# 1.1 API

- **API (Application Programming Interface)** = rozhraní pro komunikaci mezi programy
- Umožňuje jedné aplikaci používat **data nebo funkce** jiné aplikace bez znalosti její vnitřní logiky
- API se skládá z:
  - **endpointů**
  - **pravidel**, která určují, jak vypadá požadavek a odpověď

- API **není automaticky navázané na HTTP** (API může existovat i mimo HTTP)

# 1.1.1 API – endpoint

- **Endpoint** = adresa, kde se nachází zdroj nebo funkcionalita
- Příklad (HTTP, relativní URL):
  - /user/22
  - /date
  - /health

# 1.1.2 API – pravidla požadavku a odpovědi

- Dokumentace API musí jasně popsat:
  - **Jak má vypadat požadavek**
    :
    - HTTP metoda
    - parametry
    - formát
    - další pravidla

  - **Jak bude vypadat odpověď**
    :
    - jaké status kódy
    - jaký formát
    - jaké hlavičky
    - jaká data klient obdrží

---

# 1.2 REST

- **REST (Representational Stateless Transfer)** = architektonický styl pro návrh API
- Cíl: aby API bylo **konzistentní, jednoduché a dobře použitelné**

# 1.2.1 Zdroje (resources)

- **Zdroje** = data, která server nabízí
- Každý zdroj má svoji **vlastní cestu (relativní URL)**
- Pravidlo pojmenování: **podstatné jméno v množném čísle**
- Příklady:
  - správně: /users, /users/22, /orders
  - špatně: /getuser

# 1.2.2 CRUD (Create, Read, Update, Delete)

- Operace nad zdroji se mapují na HTTP metody (příklad pro resource /users ):
  - GET /users = seznam (list)
  - GET /users/22 = jeden uživatel
  - POST /users = vytvořit uživatele
  - PUT /users/22 nebo PATCH /users/22 = upravit uživatele
  - DELETE /users/22 = smazat uživatele

## 1.2.2.1 PUT vs PATCH

- Obě metody slouží k úpravě, rozdíl je v rozsahu:
  - **PUT** = nahrazení **celého** zdroje
  - **PATCH** = nahrazení **části** atributů

## 1.2.2.2 Status kódy

- Doporučené (standardní) status kódy:
  - GET /users  200
  - GET /users/22  200
  - POST /users  201
  - PUT/PATCH /users/22  200
  - DELETE /users/22  204

# 1.2.3 Reprezentace zdrojů

- **Zdroj** je uložen na serveru
- **Reprezentace** = jak server zdroj pošle klientovi (data, která klient dostane)
- Typické formáty:
  - JSON

- XML
- (API může podporovat i více formátů)

- **Content negotiation**
  = domluva na formátu:
  - klient posílá v hlavičce např.: Accept: application/json
  - server očekává data v určitém formátu např.: Content-Type: application/json

## 1.2.4 Stateless (bezstavovost)

- REST API je **stateless** = server si mezi požadavky **nepamatuje stav klienta/uřivatele**
- Každý request musí obsahovat všechny potřebné informace
- Příklad: přidání **autorizačního tokenu** ke každému požadavku

---

# 1.3 HATEOAS

- **HATEOAS (Hypermedia As The Engine Of Application State)** = pokročilý princip REST
- Myšlenka:
  - server v odpovědi přidává **odkazy na další akce**, které může klient udělat
  - klient se neřídí „natvrdo“ známými URL, ale tím, co mu server pošle v odkazech

- Výhody:
  - větší flexibilita
  - menší závislost klienta na konkrétní struktuře API

- Nevýhoda:
  - složitější implementace

## Příklad odpovědi

- Bez HATEOAS:
  - { "id": 10, "status": "created" }

- S HATEOAS:
  - odpověď obsahuje
    _links
    :
    - self = odkaz na sebe
    - pay = odkaz na zaplacení
    - cancel = odkaz na zrušení