Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

Лабораторная работа №6

По курсу «Разработка интернет приложений»

Подготовила:

Студентка группы ИУ5-55Б.

Очеретная С.В.

18.12.2020

Проверил:

Преподаватель кафедры ИУ5

Гапанюк Ю.Е.

Москва, 2021 г.

Цель лабораторной работы: изучение возможностей разработки REST API с использованием Django REST Framework.

Задание: С использованием Django REST Framework разработать REST API для одной модели (одной таблицы базы данных).

Ход работы

Создание приложения

Писать сервер с API мы будем на Django Rest Framework (DRF). Это надстройка над Django, которая позволяет нам писать REST приложения.

Войдем в виртуальное окружение и установим Django rest framework

env\Scripts\activate.bat
pip install djangorestframework

Создадим приложение stocks с помощью команды django-admin startapp stocks
Применим все миграции проекта: python manage.py migrate

В файле lab6/lab6/settings.py в листе INSTALLED_APPS добавим название нашего приложения и название модуля DRF

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'rest_framework',
    'stocks',
]
```

Написание модели

В файле lab6/stocks/models.py напишем представление моделей, как в 5 лабораторной и запустим миграции

```
(env) C:\Users\Pocht\OneDrive\Study\семестр5\РИП\webappdevelopment\lab6>py manage.py makemigrations
Migrations for 'stocks':
  stocks\migrations\0001_initial.py
    - Create model Donut
    - Create model DonutsSet

(env) C:\Users\Pocht\OneDrive\Study\семестр5\РИП\webappdevelopment\lab6>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, stocks
Running migrations:
  Applying stocks.0001_initial... OK

(env) C:\Users\Pocht\OneDrive\Study\семестр5\РИП\webappdevelopment\lab6>
```

Написание сериализатора

Напишем сериализаторы в файле lab6/stocks/serializers.py

```python
from stocks.models import Donut, DonutsSet
from rest_framework import serializers

class DonutSerializer(serializers.ModelSerializer):
    class Meta:
        # Модель, которую мы сериализуем
        model = Donut
        # Поля, которые мы сериализуем
        fields = ["pk", "name", "info", "cost"]

class DonutsSetSerializer(serializers.ModelSerializer):
    class Meta:
        # Модель, которую мы сериализуем
        model = DonutsSet
        # Поля, которые мы сериализуем
        fields = ["pk", "name", "info"]
```

Написание View

Напишем view в файле lab6/stocks/views.py

```python
from rest_framework import viewsets
from stocks.serializers import DonutSerializer, DonutsSetSerializer
from stocks.models import Donut, DonutsSet

class DonutViewSet(viewsets.ModelViewSet):
    """
    API endpoint, который позволяет просматривать и редактировать пончики
    """
    # queryset всех пончиков для фильтрации по стоимости пончика
    queryset = Donut.objects.all().order_by('cost')
    serializer_class = DonutSerializer  # Сериализатор для модели

class DonutsSetViewSet(viewsets.ModelViewSet):
    queryset = DonutsSet.objects.all()
    serializer_class = DonutsSetSerializer
```

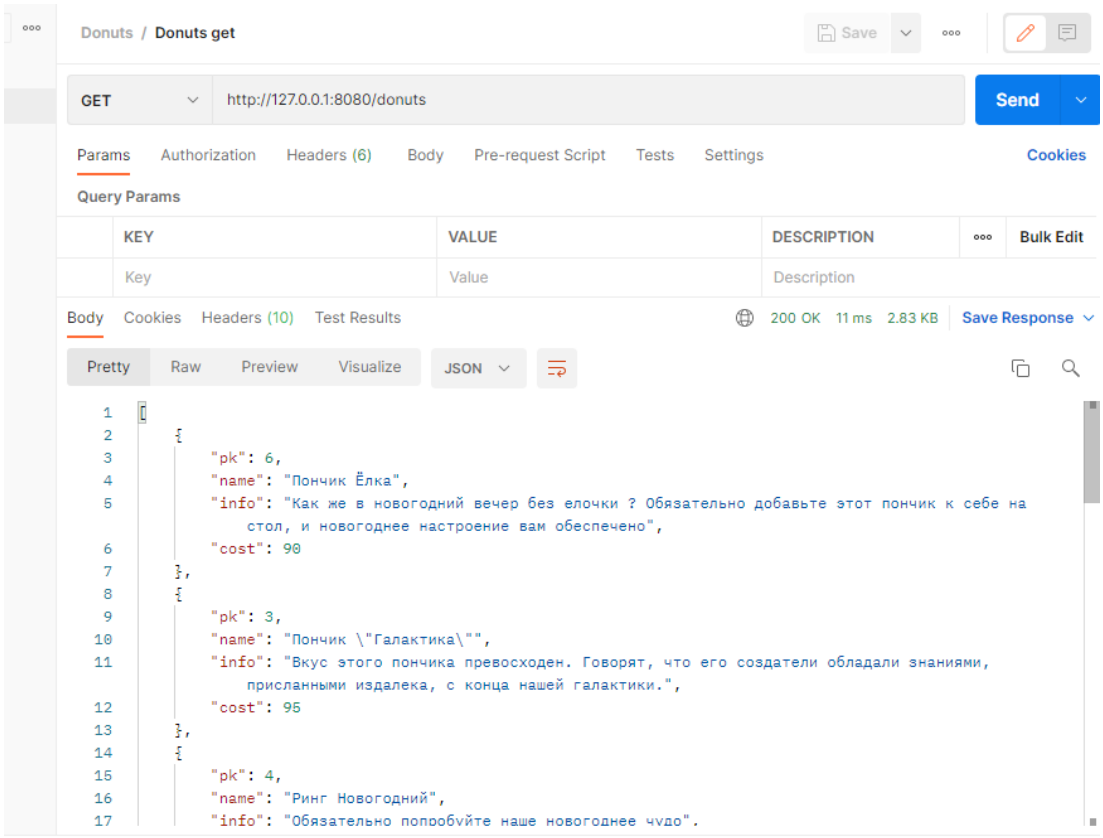
Добавление View в URL'ы нашего приложения

Добавим роутер нашего view в URL'ы приложения.
Для этого в файле lab6/lab6/urls.py напишем:

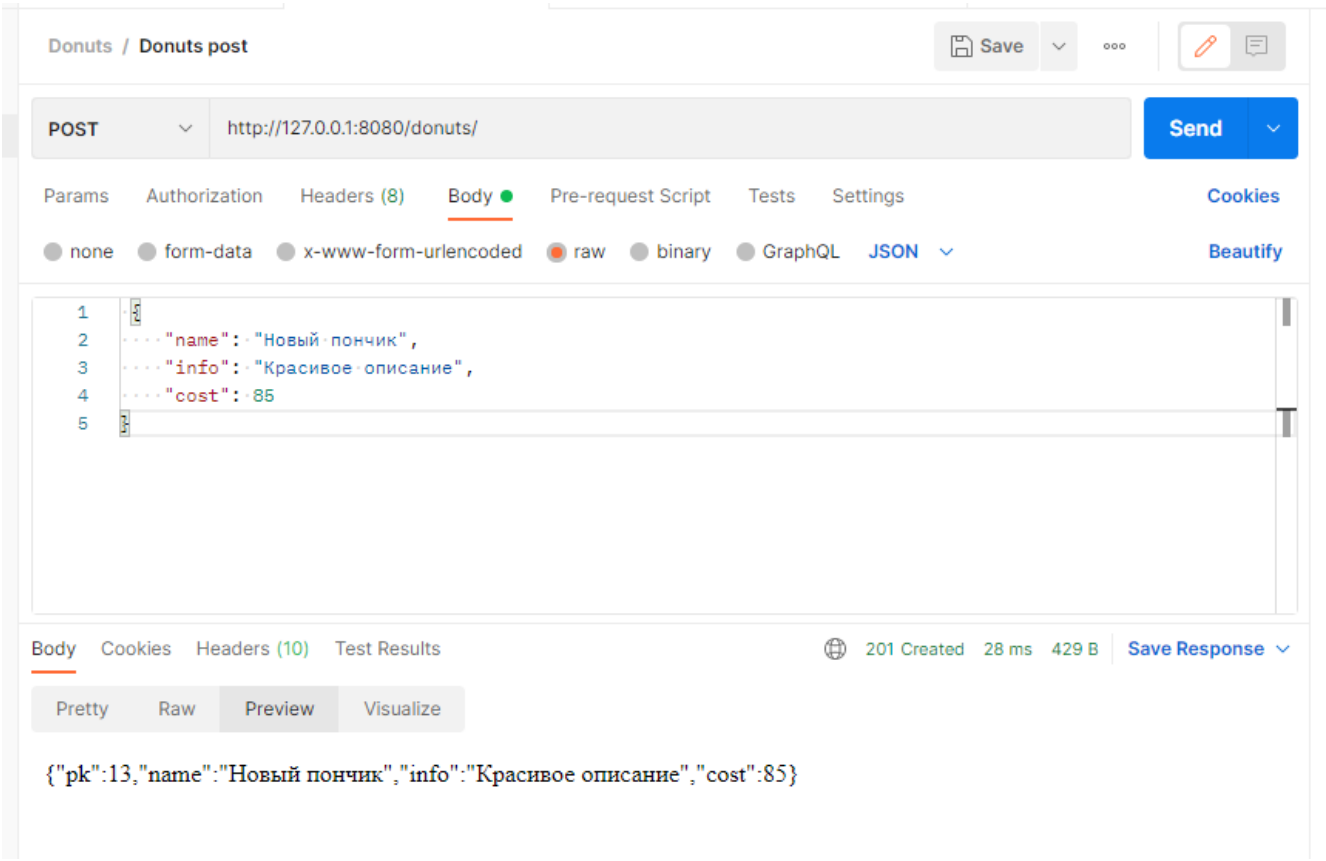
```
from django.contrib import admin
from stocks import views as stock_views
from django.urls import include, path
from rest_framework import routers

router = routers.DefaultRouter()
router.register(r'donuts', stock_views.DonutViewSet)
router.register(r'donutssets', stock_views.DonutsSetViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),

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

Проверяем правильность работы API с помощью Postman

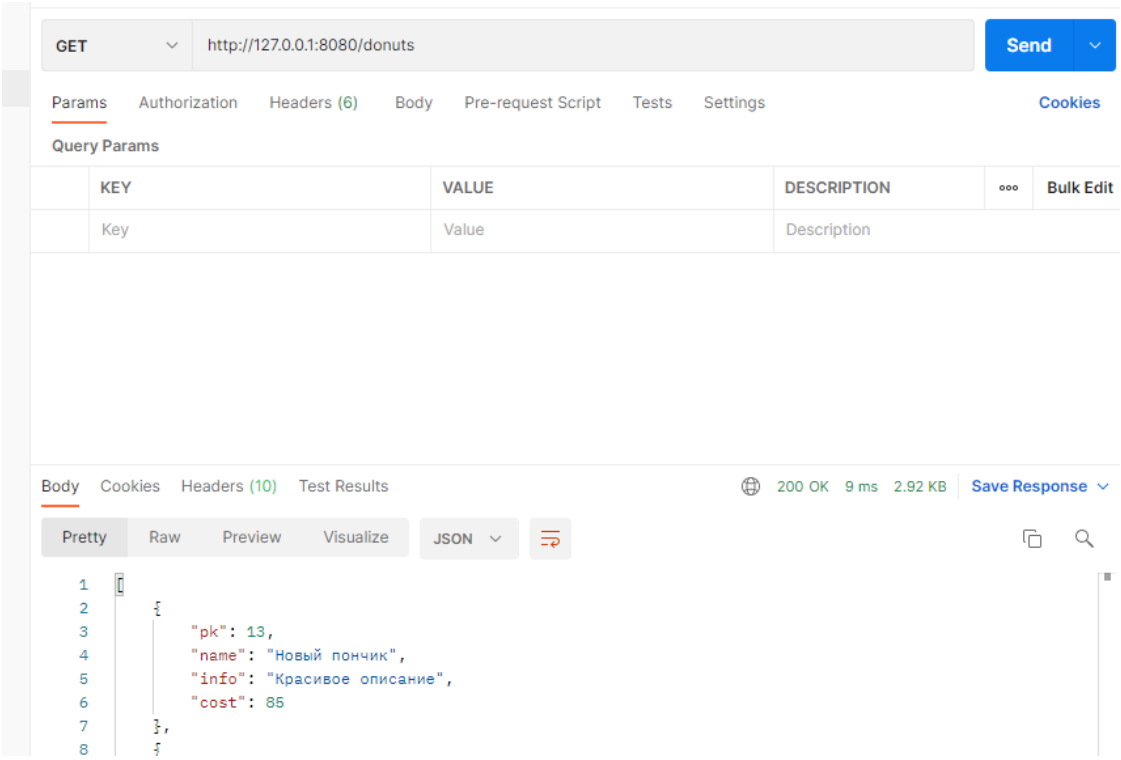
Напишем Get запрос на получение списка пончиков по возрастанию цены:



Добавим пончик с помощью метода POST:

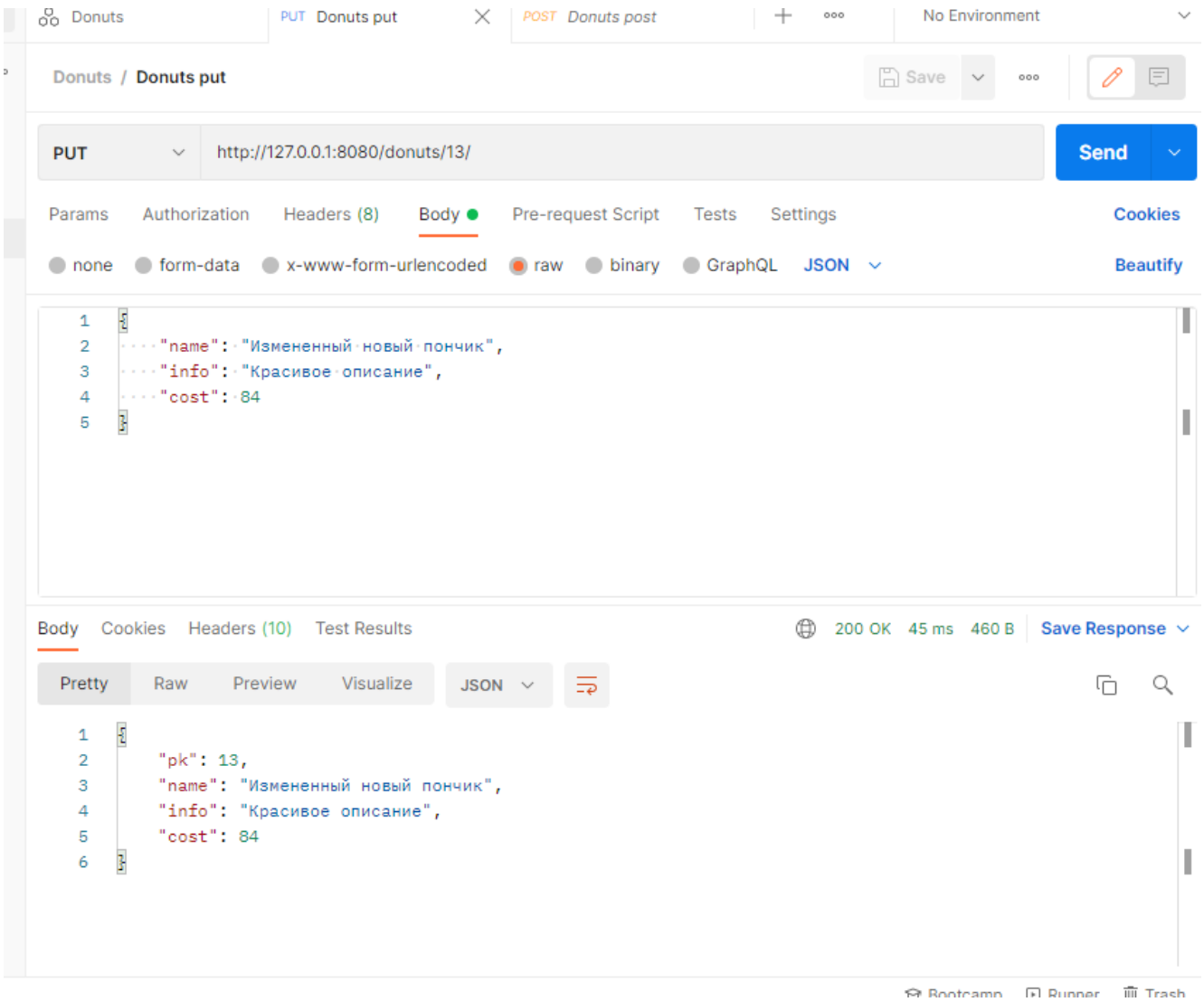


Нам пришел статус 201 и вернулся объект, который мы создали, попробуем еще раз запросить список всех объектов с помощью GET запроса.

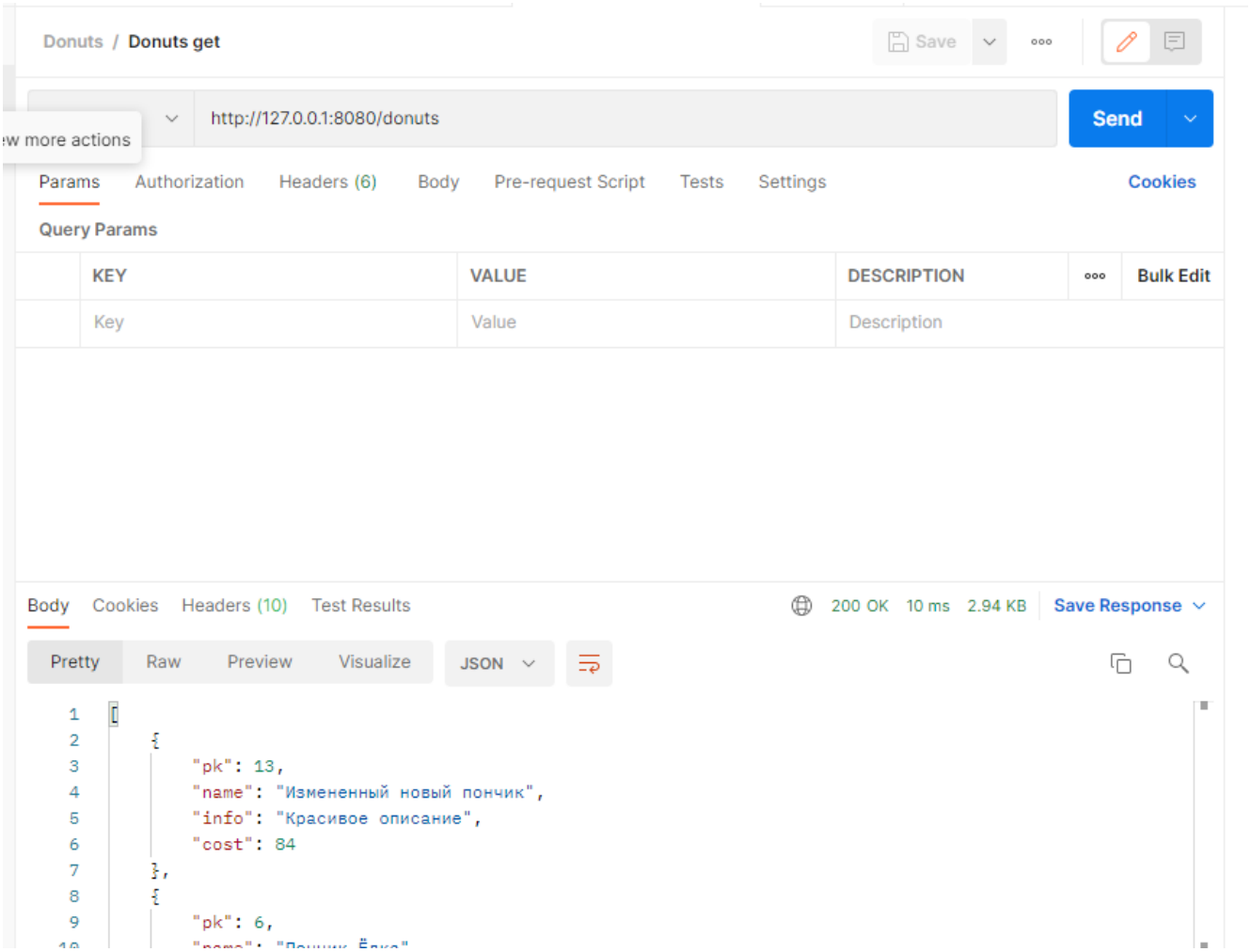


Как видим, новый пончик создан

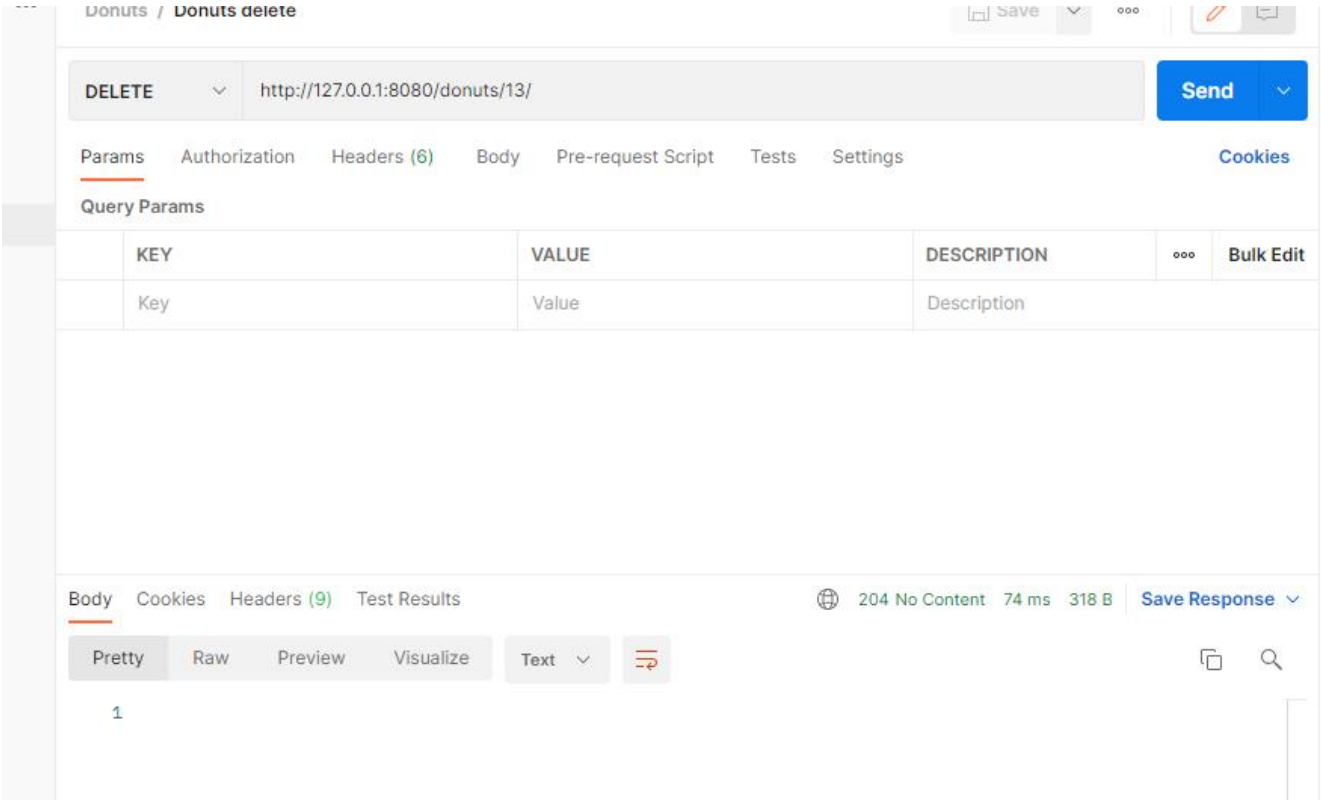
Можно заметить, что нам вернулись все объекты, которые мы создали, также им присвоились номера, которые написаны в поле pk(Primary Key), попробуем изменить объект с pk=13 и поменять название пончика, и его стоимость. Для этого поменяем метод на PUT(нужен для обновления объекта), в конце ссылки напишем id и передадим новый объект



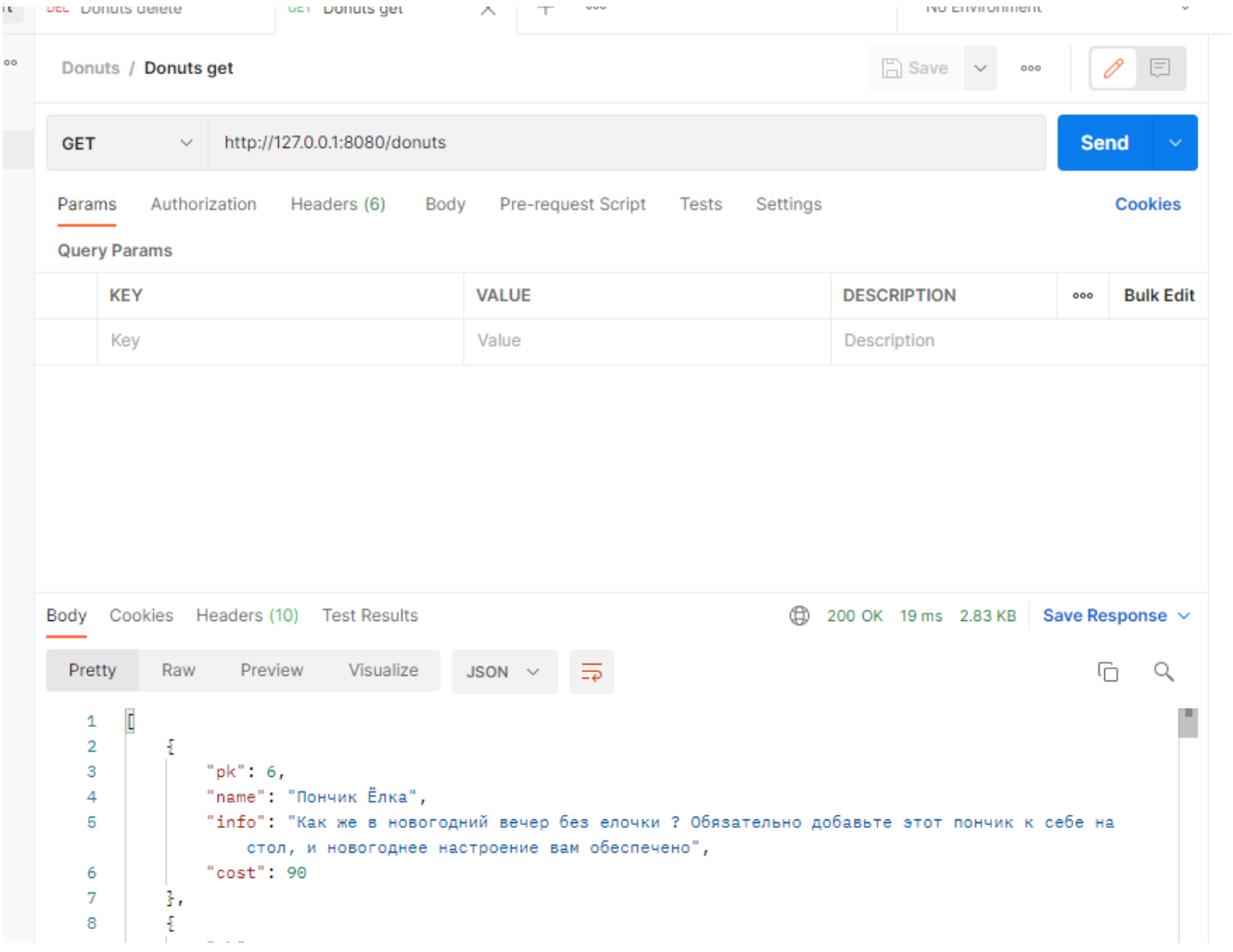
Как видим, мы поменяли описание нашего пончика и сделали скидку на 1 руб 😊



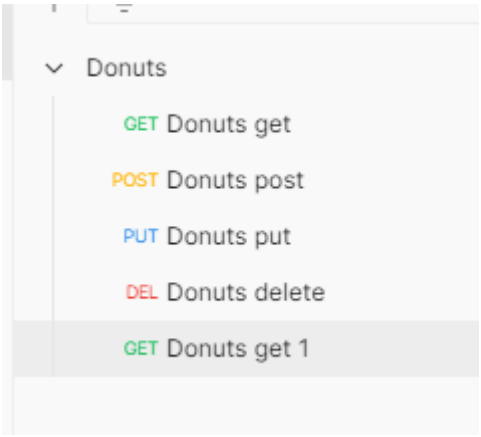
Далее удалим наш пончик с pk=13. Изменим HTTP метод на DELETE и в конце строки аргументом укажем pk.



Посмотрим список пончиков и поймем, что нашего пончика уже нет ☹



Таким образом, итог лабораторной работы – создания API для модели Пончика:



GET http://127.0.0.1:8080/donuts - Получить список всех пончиков
GET http://127.0.0.1:8080/donuts/1/ - Получить 1 пончик
POST http://127.0.0.1:8080/donuts/ - Добавить пончик
DELETE http://127.0.0.1:8080/donuts/13/ - Удалить пончик
PUT http://127.0.0.1:8080/donuts/13/ - Изменить пончик