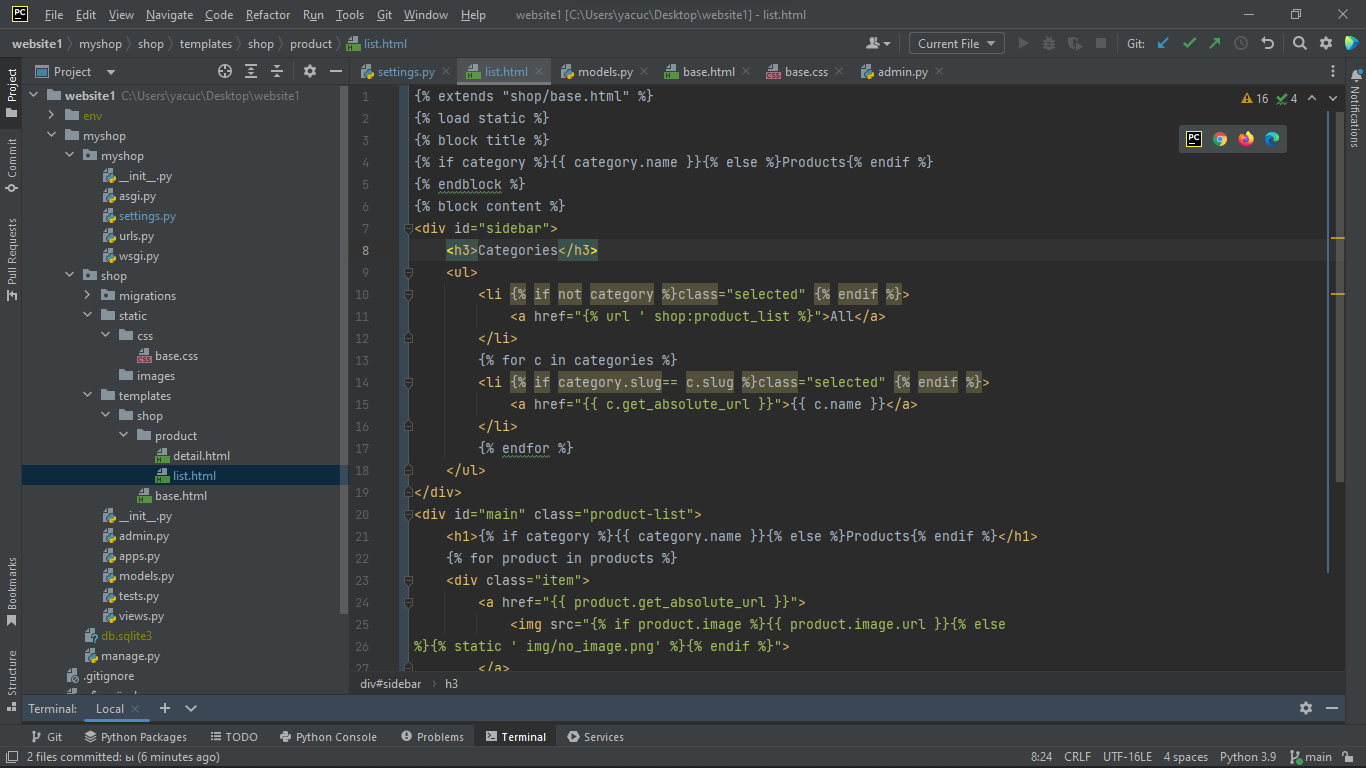
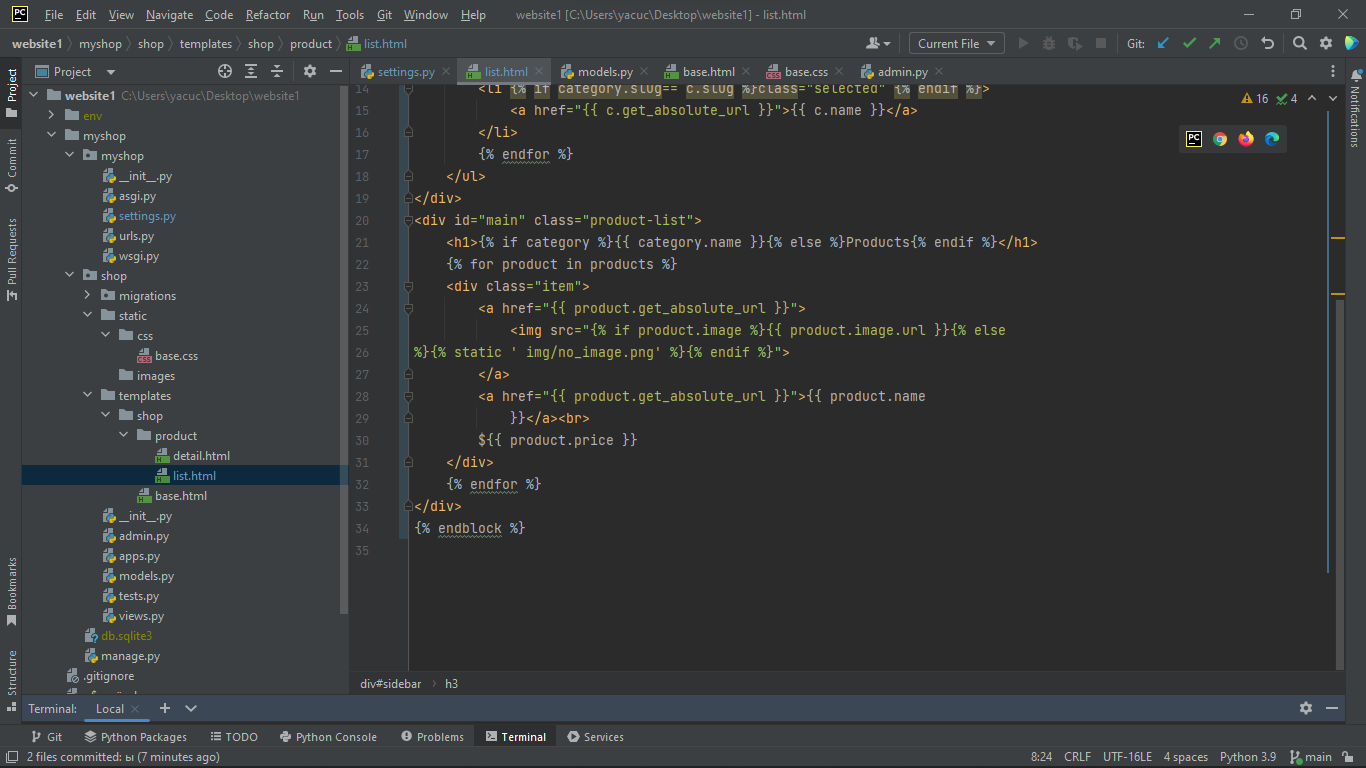
Необходимо определить базовый шаблон, а затем расширить его в list и в шаблоне detail. Отредактируем шаблон shop/base.html:



Чтобы включить требуемые стили CSS и изображения, используемые шаблонами, необходимо скопировать статические файлы, которые входят в эту главу, расположенную в каталоге static/ приложения shop. Скопируем их в ту же папку проекта.

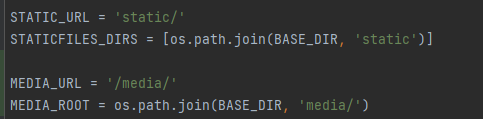
 

Изменим шаблон shop/product/list.html:

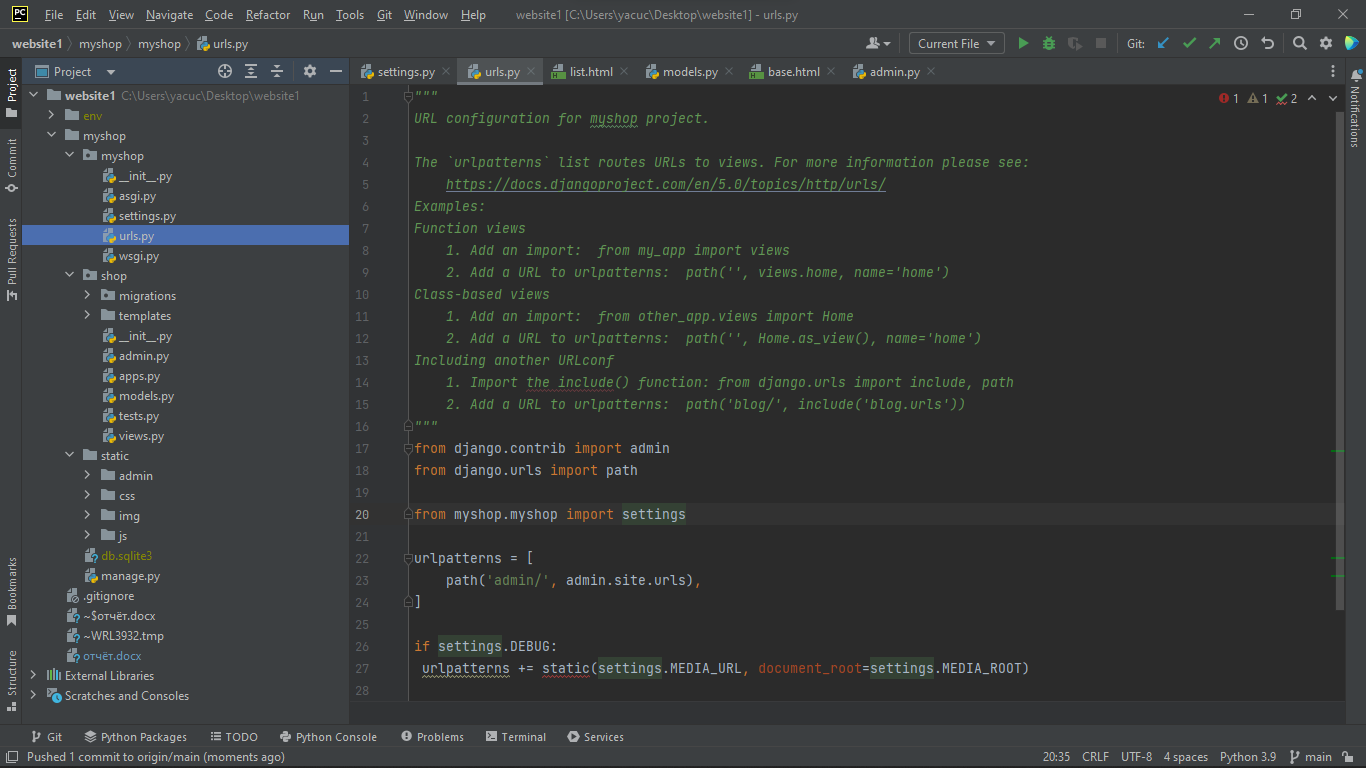
  


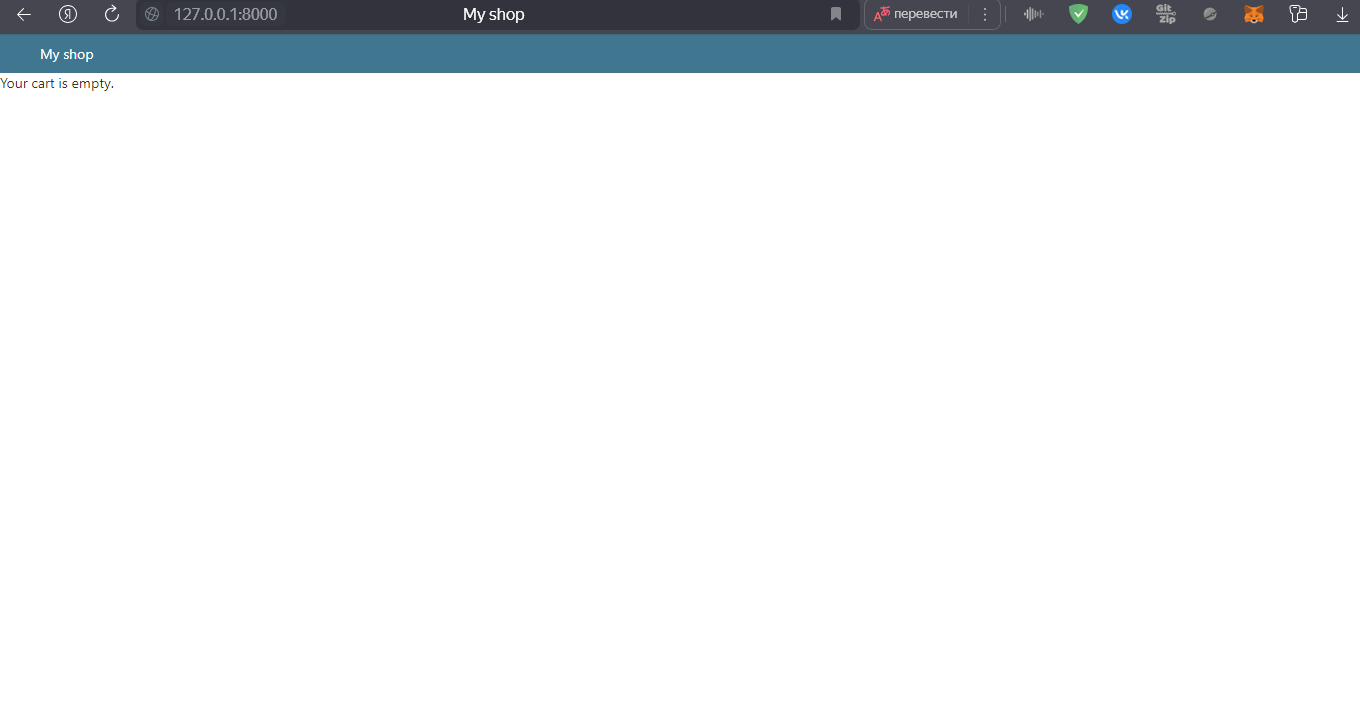
Это шаблон списка продуктов. Он расширяет шаблон shop/base.html и использует переменную контекста категорий для отображения всех категорий на боковой панели и продуктов для отображения продуктов текущей страницы. Один и тот же шаблон используется для обоих типов: список всех доступных продуктов и список продуктов, отфильтрованных по категориям. Поскольку поле изображения модели продукта может быть пустым, мы должны предоставить изображение по умолчанию для продуктов, у которых нет изображения. Изображение находится в каталоге статических файлов с относительным путем img/no\_image.png.

Отредактируем settings.py:

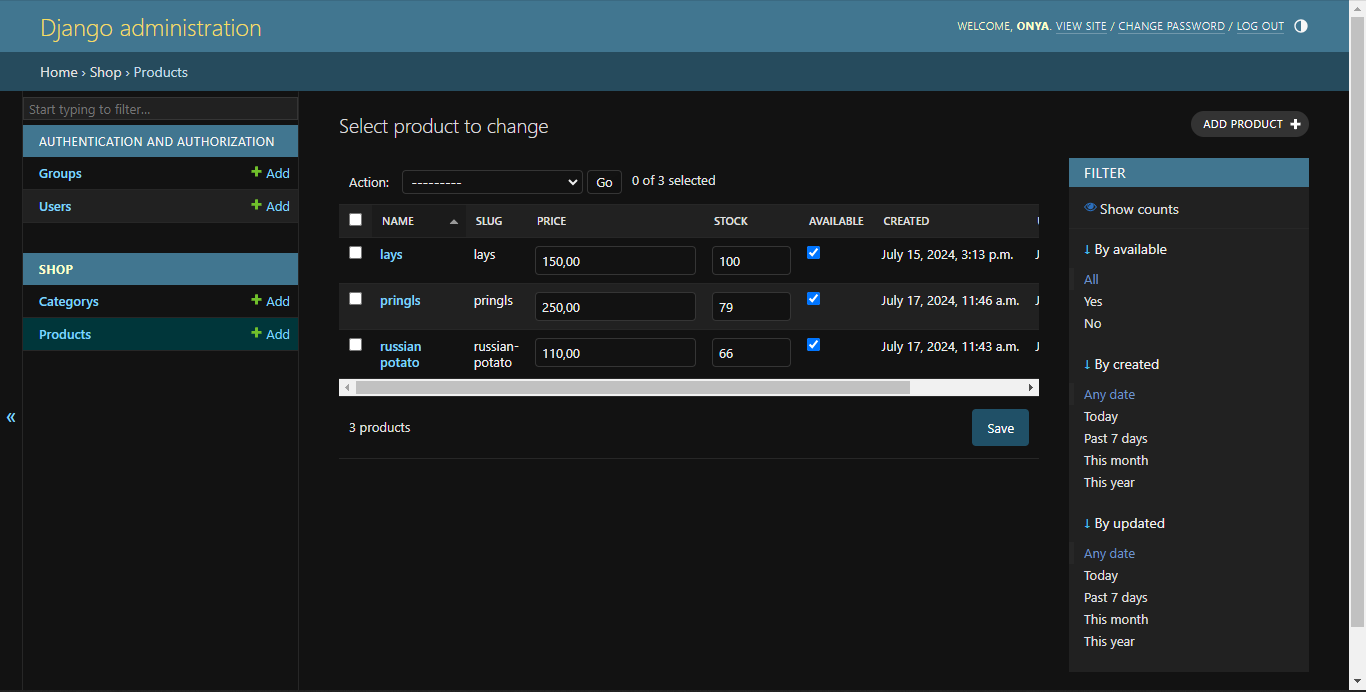


Чтобы Джанго обрабатывал загруженные мультимедийные файлы с помощью сервера разработки, отредактируем файл urls.py:

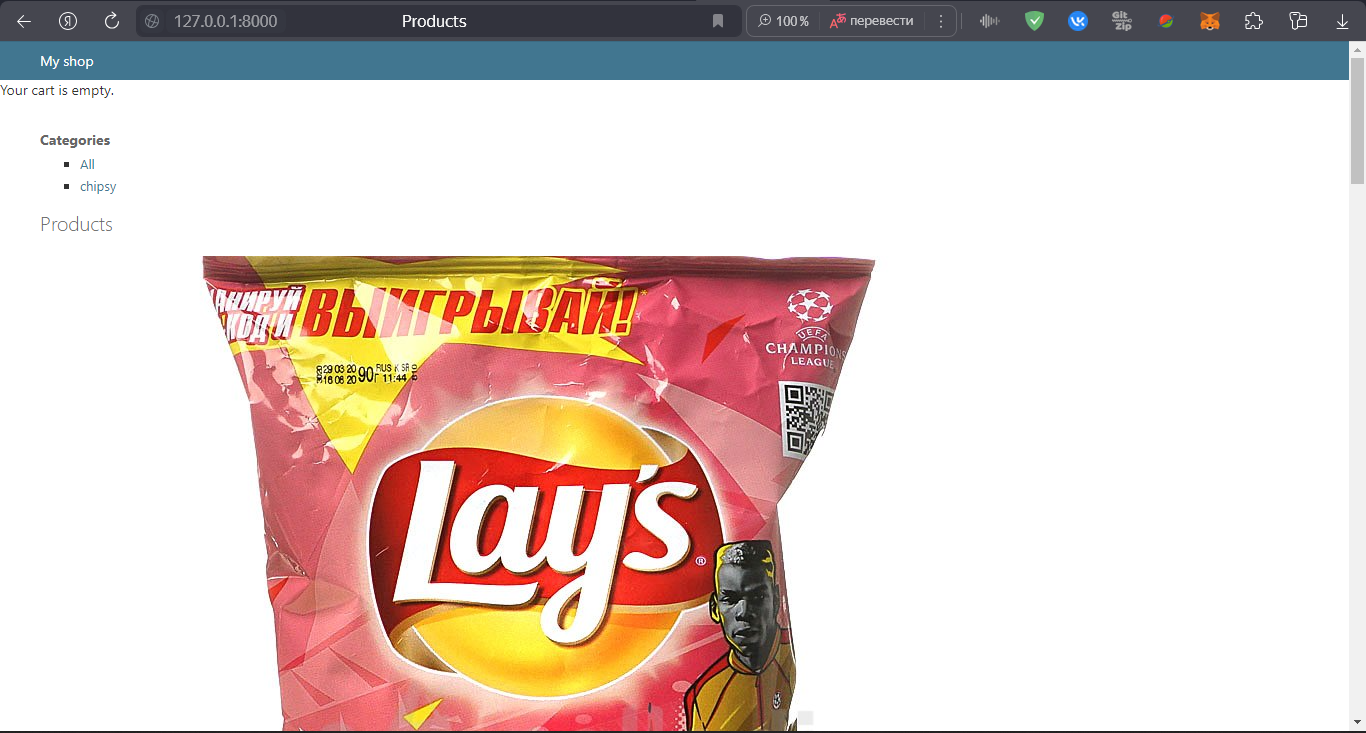


base.html будет выводить следующую страницу в соответствии с конструктором:  


Добавим в магазин несколько продуктов с помощью сайта администрирования:



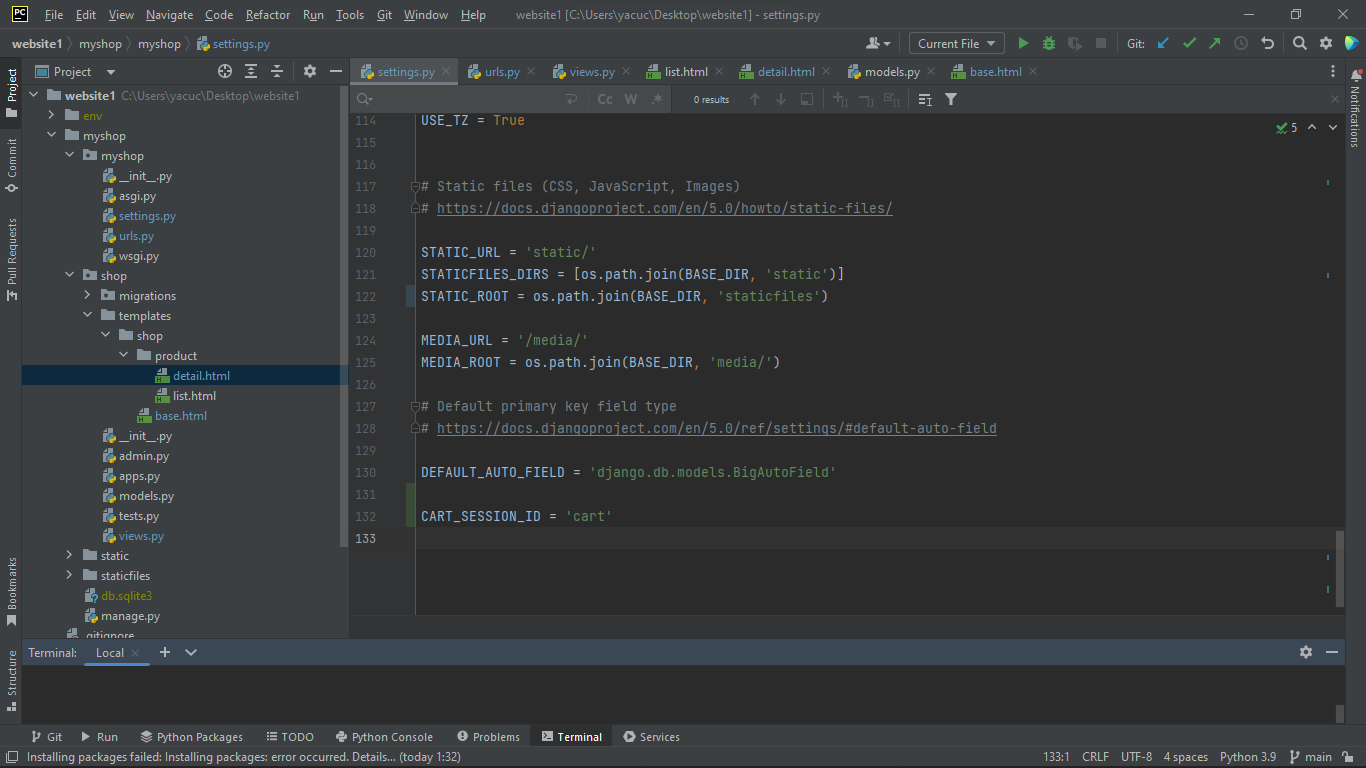
Появилась страница списка продуктов (позже надо будет поправить css)

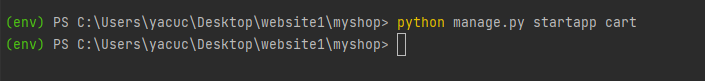

Развернуть описание пока не получается

Создаём корзину

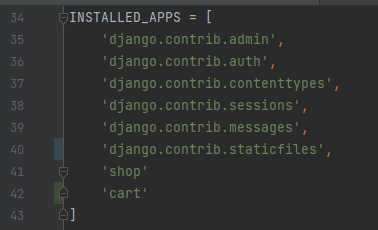
Изменим файл settings.py проекта:



Создадим приложение для управления корзинами покупок:



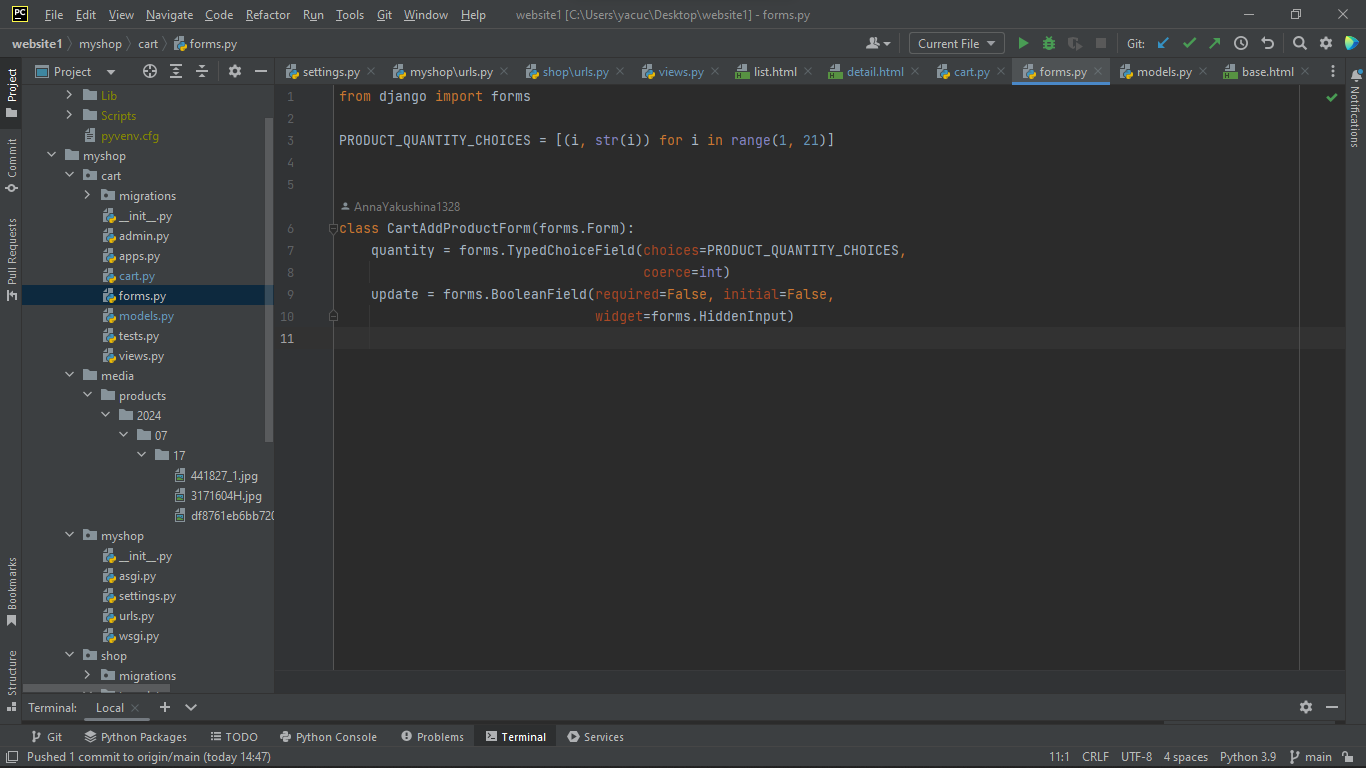
Добавим "cart" к параметру INSTALLED\_APPS:



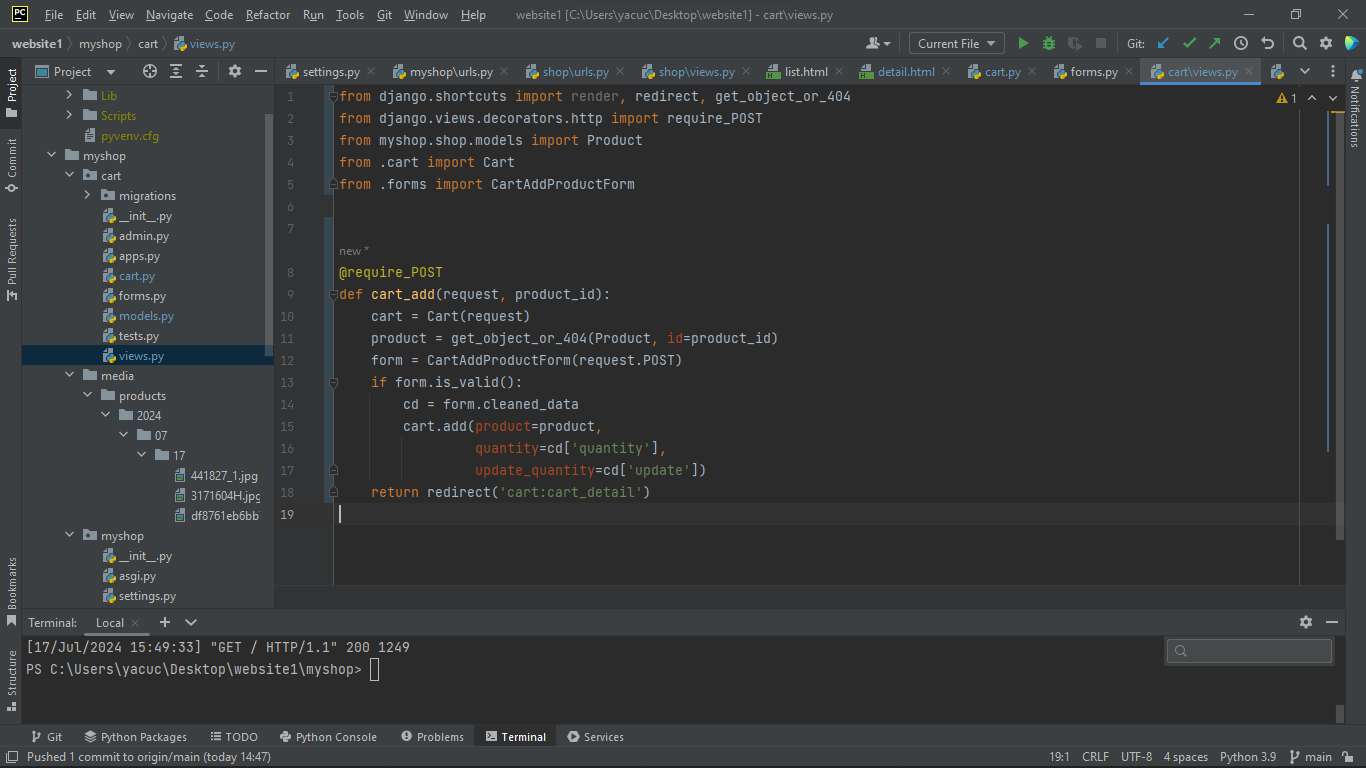
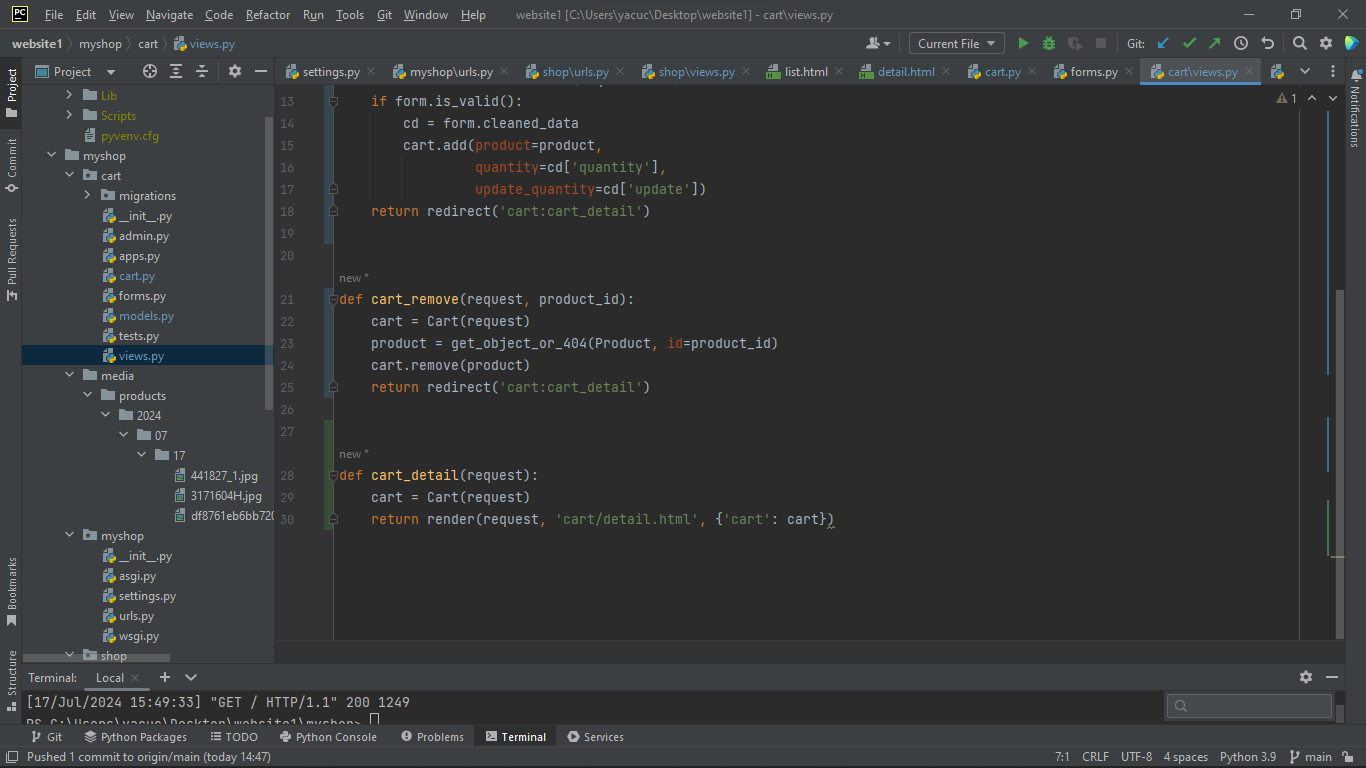
Создадим новый файл в каталоге приложения cart:

from decimal import Decimal  
from django.conf import settings  
from myshop.shop.models import Product  
  
  
class Cart(object):  
 def \_\_init\_\_(self, request):  
 *"""  
 Инициализируем корзину  
 """* self.session = request.session  
 cart = self.session.get(settings.CART\_SESSION\_ID)  
 if not cart:  
 # save an empty cart in the session  
 cart = self.session[settings.CART\_SESSION\_ID] = {}  
 self.cart = cart  
  
 def add(self, product, quantity=1, update\_quantity=False):  
 *"""  
 Добавить продукт в корзину или обновить его количество.  
 """* 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):  
 # Обновление сессии cart  
 self.session[settings.CART\_SESSION\_ID] = self.cart  
 # Отметить сеанс как "измененный", чтобы убедиться, что он сохранен  
 self.session.modified = True  
  
 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()  
  
 def \_\_iter\_\_(self):  
 *"""  
 Перебор элементов в корзине и получение продуктов из базы данных.  
 """* 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):  
 *"""  
 Подсчет всех товаров в корзине.  
 """* return sum(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

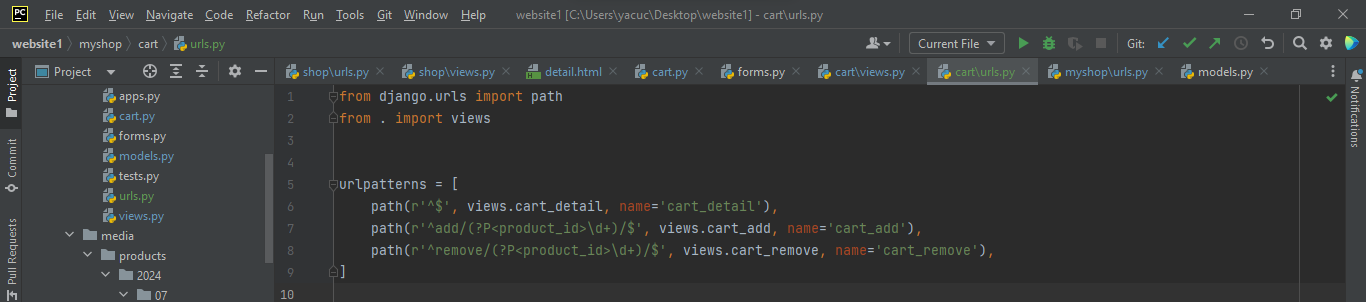
Чтобы добавить элементы в корзину, нам нужна форма, позволяющая пользователю выбрать количество добавляемого товара. Создадим файл forms.py в каталоге приложения cart:



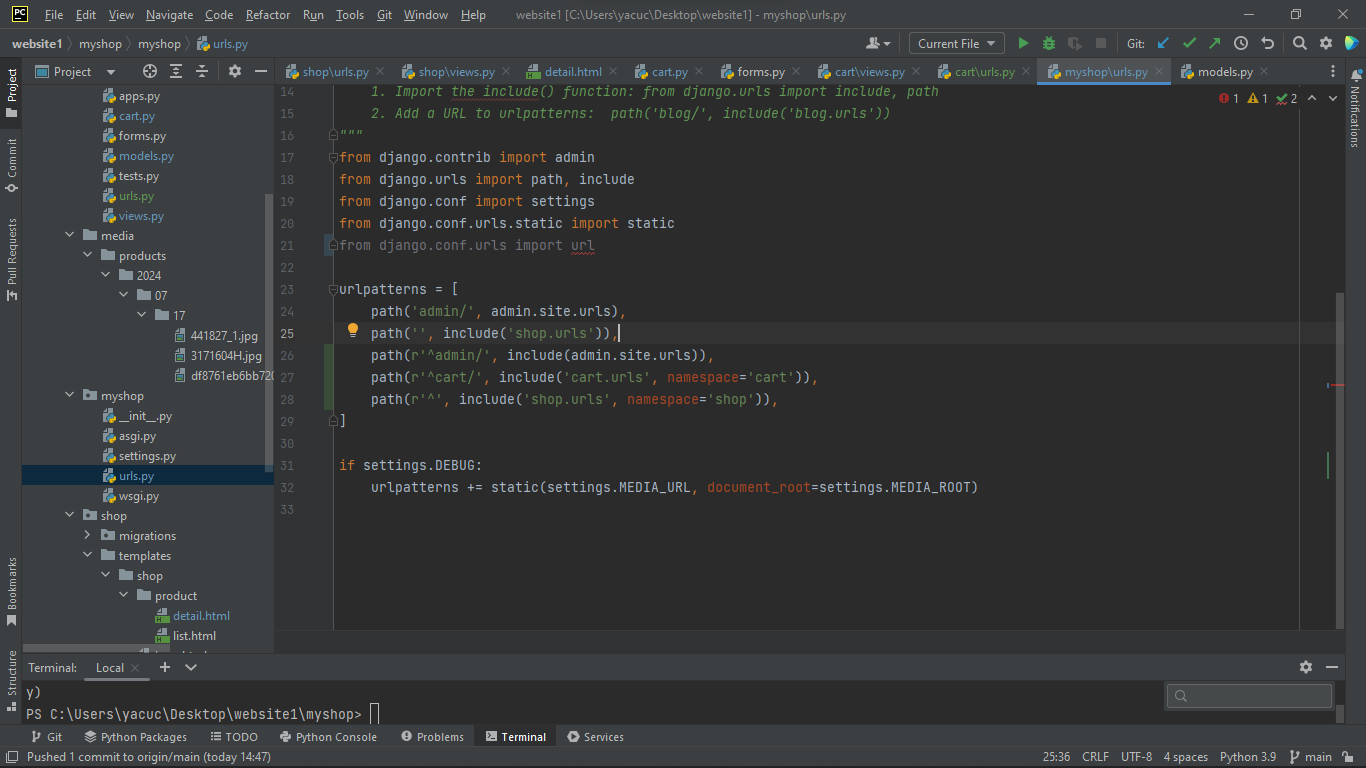
Создадим представление для добавления элементов в корзину. Отредактируем файл views.py приложения cart:

Мы создали представления для добавления товаров в корзину, обновления количества, удаления товаров из корзины и отображения корзины. Рассмотрим добавление шаблонов URL-адресов для этих представлений. Создадим новый файл в каталоге приложения cart и назовём его urls.py:



Изменим основной файл urls.py проекта myshop:



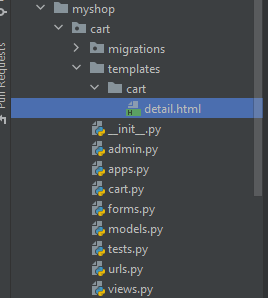
Представления cart\_add и cart\_remove не нужнаются в шаблонах, но необходимо создать шаблон для представления cart\_detail для отображения элементов корзины и итоговых значений.

Создадим следующую структуру файла в каталоге приложения cart:

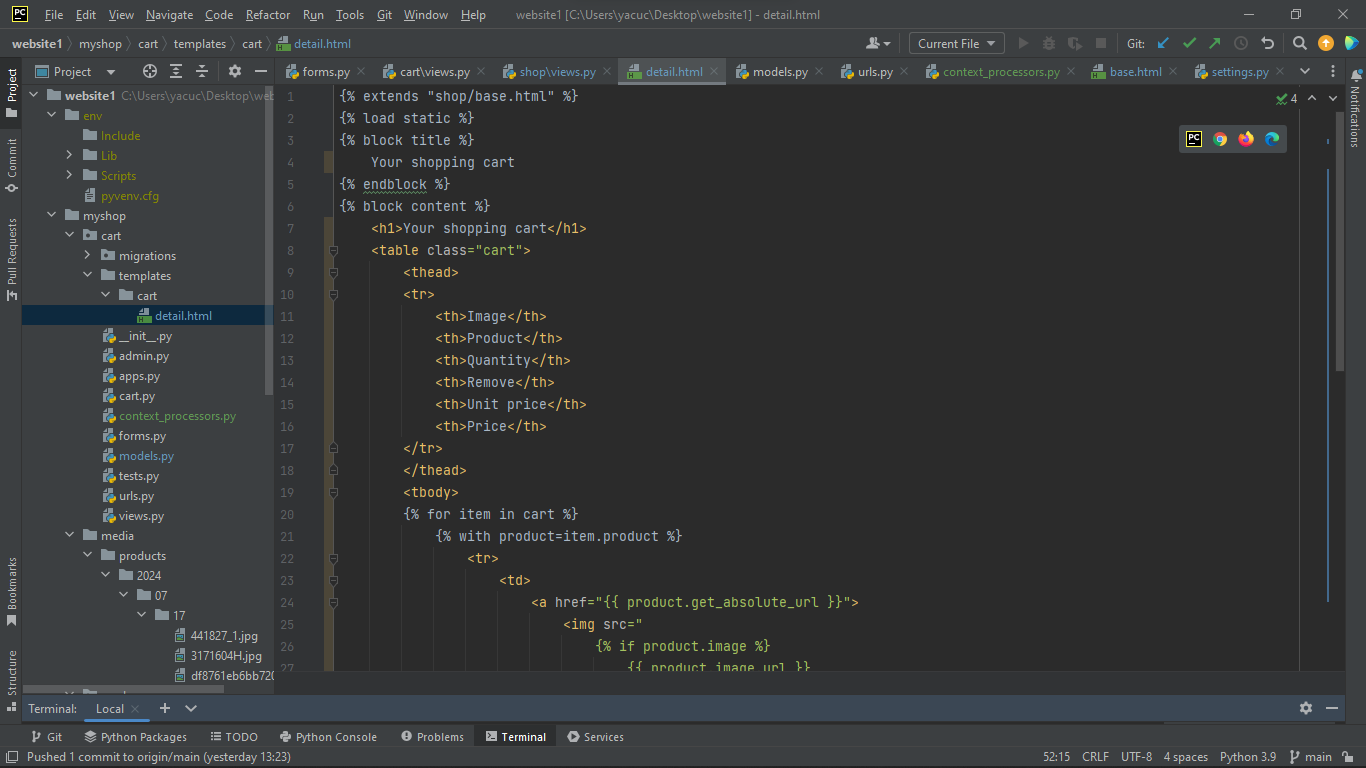
templates/

cart/

detail.html

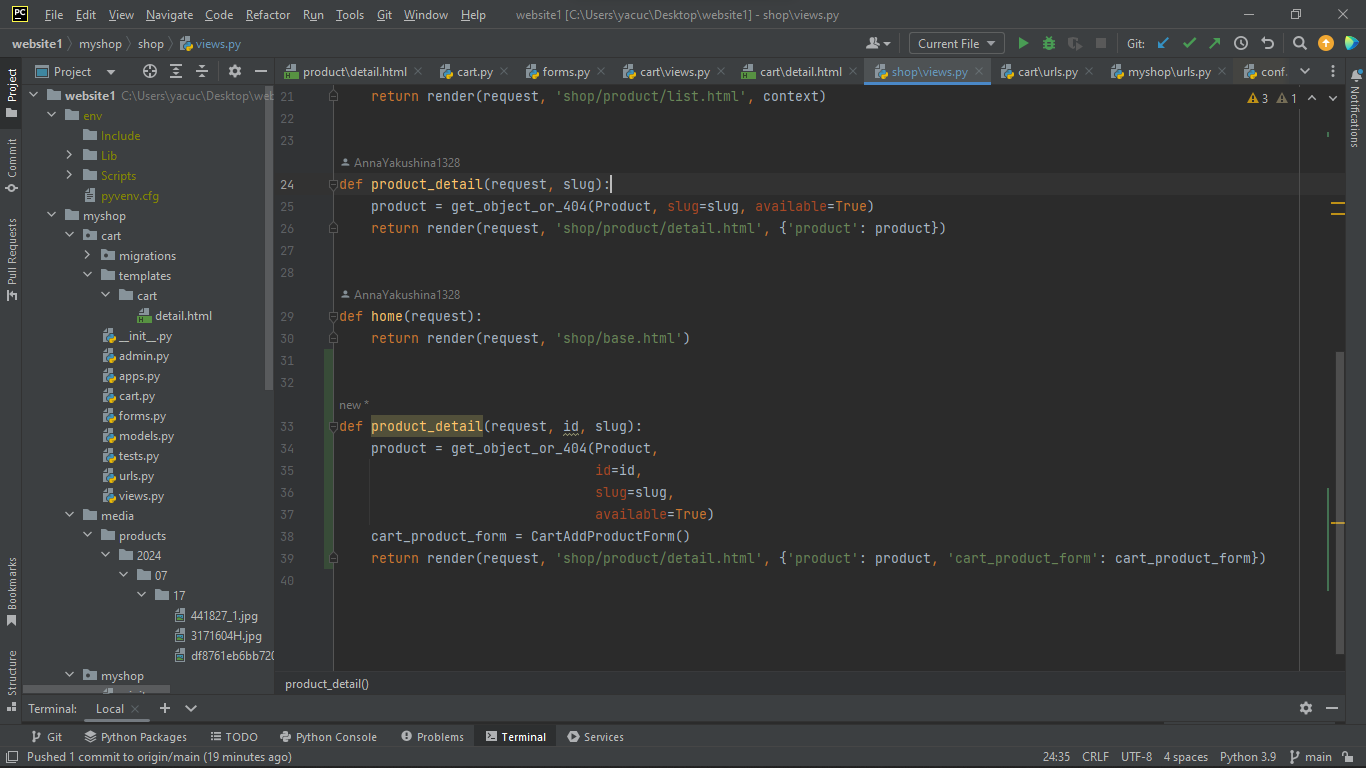


Изменим шаблон cart/detail.htm:

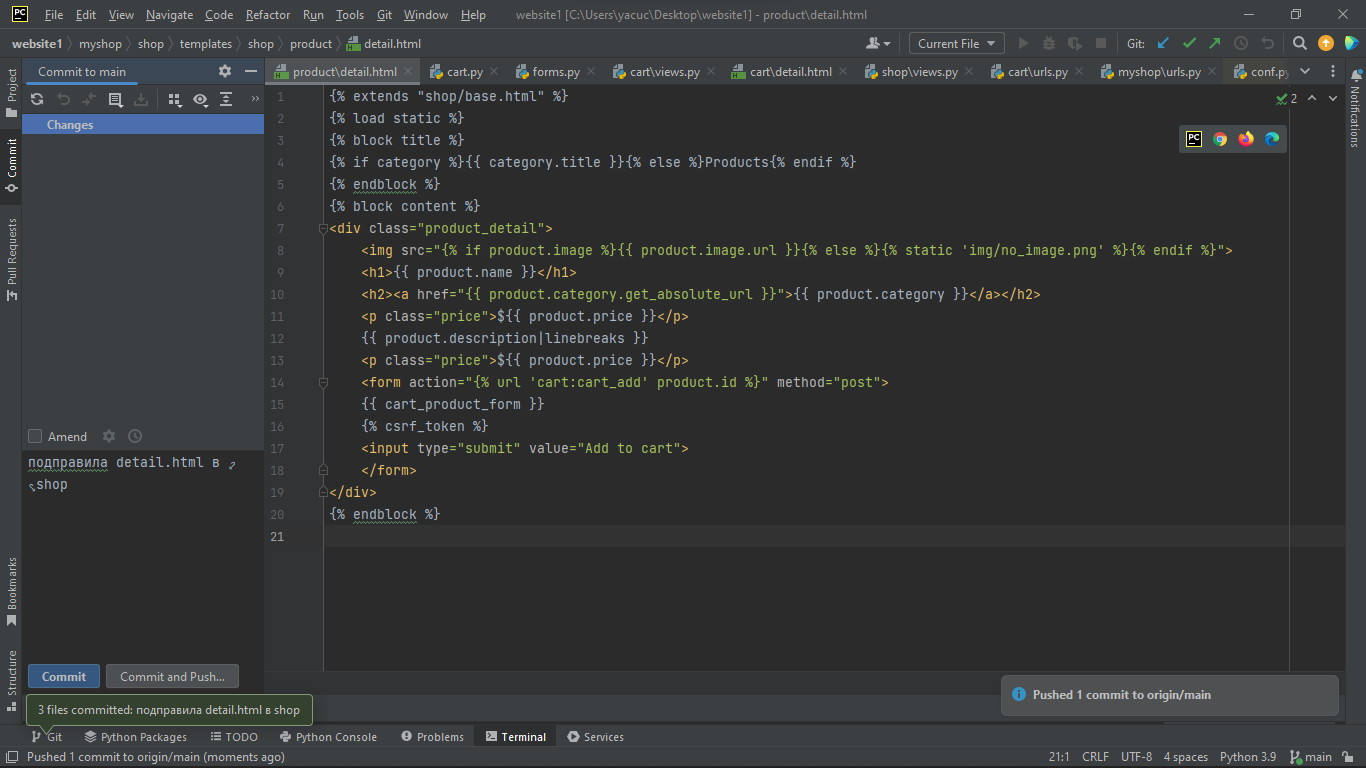


Это шаблон, используется для отображения содержимого корзины. Он содержит таблицу с элементами, хранящимися в текущей корзине.

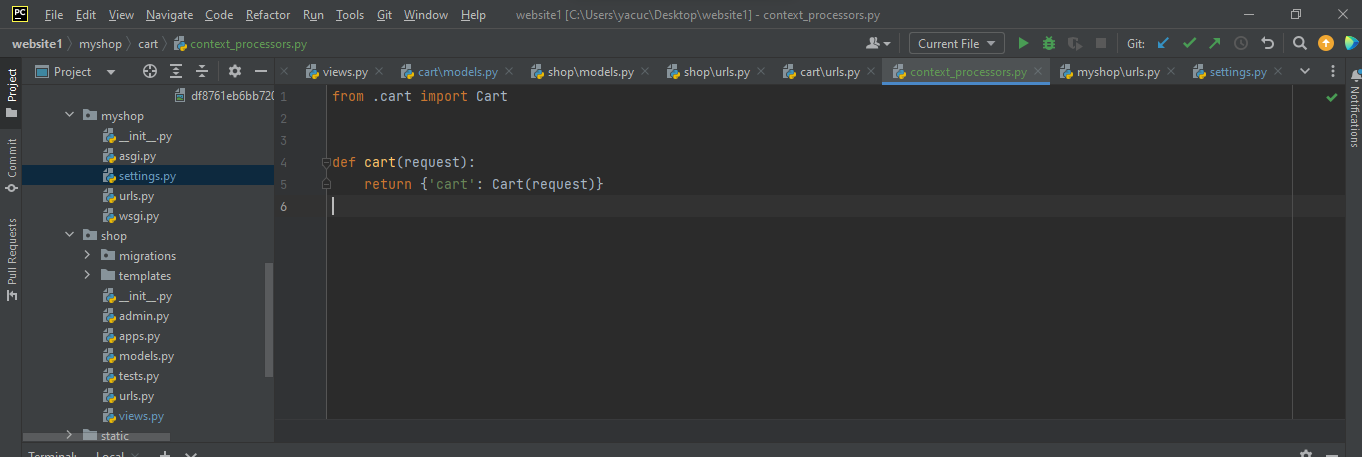
Теперь необходимо добавить кнопку «Добавить в корзину» на страницу сведений о продукте. Отредактируем файл views.py приложения shop и добавим CartAddProductForm в представление product\_detail:



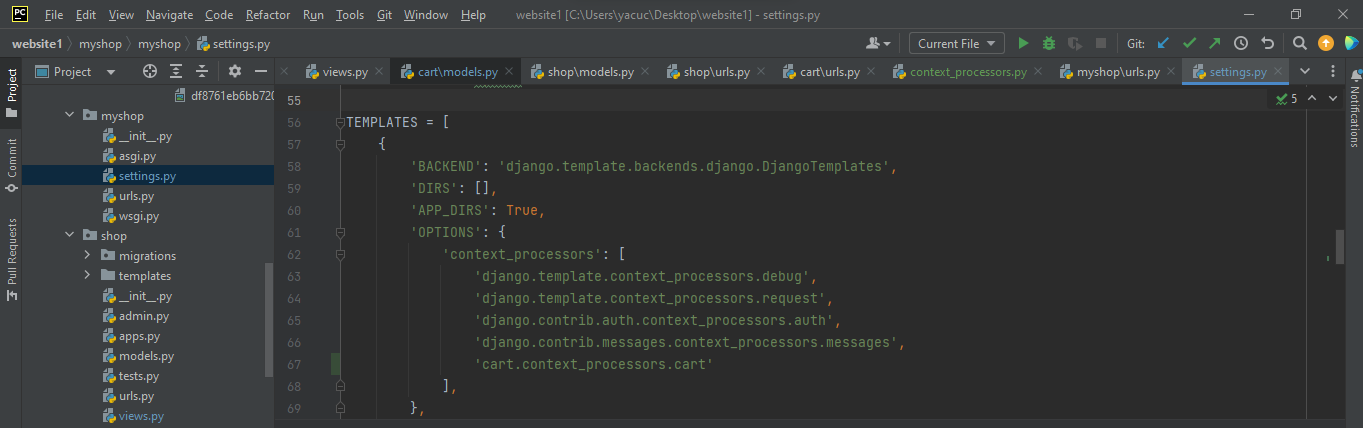
Отредактируем шаблон shop/product/detail.html приложения shop и добавим форму цены продукта:



Создадим обработчик контекста для установки текущей корзины в контекст запроса для шаблонов. Мы сможем получить доступ к этой корзине в любом шаблоне. Создадим новый файл в каталоге приложения cart и назовём его context\_processors.py:



Отредактируем файл Settings.py проекта и добавим 'cart.context\_processors.cart' для параметра context\_processors в парамертах TEMPLATES:



Теперь контекстный процессор будет выполняться при каждом просмотре шаблона с использованием RequestContext Джанго.

Теперь отредактируйем шаблон /base.html приложения shop. Заменим предыдущие строки следующим кодом:

