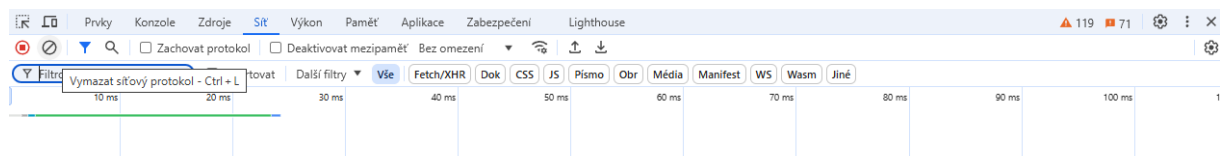


1API testování – Postman

- **Developerská konzole – Network (sít')**
- **Proč je důležité API testování?**
- **Typy API: REST, SOAP, GraphQL**
- **Klíčové pojmy pro API testování**
- **Formáty používané pro komunikaci v API**
- **Postman - Workspace, Collections, Requests**
- **Vytvoření a odeslání jednoduchého požadavku**
- **Autentizace a autorizace**
- **Swagger**

Developerská konzole – Network (sít')

Developerská konzole je nástroj integrovaný do většiny moderních webových prohlížečů (např. Chrome, Firefox, Edge), který vývojářům umožňuje analyzovat API volání mezi klientem (prohlížečem) a serverem. Je užitečná pro ladění, monitorování a optimalizaci API požadavků.



Název	Stav	Typ	Iniciátor	Velikost	Čas
PugMaster?sec=1&asyn=1&kdntuid=1&rnd=31710991&p=4...=0&gdpr=0&gd...	200	script	ads.pubmatic.com/AdServer/js/user_sync.h	39 B	47 ms
phoneDialogCheck?0.7892522465141139?service=homepage	200	fetch	login.js:35	711 B	37 ms
visibleImpress?r=1efb1688-8226-62b0-9304-013a65f39162&z=68658&i=1	200	gif	visibilityMeasure.js:49	84 B	18 ms
hit	200	xhr	hits:106	115 B	18 ms
hit	200	xhr	hits:106	115 B	59 ms
hit	200	fetch	hits:155	115 B	14 ms
40?r2=0.21814843616651491	200	jpeg	login.js:112	1.5 kB	42 ms
48?r2=0.3129934325955263	200	jpeg	login.js:112	1.7 kB	44 ms
api	101	websocket	login.js:128	0 B	169 ms
notifications	101	websocket	login.js:148	0 B	Nevyříženo
hit	200	xhr	hits:106	138 B	21 ms
graphql	200	fetch	client-bundle.es2017.min.js:1	1.7 kB	42 ms
hit	200	xhr	hits:106	115 B	21 ms
hit	200	xhr	hits:106	116 B	20 ms
hit	200	xhr	hits:106	115 B	20 ms
graphql	200	fetch	client-bundle.es2017.min.js:1	2.4 kB	61 ms
hit	200	xhr	hits:106	115 B	14 ms
hit	200	xhr	hits:106	120 B	14 ms
hit	200	xhr	hits:106	115 B	15 ms
hit	200	xhr	hits:106	117 B	15 ms
hit	200	xhr	hits:106	325 B	15 ms
hit	200	xhr	hits:106	115 B	13 ms
hit	200	xhr	hits:106	93 B	17 ms
6710bb4b3dac956101eba05a?fi=mdk;ca596730 spl2,7	200	xhr	sznplayer.common.es2022.bundle.min.js:1	(mezipaměť na disku)	1 ms
41419533?fi=mdk;f042067c spl2,7	200	xhr	sznplayer.common.es2022.bundle.min.js:1	(mezipaměť na disku)	1 ms

Monitorování požadavků

- Konzole zaznamenává každý požadavek na server.
- Co sledovat:
 - Metoda: Jaký typ požadavku byl odeslán (GET, POST, PUT, DELETE).
 - URL: Endpoint, na který byl požadavek odeslán.
 - Status code: Odpověď serveru (např. 200, 404, 500).
 - Čas: Doba zpracování požadavku.

Analýza odpovědi

- Kliknutím na konkrétní požadavek zobrazíš podrobnosti o odpovědi:
 - Headers (Hlavičky): Informace o požadavku a odpovědi (např. Content-Type, Authorization).
 - Response (Odpověď): Obsah odpovědi, obvykle ve formátu JSON nebo XML.
 - Cookies: Odeslané nebo přijaté soubory cookie.
 - Timing: Časová analýza (DNS lookup, připojení, čekání na odpověď).

Proč je důležité API testování?

1. Ověření funkčnosti API

- **API je mostem mezi různými systémy:** Testování zajišťuje, že API správně zprostředkovává požadované informace a provádí požadované akce.
- **Zajištění spolehlivosti:**
 - Správné zpracování požadavků (např. GET vrací data, POST vytvoří záznam).
 - Validace vstupů a výstupů (např. správný formát JSON/XML).
- **Příklad:** Ověření, že volání API `GET /users` vrátí seznam uživatelů s požadovanými atributy.

2. Detekce chyb na backendu

- **Odhalování problémů dříve, než je objeví uživatelé:**
 - Chyby v logice (např. výpočet ceny je špatný).
 - Nesprávné zpracování výjimečných situací (např. odpověď na neexistující záznam).
- **Ověření stability:**
 - Jak API reaguje na neplatné požadavky.
 - Jak se API chová při vysoké zátěži.
- **Příklad:** Testování, jak API reaguje na špatný vstup (např. `POST /users` s chybějícím atributem `email`).

3. Automatizace a úspora času

- **Automatizované testy umožňují:**
 - Rychlé opakování testů během vývoje.
 - Integraci s CI/CD pipeline pro nepřetržité ověřování.
- **Úspora času a zdrojů:**
 - Ruční testování API by bylo časově náročné.
 - Automatizace umožňuje rychle prověřit změny v kódu.
- **Příklad:** Po přidání nové funkcionality se automatizovaný test ihned spustí a ověří, že stávající požadavky stále fungují.

Praktický příklad výhod testování:

- **Scénář bez testování:**
 - Uživatel si stěžuje, že aplikace nezobrazuje data o produktech.
 - Vývojář zjistí, že API vrací špatný status kód (500).
 - Náprava trvá několik hodin nebo dní.
- **Scénář s testováním:**
 - Automatizovaný test zachytí chybu v API během vývoje.
 - Chyba je opravena před nasazením do produkce.
 - Uživatelé si chyby vůbec nevšimnou.

Typy API: REST, SOAP, GraphQL

1. REST (Representational State Transfer)

- **Charakteristika:**
 - Architektonický styl, ne protokol.
 - Využívá standardní HTTP metody: GET, POST, PUT, DELETE.
 - Data se přenášejí ve formátech: JSON, XML, případně CSV.
 - Založeno na zdrojích (resources), každý zdroj má unikátní URL (endpoint).
- **Výhody:**
 - Jednoduchost a snadná implementace.
 - Široká podpora v nástrojích a knihovnách.
 - Flexibilní přenos dat (např. JSON je čitelný pro lidi i stroje).
- **Nevýhody:**
 - Závislost na HTTP, což nemusí být vhodné pro všechny případy.
 - Může být méně efektivní při složitých dotazech (vyžaduje více požadavků).
- **Příklady použití:**
 - Veřejná API pro aplikace jako GitHub, Twitter, Google Maps.

2. SOAP (Simple Object Access Protocol)

- **Charakteristika:**
 - Protokol pro komunikaci mezi aplikacemi.
 - Využívá XML pro formátování zpráv.
 - Obsahuje přísně definovaný standard (WS-Security, ACID compliance).
- **Výhody:**
 - Bezpečnější díky integrované podpoře pro WS-Security.
 - Vhodné pro složité transakce a podnikové aplikace.
 - Podporuje různé transportní protokoly (HTTP, SMTP, TCP).
- **Nevýhody:**
 - Komplexnost (XML je těžkopádnější než JSON).
 - Méně populární pro moderní webové aplikace.
- **Příklady použití:**
 - Bankovní aplikace, platební systémy, ERP systémy.

3. GraphQL

- **Charakteristika:**
 - Dotazovací jazyk pro API, vyvinutý společností Facebook.
 - Uživatel definuje, jaká data potřebuje, a server poskytuje přesně to.
 - Data se přenášejí ve formátu JSON.
- **Výhody:**
 - Efektivní při dotazování na složitá data (žádné zbytečné přenosy).
 - Jediný endpoint pro všechny operace.
 - Vysoká flexibilita pro vývojáře frontendů.
- **Nevýhody:**
 - Složitější implementace na straně serveru.
 - Vyžaduje pokročilé znalosti pro optimalizaci výkonu.
- **Příklady použití:**
 - Facebook API, GitHub API, Shopify API.

Klíčové pojmy pro API testování

1. Endpoint

- **Definice:**
Jedinečná adresa (URL), na kterou se klient (např. Postman nebo frontend aplikace) připojuje, aby komunikoval s API.
 - Např.: `https://api.example.com/users`.
- **Struktura:**
 - **Base URL:** Hlavní adresa API (např. `https://api.example.com`).
 - **Path:** Konkrétní zdroj (např. `/users` nebo `/users/{id}`).

Ukázka vytvoření endpointu na backendu:

```
/// <summary>
/// Get All Companies endpoint
/// </summary>
/// <returns></returns>
[HttpGet("get-all-companies")]
public async Task<IEnumerable<CompanyValModelLight>> GetCompanyInformation()
{
    return await _companyRepository.Get();
}
```

Ukázka získání dat z databáze:

```
public async Task<IEnumerable<CompanyValModelLight>> Get()
{
    return await _context.CompanyInformation.ToListAsync();
}
```

2. Request (Požadavek)

- **Definice:**
Požadavek, který klient odesílá serveru, aby získal data nebo provedl určitou akci.
- **Obsah požadavku:**
 - **HTTP metoda:** Jakou akci má server provést (GET, POST, PUT, DELETE).
 - **Headers:** Metadata požadavku, např. Authorization pro tokeny.
 - **Body:** Data, která se odesílají (např. JSON při POST nebo PUT).
 - **Parameters:** Dotazovací parametry (`?id=123`).

3. Response (Odpověď)

- **Definice:**
Odpověď, kterou server vrací klientovi po zpracování požadavku.
- **Obsah odpovědi:**
 - **Status code:** Indikuje výsledek požadavku (např. 200, 404).
 - **Headers:** Metadata odpovědi (např. typ obsahu Content-Type: application/json).
 - **Body:** Data vrácená serverem (ve formátu JSON, XML apod.).

```
Body Cookies Headers (15) Test Results (2/5) 200 OK 631 ms 875 B Save Response
Pretty Raw Preview Visualize JSON
1 {
2   "Title": "Terminator",
3   "Year": "1991",
4   "Rated": "N/A",
5   "Released": "N/A",
6   "Runtime": "99 min",
7   "Genre": "Short, Action, Sci-Fi",
8   "Director": "Ben Hernandez",
9   "Writer": "James Cameron, Ben Hernandez",
10  "Actors": "Loris Basso, James Callahan, Debbie Medows",
11  "Plot": "A cyborg comes from the future, to kill a girl named Sarah Lee.",
12  "Language": "English",
13  "Country": "United States",
14  "Awards": "N/A",
15  "Poster": "N/A",
16  "Ratings": [
17    {
18      "Source": "Internet Movie Database",
19      "Value": "6.4/10"
20    }
21  ],
22  "Metascore": "N/A",
23  "ImdbRating": "6.1",
24  "ImdbVotes": "42",
25  "Trailer": "https://www.youtube.com/watch?v=..."
}
```

4. HTTP metody (CRUD operace)

HTTP (HyperText Transfer Protocol) má několik metod, které definují různé typy operací, které lze provést nad zdroji na webovém serveru. Zde jsou všechny běžně používané HTTP metody:

GET

- **Popis:** Načte reprezentaci specifikovaného zdroje.
- **Použití:** Používá se pro získání dat ze serveru.
- **Bezpečná a idempotentní:** Ano, nevede ke změně stavu serveru a opakované volání má stejný účinek.

POST

- **Popis:** Odesílá data ke zpracování na určený zdroj.
- **Použití:** Používá se pro odeslání dat na server (např. formuláře, nahrávání souborů).
- **Bezpečná a idempotentní:** Ne, může vést ke změně stavu serveru a opakované volání může mít různé účinky.

PUT

- **Popis:** Nahrazuje všechny aktuální reprezentace zdroje nahranými daty.
- **Použití:** Používá se pro aktualizaci nebo vytvoření zdroje.
- **Bezpečná a idempotentní:** Idempotentní, opakované volání má stejný účinek.

DELETE

- **Popis:** Odstraní specifikovaný zdroj.
- **Použití:** Používá se pro smazání zdroje ze serveru.
- **Bezpečná a idempotentní:** Idempotentní, opakované volání má stejný účinek.

PATCH

- **Popis:** Aplikuje částečné úpravy na zdroj.
- **Použití:** Používá se pro částečnou aktualizaci zdroje.
- **Bezpečná a idempotentní:** Ne vždy je idempotentní, záleží na implementaci.

HEAD

- **Popis:** Načte hlavičky odpovědi stejné jako metoda GET, ale bez těla odpovědi.
- **Použití:** Používá se pro kontrolu toho, co by GET požadavek vrátil, bez přenosu samotného obsahu.

- Bezpečná a idempotentní: Ano.

OPTIONS

- Popis: Vrací možnosti komunikace s daným zdrojem.
- Použití: Používá se pro určení metod, které jsou podporovány zdrojem.
- Bezpečná a idempotentní: Ano.

CONNECT

- Popis: Zahajuje tunel, který se obvykle používá pro proxy servery.
- Použití: Používá se pro zahájení dvoucestného spojení (tunelu) s cílovým zdrojem.
- Bezpečná a idempotentní: Ne.

TRACE

- Popis: Provede test zpětného volání, který ukazuje, jaké změny provedly servery v cestě požadavku.
- Použití: Používá se pro ladění nebo diagnostiku, sleduje cestu požadavku až k cílovému zdroji.
- Bezpečná a idempotentní: Ano.

Každá z těchto metod má specifický účel a vhodnost použití závisí na požadovaném typu operace nad zdroji v rámci webové služby.

Metoda	Popis	Příklad
GET	Získání dat	GET /users – seznam uživatelů.
POST	Vytvoření nového záznamu	POST /users – vytvoření nového uživatele.
PUT	Aktualizace záznamu	PUT /users/1 – úprava uživatele s ID 1.
DELETE	Smazání záznamu	DELETE /users/1 – smazání uživatele s ID 1.

5. Status codes

- Čísla, která indikují výsledek požadavku:

Skupina	Popis	Příklad kódů
2xx	Úspěch	200 (OK), 201 (Created)
3xx	Přesměrování	301 (Moved Permanently), 304 (Not Modified)
4xx	Chyba na straně klienta	400 (Bad Request), 404 (Not Found)
5xx	Chyba na straně serveru	500 (Internal Server Error), 503 (Service Unavailable)

Nejznámější status codes:

1xx: Informativní odpovědi

Používají se méně často a naznačují, že server přijal požadavek a klient může pokračovat.

- 100 Continue: Klient může pokračovat s požadavkem.
- 101 Switching Protocols: Server přepíná protokoly (např. z HTTP na WebSocket).

2xx: Úspěšné odpovědi

Naznačují, že požadavek byl úspěšně přijat, zpracován a server odpověděl.

- 200 OK: Požadavek byl úspěšně zpracován.
- Typické pro GET nebo POST požadavky.
- 201 Created: Úspěšně vytvořen nový zdroj (např. při POST požadavku).

- 204 No Content: Požadavek byl úspěšný, ale server nevrací žádný obsah.

3xx: Přesměrování

Naznačují, že klient musí provést další akci, aby dokončil požadavek.

- 301 Moved Permanently: URL zdroje byla trvale přesunuta na novou adresu.
- 302 Found: Zdroj byl dočasně přesunut na jinou adresu.
- 304 Not Modified: Klient může použít uloženou (kešovanou) verzi zdroje.

4xx: Chyby na straně klienta

Označují, že problém byl na straně klienta, například špatně formulovaný požadavek.

- 400 Bad Request: Požadavek obsahuje chybu (např. špatný formát JSON).
- 401 Unauthorized: Klient není autorizován. Obvykle vyžaduje přihlášení nebo token.
- 403 Forbidden: Přístup k požadovanému zdroji je zakázán, i když je klient autorizován.
- 404 Not Found: Požadovaný zdroj nebyl nalezen.
- 429 Too Many Requests: Klient překročil limit požadavků (rate limit).

5xx: Chyby na straně serveru

Znamenají, že server narazil na problém při zpracování požadavku.

- 500 Internal Server Error: Obecná chyba na serveru.
- 501 Not Implemented: Server nepodporuje požadovanou metodu.
- 502 Bad Gateway: Server obdržel neplatnou odpověď od jiného serveru.
- 503 Service Unavailable: Server je dočasně nedostupný (např. kvůli údržbě).
- 504 Gateway Timeout: Server neobdržel odpověď včas od jiného serveru.

Formáty používané pro komunikaci v API

Moderní API používají standardní formáty dat k výměně informací mezi klientem a serverem. Nejčastějšími jsou **JSON** a **XML**, přičemž každý má své výhody a omezení.

1. JSON (JavaScript Object Notation)

- **Definice:** Jednoduchý, lehký a čitelný formát, inspirovaný objekty v JavaScriptu. Používá se hlavně pro REST API díky své jednoduchosti a menší režii.
- **Struktura:**
 - Data jsou reprezentována jako páry klíč–hodnota.
 - Klíče musí být uzavřeny v uvozovkách.
 - Hodnoty mohou být texty, čísla, pole, objekty, boolean nebo null.
- **Příklad JSON:**

```
{
  "userId": 1,
  "username": "johndoe",
  "isActive": true,
  "roles": ["admin", "editor"],
  "profile": {
    "age": 30,
    "email": "johndoe@example.com"
  }
}
```

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        { "value": "New", "onclick": "CreateNewDoc()" },
        { "value": "Open", "onclick": "OpenDoc()" },
        { "value": "Close", "onclick": "CloseDoc()" }
      ]
    }
  }
}
```

- **Výhody JSON:**
 - Jednoduchost: Snadno čitelný pro lidi i stroje.
 - Lehkost: Menší velikost dat v porovnání s XML.
 - Podpora: Nativně podporován v moderních programovacích jazycích a knihovnách.
- **Nevýhody JSON:**
 - Nepodporuje atributy (oproti XML).
 - Není tak striktně definovaný (např. není schéma jako u XML).

2. XML (eXtensible Markup Language)

- **Definice:**
Značkovací jazyk používaný pro strukturovanou výměnu dat, často v SOAP API. Je robustnější a přísněji definovaný než JSON.
- **Struktura:**
 - Data jsou reprezentována pomocí značek (tags), které obsahují atributy a hodnoty.
 - Vždy musí být správně uzavřené (otevírací a zavírací značka).
- **Příklad XML:**

```
<user>
  <userId>1</userId>
  <username>johndoe</username>
  <isActive>true</isActive>
  <roles>
    <role>admin</role>
    <role>editor</role>
  </roles>
  <profile>
    <age>30</age>
    <email>johndoe@example.com</email>
  </profile>
</user>
```

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

- **Výhody XML:**
 - Flexibilita: Podpora atributů a komentářů.
 - Validace: Možnost definovat schéma (XSD), které zaručuje strukturu dat.
 - Robustnost: Lepší pro složitá data.
- **Nevýhody XML:**
 - Velikost: Větší datová režie než JSON kvůli značkám.
 - Komplexita: Hůře čitelný a zpracovatelný pro lidi.

3. Plain Text

- Použití: Jednoduché odpovědi nebo chyby (např. vrácení řetězce OK nebo Error: Not Found).
- Výhoda: Snadná implementace.
- Nevýhoda: Nevhodný pro komplexní struktury dat.

4. YAML (Yet Another Markup Language)

- Definice: Přehledný formát podobný JSON, často používaný pro konfigurace.

Instalace a přehled rozhraní Postmanu

1. Instalace Postmanu

- **Stažení:**
 - Jdi na oficiální stránku Postman.
 - Vyber verzi pro svůj operační systém (Windows, Mac, Linux).
- **Instalace:**
 - Spusť instalační balíček a postupuj podle pokynů.
- **Přihlášení (volitelné):**
 - Vytvoř si účet Postman (pro synchronizaci kolekcí mezi zařízeními).

2. Přehled rozhraní Postmanu

Komponenta	Popis
Workspace	Prostředí, kde organizuješ své projekty, kolekce a testy.
Collections	Sady požadavků seskupené do logických celků (např. pro konkrétní API).
Requests	Jednotlivé HTTP požadavky (např. GET, POST).
Tabs	Místo, kde se pracuje s jednotlivými požadavky a jejich odpověďmi.
Environment	Proměnné pro různé prostředí (např. Dev, Test, Prod).
Console	Pro ladění požadavků a zobrazení detailů odeslaných požadavků.

Postman - Workspace, Collections, Requests

1. Workspace

- **Účel:** Uspořádání práce podle projektů.
- **Vytvoření Workspace:**
 1. Klikni na **Workspaces** v levém horním rohu.
 2. Vyber **Create Workspace**.
 3. Pojmenuj svůj workspace a zadej popis.

2. Collections

- **Účel:** Organizace požadavků do skupin (např. CRUD operace na jednom API).
- **Vytvoření kolekce:**
 1. Klikni na **Collections** v levé části.
 2. Klikni na **+ New Collection**.
 3. Pojmenuj kolekci.

3. Requests

- **Účel:** Vytvoření a odeslání jednotlivých požadavků.
- **Vytvoření požadavku:**
 1. Klikni na **+ New Request**.
 2. Zadej název požadavku.
 3. Vyber metodu (GET, POST, atd.) a zadej URL endpoint.

Vytvoření a odeslání jednoduchého požadavku

- **Cíl:** Odeslat GET požadavek na veřejné API.
1. **Vytvoření požadavku:**
 - Klikni na **+ New Request**.
 - Nastav metodu na **GET**.
 - Do pole **Enter request URL** vlož URL:
`https://jsonplaceholder.typicode.com/posts.`
 2. **Odeslání požadavku:**
 - Klikni na **Send**.

https://api.thecatapi.com/v1/images/search

GET `https://api.thecatapi.com/v1/images/search` Send

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Seznámení s Response: Headers, Body, Status code

1. Headers

- **Definice:** Metadata odpovědi.
- **Příklad:**
 - `Content-Type: application/json` (odpověď je ve formátu JSON).
 - `Date: Mon, 02 Dec 2024 10:00:00 GMT` (čas odpovědi).

2. Body

- **Definice:** Hlavní obsah odpovědi (např. data z API).
- **Příklad JSON odpovědi:**

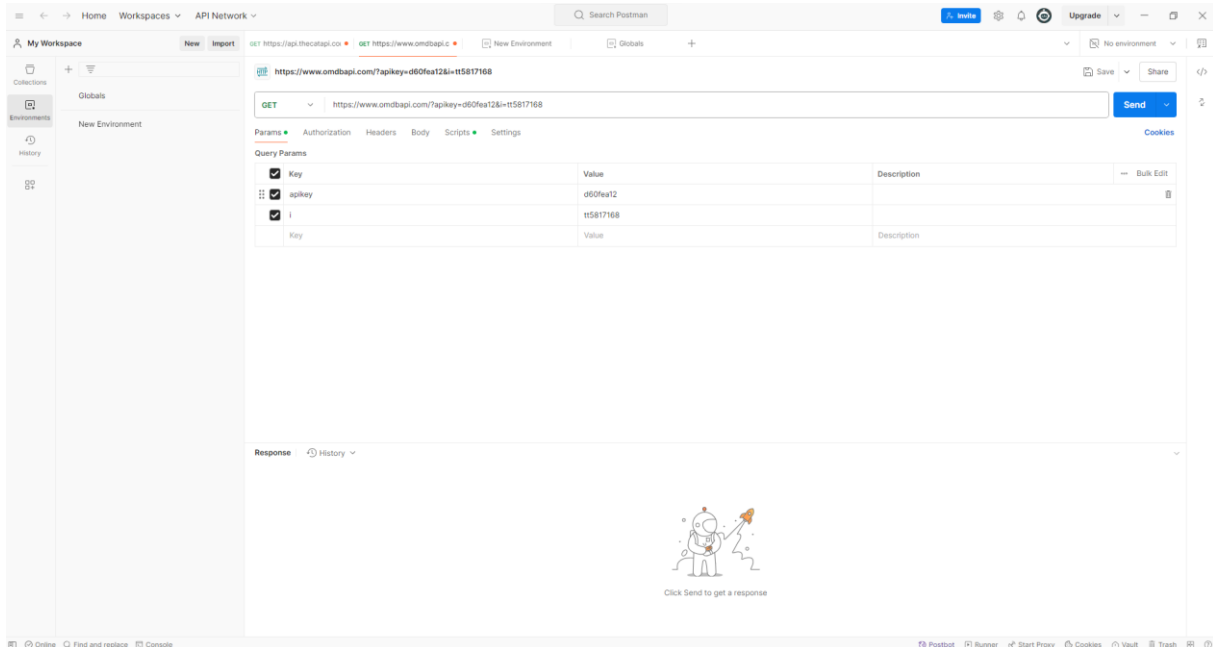
```
json
Zkopírovat kód
{
  "userId": 1,
  "id": 1,
  "title": "Sample Title",
  "body": "This is a sample post body."
}
```

3. Status code

- **Definice:** Indikuje výsledek požadavku.
- **Příklad:**
 - **200 (OK):** Požadavek byl úspěšně zpracován.
 - **404 (Not Found):** Endpoint neexistuje.
 - **500 (Internal Server Error):** Chyba na straně serveru.

Praktické cvičení

1. **Vytvoř workspace:** Pojmenuj ho „API Testování“.
2. **Vytvoř kolekci:** Pojmenuj ji „Veřejná API“.
3. **Přidej GET požadavek:** Na `https://jsonplaceholder.typicode.com/posts`.
4. **Analyzuj odpověď:** Zjisti, jaké hodnoty obsahují Headers, Body, a Status code.



Příklady:

<https://api.thecatapi.com/v1/images/search>

<https://thedogapi.com/>

<https://api.chucknorris.io/>

<https://pokeapi.co/>

<https://docs.spacexdata.com/>

<https://restcountries.com/#api-endpoints-using-this-project>

<https://github.com/public-apis/public-apis?tab=readme-ov-file>

<https://jsonplaceholder.typicode.com/>

<https://jsonplaceholder.typicode.com/users>

API key:

<https://www.omdbapi.com/?apikey=d60fea12&i=tt5817168>

<https://home.openweathermap.org/>

Autentizace a autorizace

Autentizace a autorizace jsou klíčové koncepty při zabezpečení API. Oba pojmy se často zaměňují, ale mají různé významy:

Autentizace (Authentication):

- Proces, kterým systém ověřuje identitu uživatele nebo aplikace.
- Příklad: Přihlášení uživatele pomocí uživatelského jména a hesla.

Autorizace (Authorization):

- Určuje, k jakým zdrojům nebo akcím má ověřený uživatel přístup.
- Příklad: Uživatel s rolí "admin" má přístup k API endpointu pro správu uživatelů.

1. API tokeny (Bearer token)

- API token je jedinečný řetězec vygenerovaný serverem, který reprezentuje oprávnění uživatele nebo aplikace.
- Typicky se posílá v hlavičce požadavku.
- **Použití:**
 - Po autentizaci server vrátí token, který klient ukládá (např. v lokálním úložišti nebo cookies).
 - Klient pak při každém požadavku na API zahrne token do hlavičky pro ověření.

2. Basic Auth

- Klient posílá uživatelské jméno a heslo zakódované pomocí Base64.
- Méně bezpečné, pokud není komunikace šifrována (např. přes HTTPS).

3. OAuth 2.0

- Bezpečný standard pro autorizaci, který umožňuje aplikacím přístup k datům uživatelů bez sdílení jejich přihlašovacích údajů.
- Používá přístupové tokeny (access tokens).
- Typické v aplikacích třetích stran, např. přihlášení přes Google nebo Facebook.

4. JWT (JSON Web Token)

- Samostatný token ve formátu JSON, který obsahuje informace o uživateli a jeho oprávněních.
- Skládá se ze tří částí: Header, Payload a Signature.

Swagger

Swagger je open-source nástroj a ekosystém pro návrh, dokumentaci, testování a správu REST API. Používá **OpenAPI Specification (OAS)** jako standard pro popis API. Swagger umožňuje vývojářům a uživatelům snadno porozumět a interagovat s API bez nutnosti zkoumání zdrojového kódu.

Generování dokumentace:

- Swagger generuje čitelnou a interaktivní dokumentaci API přímo z jeho specifikace.

Testování API:

- Díky Swagger UI lze odesílat požadavky přímo z prohlížeče (GET, POST, PUT, DELETE apod.).

Standardizace:

- Používá jednotný formát (OpenAPI) pro popis endpointů, parametrů, odpovědí a dalších detailů.

Příklad:

<https://petstore.swagger.io/>

<https://catfact.ninja/#/>

