Урок 14. Разработка API. Введение, что это, зачем и как работает. Виды библиотек для работы с API: Requests и FastAPI.

Теория:

Что такое API

API (application programming interface) – интерфейс программного обеспечения, чаще не имеющую пользовательский интерфейс.
API – это «прослойка» между приложением и удаленным сервером, и ее работа строится на обмене информацией между двумя сторонами: **клиентом** (приложение, которое запрашивает информацию) и **сервером** (программа, которая обрабатывает запрос и отправляет обратно нужную информацию).
Часто открытые API решения используются бизнесами для решения глобальных или локальных задач.
Примеры простых открытых API:

- Cat Facts — факты о котах
  GET https://catfact.ninja/fact *(без ключа)*
- Dog CEO — случайная картинка собаки
  GET https://dog.ceo/api/breeds/image/random *(без ключа)*
- Advice Slip — случайный совет
  GET https://api.adviceslip.com/advice *(без ключа)*
- Bored API — «чем заняться»
  GET https://www.boredapi.com/api/activity/ *(без ключа)*

Коммуникация с API происходит по средствам отправки HTTP-запросов, в том числе при помощи Python, а результат работы сервера API высылает в виде json-ответов. Правила взаимодействия и примеры запросов обычно описываются в документации к API.

Бибилиотека requests

requests — работает по принципу «**клиента**», это удобная библиотека для HTTP-запросов в Python. С её помощью можно делать GET/POST/..., читать **статус, заголовки и тело** ответа, разбирать **JSON-ответ** и обрабатывать ошибки.
Коротко о HTTP-запросах:
HTTP — это «правила общения» между клиентом (браузер/скрипт) и сервером.
Запрос содержит:

- **Метод**: что хотим сделать (GET, POST, PUT, DELETE, …).
- **URL**: адрес ресурса (/users/42?active=true).
- **Заголовки (Headers)**: метаданные (Authorization, Content-Type, Accept, …).
- **Тело (Body)**: данные (не у всех методов; чаще у POST/PUT/PATCH), обычно JSON.

Ответ содержит:

- **Код статуса** (200, 404, 500, …) + короткий текст.

- **Заголовки** (например, Content-Type: application/json).
- **Тело**: сами данные (HTML, JSON, файл и т.д.).

Основные запросы:

- **GET** — получить данные (безопасный; не меняет состояние).
- **POST** — создать/действие (отправляем данные в теле).
- **PUT** — полностью заменить ресурс по адресу.
- **PATCH** — частично изменить.
- **DELETE** — удалить.

Коды статусов (ответов):

- **2xx Успех**: 200 OK, 201 Created, 204 No Content.
- **3xx Перенаправления**: 301/302 (переезд ресурса), 304 Not Modified (кеш валиден).
- **4xx Ошибка клиента**:
  400 Bad Request (неверный формат), 401 Unauthorized (нужна аутентификация),
  403 Forbidden (нет прав), 404 Not Found (нет ресурса), 429 Too Many Requests (слишком часто).
- **5xx Ошибка сервера**: 500, 502, 503, 504.

Пример запросов в requests:

```
import requests

# GET
r = requests.get("https://api.adviceslip.com/advice", timeout=5)
# всегда ставь timeout
r.raise_for_status()              # выбросит исключение при
4xx/5xx
data = r.json()                   # распарсить JSON -> dict
print(data["slip"]["advice"])     # сам текст совета

# POST
payload = {"title": "Book", "price": 9.99, "userId": 1}
r = requests.post(
    "https://jsonplaceholder.typicode.com/posts",
    json=payload,            # requests сам поставит Content-Type:
application/json
    timeout=5
)
r.raise_for_status()
print(r.status_code)  # 201
print(r.json())       # ответ сервера (созданный «пост»)
```

- timeout=5 — не зависнем навсегда.
- raise_for_status() — ловим сетевые/HTTP-ошибки.
- .json() — превращает тело ответа (JSON) в словарь Python.

Пример основных методов:

```
r = requests.get("https://api.adviceslip.com/advice", timeout=5)
print(r.status_code)      # код статуса, например 200
print(r.headers.get("Content-Type"))  # тип данных (обычно
'application/json')
print(r.url)              # итоговый URL
```

```
print(r.elapsed.total_seconds(), "s") # время запроса
print(r.text[:80])        # «сырое» тело ответа (строка)

# Шпаргалка
requests.get(url, params={"q": "python"}, timeout=5)    # GET с
query
requests.post(url, json={"name": "Ann"}, timeout=5)     # POST с
JSON
r.ok             # True, если статус 200–399
r.status_code    # код ответа
r.headers        # заголовки
r.json()         # распарсить JSON
```

Несколько запросов:

Несколько запросов эффективнее через Session. Сессия переиспользует соединение (чуть быстрее и «дешевле» для сервера):

```
from requests import Session

with Session() as s:
    advices = []
    for _ in range(3):
        r = s.get("https://api.adviceslip.com/advice",
timeout=5)
        r.raise_for_status()
        advices.append(r.json()["slip"]["advice"])
    print("\n".join(advices))
```

Ошибки при использовании:

- Нет timeout → программа «висит». Всегда указывай timeout=....
- Не проверяешь статус → тихо обрабатываешь 404/500. Делаем raise_for_status().
- Ловишь только Exception → лучше requests.RequestException, чтобы отлавливать сетевые проблемы.
- Путаешь .text и .json() → .json() парсит JSON, .text даёт сырой текст.

Библиотека FastAPI

FastAPI — это «сервер»: помогает **быстро написать своё API** или обёртку над готовой логикой (валидация входа, красивые /docs, коды статусов, удобные схемы данных).

Пример, как обертка над существующим API

```
from fastapi import FastAPI
import requests

app = FastAPI(title="Advice wrapper")

@app.get("/advice")
def advice():
    r = requests.get("https://api.adviceslip.com/advice", timeout=5)
    data = r.json()
```

```
    # Берём текст совета и возвращаем в своём формате
    return {"advice": data["slip"]["advice"]}
```

Пример самостоятельного мини-сервера:

```
from fastapi import FastAPI

app = FastAPI(title="Mini server")

# Приветствие: /hello?name=Ann
@app.get("/hello")
def hello(name: str = "world"):
    return {"message": f"Hello, {name}!"}

# Сумма: /sum?a=2&b=3 -> {"sum": 5}
@app.get("/sum")
def sum_(a: int, b: int):
    return {"sum": a + b}
```

Самостоятельный план работы на урок:

```
pip install requests
pip install fastapi
```

1. **Совет дня**
   GET https://api.adviceslip.com/advice и выведи только текст совета, не забыть timeout, обработка ошибок (try/except, raise_for_status()).
2. **Три совета с сессией**
   Через requests.Session() получи 3 совета подряд и выведи их построчно, у каждого ответа напечатай status_code.
3. **Картинка собаки**
   GET https://dog.ceo/api/breeds/image/random → получить URL картинки → проверить статус 200 → получить и напечатать URL картинки.

4. **Аккуратные ошибки**

   GET запросить несуществующий ресурс
   https://jsonplaceholder.typicode.com/xyz.

   При 4xx/5xx печатать "<код> <текст>", но не падать.

5. **Мини-сервер FastApi**

- Что сделать: эндпоинт GET /ping → {"status":"ok"}.
- Критерий: открывается в браузере и в /docs.
- Что сделать: GET /hello?name=Ann → {"message":"Hello, Ann!"} (по умолчанию world).
- Критерий: параметр name опционален.
- Что сделать: GET /mult?a=2&b=3 → {"mult":5}; a и b — числа.
- Критерий: валидировать типы: при строках FastAPI сам вернёт 422.