REST API

A REST (Representational State Transfer) egy architekturális stílus hálózati alkalmazások tervezésére. A REST API olyan alkalmazásprogramozási interfész (API), amely a REST elveit követi, és a kommunikációhoz általában a HTTP protokollt használja.

A REST koncepcióját Roy Fielding vezette be 2000-ben doktori disszertációjában. Fontos különbség a JSON-RPC-hez képest, hogy a REST nem egy szigorúan definiált protokoll, hanem egy architekturális irányelvrendszer.

A REST erőforrás-orientált szemléletet használ. Ez azt jelenti, hogy a rendszer nem műveletek vagy függvényhívások köré szerveződik, hanem **erőforrások** köré. Egy erőforrás bármilyen entitást reprezentálhat, például egy felhasználót, terméket, rendelést vagy dokumentumot. Minden erőforrás egy egyedi URL segítségével azonosítható.

Például:

```
GET /users/15
GET /products/42
```

REST esetén a kliens nem közvetlenül függvényeket hív meg. Ehelyett a kliens erőforrásokon hajt végre műveleteket a szabványos HTTP metódusok segítségével.

A REST alapelvei

Erőforrások azonosítása

Minden erőforrásnak egyedi azonosítóval kell rendelkeznie, amely általában egy URL. Az URL az erőforrást jelöli, nem a végrehajtandó műveletet.

Helytelen (RPC-szerű) megközelítés:

```
GET /getUserById?id=10
```

REST szemléletben inkább az erőforrást azonosítjuk:

```
GET /users/10
```

HTTP metódusok (igék)

A REST a szabványos HTTP metódusokra épül a műveletek meghatározásához.

Method	Jelentés	Példa
GET	Erőforrás lekérdezése	GET /users/1
POST	Új erőforrás létrehozása	POST /users
PUT	Erőforrás teljes cseréje	PUT /users/1

Method	Jelentés	Példa
PATCH	Részleges módosítás	PATCH /users/1
DELETE	Erőforrás törlése	DELETE /users/1

Fontos, hogy a műveletet a **HTTP metódus** határozza meg, nem az URL.

Állapotmentesség (Statelessness)

A JSON-RPC-hez hasonlóan a REST is állapotmentes. Minden kérésnek tartalmaznia kell minden szükséges információt a feldolgozáshoz. A szerver alapértelmezés szerint nem tárol kliens oldali munkamenet állapotot a kérések között (kivéve ha ezt például tokenek vagy cookie-k segítségével külön megvalósítják).

Az állapotmentesség előnye, hogy javítja a rendszer skálázhatóságát és egyszerűbbé teszi az elosztott rendszerek működését.

Reprezentáció

Az erőforrások az úgynevezett **reprezentációk** formájában kerülnek átvitelre a kliens és a szerver között.

A leggyakoribb formátumok:

- JSON
- XML
- HTML
- Plain text

A modern REST API-k többsége **JSON formátumot** használ.

Példa válasz:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

REST vs JSON-RPC

Jellemző	REST	JSON-RPC
Architektúra	Erőforrás-alapú	Eljárás-alapú
Végpontok	Több endpoint	Általában egy endpoint
HTTP metódusok használata	Igen	Nem szükséges
Szabvány típusa	Architektúrális elvek	Protokoll specifikáció
Tipikus használat	Publikus web API-k	Belső API-k, blockchain

A REST API-kat gyakran használják nyilvános webes szolgáltatásoknál, míg a JSON-RPC gyakrabban jelenik meg belső rendszerekben vagy blokklánc interfészekben.

REST API szerver (Python + Flask)

Ebben a részben egy egyszerű REST API-t valósítunk meg Flask segítségével.

Függőség telepítése:

```
pip install flask
```

Hozzunk létre egy *server.py* fájlt.

```
from flask import Flask, jsonify, request

app = Flask(__name__)

users = [
    {"id": 1, "name": "Alice"},
    {"id": 2, "name": "Bob"},
]


@app.route("/users", methods=["GET"])
def get_users():
    return jsonify(users)


@app.route("/users/<int:user_id>", methods=["GET"])
def get_user(user_id):
    for user in users:
        if user["id"] == user_id:
            return jsonify(user)
    return jsonify({"error": "User not found"}), 404


@app.route("/users", methods=["POST"])
def create_user():
    new_user = request.get_json()
    new_user["id"] = len(users) + 1
    users.append(new_user)
    return jsonify(new_user), 201


@app.route("/users/<int:user_id>", methods=["DELETE"])
def delete_user(user_id):
    global users
```

```
    users = [u for u in users if u["id"] != user_id]
    return jsonify({"message": "User deleted"})


if __name__ == "__main__":
    app.run(port=5000)
```

A szerver indítása:

```
python server.py
```

Tesztelés curl segítségével

Összes felhasználó lekérdezése:

```
curl http://localhost:5000/users
```

Új felhasználó létrehozása:

```
curl -X POST http://localhost:5000/users \
-H "Content-Type: application/json" \
-d '{"name":"Charlie"}'
```

Felhasználó törlése:

```
curl -X DELETE http://localhost:5000/users/1
```

HTTP státuszkódok

A REST API-k kommunikációjában fontos szerepet játszanak a HTTP-státuszkódok.

Code	Jelentés
200	OK
201	Created
400	Bad request
401	Unauthorized
404	Not found
500	Server error

A státuszkódok a válasz fontos részét képezik és jelzik a kliens számára, hogy a kérés sikeres volt-e.

Előnyök és korlátok

A REST széles körben elterjedt, könnyen érthető és természetesen illeszkedik a HTTP-

infrastruktúrához. Támogatja a gyorsítótárazást (caching), a hitelesítési mechanizmusokat, valamint a különböző szabványos eszközöket és könyvtárakat.

Ugyanakkor a REST néha túl részletes lehet. Összetett műveletek esetén több külön endpoint használata válhat szükségessé, és nagy rendszerekben előfordulhat az úgynevezett **over-fetching** vagy **under-fetching**, amikor a kliens túl sok vagy túl kevés adatot kap egy kérésben.