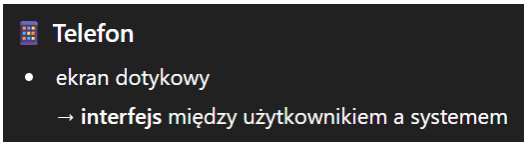

REST API – PUNKT KONTAKTU DLA PROGRAMÓW

---

1. API – (Application Programming Interface)

- interfejs dla programów, a nie dla ludzi

klient → wysyła zamówienie (`request`),

kelner (`API`) → przekazuje je do kuchni (`serwera`),

kuchnia → przygotowuje odpowiedź,

kelner → przynosi odpowiedź (`response`).

👉 **Protokół** to umowa: „jak ze sobą rozmawiamy”

To **zbiór zasad**, które mówią:
- kto mówi pierwszy,
- w jakiej formie,
- co oznacza odpowiedź

🧠 **Protokół mówi jak się komunikować, a API mówi co można zrobić w tej komunikacji.**

- oddziela **frontend** (to co widzi użytkownik np. `strona internetowa`)
  od **backendu** (to co dzieje się pod spodem np. `serwer`, `baza danych`),

- pozwala na integrację różnych systemów i aplikacji.

1. `frontend korzysta z API,`
2. `API przekazuje żądania do backendu,`
3. `backend nie musi wiedzieć, jak wygląda frontend`

---

2. REST – (Representational State Transfer)

- styl architektury, który określa, jak projektować API w prosty, czytelny i skalowalny sposób

- API zgodne z tymi zasadami nazywamy REST API

Zasady REST

1) Architektura klient-serwer

- Klient (np. `przeglądarka`, `aplikacja mobilna`) i serwer (`API`) są od siebie niezależne

- Korzyści:
  - Można rozwijać frontend i backend niezależnie

2) Bezstanowość (Stateless)

- Każde zapytanie HTTP musi zawierać wszystkie informacje potrzebne do jego obsługi.
  Serwer **nie pamięta stanu klienta** między zapytaniami

- Korzyści:
  - większa skalowalność,
  - prostsza architektura

3) Jednolity interfejs (Uniform Interface)

- Zasoby są dostępne pod adresami URL i obsługiwane przez standardowe metody http
- Składniki jednolitego interfejsu:
  - Identyfikacja zasobów:
    Każdy zasób ma unikalny URL (np. `/users/1`)
  - Manipulacja przez reprezentację:
    Dane przesyłane są w formacie JSON
  - Samoopisujące się komunikaty:
    Request i response zawierają wszystkie potrzebne informacje
- Korzyści:
  - `API` jest czytelne, przewidywalne i łatwe w użyciu

4) Podział na zasoby

- `REST` operuje na zasobach, a nie na akcjach
- Przykłady:
  - `GET /users` – lista użytkowników
  - `POST /users` – utworzenie użytkownika
  - `GET /users/1` – dane użytkownika o ID 1
- Korzyści:
  - spójność nazw,
  - łatwa dokumentacja API.

PODSUMOWANIE

✅ API to interfejs, który umożliwia komunikację między aplikacjami i pozwala im wymieniać dane w uporządkowany sposób

✅ REST API to API:

- działające zgodnie z zasadami REST,
- korzystające z protokołu HTTP,
- przesyłające dane najczęściej w formacie JSON.

Metoda	Znaczenie	Przykład
GET	Pobiera dane	`GET /users`
POST	Tworzy dane	`POST /users`
PUT	Modyfikuje dane	`PUT /users/1`
DELETE	Usuwane dane	`DELETE /users/1`

BIBLIOTEKA `REQUESTS`

- Biblioteka do wysyłania zapytań HTTP w Pythonie
- Obsługuje: `GET`, `POST`, `PUT`, `DELETE`
- Zalety:
  - bardzo prosta składnia,
  - czytelny kod,
  - łatwa obsługa nagłówków i danych JSON

1) Nagłówki HTTP (Headers)

🟥 Dodatkowe informacje przesyłane razem z zapytaniem lub odpowiedzią.
🟥 Znajdują się na początku request/response

🟥 Najczęstsze:
- **Content-Type** – format danych (np. `application/json`)
- **Authorization** – token lub klucz API (np. `Bearer <token>`)
- **Accept-Language** – oczekiwany język odpowiedzi
- **Cookies** – informacje o sesji użytkownika

🟥 Po co:
  - bezpieczeństwo i autoryzacja,
  - określenie formatu danych,
  - preferencje odpowiedzi.

2) Ciało zapytania (Body)

🟪 Zawiera właściwe dane, które są wysyłane lub odbierane.
🟪 Najczęściej w formacie JSON

🟪 Po co:
  - wysyłamy dane na serwer (`POST`, `PUT`),
  - przesyłamy większe ilości danych (np. formularze)

KODY ODPOWIEDZI HTTP

1xx – Informacyjne

[serwer otrzymał żądanie i je przetwarza]

- 100 Continue – serwer otrzymał początek i można kontynuować
- 101 Switching Protocols – akceptacja zmiany protokołu (np. http na WebSocket)

2xx – Sukces

[żądanie zostało pomyślnie odebrane, zrozumiane i przetworzone]

- 200 OK – wszystko git i odpowiedź serwera zawiera dane
- 201 Created – utworzono pomyślnie (po POST)
- 202 Accepted – przyjęte do przetworzenia, ale jeszcze robi
- 204 No Content – wszystko git, ale odpowiedź nie niesie danych

3xx – Przekierowania

[klient musi podjąć dodatkowe działania, aby uzyskać dostęp do żądanego zasobu]

- 301 Moved Permanently – przeniesiono (na stałe) pod nowy URL
- 302 Found – tymczasowo pod nowym URL
- 304 Not Modified – nie było zmian od ostatniego pobrania
- 307 Temporary Redirect – podobne do 302, ale metoda ta sama

4xx – Błędy klienta

[problem po stronie klienta]

- 400 Bad Request – błędne/niekompletne żądanie
- 401 Unauthorized – wymagane uwierzytelnienie (zalogowanie)
- 403 Forbidden – klient zalogowany, ale dostęp zabroniony
- 404 Not Found - nieznaleziono
- 405 Method Not Allowed – użyta metoda nie do tego zasobu

5xx – Błędy serwera

[problem po stronie serwera]

- 500 Internal Server Error – nieokreślony błąd
- 501 Not Implemented – nie obsługuje żądanej metody
- 502 Bad Gateway – zła odpowiedź od innego serwera
- 503 Service Unavailable – np. chwilowe przeciążenie
- 504 Gateway Timeout – brak odpowiedzi od innego serwera