API: REST и форматы обмена данными | Я.Шпора

API (от англ. **A**pplication **P**rogramming **I**nterface, «программный интерфейс приложения») — это интерфейс для обмена данными.

Консистентность и расширяемость в API

Консистентность — это согласованность данных друг с другом, их целостность и внутренняя непротиворечивость. Например, данные о каком-то объекте, полученные с одного эндпоинта, не должны отличаться от данных о том же объекте, полученных с другого эндпоинта.

Понятие консистентности включает в себя и идею **согласованности**: добавление в API новой функциональности не должно сломать API.

Расширяемость API означает возможность изменять и дополнять функциональность существующего интерфейса без нарушения совместимости с уже существующими клиентами.

JSON: формат обмена данными

По структуре JSON обычно похож на тип данных **dict**: это последовательность пар «ключ-значение»; как и словари, JSON поддерживает вложенность. Ключи в JSON пишутся строго **в двойных кавычках**.

Значениями ключей могут быть строки, числа, булевы значения, словари и списки. При этом, валидный JSON может быть списком, объектом, числом, строкой, булевым значением, null

```
{
  "name": "Captain America",
  "realName": "Steve Rogers",
  "yearCreated": 1941,
  "powers": [
    "Strength",
    "Healing ability"
  ]
}
```

XML: формат обмена данными

Внешне этот язык чем-то похож на HTML. Одно из основных отличий HTML от XML в том, что названия тегов в XML не стандартизированы, разработчик может давать имена тегам по собственному усмотрению.

Описание супер-героя из предыдущего примера в формате XML будет выглядеть так:

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <name>Captain America</name>
  <realName>Steve Rogers</realName>
  <yearCreated>1941</yearCreated>
  <powers>Strength</powers>
  <powers>Healing ability</powers>
</root>
```

Язык GraphQL

GraphQL — это язык запросов к API. У GraphQL есть три основные особенности:

- он позволяет клиенту точно указать, какие данные ему нужны;
- облегчает агрегацию данных из нескольких источников;
- использует систему типов для описания данных.

Технология gRPC

gRPC — это система удалённого вызова процедур. Основная специфика gRPC в том, что он оперирует данными в бинарном, а не в текстовом виде.

В gRPC клиентское приложение может напрямую вызывать метод серверного приложения на удалённом компьютере. Идеальный вариант использования gRPC — это общение между микросервисами и построение коммуникации между мобильным приложением и сервером.

Архитектура REST

REST (Representational State Transfer) — это архитектурный стиль, набор принципов взаимодействия компьютерных систем, основанный на методах протокола HTTP.

Одно из основных понятий в REST — **ресурс:** абстрактное понятие, под которым понимается «всё, чему можно дать имя в структуре сервиса». Ресурсом может быть что угодно, к чему разработчик REST API считает важным предоставить доступ клиенту.

Примеры ресурсов:

- Список открытых ошибок в базе данных ошибок.
- Каталог ресурсов, относящихся к медузам.
- Некоторая информация о медузах.

Принципы REST API

1. Клиент-сервер: разделение ответственности между клиентом и сервером.
2. Отсутствие состояния: сервер не хранит состояние.
3. Единый интерфейс.
4. Многоуровневость.
5. Кешируемость.
6. Код по запросу (необязательный принцип).

HTTP-методы

HTTP-метод запроса определяет действие, которое должен выполнить запрос:

- **GET** получает ресурсы;
- **POST** создаёт ресурс;
- **PUT** заменяет существующий ресурс целиком;
- **PATCH** частично изменяет существующий ресурс;
- **DELETE** удаляет ресурс;
- **HEAD** получает только заголовки ответа. HEAD похож на GET, но в ответе на этот запрос есть только заголовок, а тела ответа нет;
- **OPTIONS** получает перечень HTTP-методов, которые поддерживает ресурс.

У HTTP-методов есть дополнительные характеристики:

- **Безопасность:** если метод может изменить ресурс, то в терминах архитектуры REST он считается небезопасным (это касается методов PUT, PATCH, DELETE или POST).

- **Идемпотентность** — многократное выполнение метода возвращает тот же результат, что и однократное. Примерами таких методов являются GET, HEAD, OPTIONS, DELETE.

Запросы к API через библиотеку requests

Библиотека **requests** используется для отправки HTTP-запросов из кода на Python.

Установка библиотеки:

```
pip install requests
```

Метод get() в модуле requests

Метод `get()` модуля `requests` отправляет GET-запрос к указанному адресу и возвращает объект класса **Response**. Этот объект включает в себя полную информацию об ответе сервера.

Получить доступ к этим данным можно через атрибуты и методы класса `Response`. Например, содержимое ответа можно получить из атрибут `response.text`.

Преобразовать JSON-строку в тип данных, понятный Python, можно методом `json()` класса `Response`.

```
import requests

response = requests.get('https://swapi.dev/api/starships/9/')

# Приведем ответ сервера к типам данных Python...
response = response.json()
# ... и напечатаем его.
print(response)
# Напечатаем и тип данных объекта, полученного в результате преобр
print(type(response))
```

Для других типов HTTP-запросов есть методы `.post()`, `.patch()` и подобные.

Механизмы авторизации

Ко многим API доступ ограничен, и для контроля доступа используются механизмы авторизации.

Аутентификация — процедура проверки подлинности пользователя («подтвердите, что вы — Стас Басов»).

Авторизация — предоставление пользователю прав на выполнение каких-то действий («Стас Басов авторизован для просмотра страницы *homework*»).

OAuth — это схема авторизации, предоставляющая третьей стороне ограниченный доступ к ресурсам сервиса от имени определённого веб-сервиса. Вместо логина и пароля применяется **OAuth-токен**.

Токен — это уникальная строка из цифр и латинских букв, идентифицирующая пользователя.