

API

1. Общая информация

Многие информационные системы имеют сложную архитектуру, состоят из различных компонентов (сервисов), взаимодействуют с другими информационными системами и устройствами. В связи с этим важным является механизм эффективного взаимодействия между разными информационными системами, либо между сервисами внутри одной системы. При этом такой механизм взаимодействия должен быть унифицированным, расширяемым, функциональным, удобным в использовании. Для решения такой задачи при разработке и эксплуатации информационных систем используют прикладной программный интерфейс (Application Programming Interface, API).

2. REST API

Существует несколько способов организации API, но наиболее часто применяют REST (Representational State Transfer). В терминологии REST API каждый URL (Uniform Resource Locator) называется ресурсом, с ресурсами осуществляются следующие действия:

- GET - возвращает описание ресурса,
- POST - добавляет новый ресурс,
- PUT - изменяет ресурс,
- DELETE - удаляет ресурс.

Для этой группы действий существует общее название CRUD (Create, Read, Update, Delete) и совместно действия GET, POST, PUT и DELETE предоставляют простой CRUD интерфейс для других приложений или сервисов, взаимодействие с которым происходит через протокол HTTP. Соответствие CRUD действий и HTTP методов:

- CREATE – POST,
- READ – GET,
- UPDATE – PUT,
- DELETE – DELETE,

Во взаимодействии через REST API применяются коды HTTP ответов (например, статусы вида «2xx» означают успешную операцию, «4xx» – ошибки при обработке запроса и т.п.). Общепринятые форматы взаимодействия REST – JSON и XML. При использовании JSON необходимо помнить, что для передаваемых объектов требуется сериализация (преобразование сложных данных, таких как наборы запросов queryset или объекты моделей Django, в типы данных, которые затем можно легко преобразовать в JSON, XML).

REST API интерфейс очень удобен для межпрограммного взаимодействия. Например, мобильное приложение может выступать в роли клиента, который манипулирует данными посредством REST.

Для веб-служб, созданных с использованием идеологии REST, применяют термин «RESTful».

В отличие от веб-сервисов (веб-служб) на основе SOAP, для RESTful веб-API не существует «официального» стандарта. REST является архитектурным стилем, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом, большинство RESTful-реализаций используют такие стандарты, как HTTP, URL, JSON и XML.

3. Пример создания API в Django

Рассмотрим пример создания API на базе библиотеки Django Rest Framework (DRF). Пример для операционной системы linux.

Порядок действий

1. Создать виртуальное окружение

mkdir project

cd project

python3 -m env env

```
jche@linux-vm-jche:~/apps$ mkdir project
jche@linux-vm-jche:~/apps$ cd project
jche@linux-vm-jche:~/apps/project$ python3 -m venv venv
jche@linux-vm-jche:~/apps/project$ ls -la
total 12
drwxrwxr-x  3 jche jche 4096 Jun 10 04:19 .
drwxrwxr-x 18 jche jche 4096 Jun 10 04:19 ..
drwxrwxr-x  6 jche jche 4096 Jun 10 04:19 venv
jche@linux-vm-jche:~/apps/project$ source venv/bin/activate
(venv) jche@linux-vm-jche:~/apps/project$
```

После выполненных действий в текущей рабочей директории создастся директория venv в которой находятся скрипты для работы с виртуальным окружением. В директорию venv будут устанавливаться все библиотеки.

Активировать виртуальное окружение:

source venv/bin/activate

Установить django и djangorestframework:

pip install django

pip install djangorestframework

После этого в папке venv установятся необходимые файлы для работы с django, а также установятся необходимые значения переменных окружения.

```
(venv) jche@linux-vm-jche:~/apps/project$ ls venv/lib/python3.6/site-packages/
asgiref                                pkg_resources-0.0.0.dist-info
asgiref-3.2.7.dist-info                __pycache__
django                                 pytz
Django-3.0.7.dist-info                 pytz-2020.1.dist-info
djangorestframework-3.11.0.dist-info   rest_framework
easy_install.py                       setuptools
pip                                    setuptools-39.0.1.dist-info
pip-9.0.1.dist-info                   sqlparse
pkg_resources                         sqlparse-0.3.1.dist-info
```

2. Создать django-проект и django-приложение, базы данных

django-admin startproject project ../project

./manage.py startapp app

```
(venv) jche@linux-vm-jche:~/apps/project$ django-admin startproject project ../project
(venv) jche@linux-vm-jche:~/apps/project$ ls -la
total 20
drwxrwxr-x  4 jche jche 4096 Jun 10 04:29 .
drwxrwxr-x 18 jche jche 4096 Jun 10 04:19 ..
-rwxrwxr-x  1 jche jche  627 Jun 10 04:29 manage.py
drwxrwxr-x  2 jche jche 4096 Jun 10 04:29 project
drwxrwxr-x  6 jche jche 4096 Jun 10 04:19 venv
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py startapp app
(venv) jche@linux-vm-jche:~/apps/project$ ls -la
total 24
drwxrwxr-x  5 jche jche 4096 Jun 10 04:30 .
drwxrwxr-x 18 jche jche 4096 Jun 10 04:19 ..
drwxrwxr-x  3 jche jche 4096 Jun 10 04:30 app
-rwxrwxr-x  1 jche jche  627 Jun 10 04:29 manage.py
drwxrwxr-x  3 jche jche 4096 Jun 10 04:30 project
drwxrwxr-x  6 jche jche 4096 Jun 10 04:19 venv
(venv) jche@linux-vm-jche:~/apps/project$ █
```

./manage.py migrate

```
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
(venv) jche@linux-vm-jche:~/apps/project$ █
```

При выполнении этой команды создаются необходимые базы данных.

Редактирование файла project/settings.py

Необходимо отредактировать переменные:

```
SECRET_KEY = 'h=(n$vlb^@ls@b9ncspdp0*j2x272cz^jb@*wx+b!#*+=w*d@9'
DEBUG = True
ALLOWED_HOSTS = ['*']
```

Примечание: такие значения переменных SECRET_KEY и DEBUG используются только в отладочных целях, не рекомендуется при запуске в боевом окружении задавать SECRET_KEY в явном виде (лучше задавать через переменную окружения или из служебного файла) и устанавливать вывод внутренней информации django на экран (DEBUG=True) из соображений безопасности.

./manage.py runserver 0.0.0.0:8000

```
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 10, 2020 - 04:34:14
Django version 3.0.7, using settings 'project.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
```

При выполнении этой команды запускается встроенный web-сервер на порте 8000.

После этого можно в браузере открыть страницу django

http://<ваш IP адрес>:8000

django

[View release notes for Django 3.0](#)



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

./manage.py createsuperuser

```
(venv) jche@linux-vm-jche:~/apps/project$ vi project/settings.py
(venv) jche@linux-vm-jche:~/apps/project$
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py createsuperuser
Username (leave blank to use 'jche'): yuchernyshov
Email address:
Password:
```

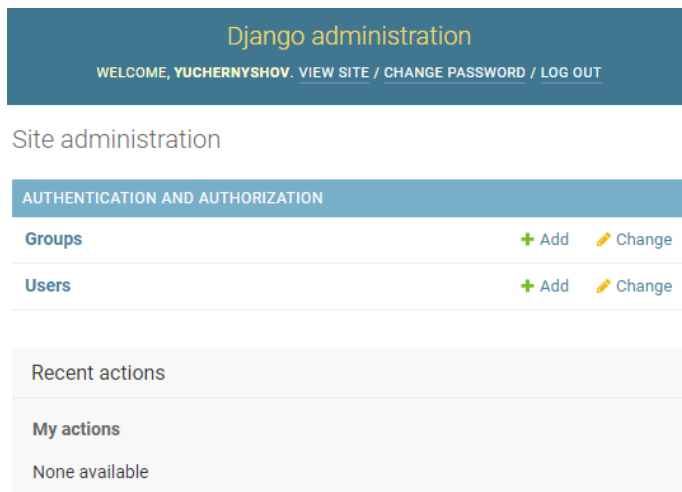
Эта команда создает суперпользователя (superuser) с помощью которого можно через панель администратора получать доступ к внутренней информации проекта django. Требуется задать имя пользователя и пароль (электронная почта - опционально).

После этого можно открыть панель администратора проекта django (web-сервер должен быть по прежнему запущен).

http://<ваш IP адрес>:8000/admin

The screenshot shows the Django administration login page. At the top, there is a blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". Below the "Password:" field, there is a blue button with the text "Log in".

После ввода имени администратора и пароля открывается панель администратора



3. Добавить приложения `app` и `rest_framework` в переменную `INSTALLED_APPS` в файле `project/settings.py`

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

4. Добавление модели `Person` в приложение `app`.
Описать класс модели в файле `app/models.py`

```
from django.db import models  
  
class Person(models.Model):  
    name = models.CharField(max_length=100)  
    number = models.IntegerField()  
  
    def __str__(self):  
        return("Object Person, name: {}, number: {}".format(self.name, self.number))
```

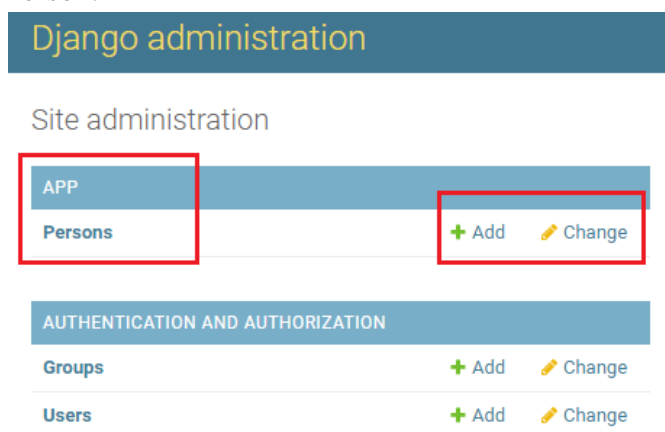
Зарегистрировать модель в `admin.py`

```
from django.contrib import admin  
  
from .models import Person  
  
@admin.register(Person)  
class PersonAdmin(admin.ModelAdmin):  
    list_display = ['name', 'number']
```

Сделать необходимые изменения в базе данных.

```
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py makemigrations
Migrations for 'app':
  app/migrations/0001_initial.py
    - Create model Person
(venv) jche@linux-vm-jche:~/apps/project$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, app, auth, contenttypes, sessions
Running migrations:
  Applying app.0001_initial... OK
```

Теперь на панели администратора <http://<ваш IP адрес>:8000/admin> появился раздел моделей и модель Person. В панели администратора можно добавлять, редактировать и удалять объекты Person.



5. Реализация представления API (API view)

В файле `app/view.py` опишем класс `person_api_view`, который является наследником класса `rest_framework.views.APIView`. Для класса `person_api_view` опишем методы

- `get` - получение информации об объектах Person в базе данных,
- `post` – публикация нового объекта Person,
- `put` – редактирование существующего объекта Person (идентификация по `Person.number`),
- `delete` – удаление существующего объекта Person (идентификация по `Person.number`).

Пример носит обучающий характер, в частности – не проверяется уникальность `number`.

Для сериализации объектов Person создается класс `PersonSerializer`, который является наследником класса `rest_framework.serializers`. Этот класс может быть описан в произвольном файле, но обычно используют `serializers.py`. В классе `PersonSerializer` описываются методы `create` – используется при создании нового объекта (POST) `update` – используется при редактировании существующего объекта (PUT).

В файлах `project/urls.py` и `app/urls.py` должны быть прописаны правила обработки URL, в том числе обработка передаваемого параметра `number` в методах PUT и DELETE..

Листинг для представления `person_api_view` и `PersonSerializer` приведен ниже. Рекомендуется реализовывать последовательно части, относящиеся к GET, POST, PUT, DELETE и проверять работу с использованием приложения chrome ARC (REST API клиент).

Описание представления person_api_view в файле app/views.py

```
from django.shortcuts import render

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.generics import get_object_or_404

from .models import Person
from .serializers import PersonSerializer

class person_api_view(APIView):

    def get(self, request):
        persons = Person.objects.all()
        serialized_persons = PersonSerializer(persons, many=True)
        return Response({"persons": serialized_persons.data})

    def post(self, request):
        person = request.data.get("person")
        serialized_person = PersonSerializer(data=person)
        if serialized_person.is_valid(raise_exception=True):
            saved_person = serialized_person.save()
            return Response({"success": "Person {} saved".format(saved_person.name)})

    def put(self, request, number):
        data = request.data.get("person")
        person = get_object_or_404(Person.objects.all(), number=number)
        serialized_person = PersonSerializer(instance=person, data=data, partial=True)
        if serialized_person.is_valid(raise_exception=True):
            updated_person = serialized_person.save()
            return Response({"success": "Person {} updated".format(updated_person)})

    def delete(self, request, number):
        person = get_object_or_404(Person.objects.all(), number=number)
        name = person.name
        number = person.number
        person.delete()
        return Response({"success": "Person {} deleted".format(name, number)})
```

Описание сериализатора PersonSerializer в файле app/serializers.py

```
from rest_framework import serializers

from .models import Person

class PersonSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
    number = serializers.IntegerField()

    def create(self, validated_data):
        return Person.objects.create(**validated_data)

    def update(self, instance, validated_data):
        instance.name = validated_data.get("name", instance.name)
        instance.number = validated_data.get("number", instance.number)
        instance.save()
        return instance
```

6. Описать правила направлений по url в файлах urls.py проекта и приложения
Установить правило для обработки **http://<ваш IP адрес>:8000/view** (перенаправлять на правила, описанные в app/urls.py)

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('view/', include('app.urls')),
]
```

Установить правила в app/urls.py

```
from django.urls import path, include

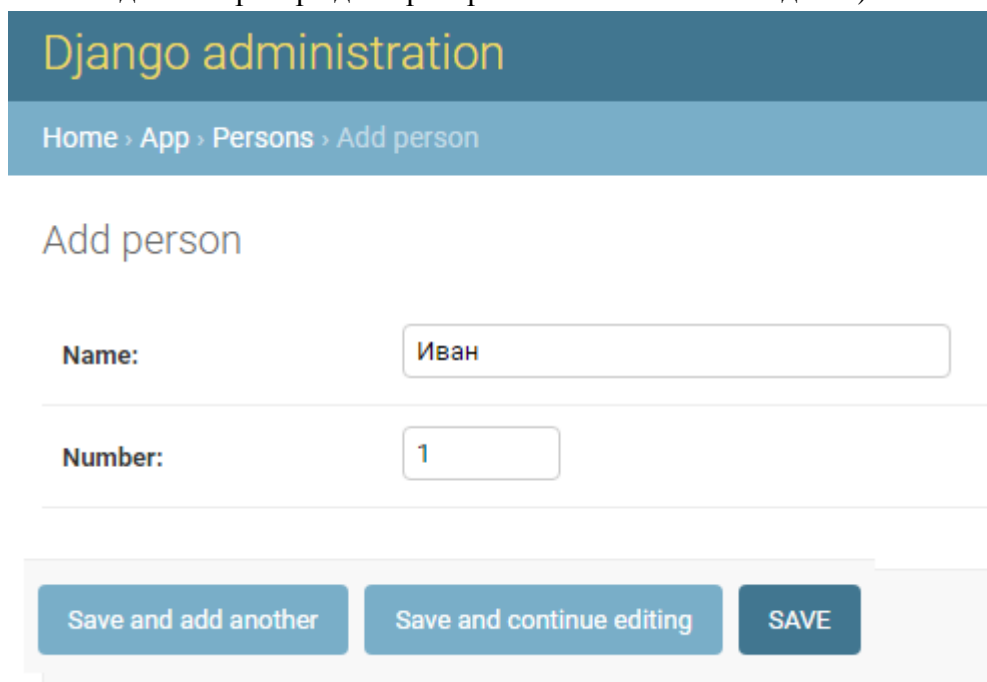
from .views import person_api_view

urlpatterns = [
    path('api/', person_api_view.as_view()),
    path('api/<int:number>', person_api_view.as_view()),
]
```

7. Проверка работы

Для тестирования работы API можно использовать chrome, но удобнее работать со специальным приложением chrome ARC (является бесплатным, легко устанавливается).

Создадим объект Person в панели администратора django (также далее можно использовать панель администратора для проверки выполнения команд API)



The screenshot shows the Django administration interface. At the top, there's a header 'Django administration' in yellow text on a dark blue background. Below it, a breadcrumb trail reads 'Home > App > Persons > Add person'. The main content area is titled 'Add person'. It contains two form fields: 'Name:' with the value 'Иван' and 'Number:' with the value '1'. At the bottom, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

Запрос GET

Request

MethodGETRequest URLhttp://130.193.35.185:8000/view/api/SEND

ParametersHeadersVariables

Header namecontent-typeHeader valueapplication/json

ADD HEADER

Headers are validHeaders size: 30 bytes

200 OK280.68 msDETAILS

```
{  - "persons": [Array[1]    - 0: {      "name": "Иван",      "number": 1    }  ],}
```

Запрос POST

Request

MethodPOSTRequest URLhttp://130.193.35.185:8000/view/api/SEND

ParametersHeadersBodyVariables

Body content typeapplication/jsonEditor viewRaw input

FORMAT JSONMINIFY JSON

```
{  "person":    {    "name": "Мария",    "number": 2  }}
```

200 OK281.91 ms

DETAILS

```
{  "success": "Person Мария saved"}
```

Запрос PUT

Request

Method

Request URL

PUT

http://130.193.35.185:8000/view/api/1

SEND

Parameters

Headers

Body

Variables

Body content type

application/json

Editor view

Raw input

FORMAT JSON

MINIFY JSON

```
{
  "person": {
    "name": "Игорь",
    "number": 1
  }
}
```

200 OK

295.39 ms

DETAILS

Copy

Download

Toggle source mode

Insert headers set

```
{
  "success": "Person Object Person, name: Игорь, number: 1 updated"
}
```

Запрос DELETE

Request

Method

Request URL

DELETE

http://130.193.35.185:8000/view/api/2

SEND

Parameters

Headers

Body

Variables

Copy

Download

Toggle source mode

Insert headers set

Header name

Header value

content-type

application/json

X

✎

ADD HEADER

✓ Headers are valid

Headers size: 30 bytes

200 OK

284.27 ms

DETAILS

Copy

Download

Toggle source mode

Insert headers set

```
{
  "success": "Person Мария deleted"
}
```

4. Полезные API для задач DataScience

Многие приложения предоставляют открытые интерфейсы для использования другими системами. Социальные сети (Google, Twitter, Facebook), глобальные технологические платформы (Amazon, Microsoft Azure, Yandex), узкоспециальные прикладные системы (gismeteo.ru) – дают инструменты использования своих данных (информация о пользователях), либо использования встроенных функций для обработки данных (распознавание и синтез речи, распознавание изображений). Ниже перечислены некоторые из имеющихся в распоряжении ресурсов.

- **Facebook API**

Facebook API предоставляет интерфейс доступа к объектам социальной сети Facebook: посты, комментарии, лайки, перепосты. Эта обширная информация дает возможность для анализа данных для социальной инженерии. Есть удобный инструмент Facebook Graph API для извлечения данных с помощью R и python.

Ссылки: https://developers.facebook.com/?locale=ru_RU

Описание: <https://developers.facebook.com/docs/graph-api>

- **Google Map API**

Google Map API – одно из наиболее часто используемых API (области применения – от сервисов заказа такси до игры Pokemon Go). Дает возможность получать информацию: координаты, расстояния между объектами, маршруты и т.п. Интересной возможностью является создание свойств расстояний, учитывающих данных.

Ссылки: <https://console.developers.google.com/apis/dashboard?pli=1>

Описание: <https://developers.google.com/maps/documentation/maps-static/intro>

Пример внедрения: <https://www.analyticsvidhya.com/blog/2015/03/hacking-google-maps-create-distance-features-model-applications/>

- **Twitter API**

Предоставляет доступ к данным: твиты, сделанные любым пользователем, твиты, содержащие отдельные термины или комбинации терминов, твиты, сделанные в определенный временной промежуток и т.п. Являются мощным инструментом в решении задач исследования социального мнения, настроения.

Описание: <https://developer.twitter.com/en/docs>

Пример практического использования: <https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/>

- **IBM Watson API**

IBM Watson предлагает набор из API для выполнения сложных задач, таких как анализ тона, преобразование документов, идентификация личности, визуальное распознавание, преобразование текста в голос и голоса в текст, и многие другие, с использованием нескольких строчек кода. Отличается от предыдущих тем, что предоставляет не инструменты доступа к данным, а обработки данных.

Описание: <https://www.ibm.com/watson>

- **Quandl API**

Quandl позволяет работать с временными рядами при анализе акций. Установка Quandl API очень проста и предоставляет ресурсы для решения задач анализа рынка акций.

Описание: <https://www.quandl.com/>

- **Yandex API**

Яндекс является одной из ведущих российских ИТ-компаний. В перечень разработок Яндекс входит множество сервисов для физических лиц и коммерческих компаний: заказ услуг (такси, еда), работа с картами, погода, распознавание и синтез речи и т.п.

Ссылка: <https://yandex.ru/dev/>

- **Gismeteo**

Ссылка <https://www.gismeteo.ru/api/>

5. Домашнее задание

Разработать проект, в котором обученная модель машинного обучения принимает через API данные и отдает на их основе прогноз.