

# POSTMAN ŠKOLENÍ

## OBSAH

|  |           |
|--|-----------|
| <b><i>Obsah</i></b> .....                | <b>1</b>  |
| <b><i>Úvod</i></b> .....                 | <b>3</b>  |
| <b>O přednášejícím</b> .....             | <b>3</b>  |
| <b>Rest API</b> .....                    | <b>3</b>  |
| Co je to Rest API .....                  | 3         |
| předávání dat v REST API.....            | 3         |
| <b>HTTP Requesty</b> .....               | <b>4</b>  |
| HTTP statusy .....                       | 4         |
| <b>Postman</b> .....                     | <b>5</b>  |
| <b>Co je to postman</b> .....            | <b>5</b>  |
| <b>První spuštění Postman</b> .....      | <b>5</b>  |
| Registrace Postman.com .....             | 5         |
| Přihlášení do Postman .....              | 5         |
| <b>Základní orientace</b> .....          | <b>7</b>  |
| Přidání prvku.....                       | 8         |
| Filtr.....                               | 9         |
| Settings.....                            | 9         |
| Console .....                            | 10        |
| <b>Requesty</b> .....                    | <b>10</b> |
| <b>Orientace v collections</b> .....     | <b>11</b> |
| Postup vytvoření kolekce .....           | 11        |
| Import, export kolekcí.....              | 11        |
| Orientace ve Složkách.....               | 14        |
| <b>HTTP Requesty</b> .....               | <b>16</b> |
| Vytvoření a provolání requestu.....      | 16        |
| URL .....                                | 25        |
| Typ HTTP callu .....                     | 26        |
| Params.....                              | 29        |
| Auth.....                                | 29        |
| Headers .....                            | 44        |
| Settings.....                            | 45        |
| Data v Requestech.....                   | 46        |
| GraphQL .....                            | 52        |
| SOAP .....                               | 54        |
| <b>Environments a proměnné</b> .....     | <b>55</b> |
| <b>Priority použití proměnných</b> ..... | <b>56</b> |
| <b>Nastavení proměnných</b> .....        | <b>56</b> |
| Initial value .....                      | 56        |
| Current value.....                       | 56        |
| Persist all/Reset All.....               | 56        |
| <b>Globals</b> .....                     | <b>57</b> |

|   |           |
|---|-----------|
| Import globals.....                         | 57        |
| Export globals.....                         | 59        |
| <b>Collection</b> .....                     | <b>60</b> |
| <b>Environment</b> .....                    | <b>60</b> |
| Vytvoření prostředí .....                   | 60        |
| Import proměnných pro prostředí .....       | 62        |
| Export proměnných pro prostředí .....       | 63        |
| <b>Data</b> .....                           | <b>64</b> |
| <b>Local</b> .....                          | <b>64</b> |
| <b>Set as variable</b> .....                | <b>64</b> |
| <b>Použití proměnných a prostředí</b> ..... | <b>64</b> |
| <b>Testování v postmanu</b> .....           | <b>65</b> |
| <b>Úvod do testování v postmanu</b> .....   | <b>65</b> |
| Javascript .....                            | 65        |
| Snippets.....                               | 65        |
| Externí knihovny .....                      | 65        |
| <b>Typy testů</b> .....                     | <b>65</b> |
| Nastavení proměnných ve skriptech .....     | 65        |
| Pre-request script .....                    | 69        |
| Tests .....                                 | 70        |
| <b>Psaní testů v Postmanovi</b> .....       | <b>71</b> |
| <b>Snippety</b> .....                       | <b>71</b> |
| Vysvětlení a použití .....                  | 71        |
| <b>Javascript testy</b> .....               | <b>73</b> |
| Kontrola statusů .....                      | 73        |
| Kontrola typu .....                         | 74        |
| Parse JSON .....                            | 74        |
| Kontrola dat .....                          | 74        |
| <b>Skriptování v postmanovi</b> .....       | <b>76</b> |
| Dynamické proměnné .....                    | 76        |
| Práce s Timestamp (BONUS).....              | 78        |
| <b>Debugging</b> .....                      | <b>79</b> |
| Debugging requestu .....                    | 79        |
| Debugging testů .....                       | 80        |
| <b>Přepoužívání kódu (BONUS hack)</b> ..... | <b>81</b> |
| <b>Collection Runner</b> .....              | <b>83</b> |
| <b>Ovládání Runneru</b> .....               | <b>83</b> |
| Konfigurace runneru .....                   | 84        |
| <b>Obecné spuštění</b> .....                | <b>85</b> |
| ENV Proměnné pro rohlik.cz.....             | 88        |
| Spuštění připravené složky/kolekce.....     | 89        |
| Výsledek běhu .....                         | 90        |
| <b>Data driven testy</b> .....              | <b>92</b> |
| Užití v praxi .....                         | 93        |

|  |     |
|--|-----|
| <i>Sdílení dat v týmu</i> .....        | 97  |
| Verzování kolekcí.....                 | 97  |
| <b>Workspace</b> .....                 | 97  |
| Vytvoření workspace .....              | 98  |
| Vytváření sdílených requestů.....      | 100 |
| Správa workspace, pozvání nových členů | 100 |
| Changelog .....                        | 101 |
| <i>Newman</i> .....                    | 102 |
| Instalace .....                        | 102 |
| <b>Spouštění testů</b> .....           | 102 |
| Základní Spuštění testu .....          | 102 |
| Globals .....                          | 103 |
| EnvirOnment.....                       | 103 |
| <b>Reporting</b> .....                 | 104 |
| <b>BATCH FILE</b> .....                | 105 |
| Komplexní příklad .....                | 106 |
| <i>MOCK servers v Postman</i> .....    | 107 |
| Příklad.....                           | 107 |
| <i>Postman 9 vs 7 (BONUS)</i> .....    | 113 |
| Přístup k detailu kolekce .....        | 113 |
| Collection Runner .....                | 114 |
| Proměnné .....                         | 114 |
| <i>Zdroje, kam dál</i> .....           | 116 |
| <i>Slovník pojmu</i> .....             | 121 |

## ÚVOD

### O PŘEDNÁŠEJÍCÍM

LinkedIn: <https://www.linkedin.com/in/petr-fifka/>

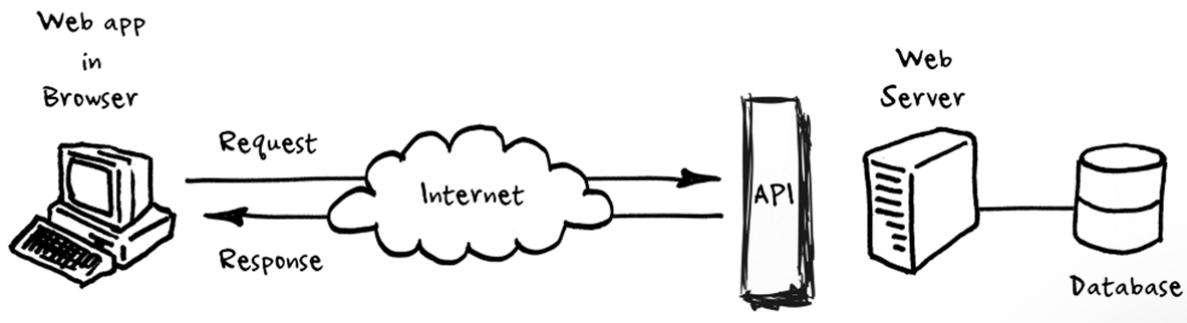
E-mail: [petrfifk@gmail.com](mailto:petrfifk@gmail.com)

## REST API

### CO JE TO REST API

API je interface (rozhraní) pro komunikaci mezi jednotlivými webovými aplikacemi a dalšími programy. Zajišťuje komunikaci mezi dvěma platformami, které si vzájemně vyměňují data. Umožňují využívat již naprogramovaná řešení a integrovat je do vlastních webů či softwaru, díky čemuž šetří programátorský čas a tím i peníze. Programátor využívající API nemusí znát, jak jsou jednotlivé aplikace napsány, stačí znát pouze, jak provolat interface.

REST je cesta, jak jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých HTTP volání. Architektura rozhraní, navržená pro distribuované prostředí. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům. Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsát konkrétními daty). REST je tedy na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.



### PŘEDÁVÁNÍ DAT V REST API

V rámci volání v REST API je potřeba serveru předit data. Toto může probíhat pomocí několika typů datových syntaxí. Můžete se setkat například s:

- JSON
- XML (SOAP)
- URL Params
- Form-data
- GraphQL
- URL parametry
- binary
- x-www-form-urlencoded

Jak dané datové syntaxe používat, naleznete v dokumentaci Postman, v tomto školení se zaměříme zejména na JSON.

### JSON

JSON je způsob zápisu dat nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována v objektech. Vstupem je libovolná datová struktura, výstupem je vždy řetězec.

## ZÁKLADNÍ STRUKTURA JSON

JSON data jsou napsána jako "jméno": "hodnota" (klíč/hodnota). Jednotlivé hodnoty jsou oddělené čárkou.

**Příklad:**

```
"jmeno": "John",  
"id": 1234
```

Objekty jsou v JSON zapsány do složených závorek.

**Příklad:**

```
{  
    "hodnotaObjektu1": "String",  
    "hodnotaObjektu2": 1234  
}
```

Array (pole více hodnot) se do JSON zapisuje do hranatých závorek

**Příklad:**

```
"array": ["hodnota1", "hodnota2", "hodnota3"]
```

## JSON DATOVÉ TYPY

| Datový typ     | Popis   |
|----------------|---|
| <b>String</b>  | Text, zapisuje se vždy do dvojitých uvozovek.<br>Příklad: "Text"                  |
| <b>Number</b>  | Číslo, zapisuje se bez uvozovek.<br>Příklad: 3.21                                 |
| <b>Object</b>  | Objekt. Může obsahovat další json data.<br>Zapisuje se vždy do složených závorek. |
| <b>Array</b>   | Pole hodnot, zapisuje se vždy do hranatých závorek                                |
| <b>Boolean</b> | Logická hodnota. Zapisuje se: true nebo false                                     |
| <b>Null</b>    | Prázdná hodnota. Zápis: null  |

## CVIČENÍ

Zadání:

Připrav JSON, který bude obsahovat:

- Tělo JSONu bude v objektu
- JSON bude obsahovat 3 klíče
  - 1. String "jmeno"
  - 2. Číslo "vek"
  - 3. Array stringů "majetek"
- Vyplň hodnoty klíčů, array bude mít nejméně 3 členy

## HTTP REQUESTY

[HTTP](#) je internetový protokol určený pro komunikaci s [WWW](#) servery. Slouží pro přenos dokumentů. Více informací naleznete například na [wikipedii](#).

Proč se zmiňujeme o [HTTP](#)? Slouží jako základ pro [API](#). Pomocí [HTTP](#) requestů posíláme zprávy pro daný [API](#) server, dostáváme přes něj také odpovědi.

## HTTP STATUSY

HTTP Requesty mají své statusy v rámci response (odpověď ze serveru), výpis všech naleznete [zde](#). Statusy slouží k identifikaci stavu odpovědi requestu.

### Nejdůležitější statusy, se kterými se setkáme:

| Status     | Význam                | Popis, co dělat?   |
|------------|-----------------------|--|
| <b>200</b> | OK                    | Vše ok, odpověď zpracována   |
| <b>201</b> | Created               | Vše ok, request zapsán   |
| <b>204</b> | No Content            | Vše ok, neposílám žádné tělo odpovědi  |
| <b>400</b> | Bad Request           | Request nezpracován, špatná struktura.<br>Něco je špatně s daty nebo parametry |
| <b>401</b> | Unauthorized          | Něco je špatně s autorizací nebo není vyplněna                                 |
| <b>404</b> | Not Found             | URL neexistuje nebo spadl celý server<br>a není možné ho provolat              |
| <b>500</b> | Internal Server Error | Ve většině případů znamená, že je API server nefunkční.                        |
| <b>503</b> | Service Unavailable   | API server nefunguje, v případě, že toto není plánováno, tak je to chyba       |

## POSTMAN

### CO JE TO POSTMAN

Postman je aplikace pro vytváření a používání API. Postman zjednoduší každý krok v API životním cyklu a řeší spolupráci s ostatními vývojáři pro efektivnější vývoj.

Postman je v základní verzi (pro testery většinou dostačující) zdarma.

V tuto chvíli postman podporuje API:

- HTTP Request
- GraphQL
- WebSocket
- SOAP

V tomto školení budeme používat ve většině případů HTTP requesty. Pro GraphQL a SOAP jsou připravené malé ukázky.

Postman má jak plnohodnotnou aplikaci pro OS, tak web aplikaci. V rámci školení se budeme věnovat pouze web aplikaci.

### PRVNÍ SPUŠTĚNÍ POSTMAN

#### REGISTRACE POSTMAN.COM

Pro plné využití aplikace Postman je potřeba se zaregistrovat na [této adrese](#). Při registraci prosím založte workspace. Co je to [workspace](#) probereme v průběhu školení.

#### PŘIHLÁŠENÍ DO POSTMAN

Spusťte Postman, stiskněte Sign in a ve webovém prohlížeči se přihlaste údaji, které jste zadali při registraci:

Postman

File Edit View Help

POSTMAN

### Create an account or sign in

[Create Free Account](#)

[Sign in](#)

Create your account or sign in later? [Skip and go to the app](#)

### A free Postman account lets you

- Organize all your API development in workspaces
- Create public workspaces to collaborate with over 10 million developers
- Back up your work on Postman's cloud
- Experience the best API development platform for free!





### Sign In

[Create Account instead?](#)

Email or Username

Password

Keep me signed in [Forgot Password?](#)

[Sign In](#)

or

 [Sign in with Google](#)

You're redirected from the Postman desktop app  
on win32 10.0.19042 to sign in.

Terms of use Privacy Policy

Po přihlášení se zobrazí domovská stránka Postman

**Postman**

File Edit View Help

Home Workspaces Reports Explore Search Postman Upgrade

**Good afternoon, Petr Fifka!**

Pick up where you left off, catch up with your team's work.

**Get started with Postman**

- Start with something new**  
Create a new request, collection, or API in a workspace  
[Create New →](#)
- Import an existing file**  
Import any API schema file from your local drive or Github  
[Import file →](#)
- Explore our public network**  
Browse featured APIs, collections, and workspaces published by the Postman community.  
[Explore →](#)
- Work smarter with Postman**  
Learn how Postman can help you at every stage of the API development.  
[Learn →](#)

**Recent workspaces**

| NAME           | LAST VIEWED   | MEMBERS |
|----------------|---------------|---------|
| My Workspace   | 8 minutes ago |         |
| Team Workspace | 18 hours ago  |         |

**Help**

- Learning Center
- Support Center
- Bootcamp



Pokud Vám vyskočí okno s instalací, aktualizujte prosím Postman.  
 Většinu akcí budeme dnes provádět ve Vašem osobním workspace "My Workspace". Co je to Workspace a jak se používá bude popsáno v samostatné kapitole později.

File Edit View Help

Home **Workspaces** API Network Reports Explore Search Postman Upgrade

**Good evening, Petr Fifka!**

Pick up where you left off, catch up with your team's work.

**Get started with Postman**

- Start with something new**  
Create a new request, collection, or API in a workspace  
[Create New →](#)
- Import an existing file**  
Import any API schema file from your local drive or Github  
[Import file →](#)
- Explore our public network**  
Browse featured APIs, collections, and workspaces published by the Postman community.  
[Explore →](#)
- Work smarter with Postman**  
Learn how Postman can help you at every stage of the API development.  
[Learn →](#)

**Recent workspaces**

| NAME           | LAST VIEWED       | MEMBERS |
|----------------|-------------------|---------|
| My Workspace   | a few seconds ago |         |
| Team Workspace | 21 hours ago      |         |

**Workspaces**

Search for a workspace + New Workspace

**Recently Visited**

- My Workspace**
- Team Workspace
- Test Workspace

**More Workspaces**

No workspaces found

[View all workspaces](#)

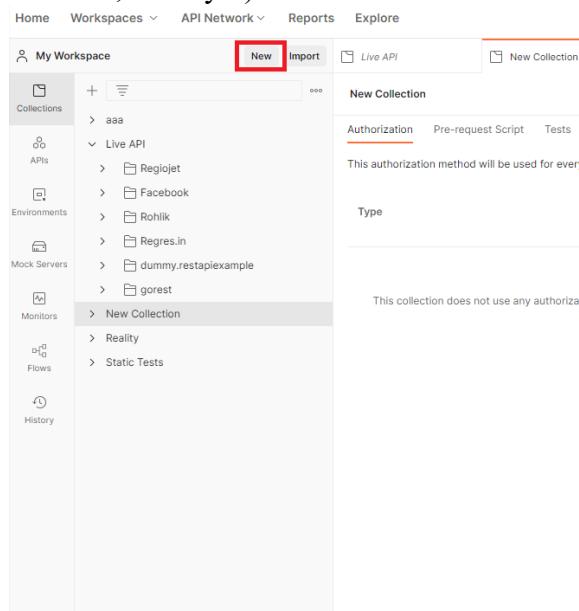


## ZÁKLADNÍ ORIENTACE

V následujících kapitolách projdeme funkčnosti UI Postman. Cílem kapitoly je seznámit se základy ovládání pro budoucí operace v programu. Základní ovládání prvků naleznete také v [Postman dokumentaci](#).

## PŘIDÁNÍ PRVKU

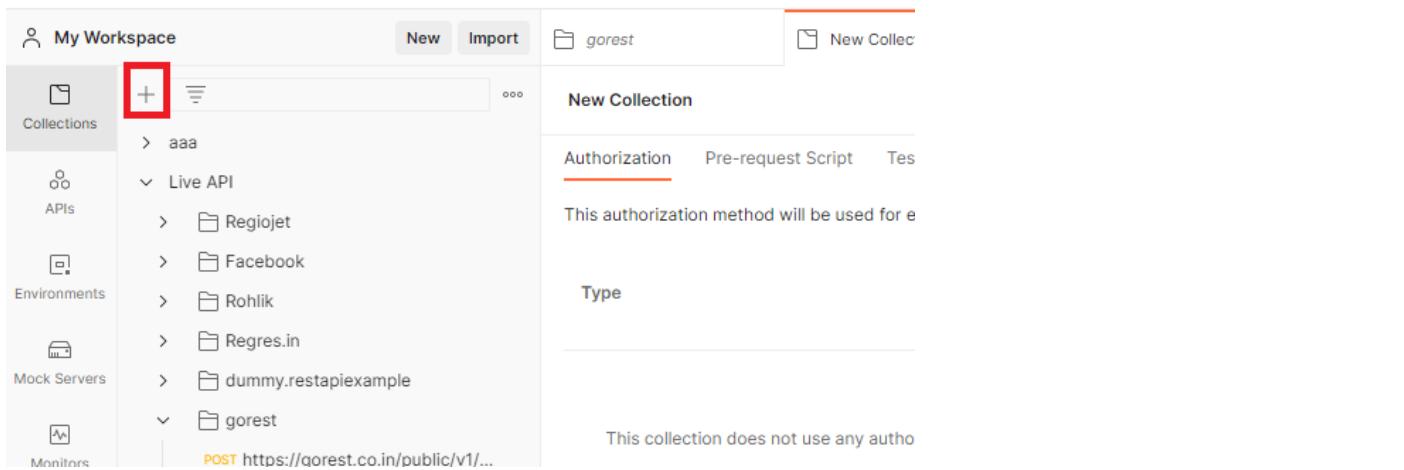
Tlačítka „New“ nám v Postman UI (CTRL + N) umožní vytvořit většinu věcí v postman (Requesty, Kolekce, Složky...)



Zobrazí se okno pro výběr jednotlivých prvků Postmanu. Prozatím okno opustíme krížkem, vrátíme se k němu později.

The screenshot shows the 'Create New' dialog box. It has sections for 'Building Blocks' and 'Advanced'. In 'Building Blocks', there are icons for 'HTTP Request' (GET), 'WebSocket Request' (BETA), 'Collection', 'Environment', 'Workspace', and 'API Documentation'. In 'Advanced', there are icons for 'Mock Server', 'Monitor', 'API', and 'Flows' (BETA). At the bottom, it says 'Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.' and 'Learn more on Postman Docs'.

Pro vytvoření prvku v dané kategorii (Collections, Environments,...) nám může posloužit tlačítko "+", které vytvoří například kolekci na jedno kliknutí.

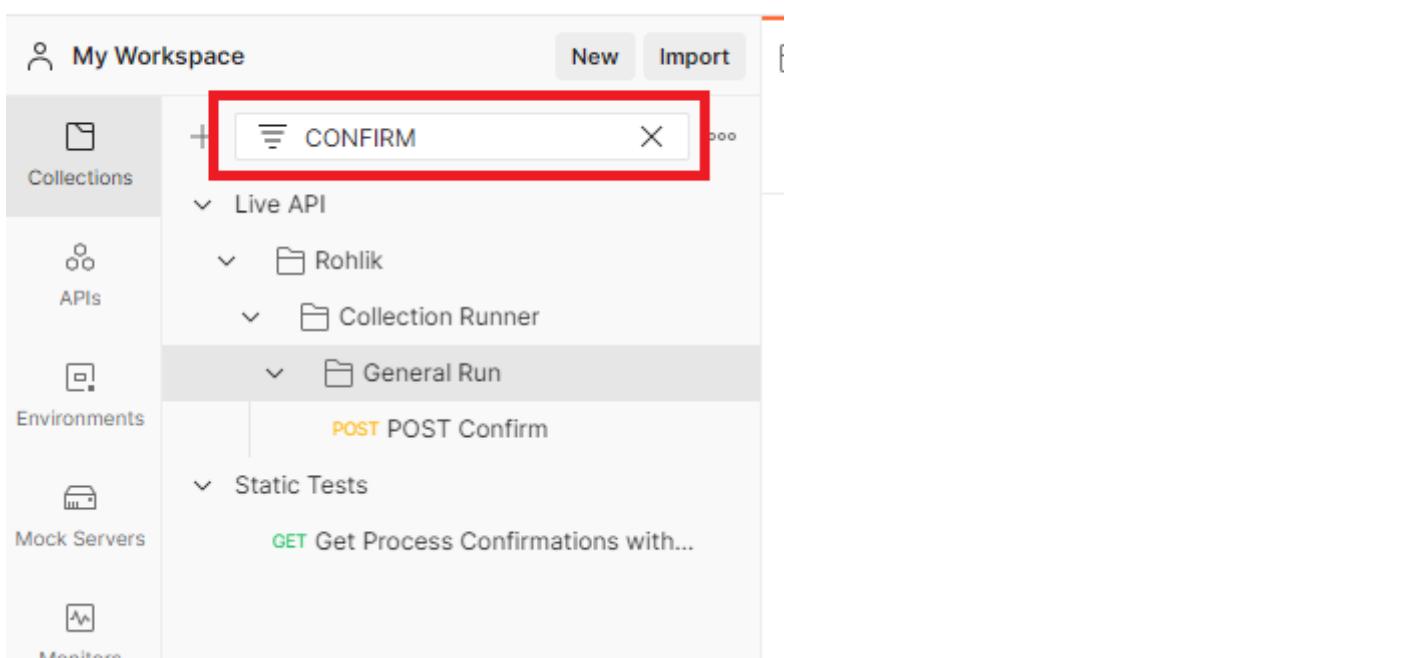


The screenshot shows the Postman interface with a "New Collection" dialog open. The left sidebar lists "My Workspace" with sections for Collections, APIs, Environments, Mock Servers, and Monitors. A red box highlights the "+" button in the Collections section. The main workspace shows a collection named "gorest" with a POST request to "https://qorest.co.in/public/v1/...". The "Authorization" tab is selected in the dialog, which states: "This authorization method will be used for every request in this collection".

## FILTR

Filtr umožňuje fulltextově hledat v rámci jednotlivých sekcí.

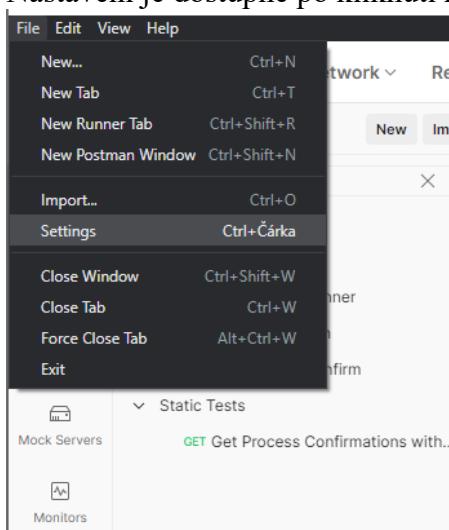
Home    Workspaces     API Network     Reports



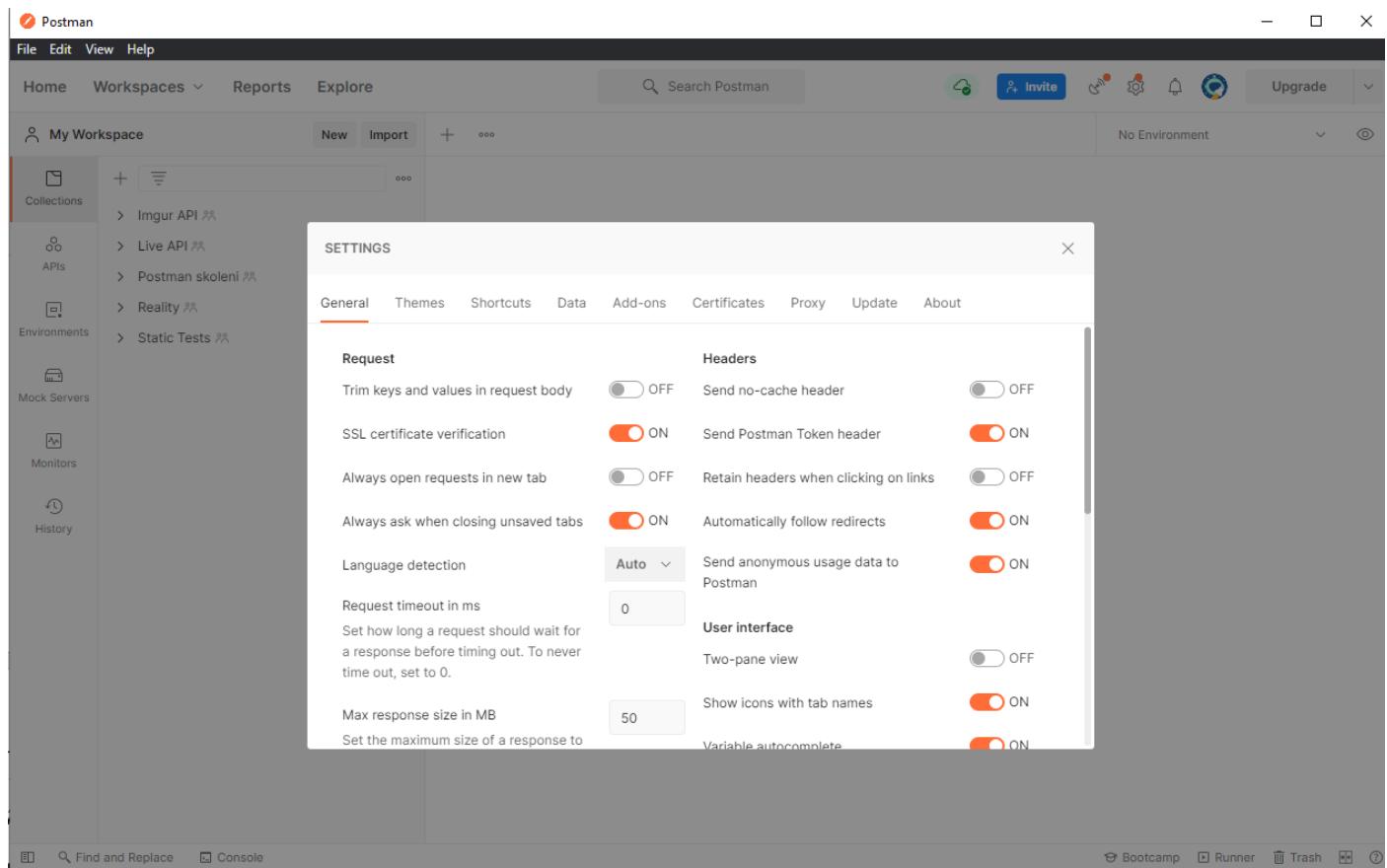
The screenshot shows the Postman interface with a search filter dialog open. The left sidebar lists "My Workspace" with sections for Collections, APIs, Environments, Mock Servers, and Monitors. A red box highlights the search input field containing the text "CONFIRM". The main workspace shows a collection named "Live API" with a POST request to "POST Confirm".

## SETTINGS

Nastavení je dostupné po kliknutí na „File“/„Settings“ (CTRL + ,)



The screenshot shows the Postman menu bar with the "File" menu open. The "Settings" option is highlighted with a red box. Other options in the menu include "New...", "New Tab", "New Runner Tab", "New Postman Window", "Import...", "Close Window", "Close Tab", "Force Close Tab", and "Exit".

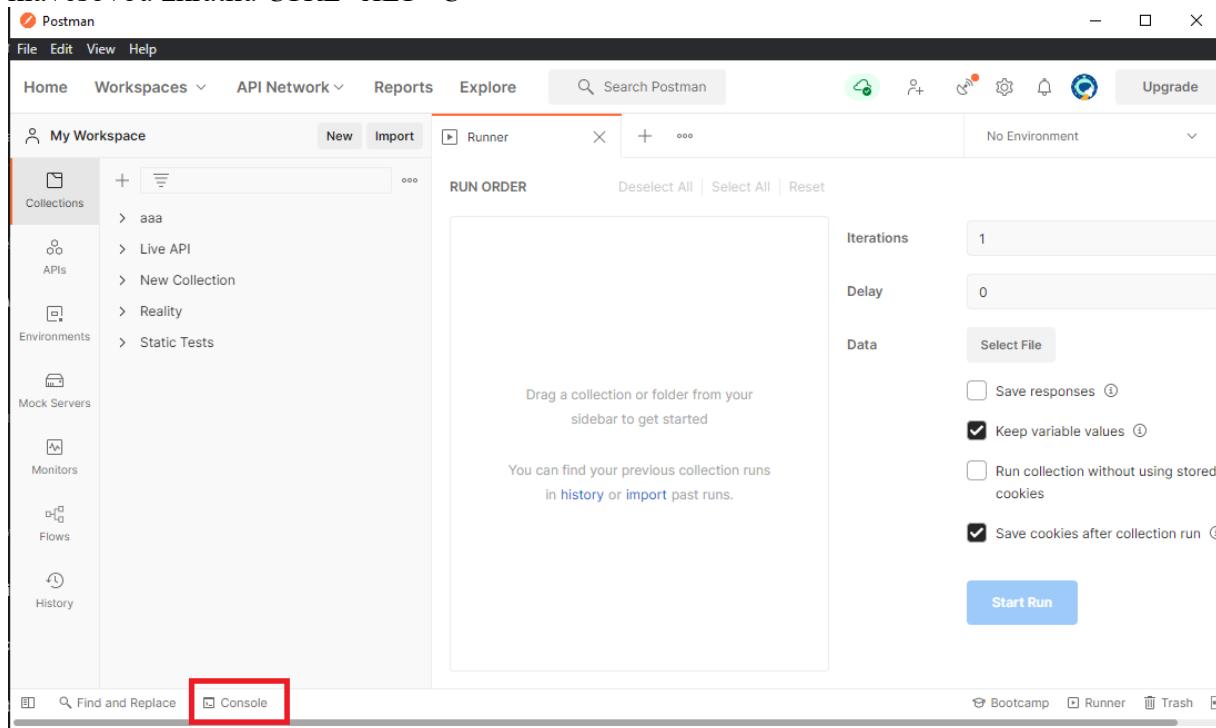


The screenshot shows the Postman application window. A 'SETTINGS' dialog box is open over the main interface. The 'General' tab is active. Other tabs include Themes, Shortcuts, Data, Add-ons, Certificates, Proxy, Update, and About. The 'Request' section contains options like 'Trim keys and values in request body' (OFF), 'SSL certificate verification' (ON), and 'Always ask when closing unsaved tabs' (ON). The 'Headers' section includes 'Send no-cache header' (OFF), 'Send Postman Token header' (ON), and 'Automatically follow redirects' (ON). The 'User interface' section has 'Two-pane view' (OFF), 'Show icons with tab names' (ON), and 'Variable autocomplete' (ON). The 'Themes' tab is also visible.

V nastavení můžete změnit konfiguraci Postman, grafické téma, proxy a mnoho dalšího

## CONSOLE

Konzole slouží k logování a debuggingu requestů a uživatelských logů. Zapneme ji buď přes tlačítko nebo klávesovou zkratku **CTRL+ALT+C**



The screenshot shows the Postman application window with the 'Runner' tab selected. The 'Console' tab at the bottom is highlighted with a red box. The main area shows a 'RUN ORDER' section with a placeholder message: 'Drag a collection or folder from your sidebar to get started'. To the right, there are configuration options: 'Iterations' set to 1, 'Delay' set to 0, and a 'Data' section with a 'Select File' button. Below these are several checkboxes: 'Save responses' (unchecked), 'Keep variable values' (checked), 'Run collection without using stored cookies' (unchecked), and 'Save cookies after collection run' (checked). A large blue 'Start Run' button is at the bottom right.

## REQUESTY

Pro testera nejdůležitější funkcionalita Postmanu je provolávání HTTP requestů. Proto, abychom je mohli efektivně vytvářet a provolávat, musíme znát to, jak je navrženo UI.

Základní rozvržení vypadá takto:

- Collection (kolekce)
- Folder (složka)
- Request

## ORIENTACE V COLLECTIONS

Kolekce v Postman slouží ke seskupování a organizaci jednotlivých requestů. Více informací o kolekcí naleznete v [Postman dokumentaci](#).

Do kolekce se dostaneme po kliknutí na ikonu složky s nápisem Collections

## POSTUP VYTVOŘENÍ KOLEKCE

### POSTUP 1

1. Kliknutí na "+" v záložce Collections
2. Zadáme jméno kolekce a potvrdíme

### POSTUP 2

1. Klik na File/New (klávesová zkratka CTRL +N)
2. Klik na "Collection"
3. Zadáme název a potvrdíme

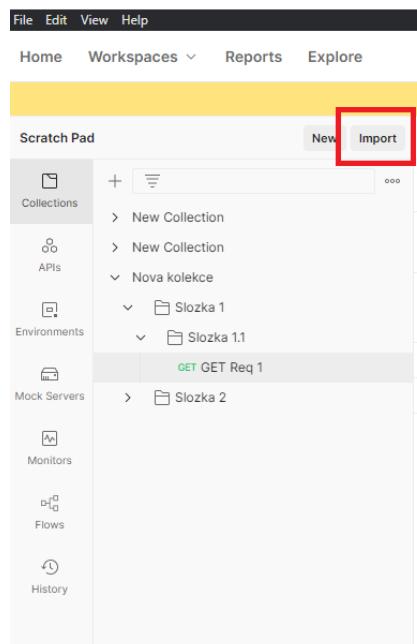
## IMPORT, EXPORT KOLEKCÍ

Import a export kolekcí slouží k distribuci requestů mimo workspace. Kolekce se ukládá v JSON.

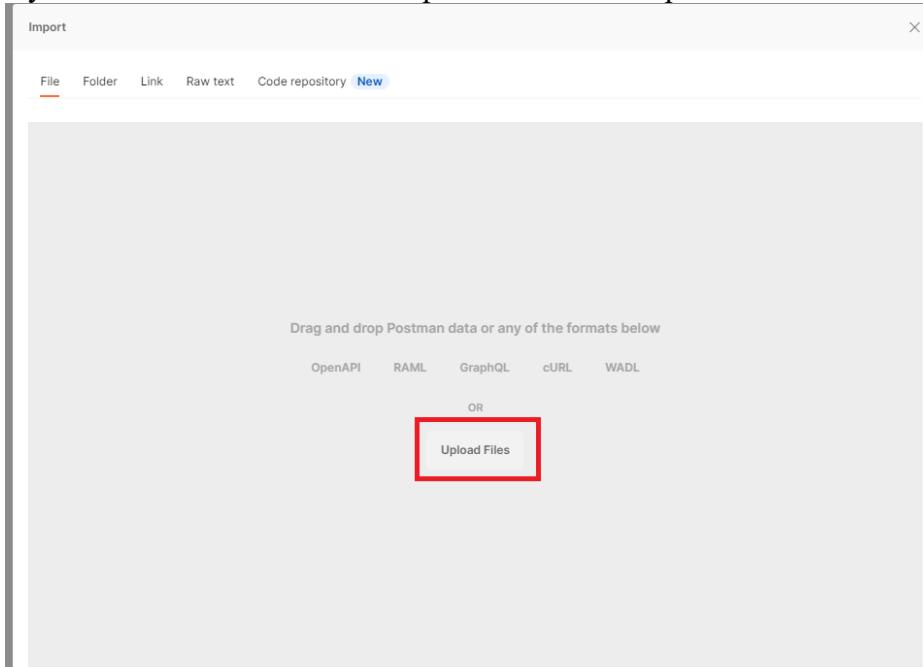
### IMPORT KOLEKCE

Stáhněte si vzorovou kolekci z tohoto [odkazu](#). Jakmile odkaz otevřete, klikněte na tlačítko raw, zobrazí se vám soubor v prázdném okně. Následně klikněte pravým tlačítkem do prostoru a stiskněte "Uložit jako..."

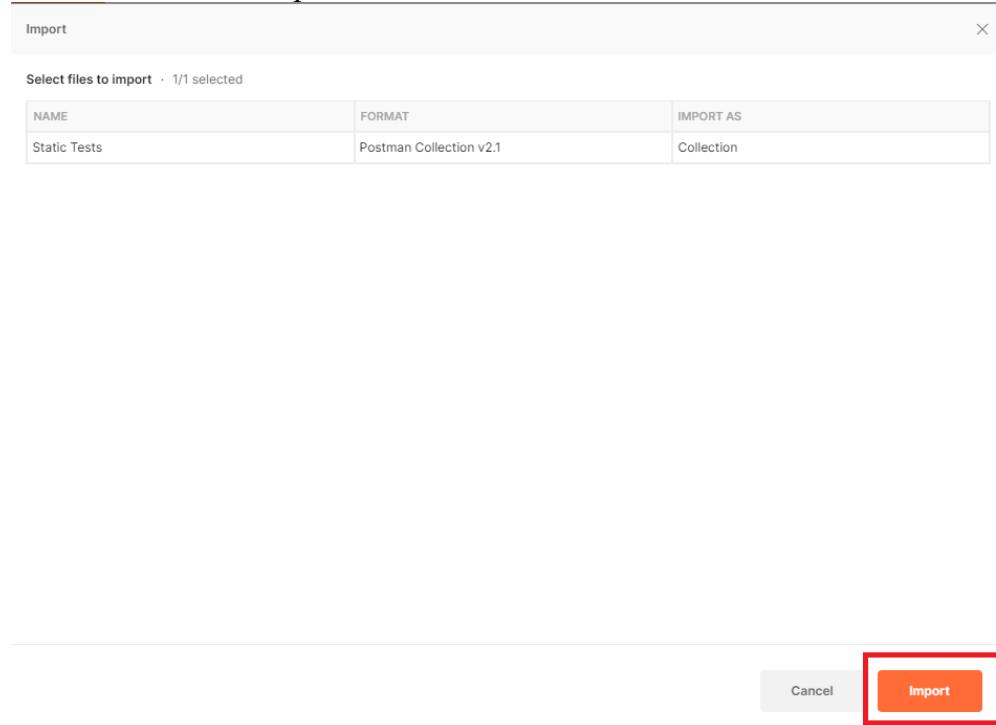
1. V záložce Collections kliknout na „Import“



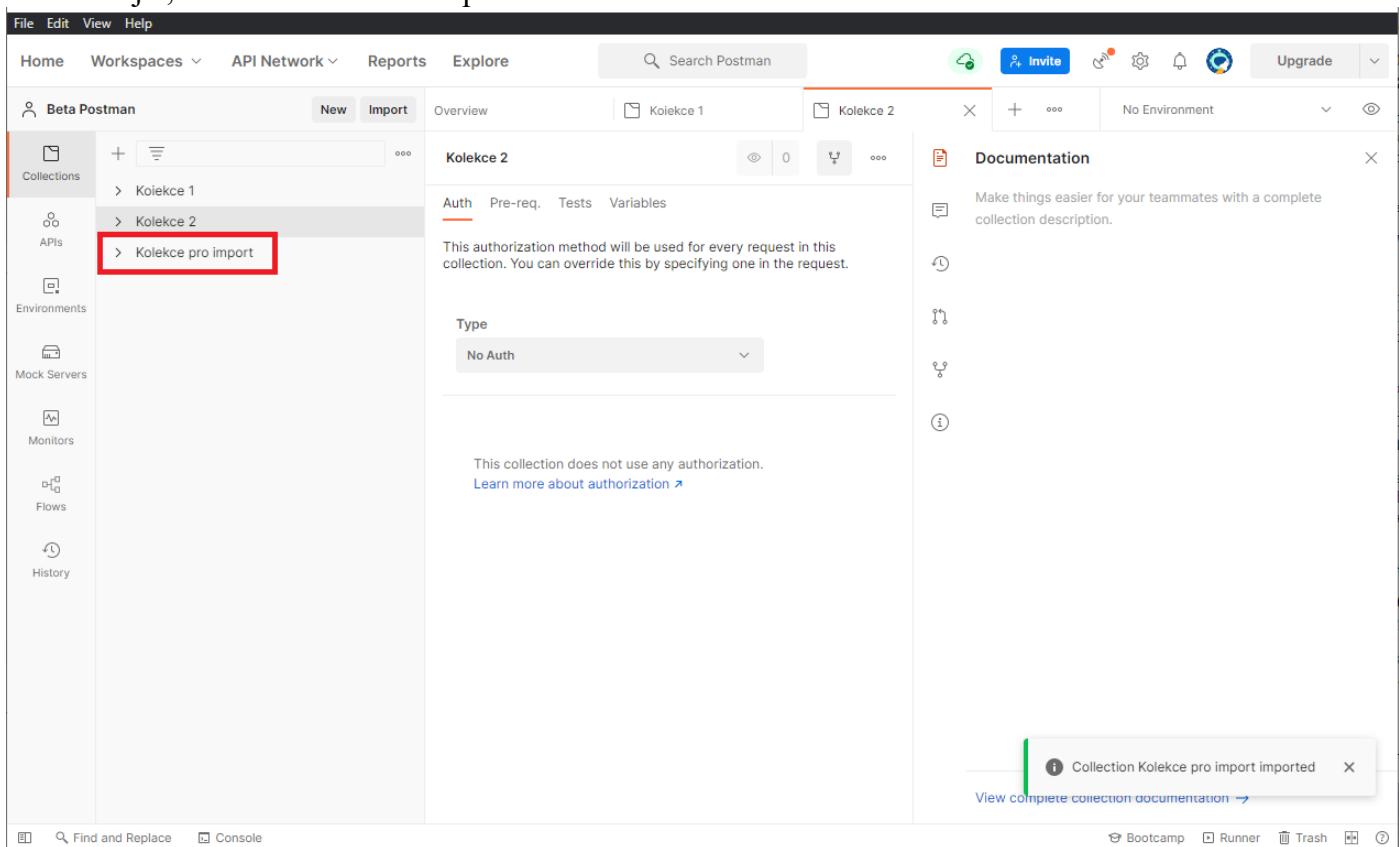
## 2. Vyberte staženou JSON kolekci pomocí tlačítka "Upload files"



## 3. Stisknout tlačítko „Import“



#### 4. Zkontrolujte, že se daná kolekce importovala



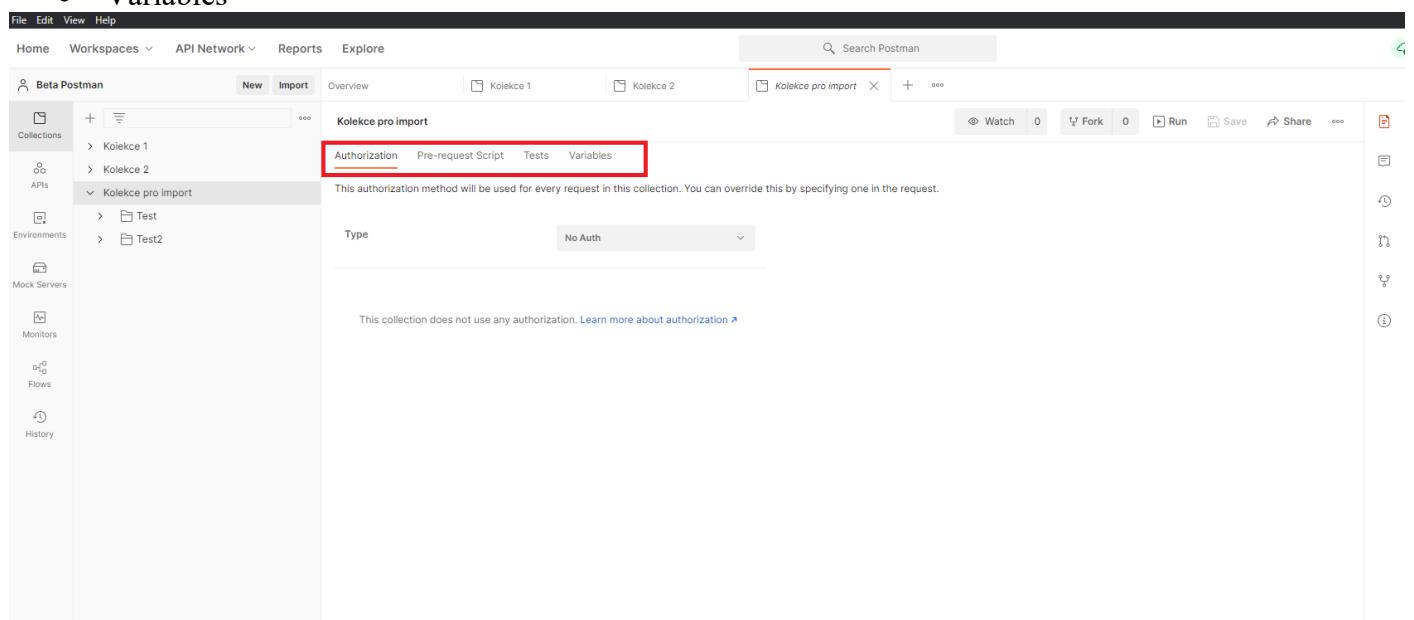
The screenshot shows the Postman interface with the following details:

- File Edit View Help** menu at the top.
- Home Workspaces API Network Reports Explore** tabs.
- Search Postman** search bar.
- Beta Postman** workspace selected.
- New Import** buttons.
- Overview** tab selected.
- Kolekce 1** and **Kolekce 2** are listed in the collections section.
- Kolekce 2** is expanded, showing **Kolekce 1**, **Kolekce 2**, and **Kolekce pro import**. The **Kolekce pro import** item is highlighted with a red box.
- Auth Pre-req. Tests Variables** tabs for the collection.
- A note: "This authorization method will be used for every request in this collection. You can override this by specifying one in the request."
- Type** dropdown set to **No Auth**.
- A note: "This collection does not use any authorization. Learn more about authorization →".
- Documentation** panel on the right with the message: "Make things easier for your teammates with a complete collection description." It includes icons for watch, fork, and info.
- Find and Replace** and **Console** buttons at the bottom.
- Collection Kolekce pro import imported** message in a toast notification.
- View complete collection documentation →** link in the toast.
- Bootcamp**, **Runner**, **Trash** buttons at the bottom right.

## STRUKTURA KOLEKCE

Záložky (do detailu projdeme později v kurzu)

- Authorization
- Pre-request script
- Tests
- Variables



The screenshot shows the Postman interface with the following details:

- File Edit View Help** menu at the top.
- Home Workspaces API Network Reports Explore** tabs.
- Search Postman** search bar.
- Beta Postman** workspace selected.
- New Import** buttons.
- Overview** tab selected.
- Kolekce 1**, **Kolekce 2**, and **Kolekce pro import** are listed in the collections section.
- Kolekce pro import** is expanded, showing **Authorization**, **Pre-request Script**, **Tests**, and **Variables** tabs. The **Authorization** tab is highlighted with a red box.
- A note: "This authorization method will be used for every request in this collection. You can override this by specifying one in the request."
- Type** dropdown set to **No Auth**.
- A note: "This collection does not use any authorization. Learn more about authorization →".
- Watch**, **Fork**, **Run**, **Save**, **Share** buttons at the top right.
- Find and Replace** and **Console** buttons at the bottom.
- Collection Kolekce pro import imported** message in a toast notification.
- View complete collection documentation →** link in the toast.
- Bootcamp**, **Runner**, **Trash** buttons at the bottom right.

## Ikony

- Watch/Unwatch
- Fork

slouží při úpravách týmem a verzování kolekcí. Na tomto školení tuto část do detailu probírat nebudeme. Jak na verzování naleznete [zde](#).

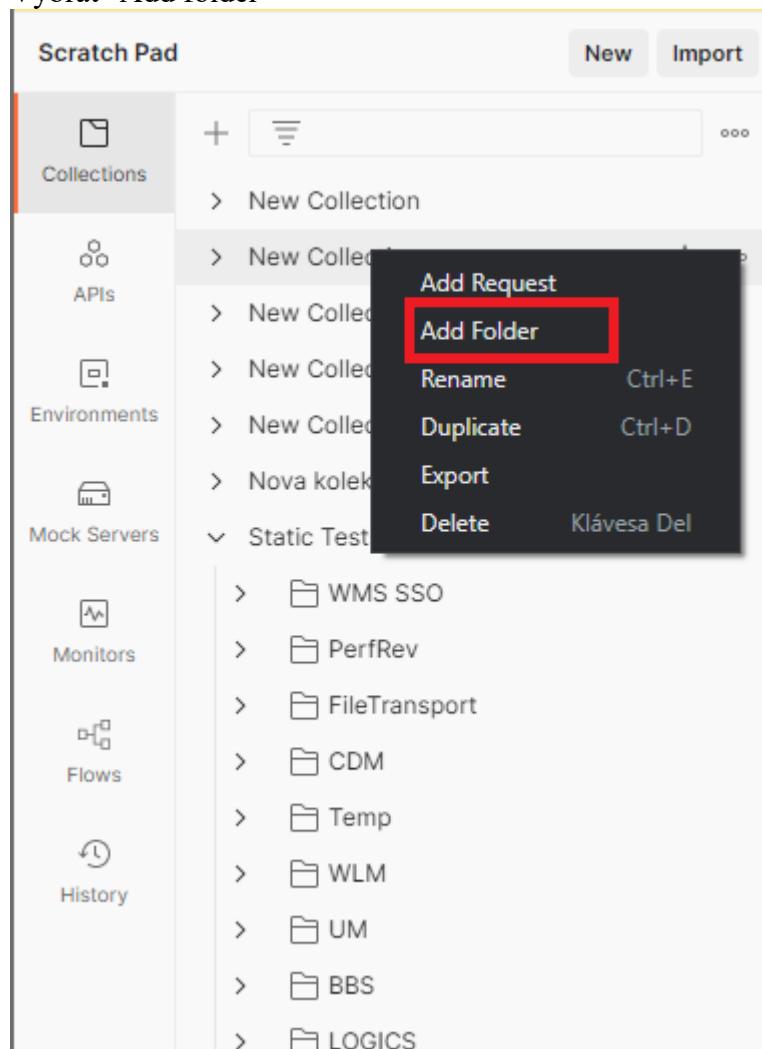
Run ikona slouží pro spuštění [Collection runneru](#), který probereme později.

## ORIENTACE VE SLOŽKÁCH

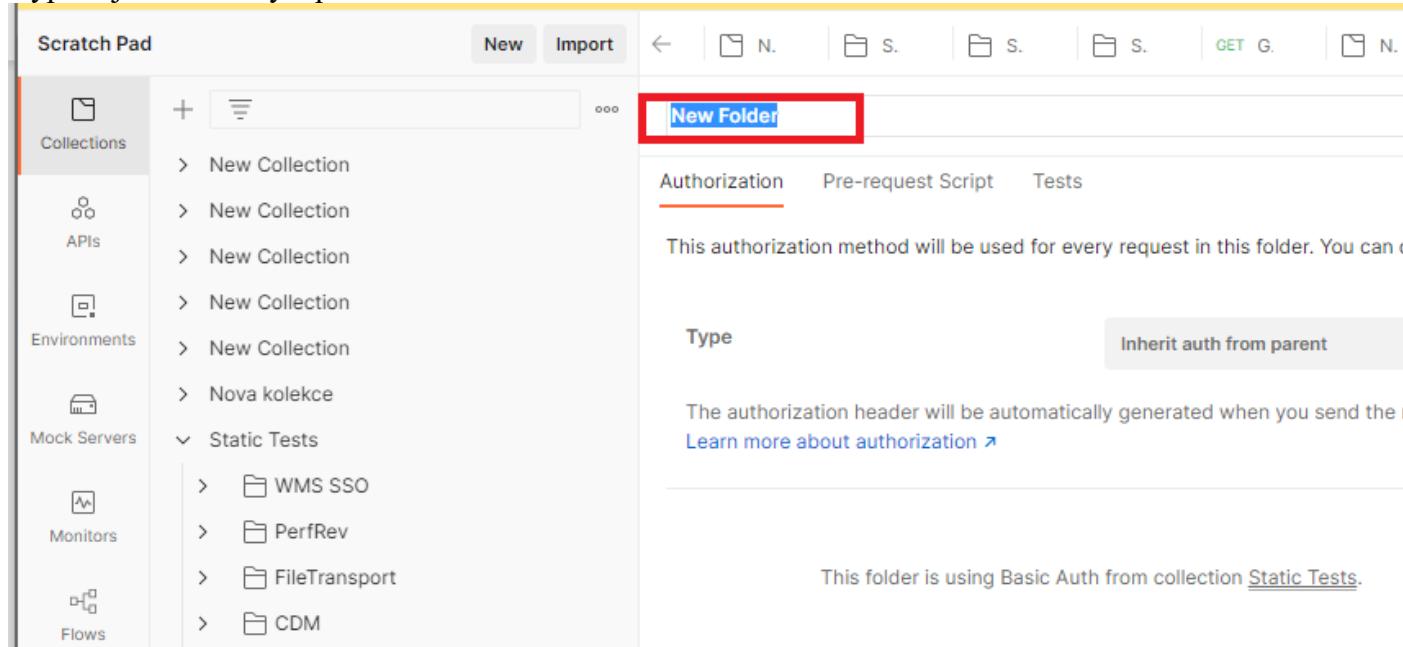
Složky v Postman slouží k třídění requestů. Může mít více úrovní.

### VYTVOŘENÍ SLOŽKY

1. Klik pravým tlačítkem myši na kolekci nebo jinou složku
2. Vybrat "Add folder"



3. Vyplnit jméno složky a potvrdit enterem



The screenshot shows the Postman interface in 'Scratch Pad' mode. On the left, there's a sidebar with categories: Collections, APIs, Environments, Mock Servers, Monitors, and Flows. Under 'Collections', several items are listed with a plus sign to add more. A red box highlights the input field where 'New Folder' is typed. To the right, there are tabs for Authorization, Pre-request Script, and Tests. Below the tabs, a note says: 'This authorization method will be used for every request in this folder. You can change it later.' Under the 'Authorization' tab, there's a 'Type' section with a button labeled 'Inherit auth from parent'. Another note below it says: 'The authorization header will be automatically generated when you send the request.' A link 'Learn more about authorization' is provided. At the bottom right, it says: 'This folder is using Basic Auth from collection Static Tests'.

### STRUKTURA SLOŽKY

Velice podobná struktuře Collection.

Obsahuje záložky:

- Authorization
- Pre-request Script
- Tests

A pro spuštění Collection Runneru tlačítko Run

Kolekce pro import / Test

Authorization Pre-request Script Tests

This authorization method will be used for every request in this folder. You can override this by specifying one in the request.

Type Inherit auth from parent

The authorization header will be automatically generated when you send the request.  
[Learn more about authorization](#)

This folder is using No Auth from collection [Kolekce pro import](#).

## HTTP REQUESTY

Jak už bylo zmíněno, provolávání HTTP requestů je pro testera to nejdůležitější, co Postman umí.

V následující části se dozvíme, jak s nimi v Postman pracovat.

Pro práci v rámci školení si vytvoříme Kolekci se jménem: Skoleni

### VYTVOŘENÍ A PROVOLÁNÍ REQUESTU

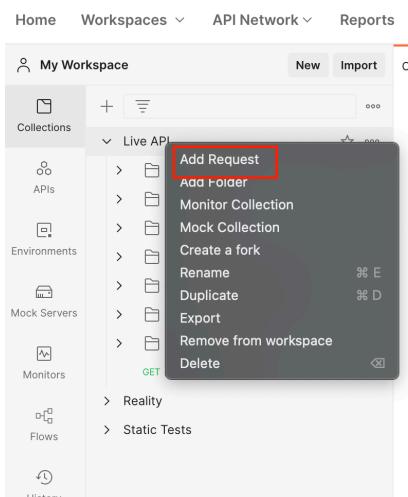
Vytvoříme složku "First calls" v kolekci Skoleni.

Existuje několik postupů, jak v Postman založit HTTP request.

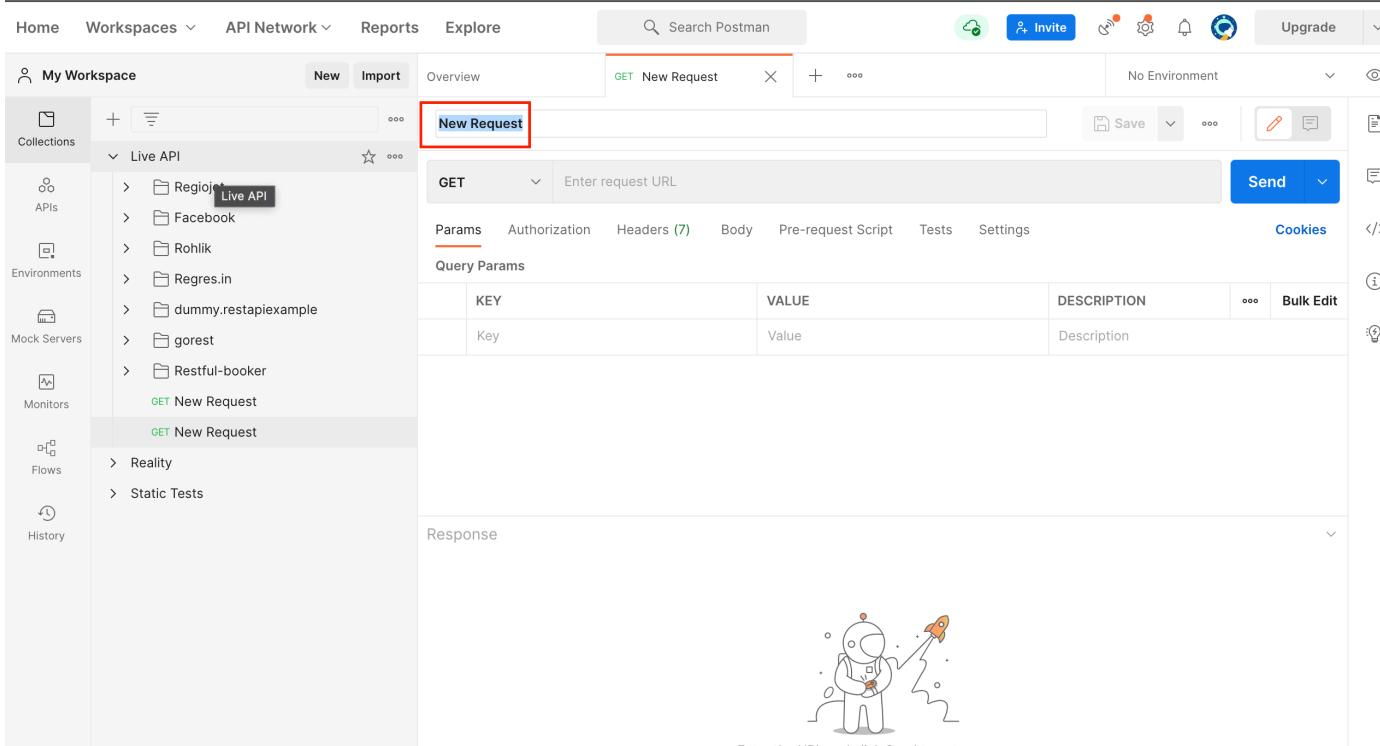
#### POSTUP 1

1. Pravý klik na kolekci nebo složku

## 2. Vybrat "Add request"



## 3. Vyplnit jméno requestu – doporučuji krátký, jasný název co request dělá. Například: GET accounts



The screenshot shows the Postman interface with a 'New Request' dialog open. The 'Headers' tab is selected, showing 'Content-Type: application/json'. The 'Body' tab is also visible. A cartoon character holding a rocket ship is in the bottom right corner.

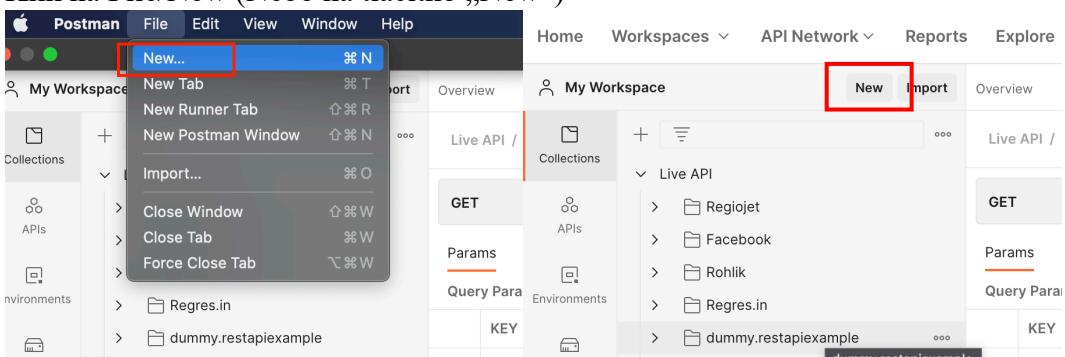
### Úkol:

Vytvořte 2 requesty:

1. V kolekci Skoleni request se jménem: HTTP Request 1
2. Ve slozce "First calls": HTTP Request 2

### POSTUP 2

#### 1. Klik na File/New (Nebo na tlačítko „New“)



## 2. Vybrat HTTP Request

Create New X

**Building Blocks**

**HTTP Request**  
GET  
Create a basic HTTP request

**WebSocket Request** BETA  
Test and debug your WebSocket connections

**Collection**  
Save your requests in a collection for reuse and sharing

**Environment**  
Save values you frequently use in an environment

**Workspace**  
Create a workspace to build independently or in collaboration

**Advanced**

**API Documentation**  
Create and publish beautiful documentation for your APIs

**Mock Server**  
Create a mock server for your in-development APIs

**Monitor**  
Schedule automated tests and check performance of your APIs

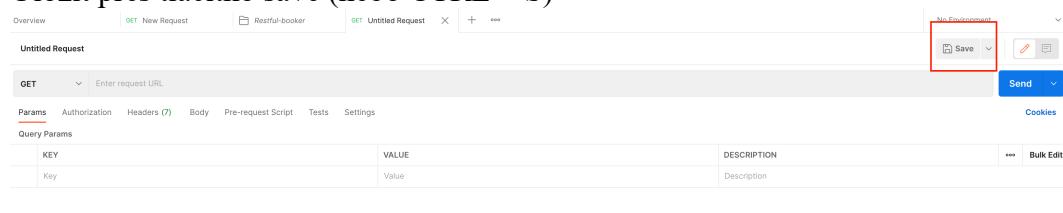
**API**  
Manage all aspects of API design, development, and testing

**Flows** BETA  
Test real-world workflows by connecting series of requests logically.

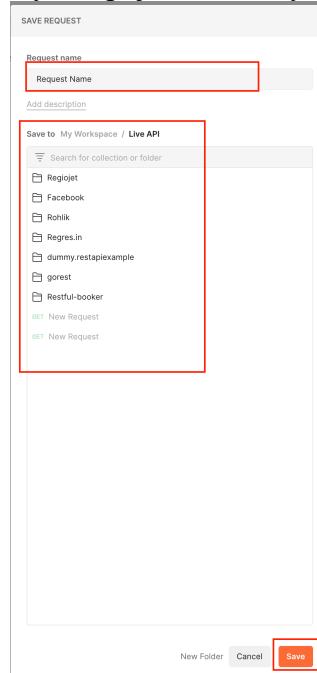
Not sure where to start? [Explore](#) featured APIs, collections, and workspaces published by the Postman community.

## 3. Uložit přes tlačítko save (nebo CTRL + S)

Learn more on [Postman Docs](#)



## 4. Vybrat pojmenovat, vybrat umístění a uložit



### Úkol:

Vytvořte request pomocí tlačítka "New", pojmenujte ho HTTP Request 3 a uložte do složky "First calls"

### CURL

curl je textový datový protokol, který se využívá pro přenos dat nejen v sítích. Je používaný například i pro nastavování TV, routerů, tiskáren, mobilních telefonů atd.

V rámci API si takto můžeme přenést request přes jedno kliknutí.

Pro přenos informací můžete použít i jiné datové formáty, v tomto školení budeme pracovat primárně s curl.

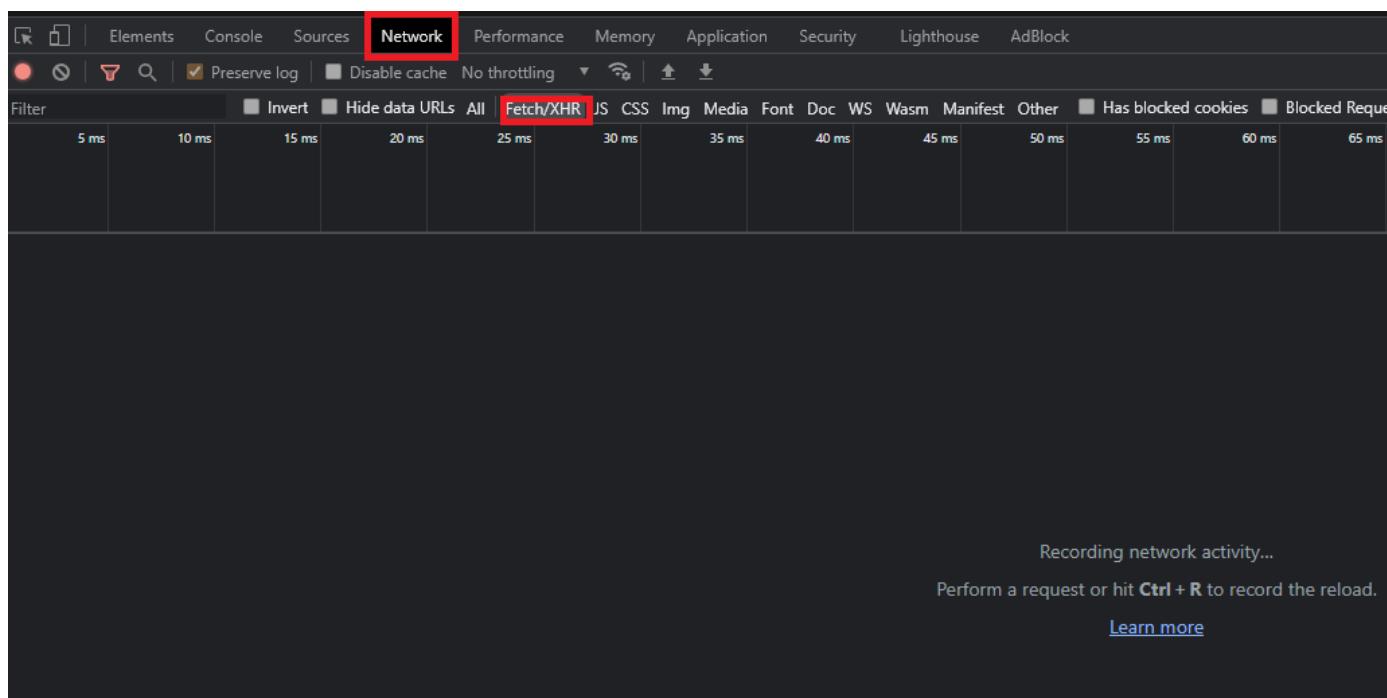
### Příklad curl:

```
curl 'https://reqres.in/api/users?page=2' \
-H 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
-H 'Referer: https://reqres.in/' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
-H 'sec-ch-ua-platform: "Windows"' \
-H 'Content-Type: application/json' \
--compressed
```

## IMPORT CURL

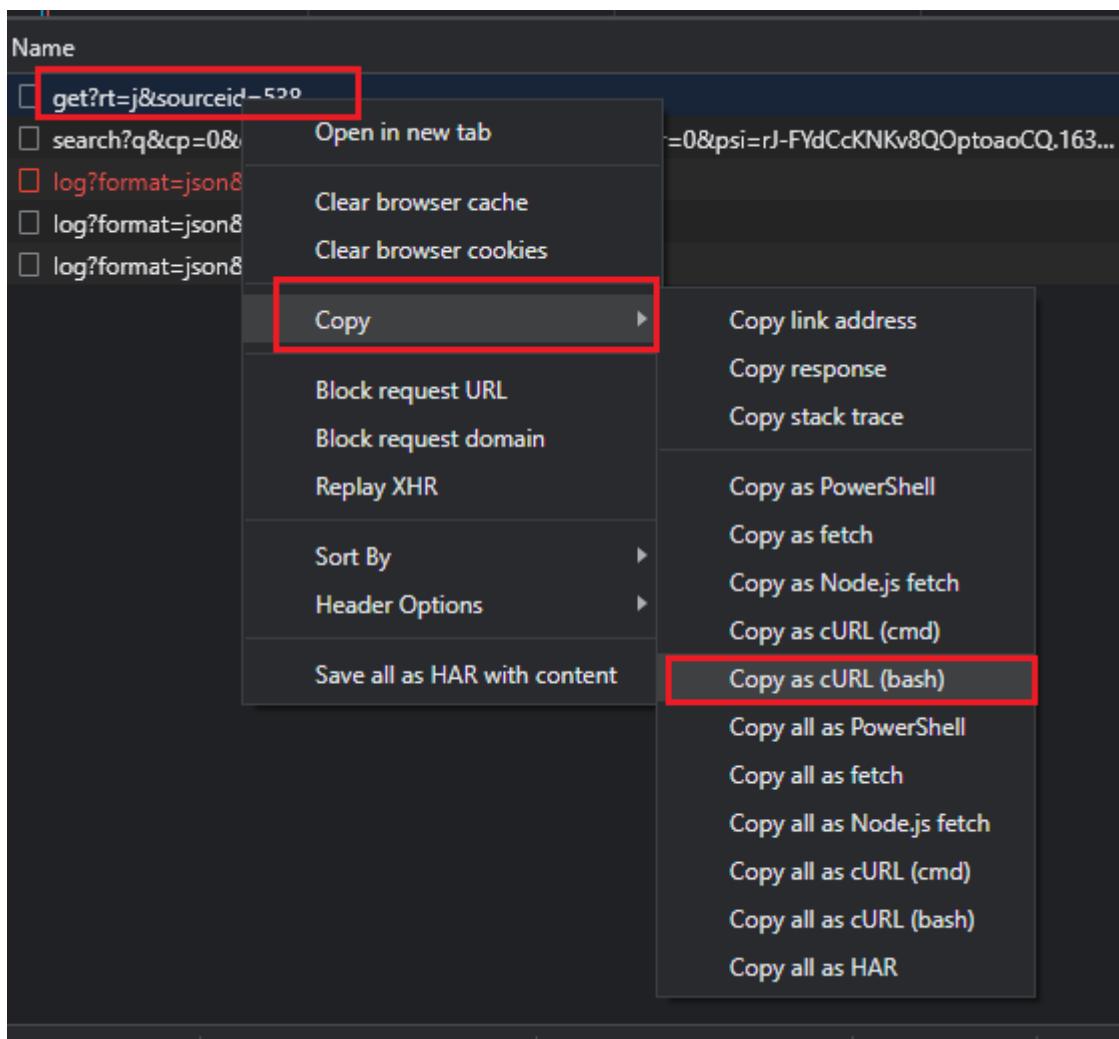
Jak fungují níže zmíněné DEV naleznete v [tomto tutoriuu](#).

1. Otevřeme prohlížeč a poté dev tooly (CTRL+SHIFT+I)
2. Otevřeme záložku Network a zvolíme Fetch/XHR

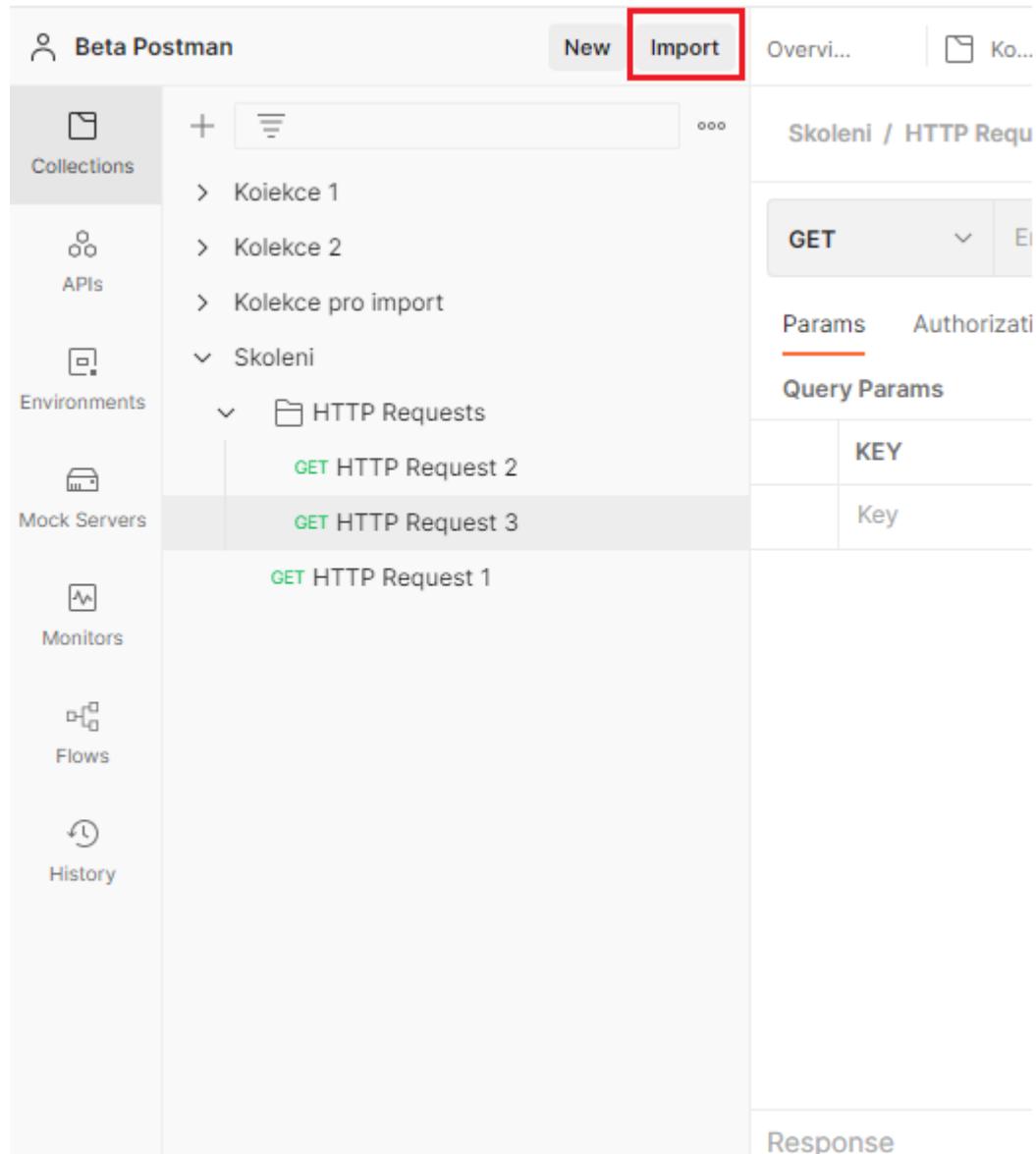


3. Otevřeme stránku google.com

4. V seznamu requestů bychom měli vidět request "get?...", klikneme na něj pravým tlačítkem, zvolíme "copy" a následně "Copy as cURL (bash)"



## 5. V Postman klikneme na tlačítko Import



The screenshot shows the Postman application interface. The top navigation bar includes 'Beta Postman', 'New', 'Import' (which is highlighted with a red box), 'Overview', and 'Ko...'. On the left, there's a sidebar with categories: Collections (Kolekce 1, Kolekce 2, Kolekce pro import, Skoleni), APIs, Environments (Skoleni), Mock Servers, Monitors, Flows, and History. The main workspace shows a collection named 'Skoleni / HTTP Requests' containing three requests: 'GET HTTP Request 2' (selected), 'GET HTTP Request 3', and 'GET HTTP Request 1'. To the right of the requests, there are tabs for 'Params' (selected) and 'Authorization', followed by sections for 'Query Params' (with columns for KEY and Value) and 'Response'.

6. Vybereme záložku "Raw text" a vložíme do pole nás cURL, stiskneme tlačítko "Continue"

Import X

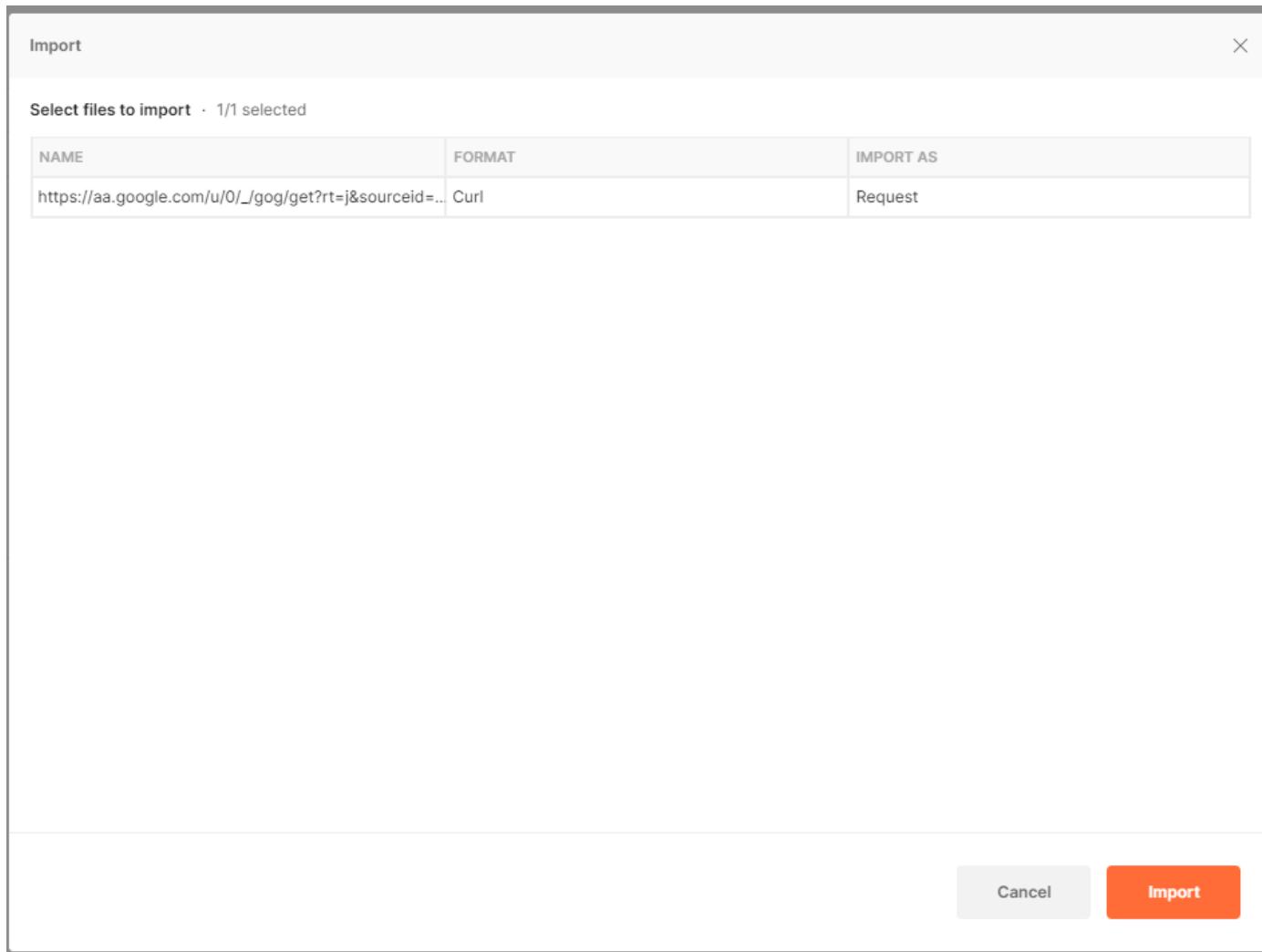
File Folder Link Raw text Code repository [New](#)

Paste raw text

```
curl 'https://aa.google.com/u/0/_/gog/get?rt=j&sourceid=538' \
-H 'authority: aa.google.com' \
-H 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69
Safari/537.36' \
-H 'sec-ch-ua-platform: "Windows"' \
-H 'content-type: application/x-www-form-urlencoded; charset=UTF-8' \
-H 'accept: */*' \
-H 'origin: https://www.google.com' \
-H 'x-client-data: CIS2yQEIo7bJAQjEtskBCKmdygEIkozKAQjr8ssBCO/ywwEInvnLA0inhMwBCLaFzAEI/4XMAQiEh8wBCMuJzAEIuYvMARitqcoB'
\
-H 'sec-fetch-site: same-site' \
-H 'sec-fetch-mode: cors' \
-H 'sec-fetch-dest: empty' \
-H 'referer: https://www.google.com/' \
-H 'upgrade-insecure-requests: 1'
```

Continue

7. Pokud vše proběhlo v pořádku, zobrazí se nám okno se shrnutím, co importujeme, potvrďme tlačítkem import

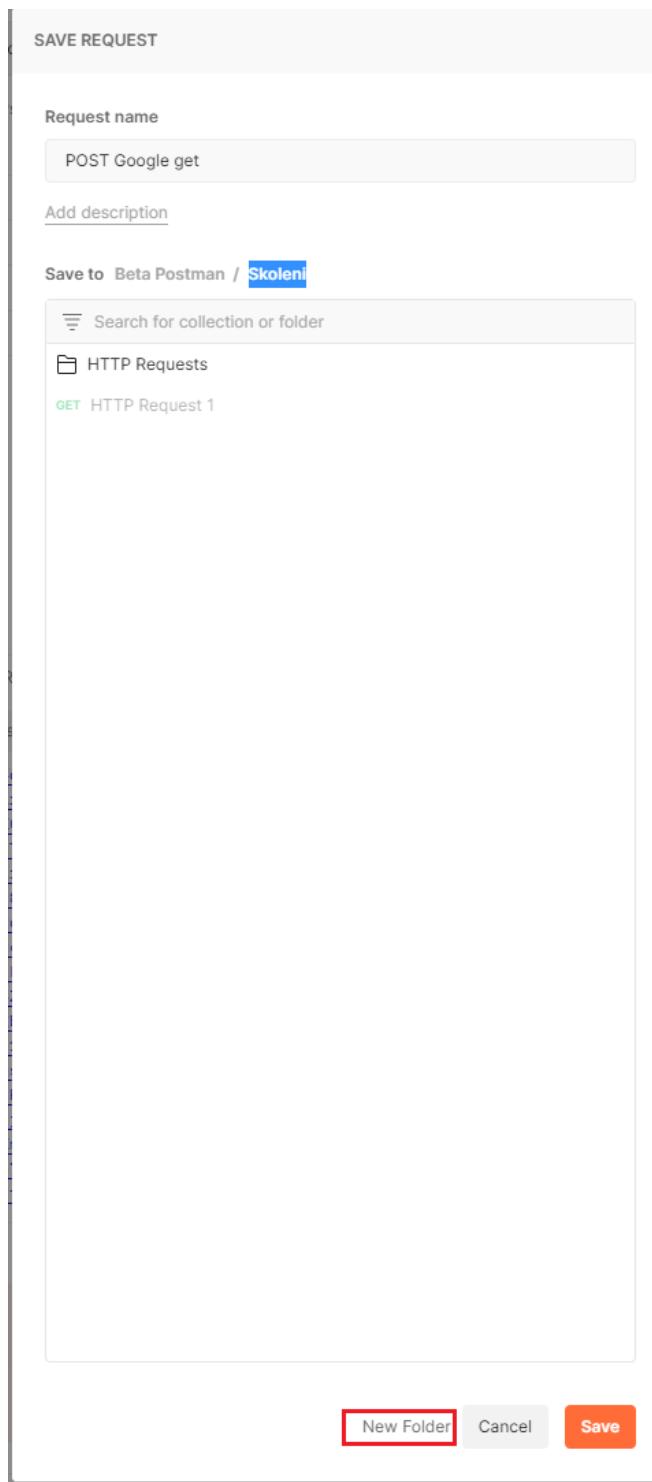


8. Importovaný Request se nám zobrazí v novém Postman tabu (záložky v horní části). Oranžová tečka u našeho tabu znamená, že nemáme uložené změny.

| KEY  | VALUE | DESCRIPTION | ... | Bulk Edit |
|--|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> rt       | j     |             |     |           |
| <input checked="" type="checkbox"/> sourceid | 538   |             |     |           |
| Key  | Value | Description |     |           |

9. Request uložíme (CTRL + S)  
10. Vyplníme jméno: "POST Google get"

11. V kolekci Skoleni vytvorime novou složku "Google" |



### Úkol:

Importujte níže zmíněné cURL, následně ho uložte, pojmenujte ho "GET Users. Vytvořte složku "regres.in" a v ní druhou složku "import req".

### cURL:

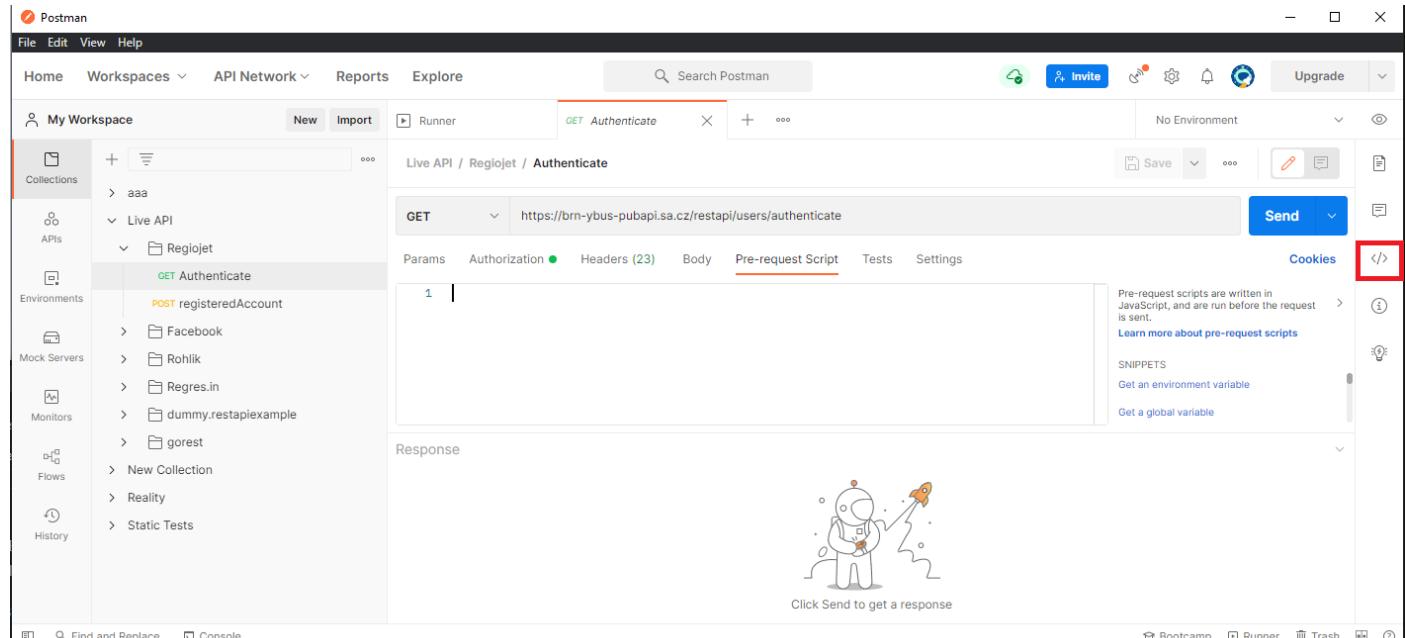
```
curl --location --request GET 'https://reqres.in/api/users' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0'
```

```
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/95.0.4638.54 Safari/537.36'
--header 'sec-ch-ua-platform: "Windows" \
--header 'Content-Type: application/json'
```

## EXPORT REQUESTU V POSTMAN JAKO cURL

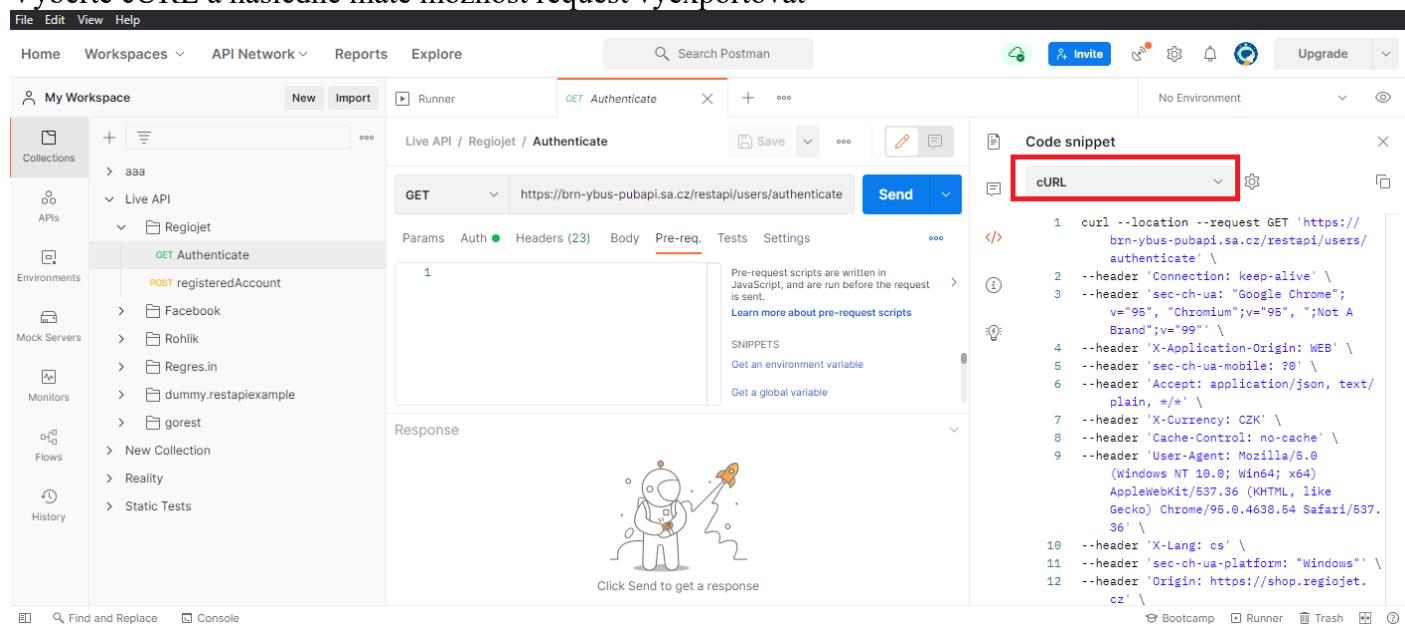
Request také můžeme pomocí cURL exportovat z Postman.  
Jak na to?

1. Otevřete request "GET Users" ve složce "import req"
2. Klikněte na ikonu Code



The screenshot shows the Postman interface with the 'Authenticate' request selected. The 'Code' icon in the top right corner of the request details panel is highlighted with a red box. The request details include the method (GET), URL (https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate), and various tabs like Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings.

3. Vyberte cURL a následně máte možnost request vyexportovat



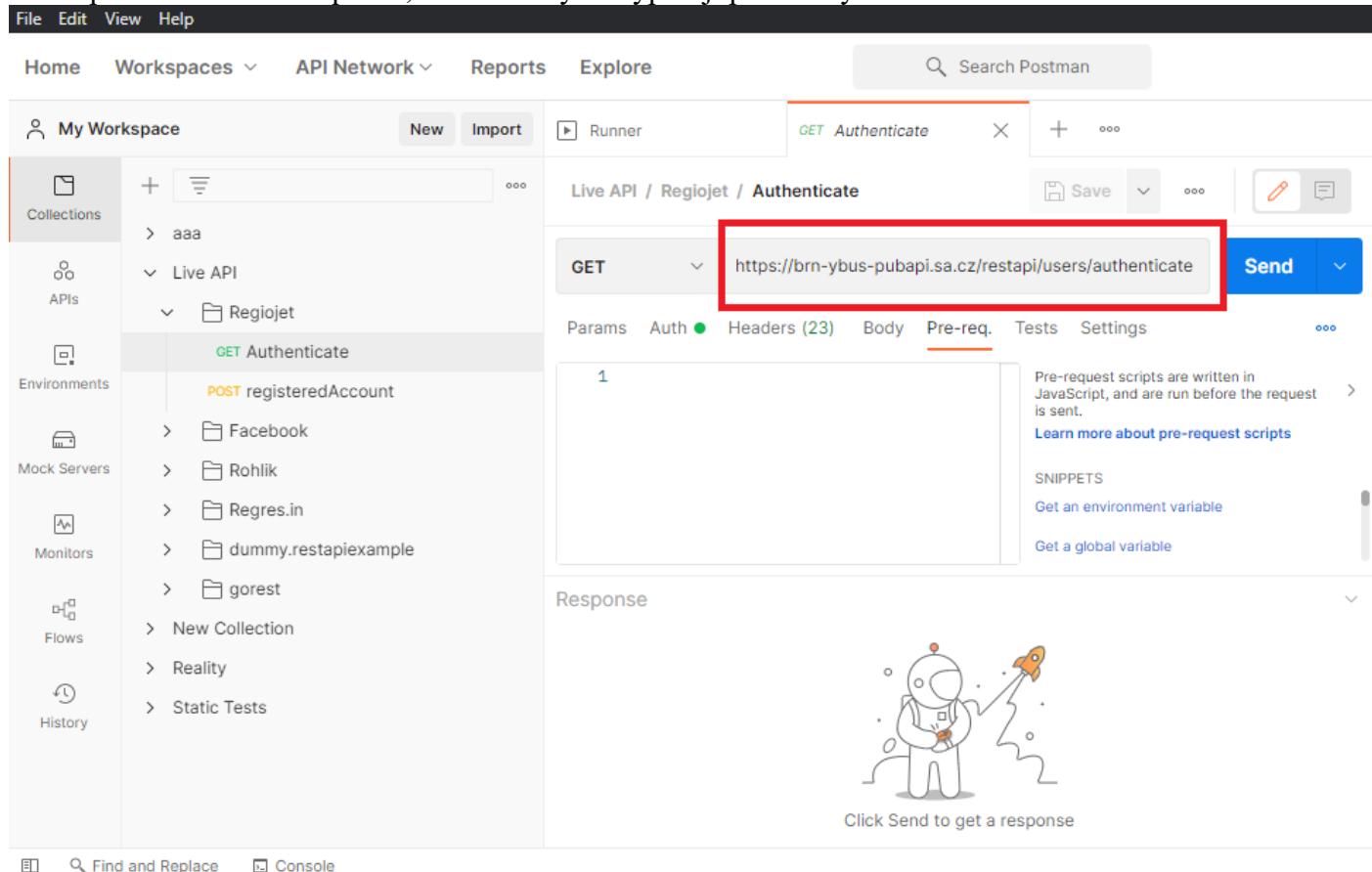
The screenshot shows the Postman interface with the 'Authenticate' request selected. The 'Code snippet' panel on the right is expanded, showing the generated cURL command. The 'curl' command includes various headers and parameters corresponding to the request settings. The 'Code snippet' panel also contains links for 'Pre-request scripts', 'SNIPPETS', 'Get an environment variable', and 'Get a global variable'.

```
curl --location --request GET 'https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate'
--header 'Connection: keep-alive'
--header 'sec-ch-ua: "Google Chrome"; v="95", "Chromium"; v="95", ";Not A Brand"; v="99"'
--header 'X-Application-Origin: WEB'
--header 'sec-ch-ua-mobile: ?0'
--header 'Accept: application/json, text/plain, */*'
--header 'X-Currency: CZK'
--header 'Cache-Control: no-cache'
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36'
--header 'X-Lang: cs'
--header 'sec-ch-ua-platform: "Windows" cz'
```

### Úkol:

Vyexportujte cURL requestu "GET Users" a následně ho do stejné složky importujte zpět, nazvěte ho "GET Users 2"

Místo pro zadání URL requestu, automaticky se vyplňují parametry z "Params" sekce



The screenshot shows the Postman application interface. On the left, there's a sidebar with sections like 'My Workspace', 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. The main area shows a 'Live API / Regiojet / Authenticate' collection. A specific 'GET Authenticate' request is selected. The 'Pre-request' tab is open, showing a script for pre-request scripts. The 'Send' button is visible at the top right of the request card.

### **Úkol:**

Vložte URL: <https://www.principal.cz/cz/> do requestu: HTTP Request 1

### **TYP HTTP CALLU**

Detailedy ohledně HTTP request typů naleznete například na [w3schools](#).

V rámci školení probereme základní HTTP typy:

- GET
- POST
- PUT
- PATCH
- DELETE

V Postman typ callu naleznete vlevo vedle URL.

File Edit View Help

Home Workspaces API Network Reports Explore Search Postman

My Workspace New Import Runner GET Authenticate + ...

Collections + Live API / Regiojet / Authenticate Save + ...

APIs + GET GET Authenticate https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate Send + ...

Environments + Params Auth Headers (23) Body Pre-req. Tests Settings ...

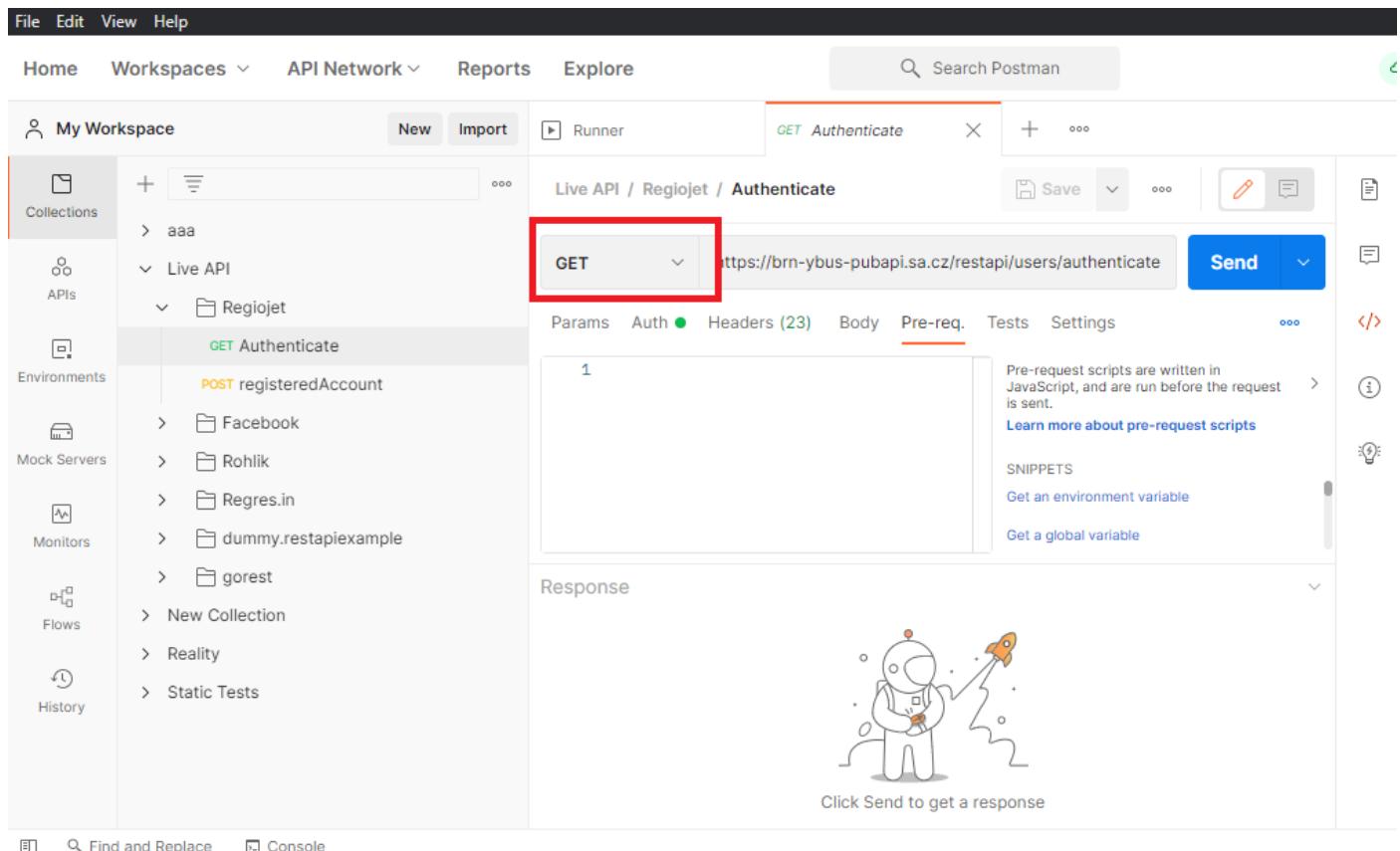
Mock Servers + 1 Pre-request scripts are written in JavaScript, and are run before the request is sent. Learn more about pre-request scripts

Monitors + SNIPPETS Get an environment variable

Flows + Get a global variable

History + Response Click Send to get a response

Find and Replace Console



Pro práci s typy requestu vytvořte novou složku "HTTP Request" v kolekci "Skolení"

## GET

Využívá se pro žádost o data ze specifického zdroje.

Každý nový request v Postman má typ GET.

## PŘÍKLAD

Importujte GET request do Postman. Request pojmenujte "GET Regres.in users" a uložte do složky "HTTP Requests". Následně request provolejte.

### cURL:

```
curl --location --request GET 'https://reqres.in/api/users' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

## POST

Využívá se na poslaní dat na server pro vytvoření dat.

## PŘÍKLAD

Importujte POST request do Postman. Request pojmenujte "POST Regres.in users" a uložte do složky "HTTP Requests". Následně request provolejte.

### cURL:

```
curl --location --request POST 'https://reqres.in/api/users' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36'
```

```
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "Morpheus",
  "job": "leader"
}'
```

### Úkol:

Najdete a otevřete request "HTTP Request 2", přesuňte ho do složky "HTTP Requests", následně ho změňte na typ POST a url změňte na <https://reqres.in/api/users>

### PUT

Velice podobný POST metodě. Rozdíl je v tom, že PUT vytvoří nový záznam nebo upraví již existující.

### PŘÍKLAD

Importujte PUT request do Postman. Request pojmenujte "PUT Regres.in users" a uložte do složky "HTTP Requests". Následně request provolejte.

#### curl:

```
curl --location --request PUT 'https://reqres.in/api/users/2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "Petr",
  "job": "Tester"
}'
```

### Úkol:

přesuňte request "HTTP Request 3" do složky "HTTP Requests", vložte URL

<https://reqres.in/api/users/2>, změňte typ HTTP requestu na PUT, vložte Raw/JSON body:

```
{
  "name": "Principal",
  "job": "PUT"
}
```

### PATCH

PATCH slouží k úpravě stávajících dat.

### PŘÍKLAD

Importujte PATCH request do Postman. Request pojmenujte "PATCH Regres.in users" a uložte do složky "HTTP Requests". Následně request provolejte.

#### curl:

```
curl --location --request PATCH 'https://reqres.in/api/users/22' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "Adam",
  "job": "manager"
}'
```

## DELETE

DELETE slouží ke smazání stávajících dat.

## PŘÍKLAD

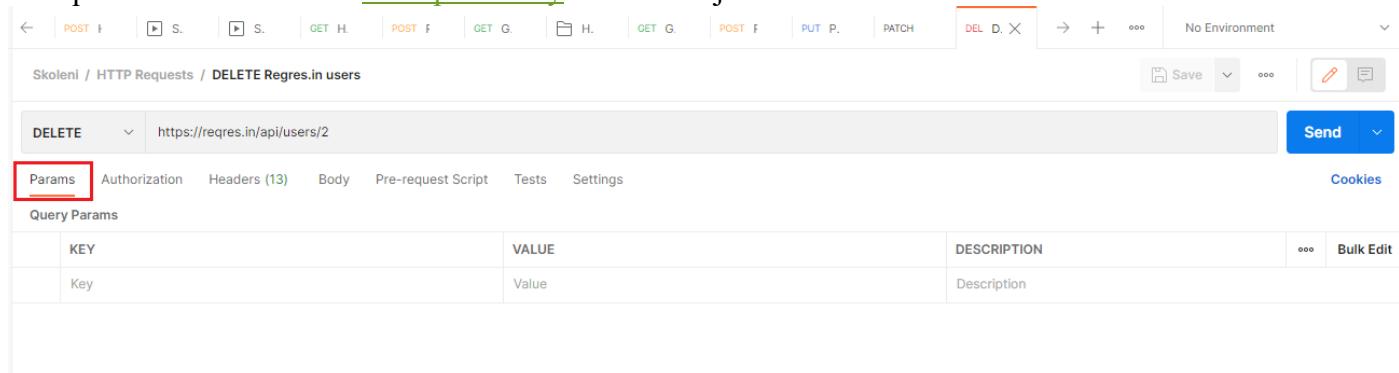
Importujte DELETE request do Postman. Request pojmenujte "DELETE Regres.in users" a uložte do složky "HTTP Requests". Následně request provolejte.

### CURL:

```
curl --location --request DELETE 'https://reqres.in/api/users/2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

## PARAMS

V requestu můžete zadávat URL parametry. Naleznete je v záložce Params.



| KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description |     |           |

Každý URL parametr musí mít svůj klíč (key) a hodnotu (value).

Obecná syntaxe URL parametrů:

```
http://nejakyweb.cz/get?KEY1=VALUE1&KEY2=VALUE2
```

Postman obsahuje ještě pole Description. To slouží pro popis daného klíče a hodnoty.

## PŘÍKLAD

Vytvořte novou složku "Params" v kolekci školení.

Importujte následující request do nově vytvořené složky. Pojmenujte ho "GET Users params". Request provolejte.

### CURL:

```
curl 'https://reqres.in/api/users?page=2' \
-H 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
-H 'Referer: https://reqres.in/' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
-H 'sec-ch-ua-platform: "Windows"' \
-H 'Content-Type: application/json' \
--compressed
```

## Úkol:

V nově vytvořeném requestu změňte parametr "page" na hodnotu 3. Request uložte a provolejte.

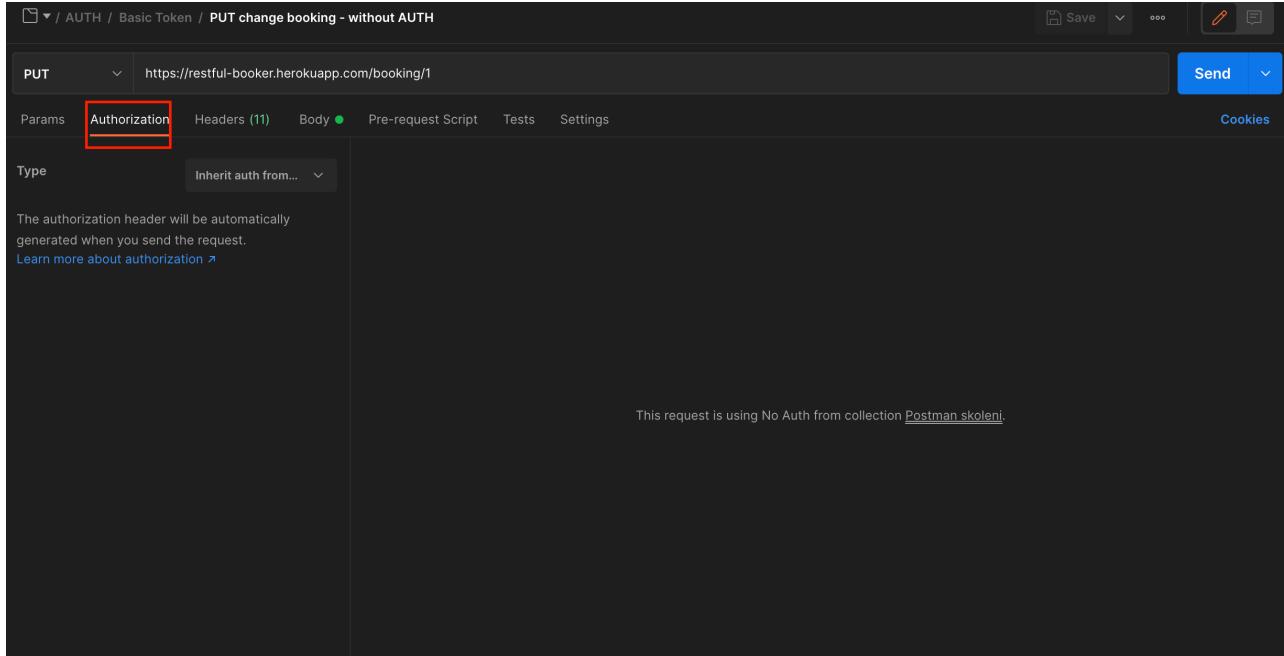
## AUTH

Autentikace je ověření volání requestu v případě, že to API vyžaduje (podobně jako přihlášení například do e-mailu). Existuje několik různých typů autentikace.

Pokud autentikaci nezadáme, dostaneme jako odpověď status 401.

Běžně používané typy autorizace v Postman:

- Inherit from parent (je možné autentikaci nastavit na úrovni kolekce nebo složky, jak na to si ukážeme [níže](#))
- Basic Auth
- API key
- Bearer token
- OAUTH 2.0



The screenshot shows the Postman interface with a request configuration. The method is set to PUT, the URL is <https://restful-booker.herokuapp.com/booking/1>, and the collection is AUTH / Basic Token / PUT change booking - without AUTH. The Authorization tab is selected, highlighted with a red border. Other tabs include Params, Headers (11), Body (green dot), Pre-request Script, Tests, Settings, and Cookies. A note below the tabs states: "The authorization header will be automatically generated when you send the request." and provides a link to "Learn more about authorization". At the bottom right of the main area, there is a message: "This request is using No Auth from collection Postman skolení." A blue "Send" button is located at the top right of the interface.

Vytvořte novou složku "Auth" v kolekci "Skolení"

### BASIC AUTH

Basic auth je nejpodobnější běžnému přihlášení na webové stránce.

Do requestu se přidá jméno a heslo, které se pošlou s requestem.

### PŘÍKLAD

Ve složce "auth" vytvořte novou: "Basic auth".

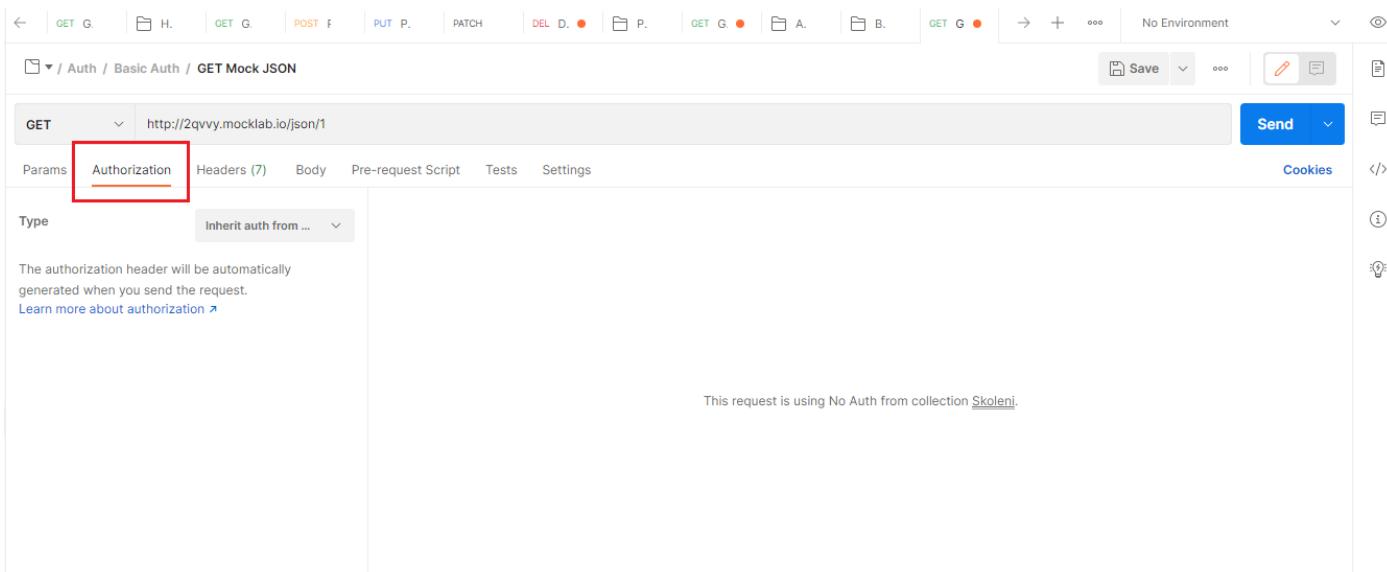
Vytvořte nový HTTP Request.

Pojmenujte ho: GET Mock json

URL: <http://2qvvy.mocklab.io/json/1>

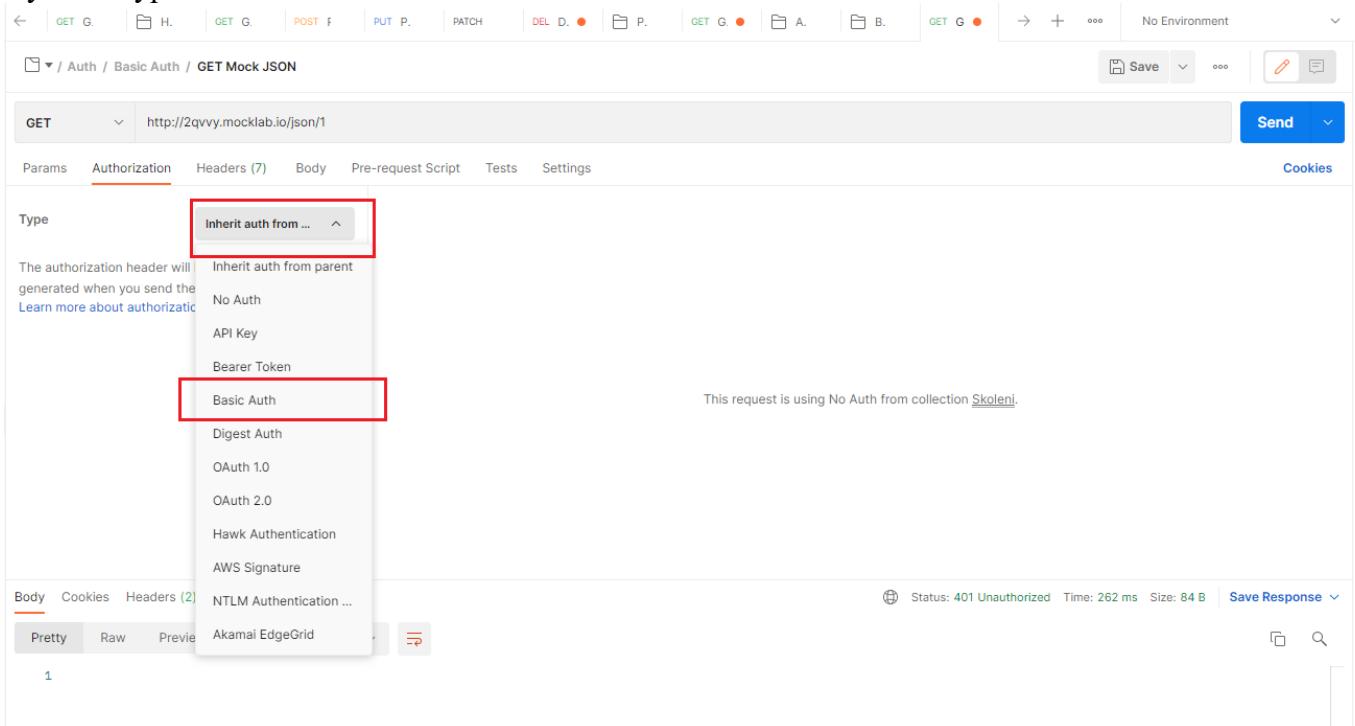
Nejdříve ho zkuste provolat bez autentikace. Dostanete odpověď se statusem 401.

Otevřete záložku Authorization v requestu.



This request is using No Auth from collection [Skolení](#).

## Vyberte Type: Basic Auth



The authorization header will be automatically generated when you send the request.  
[Learn more about authorization](#)

This request is using No Auth from collection [Skolení](#).

Status: 401 Unauthorized Time: 262 ms Size: 84 B | Save Response

Vložte:

- Username: userSkoleni
- Password: Test123



The screenshot shows the Postman interface with a GET request to `http://2qvvy.mocklab.io/json/1`. The **Authorization** tab is selected, showing `Basic Auth` as the type. A warning message is displayed: **Heads up!** These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#). The **Username** field contains `userSkoleni` and the **Password** field contains `Test123`, both highlighted with a red box. A [Show Password](#) checkbox is checked.

Request uložte a znova provolejte. Měli byste dostat response 200 s body:

```
{  
  "id": 1,  
  "value": "úspěšně jsi provolal request."  
}
```

## Úkol:

Vytvořte nový request ve složce "Basic Auth":

| Pole               | Hodnota   |
|--------------------|---|
| Název              | POST Mock JSON  |
| URL                | <a href="http://2qvvy.mocklab.io/json">http://2qvvy.mocklab.io/json</a> |
| Typ callu          | POST  |
| Authorization type | Basic Auth  |
| Username           | userSkoleni   |
| Password           | Test123   |

## Body/raw/JSON:

```
{  
  "id": 12345,  
  "value": "abc-def-ghi"  
}
```

Provolejte request, cíl je dostat response 201:

```
{  
    "status": "úspěšně jsi vytvořil záznam"  
}
```

## API KEY

Vytvoříme složku "API key" ve složce "Auth"

Importujeme následující request, pojmenujeme ho "PUT Restful-Booker" a uložíme do složky "API key".  
cURL:

```
curl --location --request PUT 'https://restful-booker.herokuapp.com/booking/1' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--data-raw '{
    "firstname" : "James",
    "lastname" : "Brown",
    "totalprice" : 111,
    "depositpaid" : true,
    "bookingdates" : {
```

```
{
    "checkin": "2018-01-01",
    "checkout": "2019-01-01"
},
    "additionalneeds": "Breakfast"
}'
```

Request provolejte. Dostanete response 403.

Importujte následující cURL do složky "API key", pojmenujte ho "POST Restful-booker auth" **cURL:**

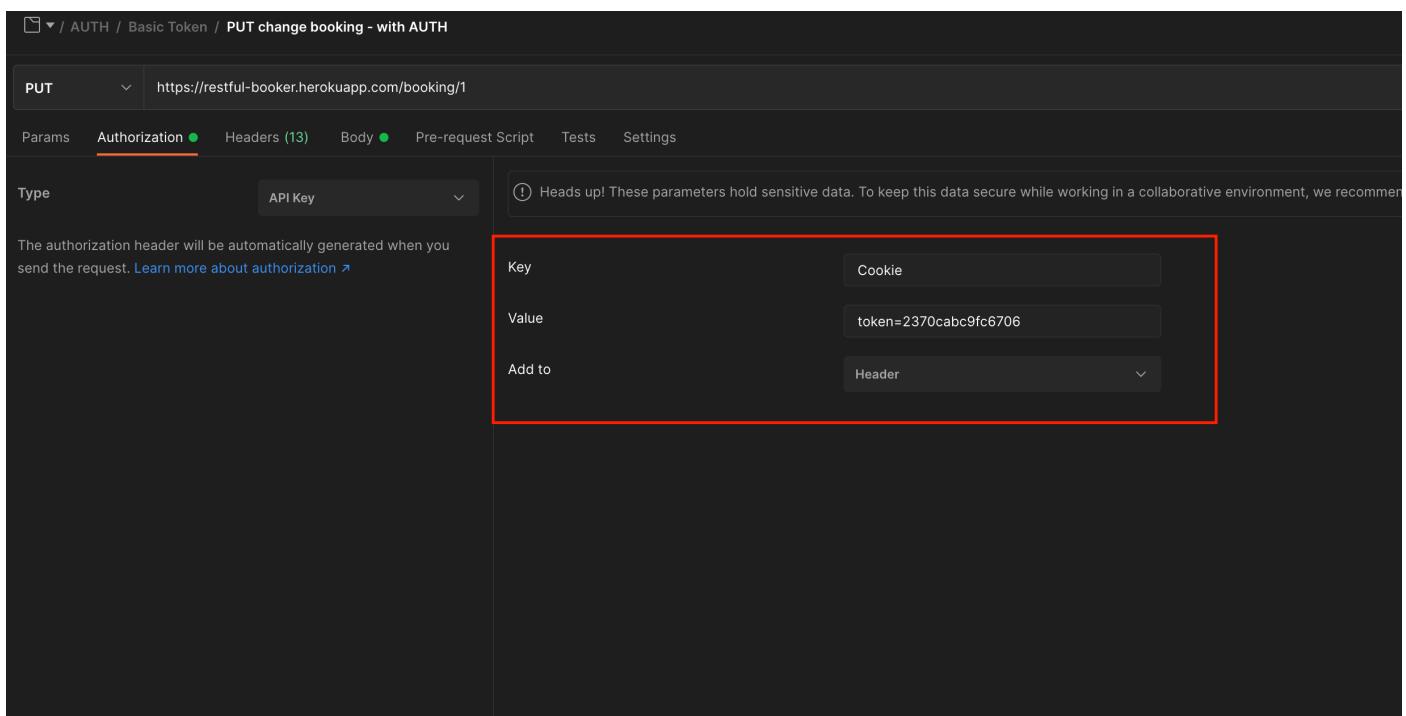
```
curl --location --request POST 'https://restful-booker.herokuapp.com/auth' \
--header 'Content-Type: application/json' \
--data-raw '{
    "username": "admin",
    "password": "password123"
}'
```

Provolejte ho, vrátit by se vám měla response s autorizačním tokenem:

```
{
    "token": "fdd0917475c0553"
}
```

Hodnotu z fieldu "token" zkopírujte a vložte jako autorizaci "API KEY" do requestu "PUT Restful-Booker" jako:

| Pole   | Hodnota             |
|--------|---------------------|
| Key    | Cookie              |
| Value  | token=TOKEN_HODNOTA |
| Add to | Header              |



The screenshot shows the Postman interface with the following details:

- Method:** PUT
- URL:** https://restful-booker.herokuapp.com/booking/1
- Authorization:** Type: API Key
- Body:** (Empty)
- Headers:** (13 items)
- Pre-request Script:** (Empty)
- Tests:** (Empty)
- Settings:** (Empty)

In the Authorization section, there is a warning message: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using environment variables or a vault." Below this, the token input field is highlighted with a red box, showing the value "token=2370cab9fc6706". The "Add to" dropdown is set to "Header".

následně request provolejte (pozor, token platí pouze chvíli). Dostanete odpověď:

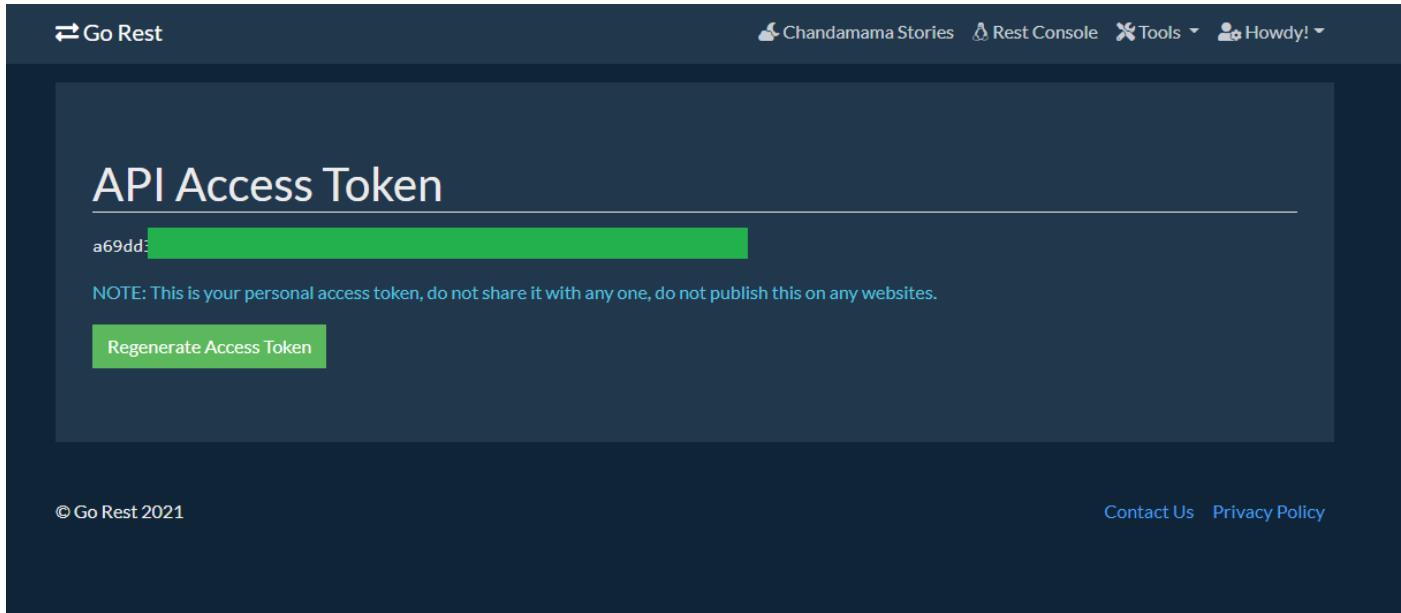
```
{
    "firstname": "James",
    "lastname": "Brown",
    "totalprice": 111,
    "depositpaid": true,
    "bookingdates": {
        "checkin": "2018-01-01",
        "checkout": "2019-01-01"
    },
}
```

```
        "additionalneeds": "Breakfast"  
    }
```

## TOKEN AUTH

Vytvořte složku "Token" ve složce "Auth".

Otevřete stránku [gorest](#), zaregistrujte se a vygenerujte API Access Token.



The screenshot shows a dark-themed web application for generating API access tokens. At the top, there are navigation links: 'Go Rest', 'Chandamama Stories', 'Rest Console', 'Tools', and 'Howdy!'. The main title is 'API Access Token'. Below the title, a large green bar contains the generated token: 'a69dd...'. A note below the bar reads: 'NOTE: This is your personal access token, do not share it with any one, do not publish this on any websites.' A green button labeled 'Regenerate Access Token' is located at the bottom left of the token area. At the bottom of the page, there is footer text: '© Go Rest 2021' on the left and 'Contact Us Privacy Policy' on the right.

Importujte request níže, pojmenujte ho "POST gorest users", uložte do složky "Token".

### cURL:

```
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer ACCESS-TOKEN" -XPOST "https://gorest.co.in/public/v1/users" -d '{"name":"Tenali Ramakrishna", "gender":"male", "email":"tenali.ramakrishna@15ce.com", "status":"active"}'
```

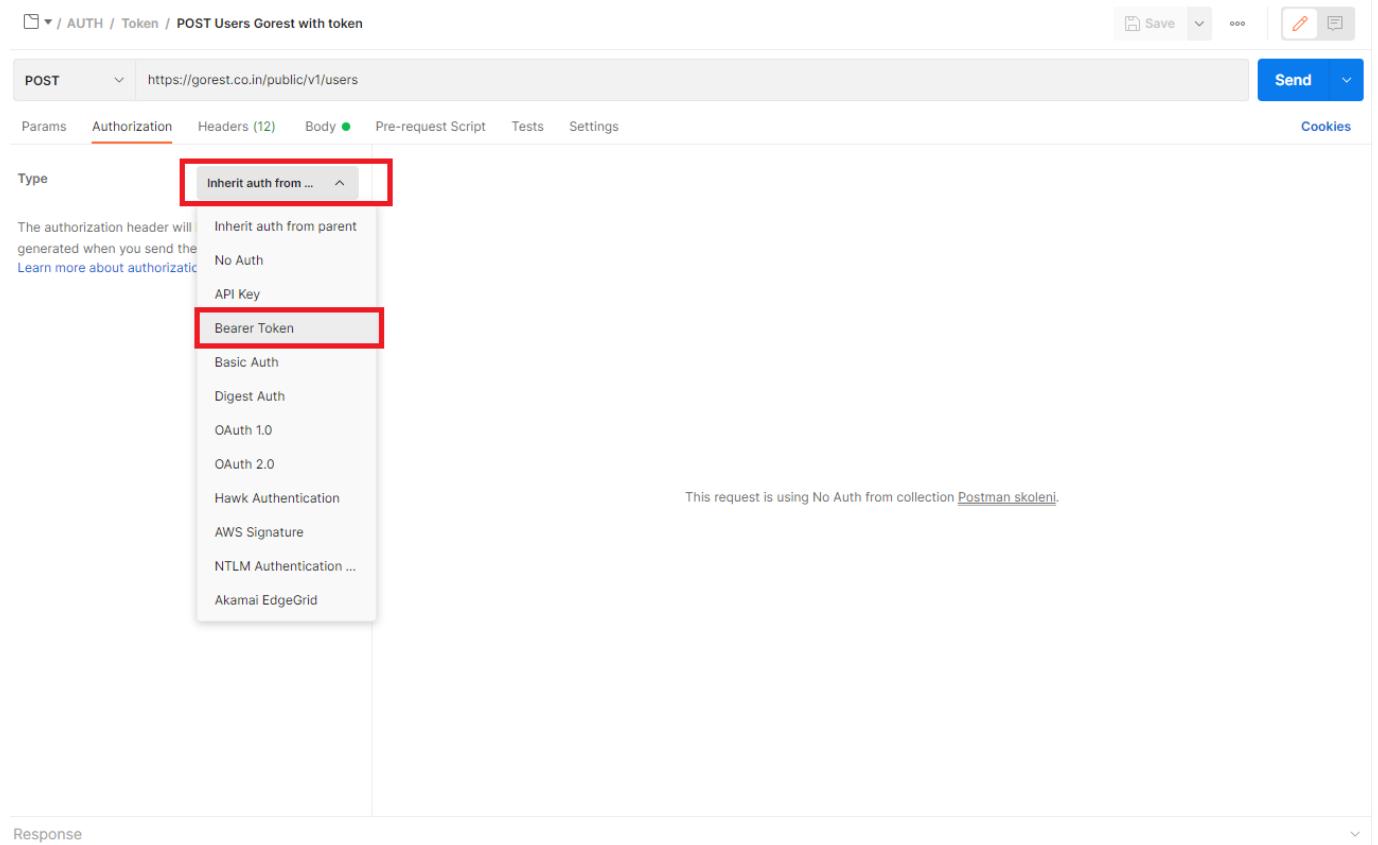
Zkuste ho provolat. Dostaneme response se statusem 401:

```
{  
    "meta": null,  
    "data": {  
        "message": "Authentication failed"  
    }  
}
```

Vyplňte Token do Authorization:

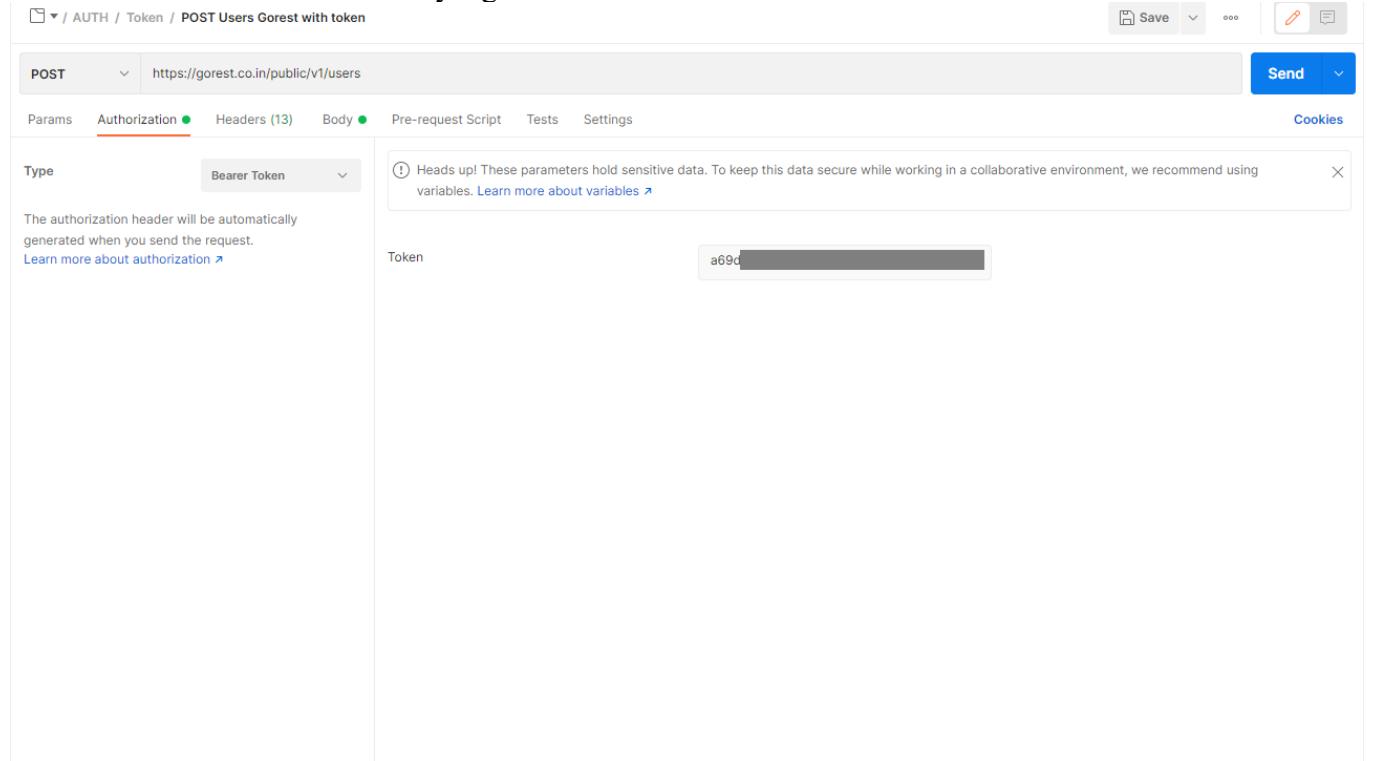
- Otevřete záložku Authorization

- Zvolte Type/Bearer token



The screenshot shows the Postman interface for a POST request to <https://gorest.co.in/public/v1/users>. The 'Authorization' tab is selected. A dropdown menu is open under 'Type', with 'Inherit auth from ...' at the top and 'Bearer Token' highlighted with a red box. Other options like 'No Auth', 'API Key', 'Basic Auth', etc., are listed below. A note on the left says: 'The authorization header will be automatically generated when you send the request.' A note on the right says: 'This request is using No Auth from collection Postman skolení.'

- Do fieldu Token vložte access key z gorest webu



The screenshot shows the same Postman interface as above, but now the 'Token' field contains the value 'a69d...'. A tooltip message appears above the field: '(?) Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)'.

Následně request znovu provolejte. Vrátí se response se statusem 422 s body:

```
{
  "meta": null,
```

```
"data": [
  {
    "field": "email",
    "message": "has already been taken"
  }
]
```

## OAUTH 2.0

OAuth 2.0 je otevřený protokol používaný pro autorizační flow. Cílem je poskytnout bezpečnou autentizaci a autorizaci oproti API různých služeb, a to jednotně pro desktopové, mobilní i webové aplikace. Provozovatelům služby, která OAuth identity poskytuje, dává OAuth možnost sdílet uživatelská data a identity, aniž by uživatelé museli prozrazovat své heslo komukoliv dalšímu.

## PŘÍKLAD

---

Pro vyzkoušení, jak funguje OAuth 2.0 využijeme API Imguru.

1. Zaregistrujte se na: [imgur.com](https://imgur.com), poté se přihlašte
2. Vytvořte novou [registraci](#) aplikace pro imgur, vyberte "OAuth 2 authorization without a callback URL".

## Register an Application

ERROR: Invalid Captcha

Application name:

Nejaky nazev

Authorization type:

- OAuth 2 authorization with a callback URL
- OAuth 2 authorization without a callback URL
- Anonymous usage without user authorization

Authorization callback URL:

The callback URL is used to determine where Imgur redirects the user after they authorize your access request, and it can include query parameters. The redirect will include the same query parameters, as well as the access token, which your application must be able to parse. It can also be changed in the "applications" section of your account settings.

Application website (optional):

Email:

Description:



I'm not a robot



reCAPTCHA  
[Privacy](#) • [Terms](#)

[submit](#)

[Have Imgur? Join our team!](#) [about](#) [store](#) [help](#) [blog](#) [request deletion](#) [forum](#) [terms](#) [privacy](#) [ccpa](#) [apps](#) [api](#) [advertise](#) [ad choices](#)

3. Uložte si Client ID a Client Secret

## Great! Now you can get started with the API!

For public read-only and anonymous resources, such as getting image info, looking up user comments, etc. all you need to do is send an authorization header with your client\_id in your requests. This also works if you'd like to upload images anonymously (without the image being tied to an account), or if you'd like to create an anonymous album. This lets us know which application is accessing the API.

**Authorization: Client-ID YOUR\_CLIENT\_ID**

For accessing a user's account, please visit the [OAuth2 section of the docs](#).

**Client ID:**

5ae2ad751245d6a

**Client secret:**

ce44b334d

Vytvořte novou složku "OAuth" ve složce "Auth".

Nainportujte request pro autorizaci imguru, pojmenujte ho "POST imgur auth". V autorizaci zvolte Bearer Token, ale zatím ho nevyplňujte

**cURL:**

```
curl --location --request POST 'https://api.imgur.com/oauth2/token' \
--header 'Authorization: Bearer ee928db221212f9b8812d55de319de389e9fa890' \
--form 'refresh_token=#refreshToken' \
--form 'client_id=#clientId' \
--form 'client_secret=#clientSecret' \
--form 'grant_type=refresh_token'
```

Pokud ho ted' provoláte, dostanete response 400:

```
{
  "data": {
    "error": "The client credentials are invalid",
    "request": "/oauth2/token",
    "method": "POST"
  },
  "success": false,
  "status": 400
}
```

Musíme nejdříve nastavit OAuth 2.0 autorizaci

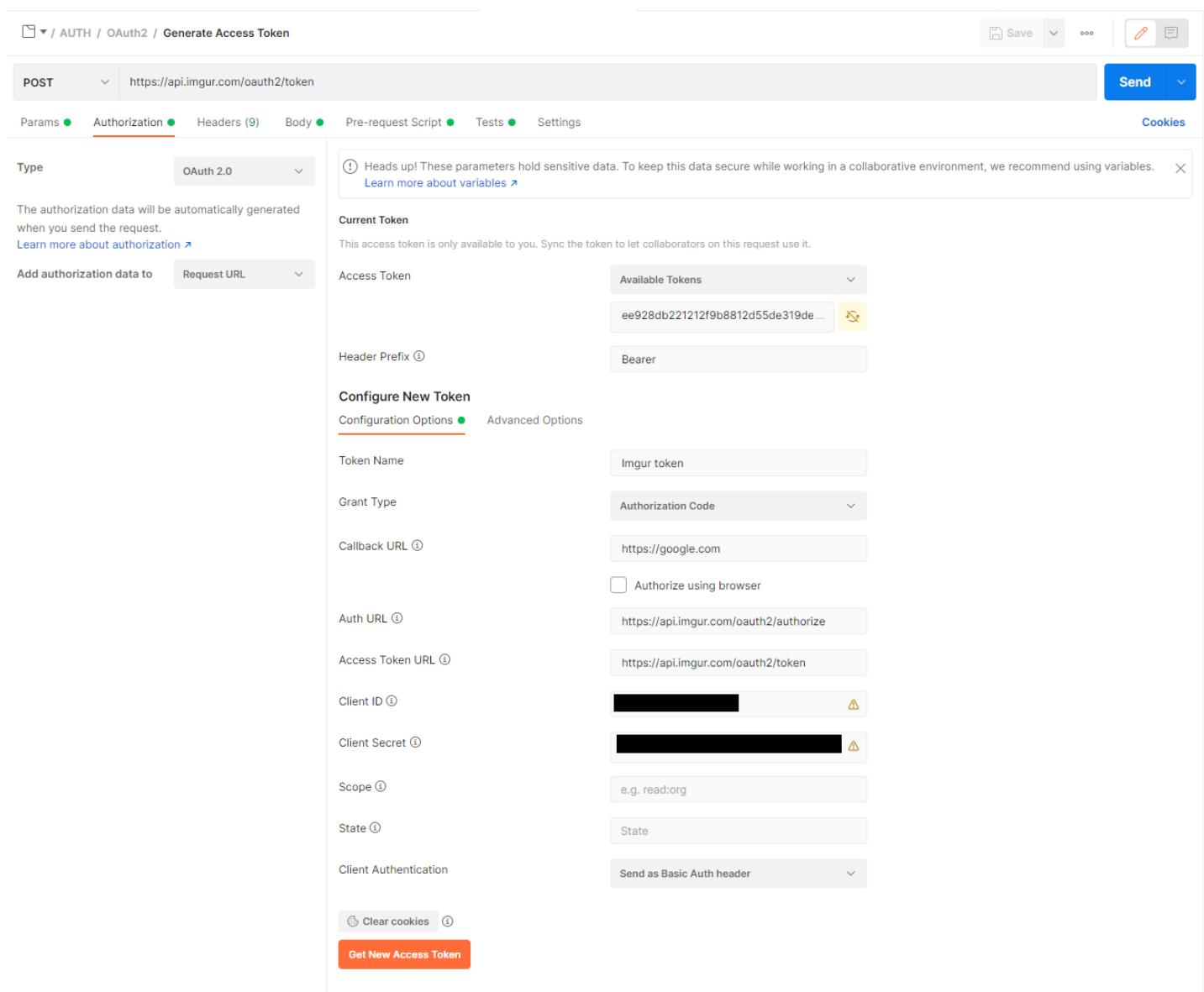
- Otevřete záložku Authorization
- Zvolte hodnotu Type/OAUTH 2.0

**Hodnoty pro Token:**

| Pole             | Hodnota   |
|------------------|---|
| Access Token     | Available Tokens  |
| Header Prefix    | Bearer  |
| Token name       | imgurToken  |
| Auth URL         | <a href="https://api.imgur.com/oauth2/authorize">https://api.imgur.com/oauth2/authorize</a> |
| Access Token URL | <a href="https://api.imgur.com/oauth2/token">https://api.imgur.com/oauth2/token</a>         |
| Callback URL     | <a href="https://imgur.com">https://imgur.com</a>   |
| Client ID        | Vyplňte z imgur   |
| Client Secret    | Vyplňte z imgur   |
| Scope            | Nechat prázdné  |
| State            | Nechat prázdné  |

## Client Authentication

Send as Basic Auth header



The screenshot shows the Postman interface with the following configuration:

- Method:** POST
- URL:** https://api.imgur.com/oauth2/token
- Type:** OAuth 2.0
- Current Token:** Access Token (Available Tokens: ee928db221212f9b8812d55de319de...)
- Header Prefix:** Bearer
- Configure New Token:**
  - Configuration Options:** Advanced Options (selected)
  - Token Name:** Imgur token
  - Grant Type:** Authorization Code
  - Callback URL:** https://google.com
  - Auth URL:** https://api.imgur.com/oauth2/authorize
  - Access Token URL:** https://api.imgur.com/oauth2/token
  - Client ID:** [REDACTED]
  - Client Secret:** [REDACTED]
  - Scope:** e.g. read:org
  - State:** State
  - Client Authentication:** Send as Basic Auth header

At the bottom, there are buttons for **Clear cookies** and **Get New Access Token**.

Stiskněte tlačítko Get New Access Token a přihlaste se do imguru (pokud Vám vyskočí okno s chybou, zkонтrolujte zadáne údaje, zejména callback URL a zkuste to znova). Následně Vám Postman zobrazí stránku s tokenem. Access Token a refresh\_token si uložte bokem, budete je ještě potřebovat.

**OAuth 2.0**

**MANAGE ACCESS TOKENS**

| All Tokens  | Delete |
|-------------|--------|
| Imgur token |        |

**Token Details**

|                  |                       |
|------------------|-----------------------|
| Token Name       | Imgur token           |
| Access Token     | ee928db2[REDACTED]    |
| Token Type       | bearer                |
| expires_in       | 315360000             |
| scope            | null                  |
| refresh_token    | e9e9e40b7af[REDACTED] |
| account_id       | 156602709             |
| account_username | petrfifk              |

**Use Token**

Do okna Manage Access Tokens se dostanete také kliknutím na Available Tokens/Manage tokens v tabu Authorization.

**/ AUTH / OAuth2 / Generate Access Token working**

**POST** <https://api.imgur.com/oauth2/token> **Send**

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Type OAuth 2.0

**Current Token**  
This access token is only available to you. Sync the token to let collaborators on this request use it.

Access Token **Available Tokens**

- Imgur token
- Manage Tokens

Header Prefix [①](#)

**Configure New Token**

Configuration Options Advanced Options

Token vložte do pole pod Access Token (pokud se nevyplní automaticky).

**Current Token**  
This access token is only available to you. Sync the token to let collaborators on this request use it.

Access Token **Available Tokens**

ee928db221212f9b8812d55de319de ... [🔗](#)

Header Prefix [①](#)

**Configure New Token**

Configuration Options **Advanced Options**

Token Name Imgur token

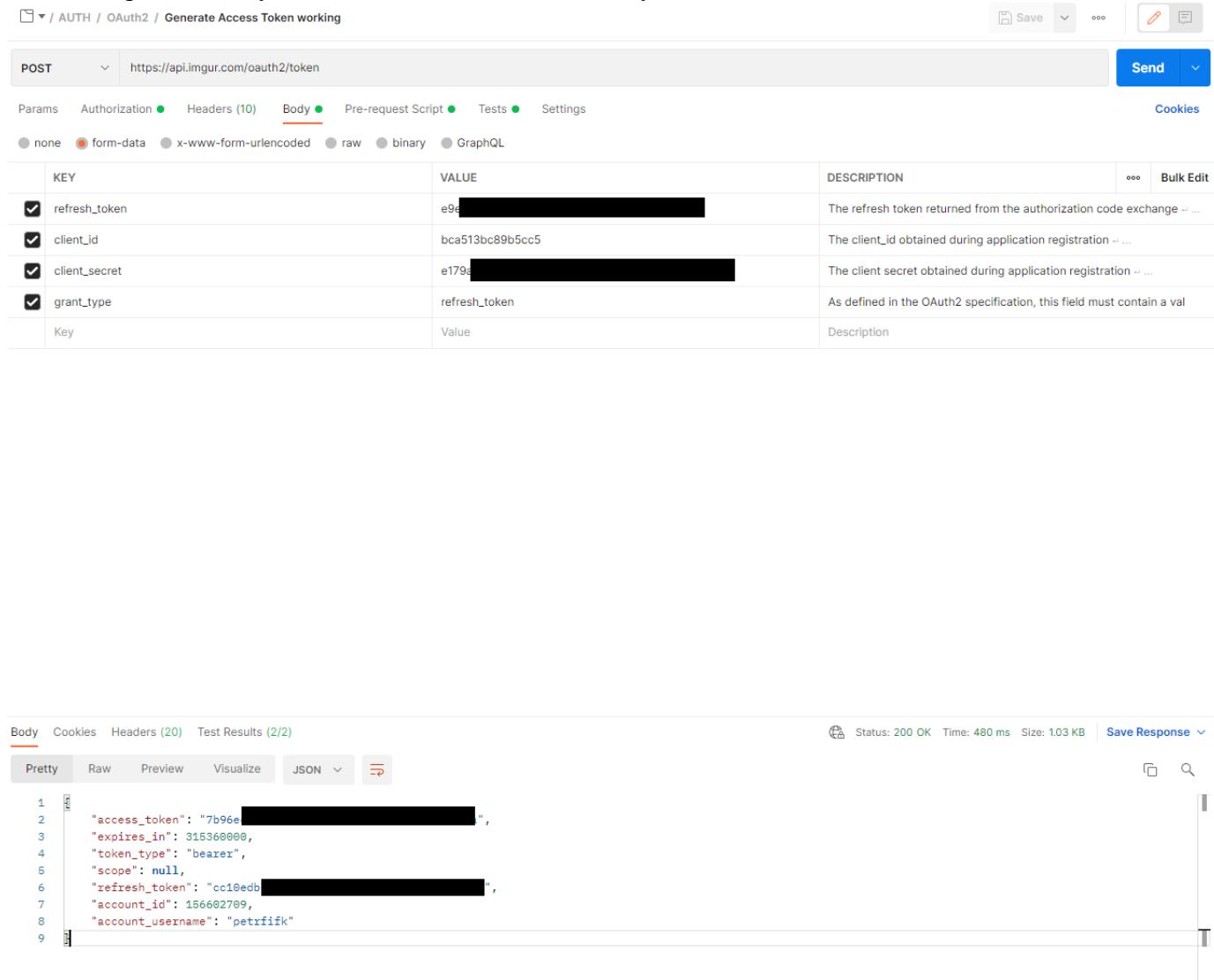
Grant Type Authorization Code

V Body Requestu vyplňte:

- refresh\_token

- client\_id
- client\_secret

Pošlete request. Měl by se Vám vrátit status 200 s detaily o Tokenu.



The screenshot shows the Postman interface with a POST request to <https://api.imgur.com/oauth2/token>. The body is set to form-data with the following parameters:

| KEY           | VALUE           | DESCRIPTION   |
|---------------|-----------------|---|
| refresh_token | e9e[REDACTED]   | The refresh token returned from the authorization code exchange - ... |
| client_id     | bca513bc89b5cc5 | The client_id obtained during application registration - ...          |
| client_secret | e179a[REDACTED] | The client secret obtained during application registration - ...      |
| grant_type    | refresh_token   | As defined in the OAuth2 specification, this field must contain a val |
| Key           | Value           | Description   |

Below the request, the response is shown with a status of 200 OK, time 480 ms, and size 1.03 KB. The response body is a JSON object:

```

1 "access_token": "7b96e[REDACTED]",
2   "expires_in": 315360000,
3   "token_type": "bearer",
4   "scope": null,
5   "refresh_token": "cc10edb[REDACTED]",
6   "account_id": 156602709,
7   "account_username": "petrififik"
8

```

Tento token pak můžete používat v dalších requestech.

### Úkol:

Nainportujte následující soubor do složky "OAuth", nazvěte jej "GET imgur images" **cURL:**

```
curl --location --request GET 'https://api.imgur.com/3/account/me/images' \
```

Provolejte request. Jako response se vrátí 401.

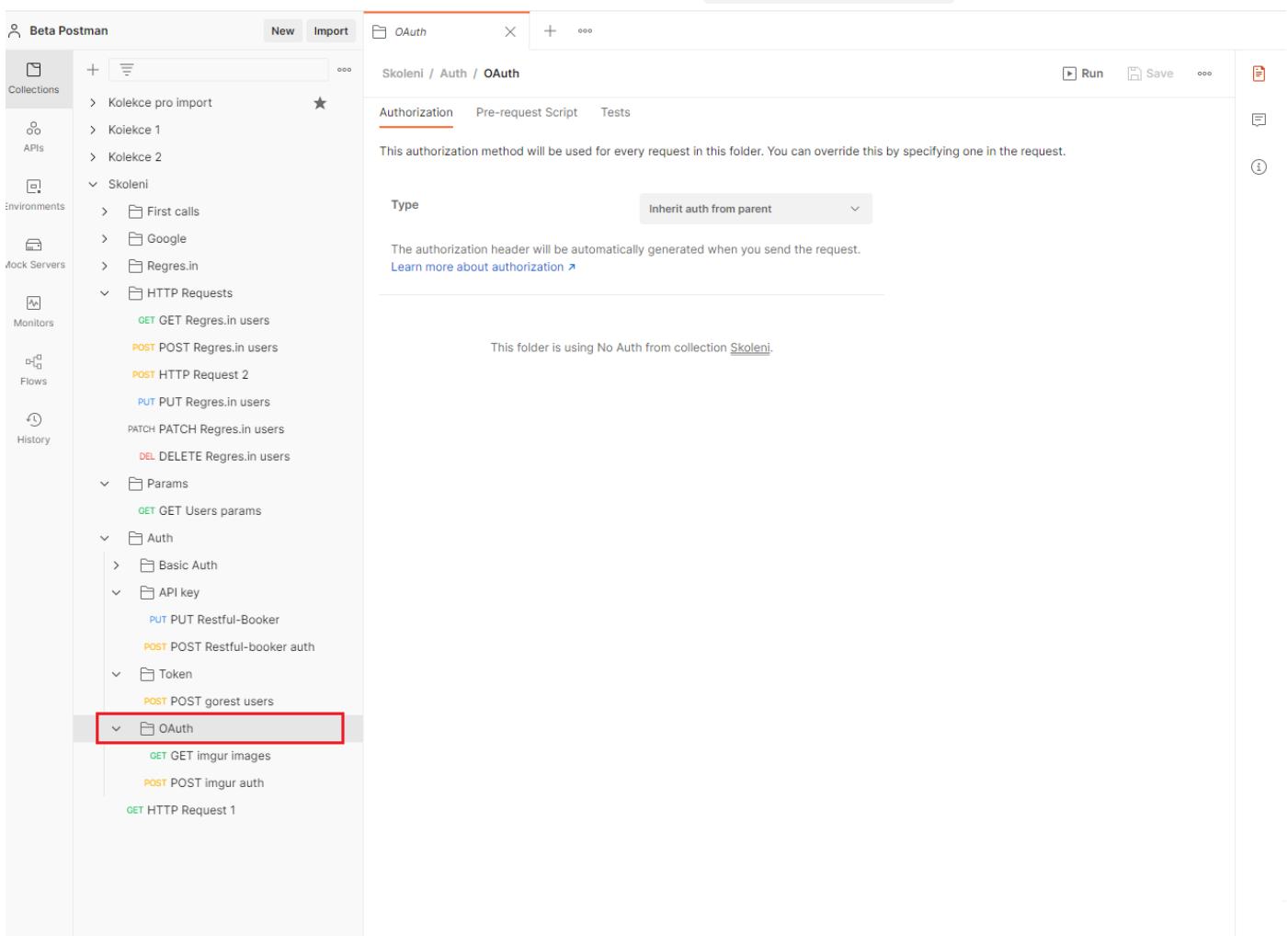
V záložce Authorization zvolte OAuth 2.0 vložíme token do Access Token z minulého a znova request provolejte.

---

### AUTENTIKACE COLLECTION/SLOŽKY

Omezení duplicit je důležité, proto Postman umožňuje nastavit Autentikaci na úrovni kolekce nebo složky.

Otevřete složku "OAuth" levým kliknutím myši. Otevře se detail folder.



The screenshot shows the Postman interface with the left sidebar expanded. The 'Collections' section lists several items under the 'Skoleni' folder, including 'Kolekce pro import', 'Kolekce 1', 'Kolekce 2', 'First calls', 'Google', 'Regres.in', and 'HTTP Requests'. Under 'HTTP Requests', there are several requests: 'GET Regres.in users', 'POST POST Regres.in users', 'POST HTTP Request 2', 'PUT PUT Regres.in users', 'PATCH PATCH Regres.in users', 'DEL DELETE Regres.in users', 'GET GET Users params', 'Auth' (with sub-options 'Basic Auth' and 'API key'), 'PUT PUT Restful-Booker', 'POST POST Restful-booker auth', 'Token' (with 'POST POST gorest users'), and 'OAuth' (which is highlighted with a red box). The right panel shows the 'OAuth' tab for the 'Skoleni / Auth / OAuth' folder. It has tabs for 'Authorization', 'Pre-request Script', and 'Tests'. The 'Authorization' tab is selected, showing the message: 'This authorization method will be used for every request in this folder. You can override this by specifying one in the request.' Below it, it says 'Type' and 'Inherit auth from parent'. A note states: 'The authorization header will be automatically generated when you send the request.' and a link to 'Learn more about authorization'. At the bottom, it says 'This folder is using No Auth from collection [Skoleni](#)'.

Následně v záložce Authorization zvolte OAuth 2.0, složka automaticky převezme nastavení konfigurace. Vy jen musíte vybrat současný token z dropdownu "Access Token"

OAuth

POST POST imgur auth

GET GET imgur images

...  
oo

Skolení / Auth / OAuth

Authorization ● Pre-request Script Tests

This authorization method will be used for every request in this folder. You can override this by specifying one in the request.

Type

OAuth 2.0



The authorization data will be automatically generated when you send the request.

[Learn more about authorization ↗](#)

Add auth data to

Request Headers



! Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables ↗](#)



#### Current Token

This access token is only available to you. Sync the token to let collaborators on this request use it.

Access Token

Available Tokens



imgurToken

Manage Tokens

Header Prefix ⓘ

|                  |   |
|------------------|---|
| Available Tokens | ^ |
| imgurToken       |   |
| Manage Tokens    |   |

#### Configure New Token

##### Some changes to the Token configurations

Token configuration settings are now shared as a part of the folder. The fields under reflect the last settings you used to configure an OAuth2.0 token. They are not shared with anyone just yet. You can still use these settings to generate a new access token. However, if you wish to edit them, they will be shared as a part of the folder and will be visible to everyone who can view this folder.

[Edit token configuration](#)

Configuration Options ● Advanced Options

Token Name

imgurToken

...  
oo

Upravte v obou requestech ve složce OAuth autorizaci na "Inherit auth from parent"

Auth / OAuth / GET imgur images

GET https://api.imgur.com/3/account/me/images

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type Inherit auth from ... ^

The authorization header will be generated when you send the request. Learn more about authorization.

Inherit auth from parent (highlighted with a red box)

No Auth API Key Bearer Token Basic Auth Digest Auth OAuth 1.0 OAuth 2.0 Hawk Authentication AWS Signature NTLM Authentication ... Akamai EdgeGrid

This request is using No Auth from collection Skolení.

Response

Click Send to get a response



Zkuste requesty znovu provolat (nezapomínejte své změny ukládat).

## HEADERS

Headers neboli hlavičky HTTP requestu slouží pro dodatečné informace requestu. Mohou zde být například:

- Cookies
  - Cookies jsou malé datové soubory z webových stránek, která obsahují určitá data. Při každé další návštěvě webu prohlížeč tyto cookies automaticky posílá dané stránce.
  - Cookies mohou obsahovat data jako například:
    - Údaje o přihlášení, uživatel se nemusí přihlašovat vždy, když přijde na stránku
    - Informace o chování uživatele
    - Data pro účely reklamy cílené na uživatele
- Typ requestu
- Jazyk
- Informace o prohlížeči
- A další

Více o hlavičkách najeznete například [zde](#).

## PŘÍKLAD

Vytvořte novou složku v kolekci "Skolení", nazvěte ji "Headers"-

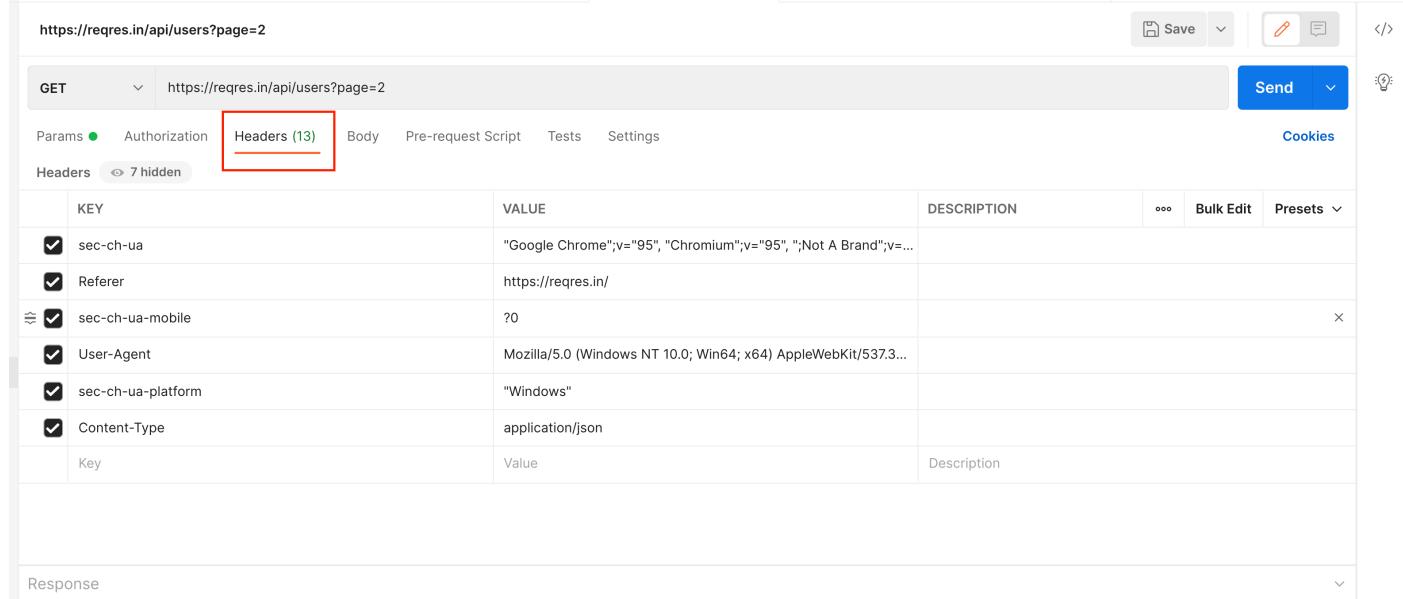
Importujte request pro regres.in API, nazvěte ho "GET regres.in users". Request má která má hlavičky, některé si vysvětlíme:

## URL:

```
curl --location --request GET 'https://reqres.in/api/users?page=2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
```

```
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

Hlavičky naleznete v requestu, záložka "Headers"



The screenshot shows the Postman interface with a request to `https://reqres.in/api/users?page=2`. The 'Headers' tab is selected and highlighted with a red box. Other tabs visible include 'Params', 'Authorization', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Headers' table lists the following key-value pairs:

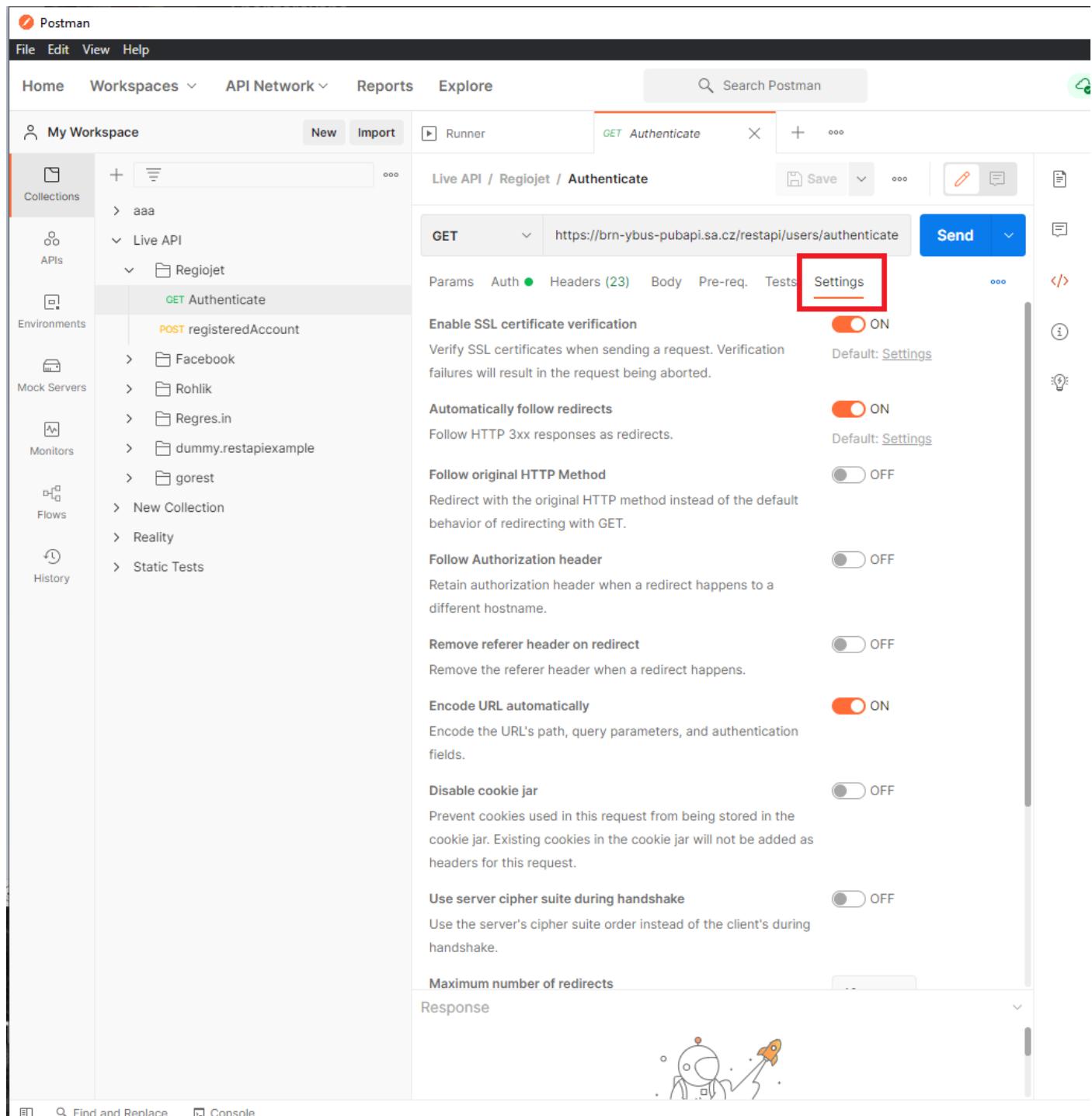
| KEY  | VALUE   | DESCRIPTION | ... | Bulk Edit | Presets |
|--|---|-------------|-----|-----------|---------|
| <input checked="" type="checkbox"/> sec-ch-ua          | "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v=... |             |     |           |         |
| <input checked="" type="checkbox"/> Referer            | https://reqres.in/  |             |     |           |         |
| <input checked="" type="checkbox"/> sec-ch-ua-mobile   | ?0  |             | x   |           |         |
| <input checked="" type="checkbox"/> User-Agent         | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.3...  |             |     |           |         |
| <input checked="" type="checkbox"/> sec-ch-ua-platform | "Windows"   |             |     |           |         |
| <input checked="" type="checkbox"/> Content-Type       | application/json  |             |     |           |         |
| Key  | Value   | Description |     |           |         |

V Requestu vidíme například tyto hlavičky:

| Header              | Význam  |
|---------------------|---|
| <b>Referer</b>      | Slouží serveru k identifikaci odkud uživatel přichází                               |
| <b>User-Agent</b>   | Slouží serveru k identifikaci aplikace, operačního systému...                       |
| <b>Content-Type</b> | Slouží k určení typu obsahu, který je na server posílan, například application/json |

## SETTINGS

Konfiguraci můžeme měnit i pro jednotlivý request. Naleznete je v záložce "Settings". Nastavuje se zde například SSL verifikace.



The screenshot shows the Postman application interface. On the left, there's a sidebar with sections like 'My Workspace', 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. The main area displays a 'GET Authenticate' request for 'Live API / Regiojet / Authenticate' with the URL <https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate>. A red box highlights the 'Settings' tab in the top right of the request details panel. Below it, several configuration options are listed with toggle switches:

- Enable SSL certificate verification**: ON (Default: Settings)
- Automatically follow redirects**: ON (Default: Settings)
- Follow original HTTP Method**: OFF
- Follow Authorization header**: OFF
- Remove referer header on redirect**: OFF
- Encode URL automatically**: ON
- Disable cookie jar**: OFF
- Use server cipher suite during handshake**: OFF
- Maximum number of redirects**: [input field]

## DATA V REQUESTECH

### PARAMS

Postman podporuje parametrizaci v URL a URI. Poznámka: URL je součásti URI. Více detailu o URI naleznete například [zde](#).

### URL PARAMETRY

URL Parametry zadáváme v záložce "Params" v requestu.

https://reqres.in/api/users?page=2

Save Send </>

GET https://reqres.in/api/users?page=2

Params Authorization Headers (13) Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

| KEY  | VALUE   | DESCRIPTION | ... | Bulk Edit | Presets |
|--|---|-------------|-----|-----------|---------|
| <input checked="" type="checkbox"/> sec-ch-ua          | "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v=... |             |     |           |         |
| <input checked="" type="checkbox"/> Referer            | https://reqres.in/  |             |     |           |         |
| <input checked="" type="checkbox"/> sec-ch-ua-mobile   | ?0  |             |     |           |         |
| <input checked="" type="checkbox"/> User-Agent         | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.3...  |             |     |           |         |
| <input checked="" type="checkbox"/> sec-ch-ua-platform | "Windows"   |             | ×   |           |         |
| <input checked="" type="checkbox"/> Content-Type       | application/json  |             |     |           |         |
| Key  | Value   | Description |     |           |         |

Response

## PŘÍKLAD

V příkladu zadáme vyhledání spoje v Regiojet API.

Vytvořte složku v kolekci "Skolení" a nazavěte ji "Params"

Importujte cURL níže, nazavěte ho "GET regiojet routes" a uložte do složky "Params"

cURL:

```
curl --location --request GET 'https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple' \
--header 'Connection: keep-alive' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'X-Application-Origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/95.0.4638.54 Safari/537.36' \
--header 'Accept: application/json, text/plain, */*' \
--header 'X-Currency: CZK' \
--header 'Cache-Control: no-cache' \
--header 'X-Lang: cs' \
--header 'sec-ch-ua-platform: "macOS"' \
--header 'Origin: https://novy.regiojet.cz' \
--header 'Sec-Fetch-Site: cross-site' \
--header 'Sec-Fetch-Mode: cors' \
--header 'Sec-Fetch-Dest: empty' \
--header 'Referer: https://novy.regiojet.cz/' \
--header 'Accept-Language: en-US,en;q=0.9,cs-CZ;q=0.8,cs;q=0.7'
```

Zkuste request provolat, vrátí se Vám status 400, protože nám v requestu chybí povinné parametry.

V následující tabulce naleznete klíče a hodnoty, které je potřeba vyplnit do parametrů.

| Klíč                    | Hodnota                       |
|-------------------------|-------------------------------|
| <b>tariffs</b>          | REGULAR                       |
| <b>toLocationType</b>   | CITY                          |
| <b>departureDate</b>    | Datum ve formátu "YYYY-MM-DD" |
| <b>toLocationId</b>     | 2147875000                    |
| <b>fromLocationType</b> | CITY                          |
| <b>fromLocationId</b>   | 10202003                      |

Doplňte parametry do importovaného požadavku a provolejte ho. Vrátí se vám odpověď 200 s výpisem jednotlivých spojů.

Skolení / Params / GET **regiojet routes**

GET https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocatio

Params Authorization Headers (23) Body Pre-request Script Tests Settings Cookies </>

Query Params

| KEY  | VALUE      | DESCRIPTION | ... | Bulk Edit |
|--|------------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> tariffs          | REGULAR    |             |     |           |
| <input checked="" type="checkbox"/> toLocationType   | CITY       |             |     |           |
| <input checked="" type="checkbox"/> toLocationId     | 2147875000 |             |     |           |
| <input checked="" type="checkbox"/> fromLocationType | CITY       |             |     |           |
| <input checked="" type="checkbox"/> fromLocationId   | 10202003   |             |     |           |
| <input checked="" type="checkbox"/> departureDate    | 2021-12-23 |             |     |           |
| Key  | Value      | Description |     |           |

Body Cookies Headers (16) Test Results Status: 200 200 Time: -- Size: 6.23 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 "routes": [
2   {
3     "id": "5855269234,5853509534",
4     "departureStationId": 372825000,
5     "departureTime": "2021-12-23T06:18:00.000+01:00",
6     "arrivalStationId": 4961583004,
7     "arrivalTime": "2021-12-23T10:30:00.000+01:00",
8     "vehicleTypes": [
9       "TRAIN"
10      ],
11      "transfersCount": 1,
12      "freeSeatsCount": 222,
13      "priceFrom": 343,
14      "priceTo": 666,
15      "creditPriceFrom": 335,
16      "creditPriceTo": 661,
17      ...
18    ]
19  ]
20 ]
```

## Úkol:

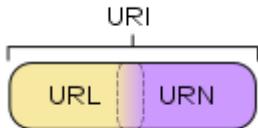
Upravte request následovně:

toLocationID: 10202052

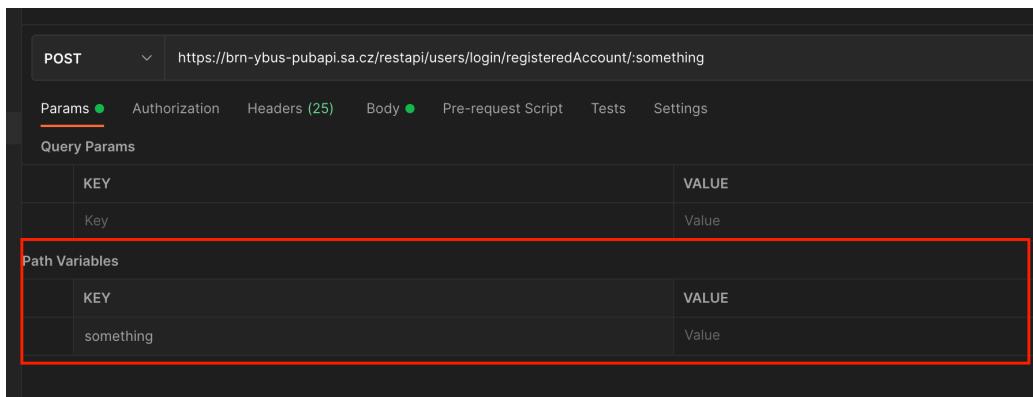
fromLocationID: 17554000

## URI PARAMETRY

Je nejobecnější z rodiny identifikátorů URN, URL, URI a slouží k identifikování jména nebo zdroje na internetu.



V Postmanu můžeme URI využít například u dynamických URL, kde součástí cesty je dynamická hodnota. Do cesty URL zadáme ":Name", následně nám Postman odemkne v záložce params novou část "Path Variables"



The screenshot shows a POST request in Postman. The URL is <https://brn-ybus-pubapi.sa.cz/restapi/users/login/registeredAccount/:something>. The 'Path Variables' section is highlighted with a red box, showing a key 'something' with a value 'Value'.

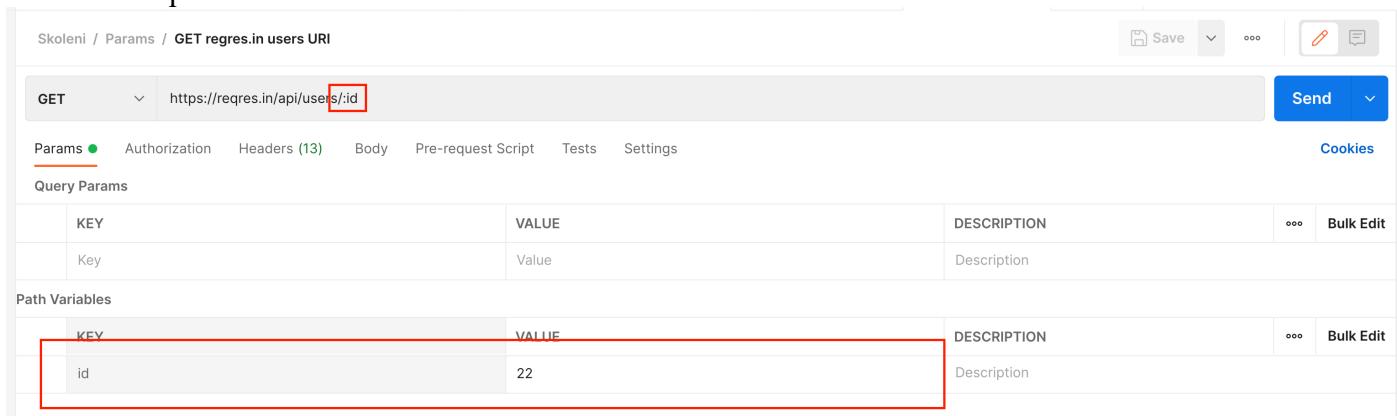
## PŘÍKLAD

Importujte následující cURL do složky "Params", nazvěte ho "GET regres.in users URI".

**cURL:**

```
curl --location --request GET 'https://reqres.in/api/users/2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

Změňte v requestu číslo 2 v URL za :id a následně v Path Variables dosaďte číslo 22.



The screenshot shows a GET request in Postman. The URL is <https://reqres.in/api/users/:id>. The 'Path Variables' section is highlighted with a red box, showing a key 'id' with a value '22'.

Kdy použít path variables? Doporučuji je používat pouze v případě, že je dynamická hodnota součástí URL, pokud se jedná o parametr, pak použijte standardní přístup, který jsme si ukazovali výše. Jaký je rozdíl mezi path variable a param?

Path Variable: <https://reqres.in/api/users/3>

Param: <https://reqres.in/api/users?page=3>

## JSON

Vytvořte novou složku "JSON" v kolekci "Skolení".

Importujte request z cURL níže pro přidání produktu do košíku na rohlik.cz. Pojmenujte ho "POST rohlik.cz cart" a uložte jej do složky "JSON"

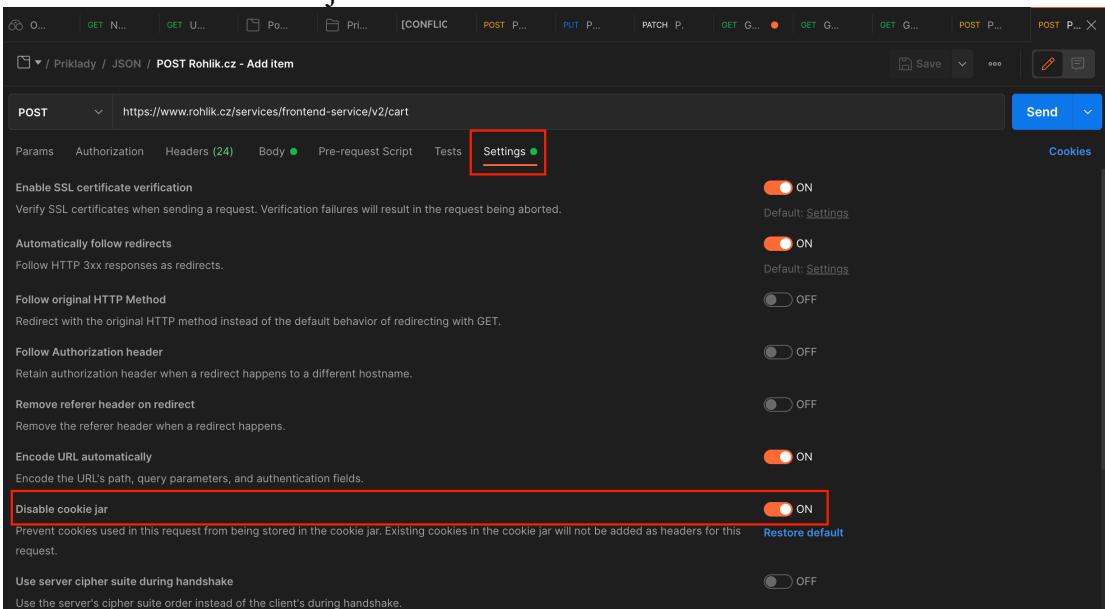
**cURL:**

```
curl --location --request POST 'https://www.rohlik.cz/services/frontend-service/v2/cart' \
--header 'authority: www.rohlik.cz' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'accept: application/json' \
--header 'content-type: application/json' \
--header 'x-origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36'
```

```
--header 'sec-ch-ua-platform: "Windows"' \
--header 'origin: https://www.rohlik.cz' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'referer: https://www.rohlik.cz/c300102000-ovoce-a-zelenina' \
--header 'accept-language: en-US,en;q=0.9' \
--data-raw '{
    "productId": 3344,
    "quantity": 1,
    "source": ":ProductCategory:300102000",
    "actionId": null,
    "recipeId": null
}'
```

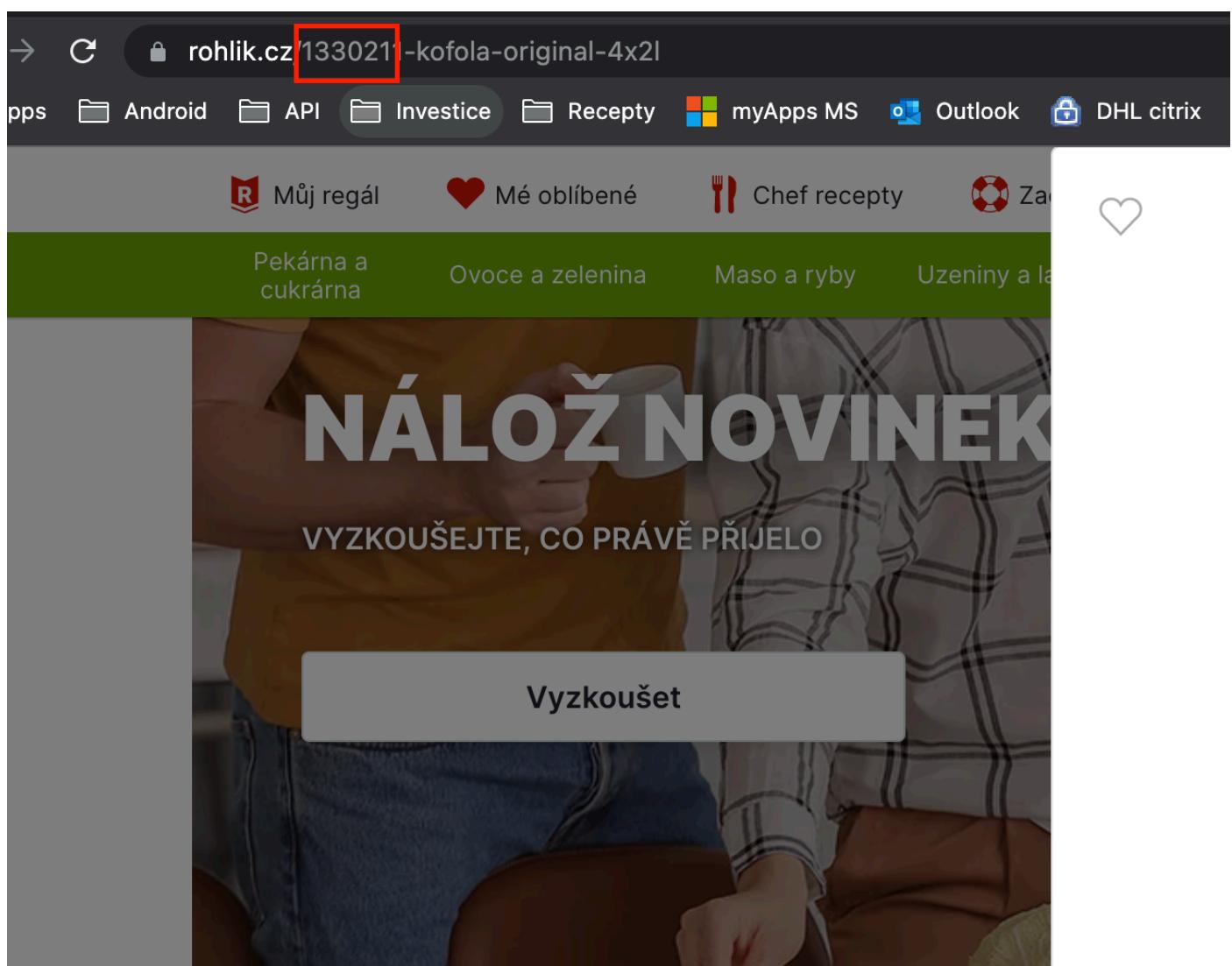
V cURL je chybné ID produktu.

Jelikož u rohlíku zatím nechceme používat cookies, otevřete request, běžte do settings requestu a přepněte hodnotu "Disable cookie jar" na "ON"



The screenshot shows the Postman interface with a request configuration for a POST method to <https://www.rohlik.cz/services/frontend-service/v2/cart>. The 'Settings' tab is selected, and the 'Cookies' section is visible. A red box highlights the 'Disable cookie jar' setting, which is currently set to 'ON'. Other settings like 'Enable SSL certificate verification' and 'Follow original HTTP Method' are also shown.

Otevřete si stránku [rohlik.cz](http://rohlik.cz), vyberte detail jakéhokoliv produktu, z URL vykopírujte ID a vložte ho do JSON fieldu: productId



Request následně provolejte. Při správném nastavení dostanete response 200.

POST https://www.rohlik.cz/services/frontend-service/v2/cart

Params Authorization Headers (24) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▼

Cookies ▼

Beautify

```

1
2   ...
3     "productId": 1330211,
4     "quantity": 1,
5     "source": ":ProductCategory:300102000",
6     "actionId": null,
7     "recipeId": null

```

Body Cookies (1) Headers (25) Test Results

Status: 200 OK Time: 305 ms Size: 2.61 KB Save Response ▼

Pretty Raw Preview Visualize **JSON** ▼

```

1
2   "status": 200,
3   "messages": [],
4   "data": {
5     "totalPrice": 109.90,
6     "totalsavings": 0,
7     "minimalStandardOrderPrice": 500.00,
8     "minimalDeliveryPointOrderPrice": 100,
9     "deliveryPointEnabled": true,
10    "submitConditionPassed": false,
11    "companies": [
12      {
13        "id": 1,
14        "name": "Velká Pecka s.r.o.",
15        "label": "Supermarket Rohlik.cz",
16        "link": "/",
17        "categories": [
18          ...
19        ]
20      }
21    ]
22  }
23}

```

## GRAPHQL

GraphQL je jedním z dotazovacích jazyků pro tvorbu API. GraphQL byl vyvinut firmou Facebook a od roku 2015 je veřejně dostupný.

Více informací o GraphQL a jak jej používat naleznete [zde](#).

Pro GraphQL využijeme SpaceX API. Konkrétně získáme 10 posledních startů.

## PŘÍKLAD

Vytvořte novou složku v kolekci "Skolení", nazvěte ji "GraphQL".

Vytvořte nový HTTP request, nazvěte ho "POST SpaceX launchesPast" a uložte ho do složky "GraphQL"



Create New

X

#### Building Blocks

**HTTP Request**

**GET** Create a basic HTTP request



**WebSocket Request** BETA

Test and debug your WebSocket connections



**Collection**

Save your requests in a collection for reuse and sharing



**Environment**

Save values you frequently use in an environment



**Workspace**

Create a workspace to build independently or in collaboration

#### Advanced



**API Documentation**

Create and publish beautiful documentation for your APIs



**Mock Server**

Create a mock server for your in-development APIs



**Monitor**

Schedule automated tests and check performance of your APIs



**API**

Manage all aspects of API design, development, and testing



**Flows** BETA

Test real-world workflows by connecting series of requests logically.

Not sure where to start? [Explore](#) featured APIs, collections, and workspaces published by the Postman community.

[Learn more on Postman Docs](#)

## Typ Requestu: POST

URL: <https://api.spacex.land/graphql/>

Zvolte: Body/GraphQL

The screenshot shows the Postman interface with a POST request to <https://api.spacex.land/graphql/>. The 'Body' tab is selected, containing the following GraphQL query:

```
1
```

The 'GRAPHQL VARIABLES' section contains a single variable:

|   |
|---|
| 1 |
|---|

Do Query vložte:

```
{
  launchesPast(limit: 10) {
    mission_name
    launch_date_local
    launch_site {
      site_name_long
    }
    links {
      article_link
      video_link
    }
    rocket {
      rocket_name
    }
  }
}
```

Provolejte request. Dostanete zpět JSON s posledními starty raket SpaceX:

Body Cookies Headers (0) Test Results

Pretty Raw Preview Visualize JSON  

```

1 "data": {
2   "launchesPast": [
3     {
4       "mission_name": "Starlink-15 (v1.0)",
5       "launch_date_local": "2020-10-24T11:31:00-04:00",
6       "launch_site": {
7         "site_name_long": "Cape Canaveral Air Force Station Space Launch Complex 40"
8       },
9       "links": {
10         "article_link": null,
11         "video_link": "https://youtu.be/J442-ti-Dhg"
12       },
13       "rocket": {
14         "rocket_name": "Falcon 9"
15       }
16     },
17     {
18       "mission_name": "Sentinel-6 Michael Freilich",
19       "launch_date_local": "2020-11-21T09:17:00-08:00",
20       "launch_site": {
21         "site_name_long": "Vandenberg Air Force Base Space Launch Complex 4E"
22       },
23       "links": {
24         "article_link": "https://spaceflightnow.com/2020/11/21/international-satellite-launches-to-extend-measurments-of-sea-level-rise/",
25         "video_link": "https://youtu.be/aVFPzTDCihQ"
26       },
27       "rocket": {
28         "rocket_name": "Falcon 9"
29       }
30     },
31     {
32       "mission_name": "Crew-1",
33       "launch_date_local": "2020-11-15T19:27:00-05:00",
34       "launch_site": {
35         "site_name_long": "Kennedy Space Center Historic Launch Complex 39A"
36       },
37       "links": {
38         "article_link": "https://spaceflightnow.com/2020/11/16/astronauts-ride-spacex-crew-capsule-in-landmark-launch-for-commercial-spaceflight/",
39         "video_link": "https://youtu.be/bnChQbxLkkI"
40       },
41     }
42   ]
43 }
```

Status: 200 OK Time: 430 ms Size: 3.79 KB Save Response 

## SOAP

SOAP je protokolem pro výměnu zpráv založených na XML přes síť, hlavně pomocí HTTP. Nejznámější program na provolávání SOAP je [SOAPUI](#). Postman však v nejnovějších verzích také SOAP umožňuje provolávat.

Více detailů o provolávání SOAP v Postman naleznete na [learning stránkách Postman](#).

## PŘÍKLAD

Vytvořte novou složku v kolekci "Skolení", nazvěte ji "SOAP".

Založte nový POST HTTP Request, pojmenujeme ho "SOAP oorsprong countries"

**Vložte URL:** <http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso>

Přejděte do Body, vyberte Raw/XML

<http://webservices.orsprong.org/websamples.countryinfo/CountryInfoService.ws>

Save   

POST <http://webservices.orsprong.org/websamples.countryinfo/CountryInfoService.ws>

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **XML**

1 |

## Vložte XML:

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <ListOfCountryNamesByName xmlns="http://www.orsprong.org/websamples.countryinfo">
      </ListOfCountryNamesByName>
    </soap12:Body>
  </soap12:Envelope>
```

Request uložte a provolejte, dostanete odpověď v XML, která bude obsahovat jednotlivé země a jejich kódy.

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize XML

1 <?xml version="1.0" encoding="utf-8"?>
2 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
3 <soap:Body>
4 <m:ListOfCountryNamesByNameResponse xmlns:m="http://www.orsprong.org/websamples.countryinfo">
5 <m:ListOfCountryNamesByNameResult>
6 <m:tCountryCodeAndName>
7 <m:sISOCode>AX</m:sISOCode>
8 <m:sName>Åland Islands</m:sName>
9 </m:tCountryCodeAndName>
10 <m:tCountryCodeAndName>
11 <m:sISOCode>AF</m:sISOCode>
12 <m:sName>Afghanistan</m:sName>
13 </m:tCountryCodeAndName>
14 <m:tCountryCodeAndName>
15 <m:sISOCode>AL</m:sISOCode>
16 <m:sName>Albania</m:sName>
17 </m:tCountryCodeAndName>
18 <m:tCountryCodeAndName>
19 <m:sISOCode>DZ</m:sISOCode>
20 <m:sName>Algeria</m:sName>
21 </m:tCountryCodeAndName>
22 <m:tCountryCodeAndName>
23 <m:sISOCode>AS</m:sISOCode>
24 <m:sName>American Samoa</m:sName>
25 </m:tCountryCodeAndName>
26 <m:tCountryCodeAndName>
27 <m:sISOCode>AD</m:sISOCode>
28 <m:sName>Andorra</m:sName>
29 </m:tCountryCodeAndName>
30 <m:tCountryCodeAndName>
31 <m:sISOCode>AO</m:sISOCode>
32 <m:sName>Angola</m:sName>
33 </m:tCountryCodeAndName>
34 <m:tCountryCodeAndName>
35 <m:sISOCode>AI</m:sISOCode>
36 <m:sName>Anguilla</m:sName>
37 </m:tCountryCodeAndName>
38 <m:tCountryCodeAndName>
39 <m:sISOCode>AQ</m:sISOCode>
40 <m:sName>Antarctica</m:sName>
41 </m:tCountryCodeAndName>
42 <m:tCountryCodeAndName>
43 <m:sISOCode>AG</m:sISOCode>
44 <m:sName>Antigua &amp; Barbuda</m:sName>
45 </m:tCountryCodeAndName>
46 <m:tCountryCodeAndName>
47 <m:sISOCode>AR</m:sISOCode>
48 <m:sName>Argentina</m:sName>
49 </m:tCountryCodeAndName>
50 <m:tCountryCodeAndName>

## ENVIRONMENTS A PROMĚNNÉ

Proměnné v Postman slouží k práci s přepoužitelnými hodnotami napříč requesty. Využijeme je například, když potřebujeme data využít ve více requestech, uložit konstanty (pevně dané hodnoty) nebo také pro dynamické datové hodnoty.

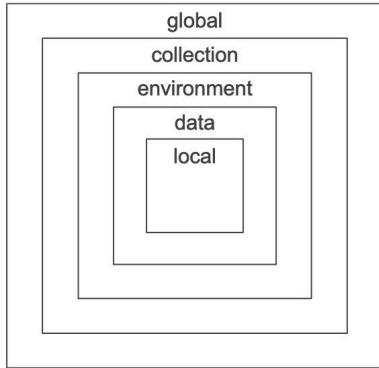
Proměnné v Postman mají několik úrovní (scope):

- Globals
- Environment

- Collection
- Data
- Local

## PRIORITY POUŽITÍ PROMĚNNÝCH

Pokud máme proměnnou uloženou se stejným klíčem ve více úrovní, vyhodnocuje se priorita proměnné, a to dle schéma níže. Například pokud je definovaná proměnná v úrovni *local* a *global*, prioritu má vždy hodnota *local*.



## NASTAVENÍ PROMĚNNÝCH

Proměnná nabývá v Postman 2 hodnot

- Initial value
- Current value

| VARIABLE | INITIAL VALUE | CURRENT VALUE |
|----------|---------------|---------------|
| abcd     | aaa           | aaa           |

### INITIAL VALUE

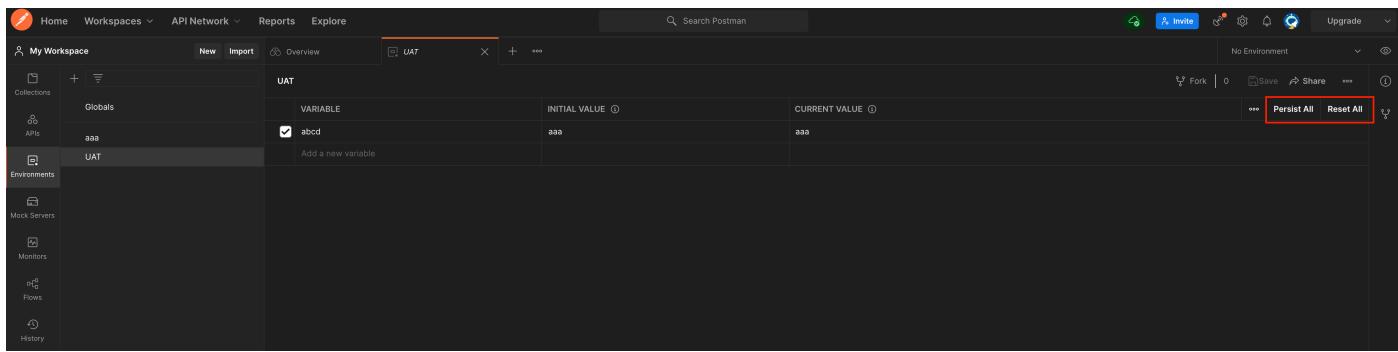
Je hodnota proměnné, která se sdílí v rámci Workspace či při sdílení vyexportovaného souboru

### CURRENT VALUE

Tato hodnota proměnné se nikdy nesdílí. Je vhodná například pro citlivější nebo user údaje. Pokud není vyplněná, tak se automaticky použije Initial value, pokud ani tato hodnota není vyplněna, poté se vrací prázdný text.

### PERSIST ALL/RESET ALL

Volba "Persist All" vezme všechny "Current Values" v prostředí a přepíše "Init Values". "Reset All" udělá opak. Přepíše "Current Values" "Initial Hodnotami".



## GLOBALS

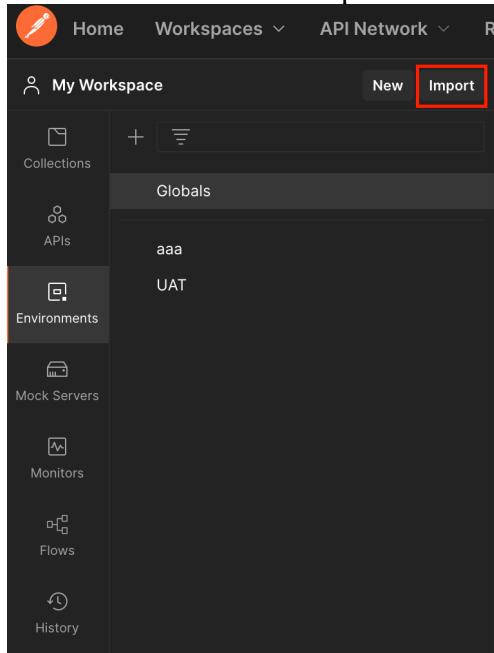
Globální proměnné slouží k přepoužití dat v celé aplikaci postman. Vhodné je to například ve chvíli, kdy máme data, která jsou využívány v celém projektu, nad kolekcemi.

### IMPORT GLOBALS

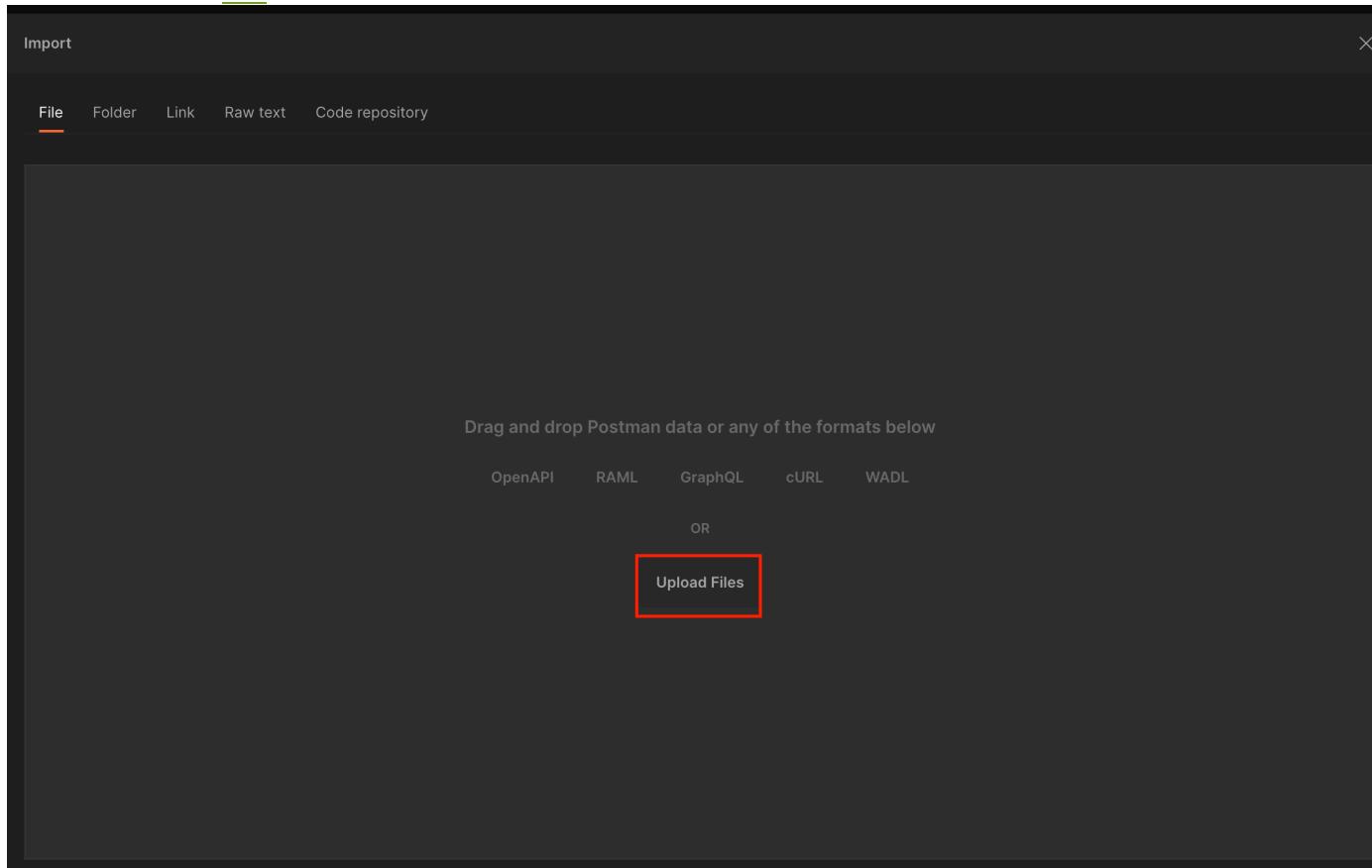
Globalní proměnné se importují pomocí import tlačítka. Musí se jednat o JSON, který je označený jako postman global variables.

**Pozor! Pokud importovaný soubor obsahuje již existující proměnnou v aktuálním globals, přepíše ji novou hotnotou v "Initial Value"**

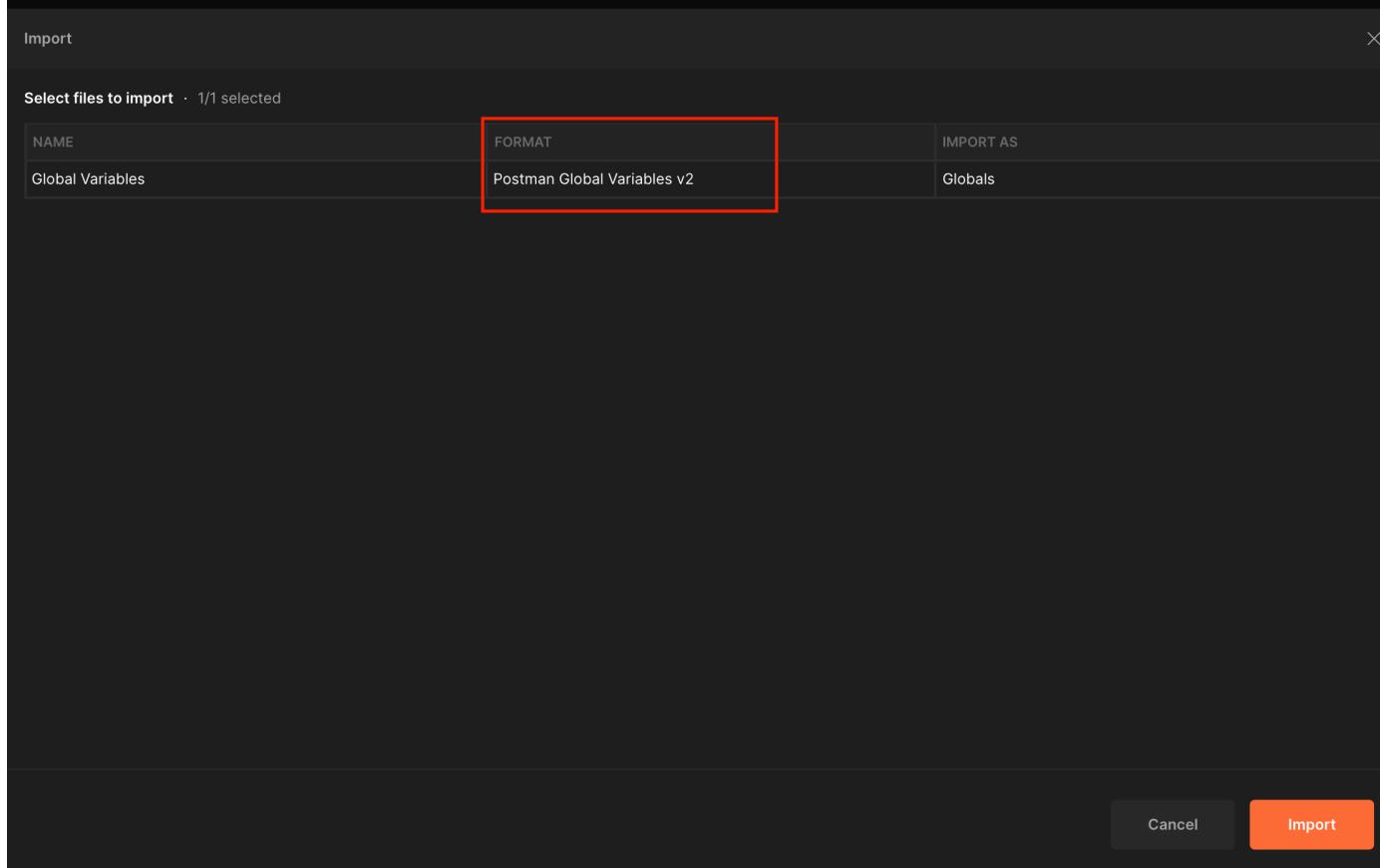
1. Klikněte na tlačítko na Import



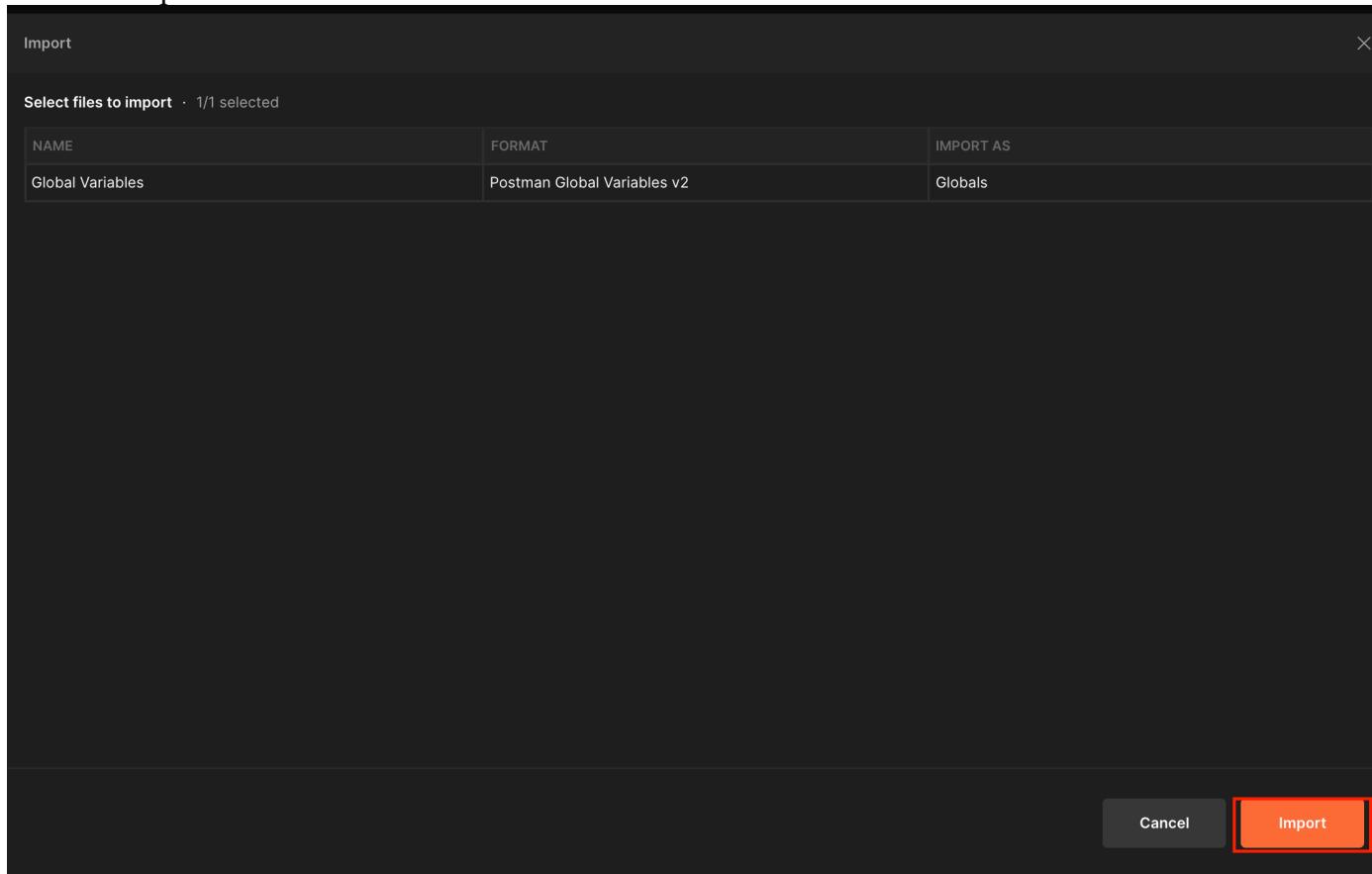
2. Klikněte na "Upload files" a vyberte JSON s globálními proměnnými. Soubor s globals proměnnými můžete stáhnout [zde](#).



3. Zkontrolujte, že se jedná o JSON s globals



#### 4. Potvrďte import



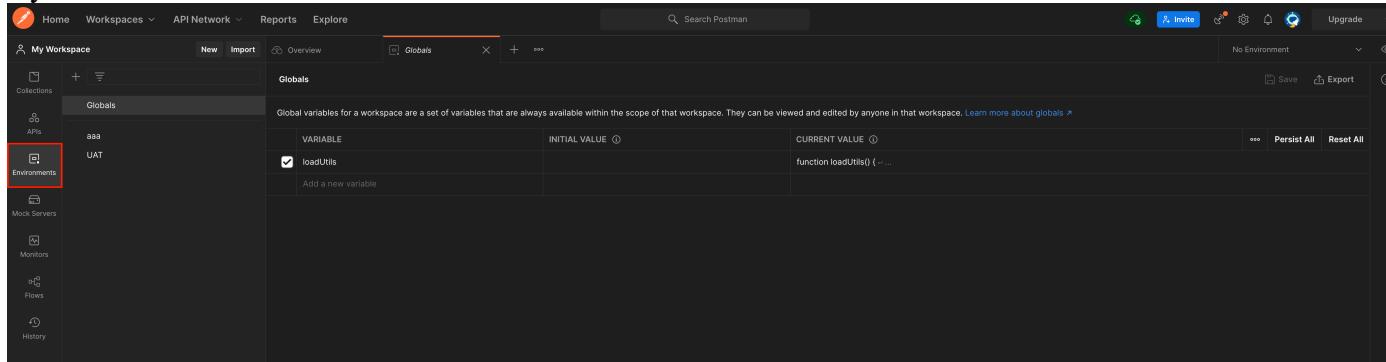
The screenshot shows the Postman 'Import' dialog. At the top, it says 'Select files to import · 1/1 selected'. Below is a table with one row:

| NAME             | FORMAT                      | IMPORT AS |
|------------------|-----------------------------|-----------|
| Global Variables | Postman Global Variables v2 | Globals   |

At the bottom right are two buttons: 'Cancel' and 'Import', with 'Import' being highlighted by a red box.

### EXPORT GLOBALS

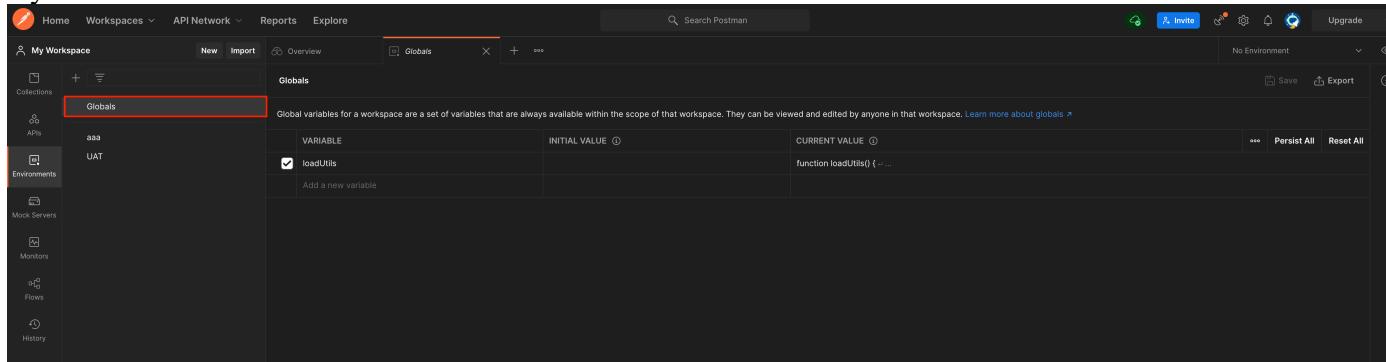
#### 1. Vyberte Environments



The screenshot shows the Postman workspace interface. On the left sidebar, under 'Collections', there is a 'Environments' section which is highlighted with a red box. The main area shows a 'Globals' table with one entry:

| VARIABLE                                      | INITIAL VALUE ⓘ | CURRENT VALUE ⓘ              |
|---|-----------------|------------------------------|
| <input checked="" type="checkbox"/> loadUtils |                 | function loadUtils() { ... } |

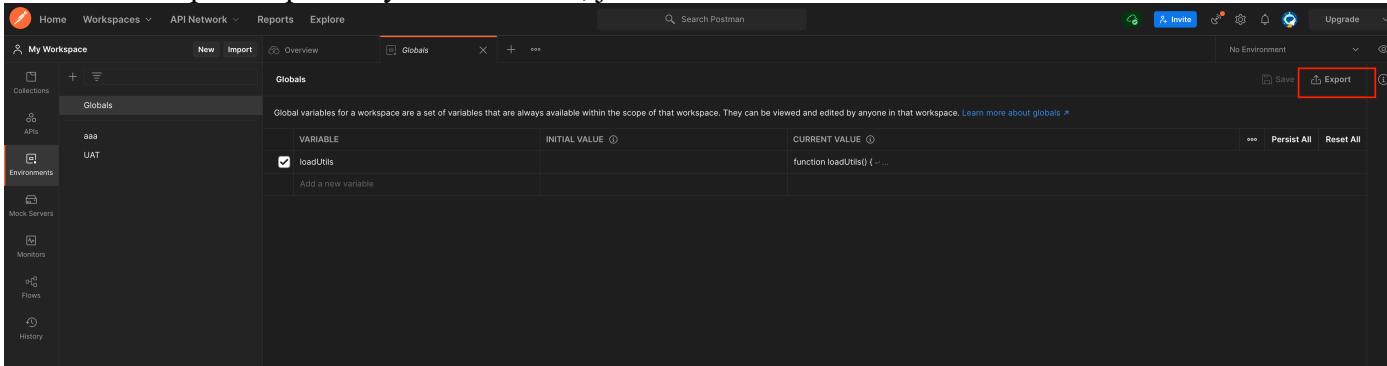
#### 2. Vyberte Globals



The screenshot shows the Postman workspace interface. On the left sidebar, under 'Collections', there is a 'Globals' section which is highlighted with a red box. The main area shows a 'Globals' table with one entry:

| VARIABLE                                      | INITIAL VALUE ⓘ | CURRENT VALUE ⓘ              |
|---|-----------------|------------------------------|
| <input checked="" type="checkbox"/> loadUtils |                 | function loadUtils() { ... } |

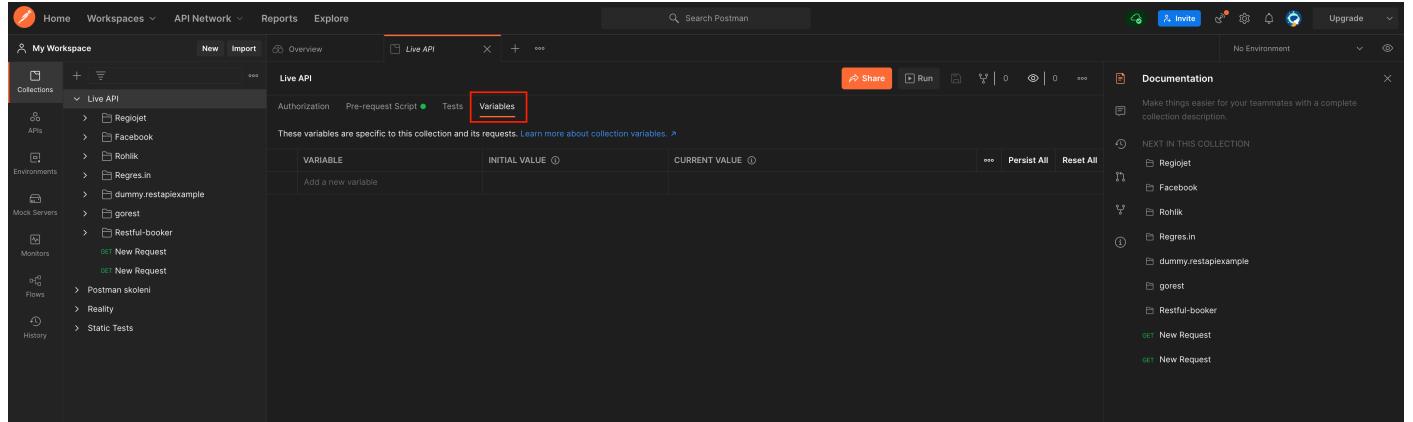
### 3. Klikněte na export Export a vyberte umístění, jméno souboru.



## COLLECTION

Proměnné v kolekci využijeme pro hodnoty, které jsou specifické pro danou skupinu requestů. Může se jednat například o specifické data pro projekt.

Import a export samostatně pro proměnné v kolekci nelze využít. Importuje/exportuje se v rámci celé kolekce.

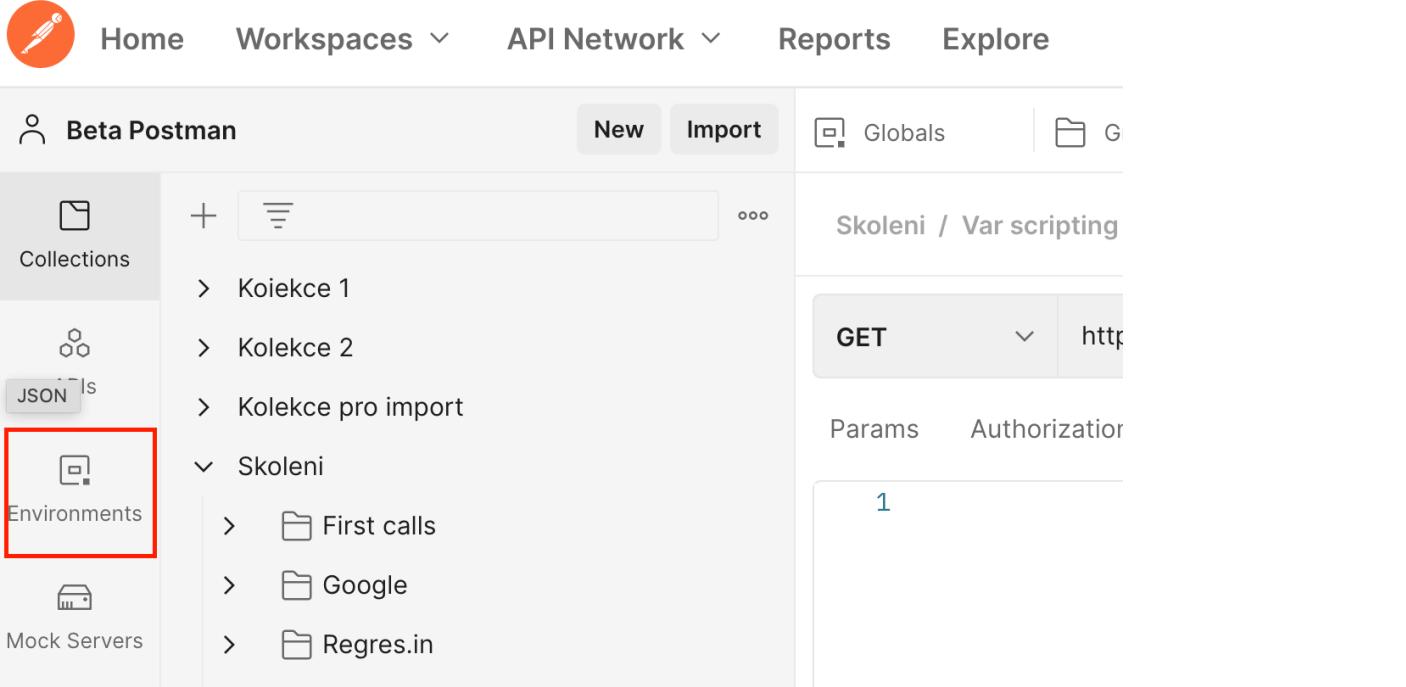


## ENVIRONMENT

Proměnné v prostředích slouží pro definici dat, které využijeme napříč kolekcemi. Jsou vhodné používat, pokud máme více testovacích prostředích.

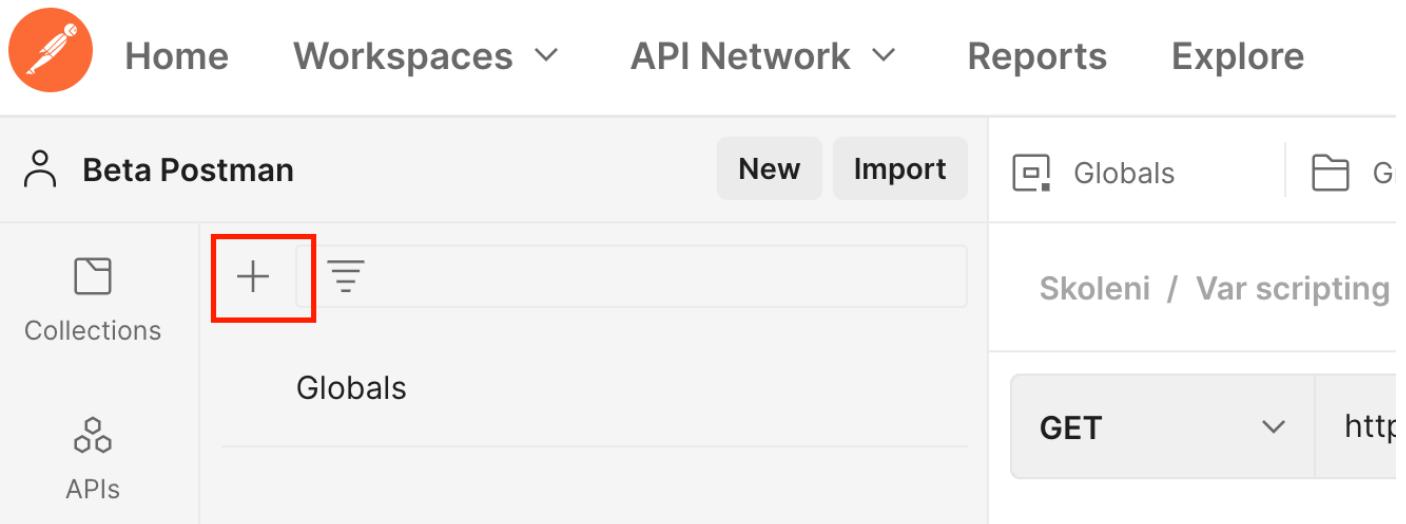
## VYTVOŘENÍ PROSTŘEDÍ

Otevřete záložku "Environments" v levém panelu.

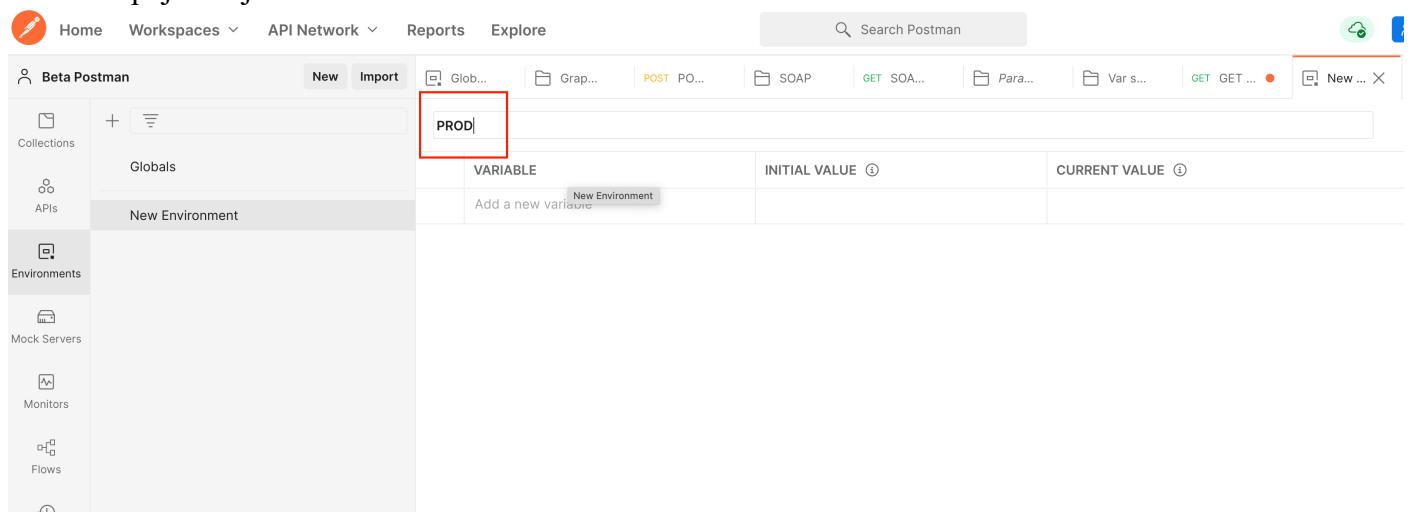


Vytvořte nové prostředí a pojmenujte ho "PROD". Toto prostředí následně budeme využívat v dalších částech Syllabusu.

Klikněte na ikonu "+" vedle vyhledávače.



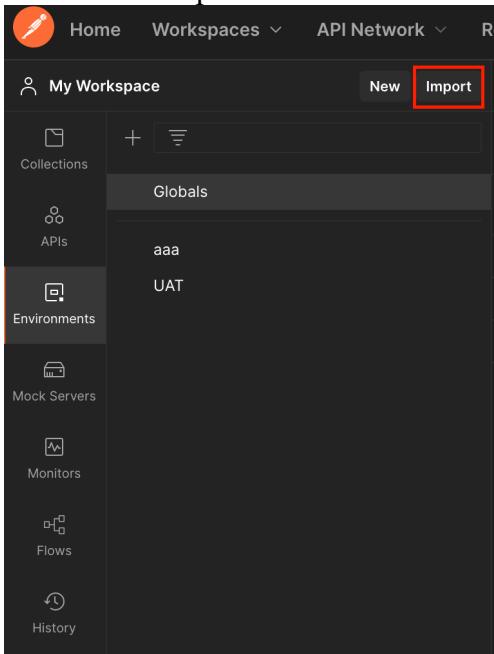
Prostředí pojmenujte PROD a uložte.



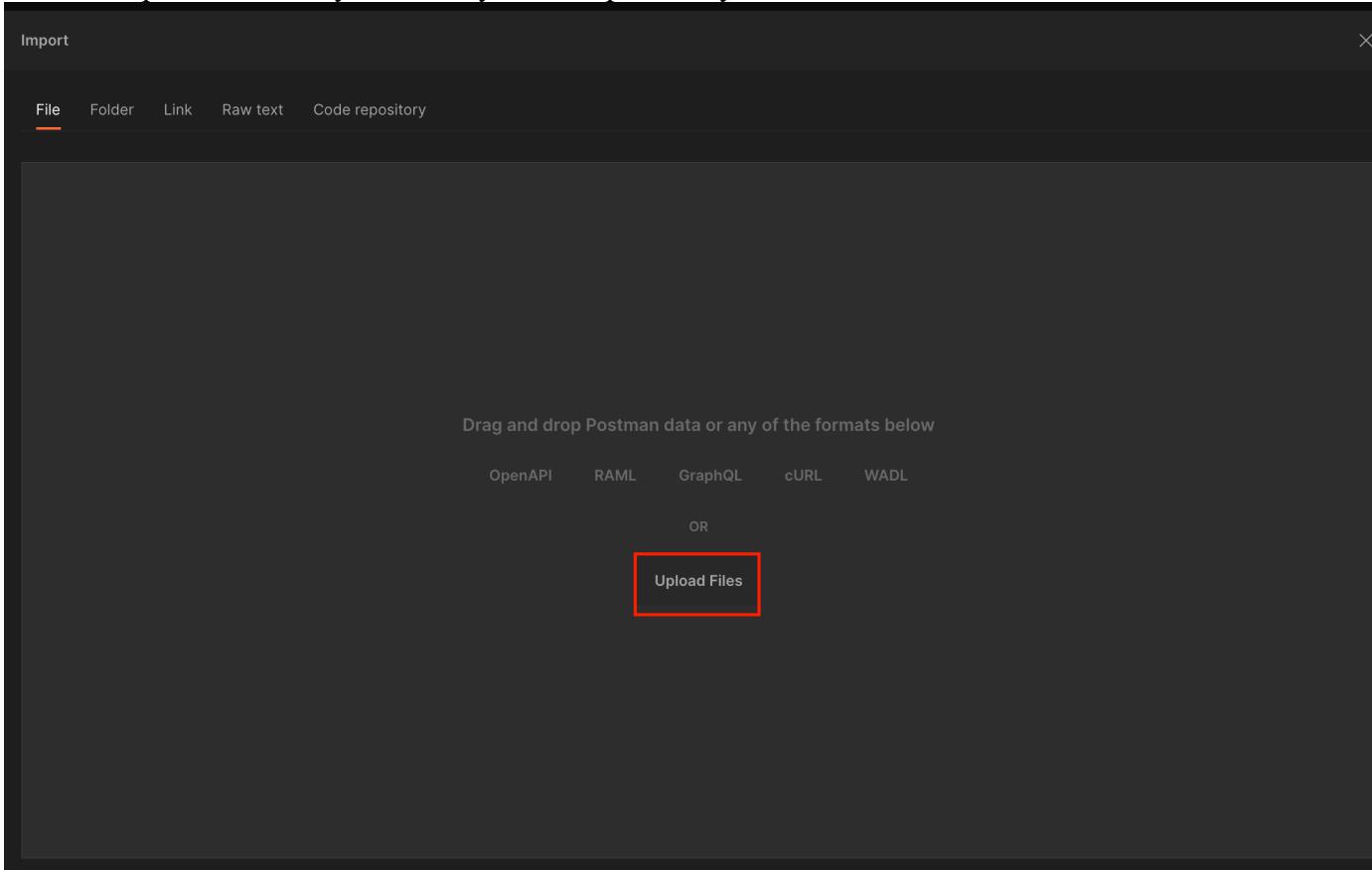
## IMPORT PROMĚNNÝCH PRO PROSTŘEDÍ

Soubor pro vyzkoušení importu naleznete [zde](#).

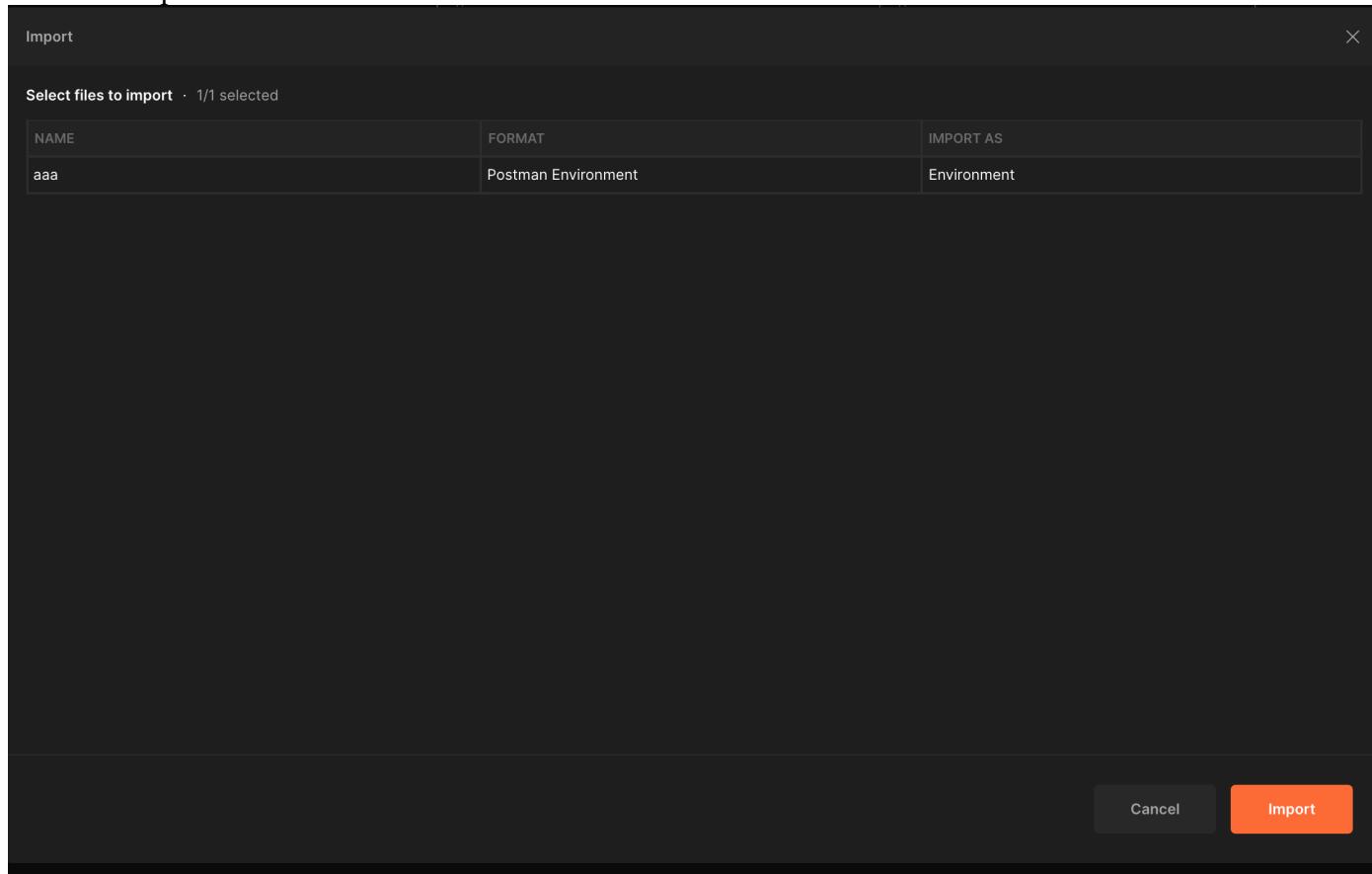
## 1. Klikněte na Import



## 2. Klik na "Upload files" a vyber stažený JSON s proměnnými



### 3. Potvrďte import




---

## EXPORT PROMĚNNÝCH PRO PROSTŘEDÍ

### 1. Vyberte Environments

The screenshot shows the Postman interface with the 'Environments' section selected in the sidebar. The main area displays a table of global variables for the 'aaa' environment. One variable, 'loadUtils', is checked. The table includes columns for VARIABLE, INITIAL VALUE, and CURRENT VALUE.

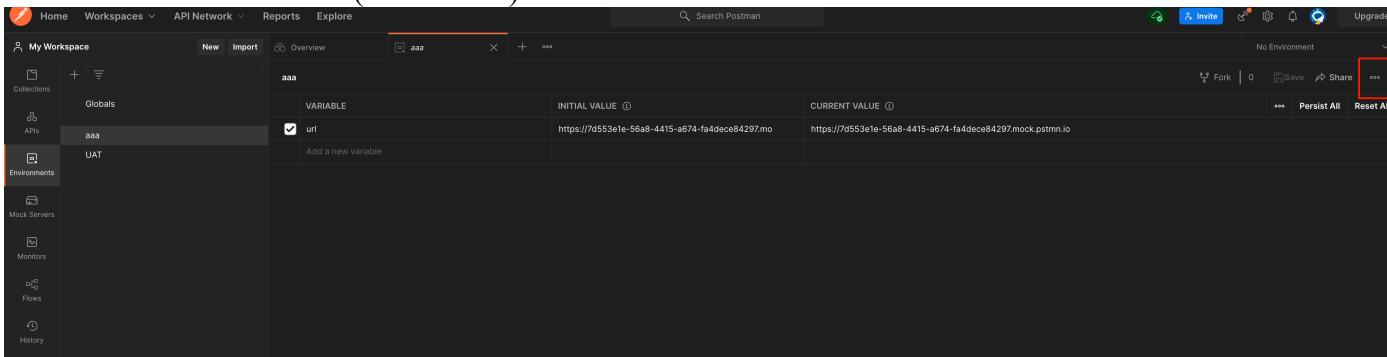
| VARIABLE  | INITIAL VALUE | CURRENT VALUE                |
|-----------|---------------|------------------------------|
| loadUtils |               | function loadUtils() { ... } |

### 2. Vyberte konkrétní prostředí

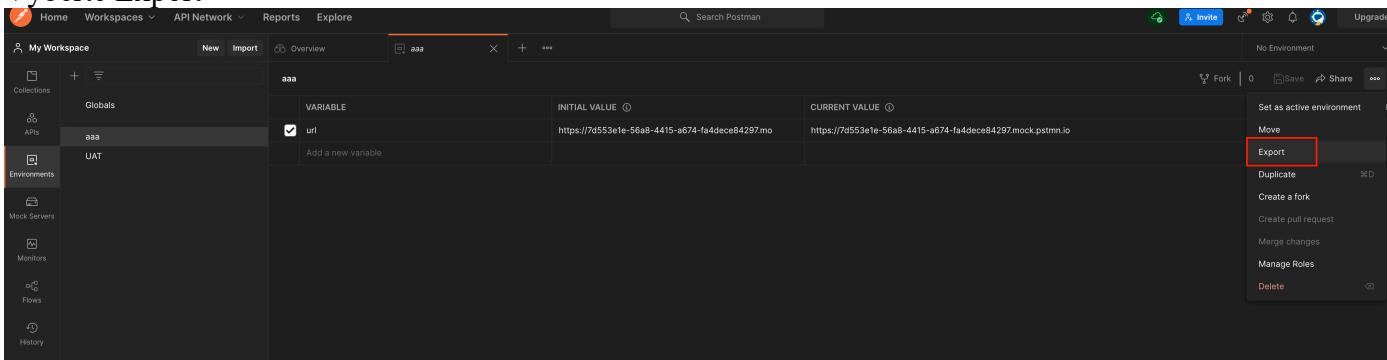
The screenshot shows the Postman interface with the 'aaa' environment selected in the sidebar. The main area displays a table of variables for the 'aaa' environment. One variable, 'url', is checked. The table includes columns for VARIABLE, INITIAL VALUE, and CURRENT VALUE.

| VARIABLE | INITIAL VALUE  | CURRENT VALUE  |
|----------|--|--|
| url      | https://7d553e1e-56a8-4415-a674-fa4dece84297.mock.pstmn.io | https://7d553e1e-56a8-4415-a674-fa4dece84297.mock.pstmn.io |

### 3. Klikněte na ikonu tří teček (more menu)



### 4. Vyberte Export



## DATA

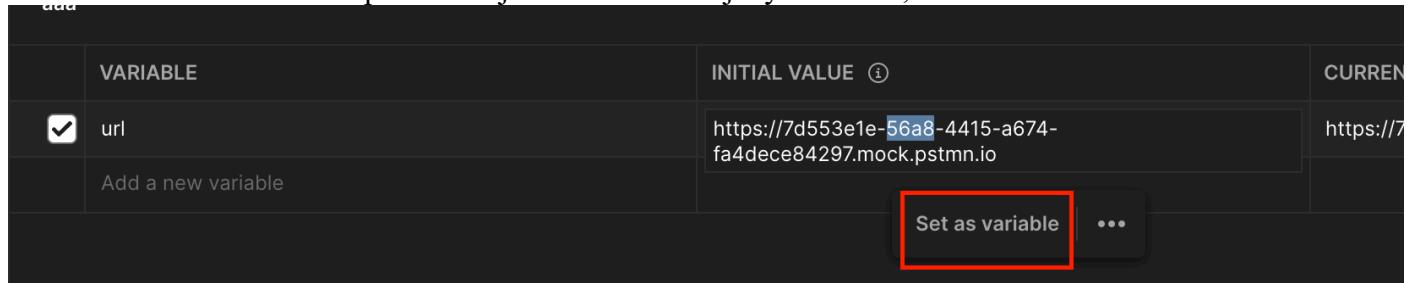
Data typ proměnných se nastavují pro Collection runner. Který probereme v [dané kapitole](#). Tyto proměnné jsou nastavené pouze v rámci běhu Runneru, po dokončení se mažou z paměti.

## LOCAL

Proměnné v této úrovni se vytváří pouze v rámci skriptů. Blíže se tomu venujeme v [kapitole o skriptování](#). Tyto proměnné jsou nastavené pouze v rámci běhu Collection Runneru nebo při posílání requestu, po dokončení se mažou z paměti.

## SET AS VARIABLE

Mimo standardní nastavení proměnné je možné označit jakýkoliv text, zobrazí se možnost *Set as variable*.



|  | VARIABLE | INITIAL VALUE ⓘ  | CURRENT VALUE ⓘ  |
|--|----------|--|--|
| <input checked="" type="checkbox"/>  | url      | https://7d553e1e-56a8-4415-a674-fa4dece84297.mock.pstmn.io | https://7d553e1e-56a8-4415-a674-fa4dece84297.mock.pstmn.io |
| <a href="#">Add a new variable</a> <div style="text-align: right;"> <span style="border: 1px solid red; padding: 2px;">Set as variable</span> ...         </div> |          |  |  |

## POUŽITÍ PROMĚNNÝCH A PROSTŘEDÍ

Proměnné můžeme použít kdekoli v Postman. Například v parametrech, URL, body, kolekcích...

### Použití:

```
{{klic_promenne}}
```

### Příklad:

```
"klic": "{{promenna_text}}",
"klic2": {{promenna_cislo}}
```

Pro použití ve skriptech je postup trochu odlišný. Detail rozebereme v kapitole věnující se [skriptování a testování](#).

## TESTOVÁNÍ V POSTMANU

### ÚVOD DO TESTOVÁNÍ V POSTMANU

Testování a test automatizace v Postman představuje jednu z jeho největších výhod. Test skripty dokážou urychlit testování zvláště u opakovaných průchodů.

Dokumentaci Postman o skriptování naleznete na stránkách [Postman learning](#).

### JAVASCRIPT

Testy a skripty se v Postman píšou v Javascript. Pokročilé skriptování v Javascriptu vyžaduje znalosti programování a [OOP](#), proto veškeré zde uvedené příklady nebudu do detailu vysvětlovat. Jen vysvětlím jak je přepoužít pro Vaše účely.

### SNIPPETS

Snippety jsou malé části kódu uložené v Postman, které zjednodušují psaní testů a skriptů.

### EXTERNÍ KNIHOVNY

Postman podporuje využití externích knihoven, jejich seznam nalezneš [zde](#).

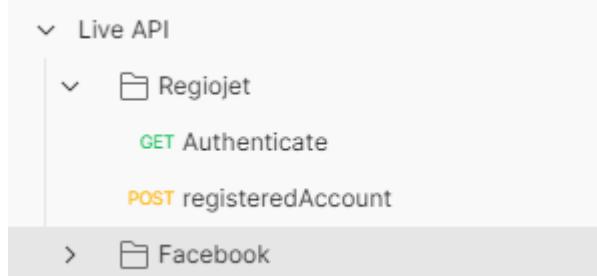
### TYPY TESTŮ

Testy a skripty mohou probíhat v

- Kolekci
- Složce
- Requestu

Testy napsané v Kolekci a složce se provedou vždy u každého děděného requestu, například:

Struktura kolekce:



Při spuštění requestu *Authenticate* se spustí testy napsané v:

- Kolekci *Live API*
- Složce *Regiojet*
- Requestu *Authenticate*

Neprovede se však test napsaný ve složce *Facebook*.

### NASTAVENÍ PROMĚNNÝCH VE SKRIPTECH

Základní logika volání proměnných v testu.

| Proměnná                | Get   | Set   |
|-------------------------|---|---|
| <b>Globals</b>          | pm.globals.get("variable_key");             | pm.globals.set("variable_key", "variable_value");             |
| <b>Collection</b>       | pm.collectionVariables.get("variable_key"); | pm.collectionVariables.set("variable_key", "variable_value"); |
| <b>Environment</b>      | pm.environment.get("variable_key");         | pm.environment.set("variable_key", "variable_value");         |
| <b>Data</b>             | pm.iterationData.get("variable_key");       | -   |
| <b>Local</b>            | pm.variables.get("variable_key");           | pm.variables.set("variable_key", "variable_value");           |
| <b>Jakákoli v rstva</b> | pm.variables.get("variable_key");           | -   |

Pro smazání proměnné můžeme použít funkci *.unset()*, stejně jako *.get()*

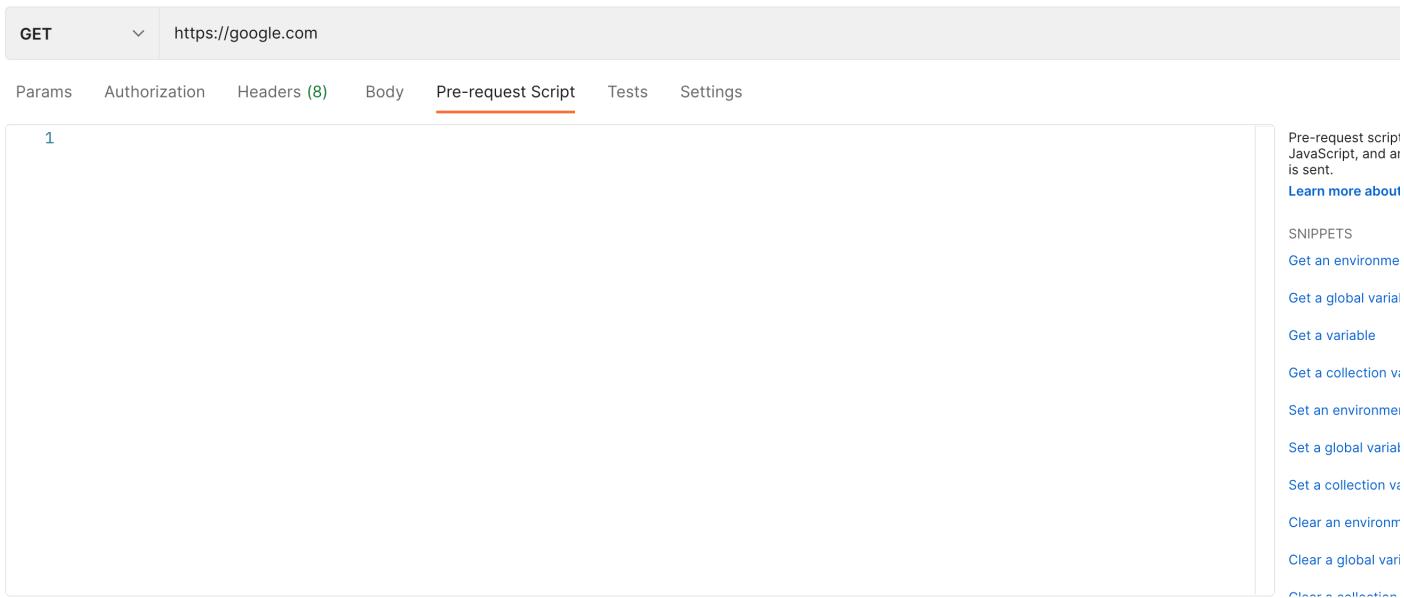
## PŘÍKLAD

V následujícím příkladu zkusíme nastavit a provolat proměnné

Vytvořte novou složku v kolekci "Skolení" a nazvete ji "Var scripting"

V ní vytvořte nový GET HTTP Request, URL: <https://google.com>, nazvěte ho "GET google".

Otevřete záložku "Pre-request scripts".



The screenshot shows the Postman interface with a GET request to <https://google.com>. The 'Pre-request Script' tab is selected. On the right, a sidebar provides links to snippets for setting variables, such as 'Get a global variable', 'Set a collection variable', and 'Clear a collection variable'.

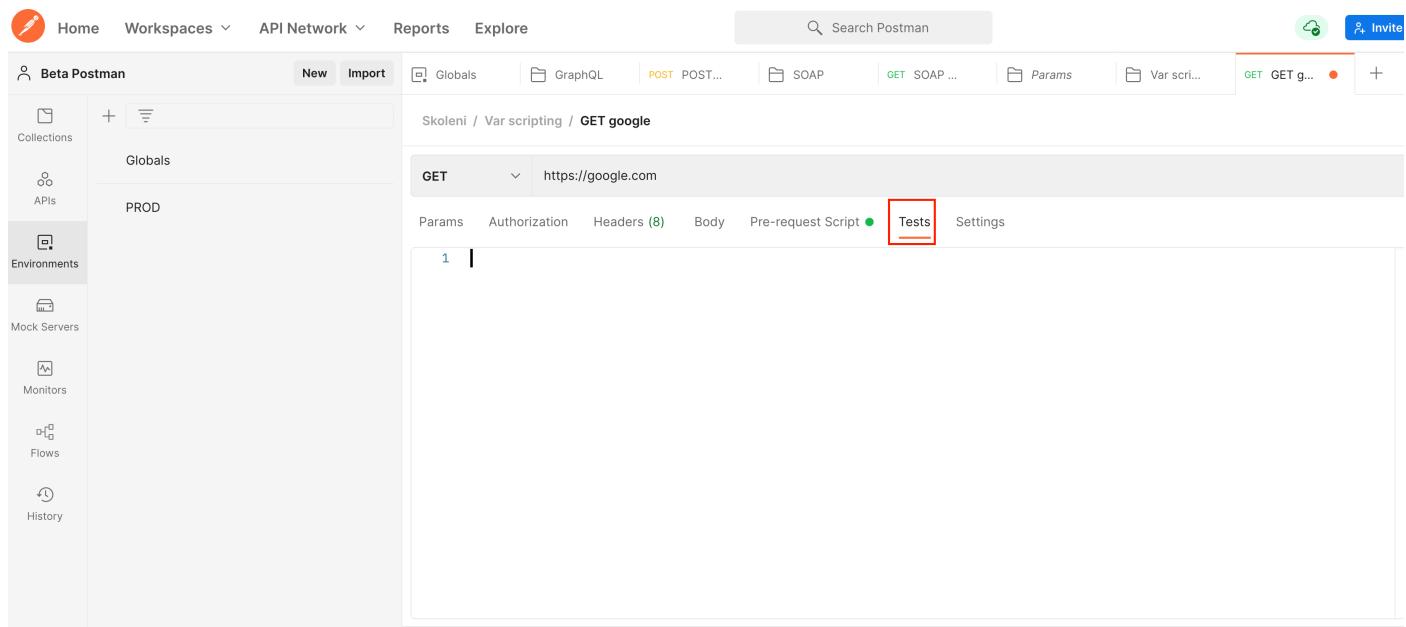
Vytvořte následující proměnné pomocí příkazů:

| Typ Proměnné | Název           | Hodnota            |
|--------------|-----------------|--------------------|
| Globals      | globalsVar1     | globální proměnná  |
| Collection   | collectionVar1  | proměnná kolekce   |
| Environment  | environmentVar1 | proměnná prostředí |
| Local        | localVar1       | lokální proměnná   |

Následující kód vložíme do Pre-request skriptu. Co přesně dělá "Pre-request skript" se dozvídáme v následující kapitole.

```
pm.globals.set("globalsVar1", "globální proměnná")
pm.collectionVariables.set("collectionVar1", "proměnná kolekce")
pm.environment.set("environmentVar1", "proměnná prostředí")
pm.variables.set("localVar1", "lokální proměnná")
```

Otevřeme záložku Tests:



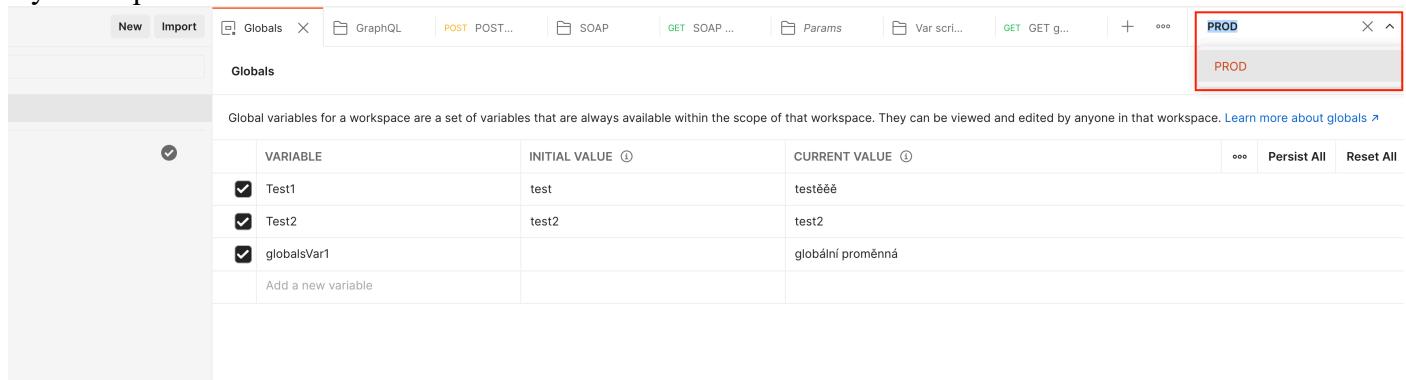
The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, APIs, Environments (selected), Mock Servers, Monitors, Flows, History.
- Top Bar:** Home, Workspaces, API Network, Reports, Explore, Search Postman, Invite.
- Collection:** Skoleni / Var scripting / GET google
- Request:** GET https://google.com
- Test Tab:** The 'Tests' tab is highlighted with a red box.

Do pole s textem vložíme následující kód:

```
console.log(pm.globals.get("globalsVar1"))
console.log(pm.collectionVariables.get("collectionVar1"))
console.log(pm.environment.get("environmentVar1"))
console.log(pm.variables.get("localVar1"))
```

Vyberete prostředí PROD.



| VARIABLE           | INITIAL VALUE | CURRENT VALUE     |
|--------------------|---------------|-------------------|
| Test1              | test          | testěěě           |
| Test2              | test2         | test2             |
| globalsVar1        |               | globální proměnná |
| Add a new variable |               |                   |

Request provolejte, otevřete konzoli (CTRL+ALT+C). Ve výpisu měli vidět jednotlivé proměnné.

Postman Console

Search messages All Logs Clear

- ▶ GET https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&t oLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocationId= 10202003&departureDate=2021-12-23 200 | 2.94 s
- ▶ POST https://www.rohlik.cz/services/frontend-service/v2/cart 200 | 305 ms
- ▶ GET http://webservices.orsprong.org/websamples.countryinfo/CountryInfoService.w 200 | 197 ms
- ▶ GET https://google.com/ 301 ↗
- ▶ GET https://www.google.com/ 302 ↗
- ▶ GET https://consent.google.com/ml?continue=https://www.google.com/&gl=CZ&m=0&pc= 200 | 178 ms
- ▶ GET https://google.com/ 301 ↗
- ▶ GET https://www.google.com/ 302 ↗
- ▶ GET https://consent.google.com/ml?continue=https://www.google.com/&gl=CZ&m=0&pc= 200 | 155 ms

"globální proměnná"

"proměnná kolekce"

"proměnná prostředí"

"lokální proměnná"

Show timestamps  Hide network

Otevřete Globals proměnné (Environment/Globals), zkontrolujte zapsanou proměnnou.

Beta Postman

New Import Globals GraphQL POST POST... SOAP GET SOAP ... Params Var scri... GET GET g... + ... No Environment

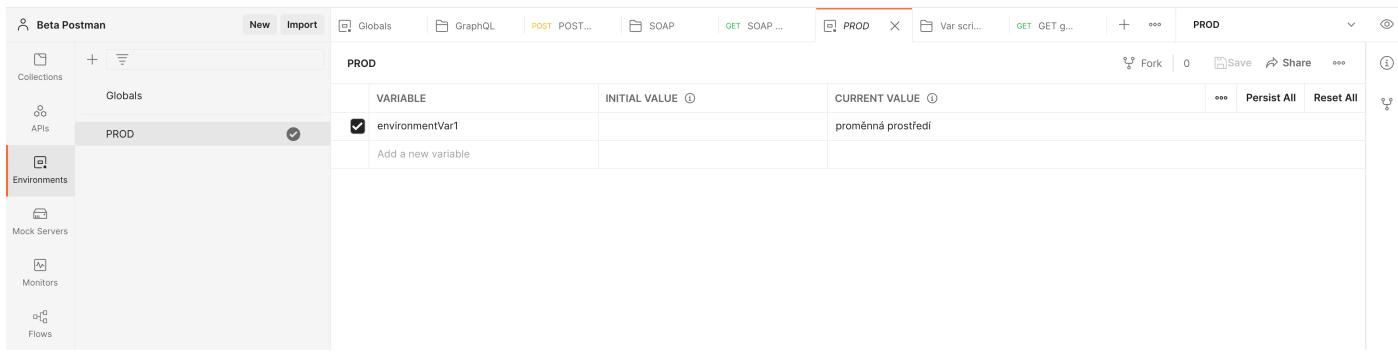
Collections APIs Environments Mock Servers Monitors Flows History

Globals PROD

|                                     | VARIABLE    | INITIAL VALUE ⓘ | CURRENT VALUE ⓘ   | ... | Persist All | Reset All |
|-------------------------------------|-------------|-----------------|-------------------|-----|-------------|-----------|
| <input checked="" type="checkbox"/> | Test1       | test            | testééé           | ... |             |           |
| <input checked="" type="checkbox"/> | Test2       | test2           | test2             | ... |             |           |
| <input checked="" type="checkbox"/> | globalsVar1 |                 | globální proměnná | ... |             |           |
| Add a new variable                  |             |                 |                   |     |             |           |

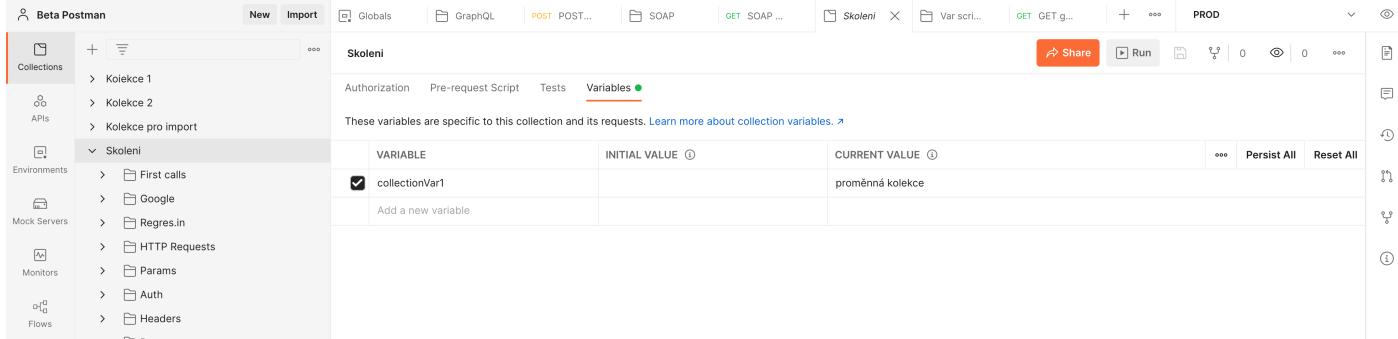
Save Export

Otevřete prostředí PROD v enviroments a zkontrolujte vytvořenou proměnnou



The screenshot shows the Postman interface with a collection named 'PROD'. In the 'Variables' section, there is a variable named 'environmentVar1' with the initial value 'promenná prostředí'.

Otevřete kolekci Skoleni, ježi proměnné a zkонтrolujte vytvořenou proměnnou.



The screenshot shows the Postman interface with a collection named 'Skoleni'. In the 'Variables' section, there is a variable named 'collectionVar1' with the initial value 'proměnná kolekce'.

## PRE-REQUEST SCRIPT

Se vyhodnucuje vždy před posláním requestu. Vhodný je například na nastavování proměnných nebo timestamp do requestu.

## PŘÍKLAD

Zavoláme regiojet.cz API, kde v pre-request skriptu nastavíme departure date jako lokální proměnnou (nikam se neuloží, po doběhnutí requestu se smaže).

Vytvořte novou složku "Pre-request scripts" v kolekci "Skoleni". Request importujte do nově uložené složky a nazvěte ho "GET regiojet routes – variable"

## CURL:

```
curl --location -g --request GET 'https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocationId=10202003&departureDate={{depDate}}'
--header 'Connection: keep-alive'
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"'
--header 'X-Application-Origin: WEB'
--header 'sec-ch-ua-mobile: ?0'
--header 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36'
--header 'Accept: application/json, text/plain, */*'
--header 'X-Currency: CZK'
--header 'Cache-Control: no-cache'
--header 'X-Lang: cs'
--header 'sec-ch-ua-platform: "macOS"'
--header 'Origin: https://novy.regiojet.cz'
--header 'Sec-Fetch-Site: cross-site'
--header 'Sec-Fetch-Mode: cors'
--header 'Sec-Fetch-Dest: empty'
--header 'Referer: https://novy.regiojet.cz/'
--header 'Accept-Language: en-US,en;q=0.9,cs-CZ;q=0.8,cs;q=0.7'
```

Všimněte si, že v novém requestu již máme nastavenou proměnnou v parametru departureDate.

Skolení / Pre-request scripts / GET regiojet routes - variables

GET https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocat... Save + PROD

Params Authorization Headers (23) Body Pre-request Script Tests Settings Cookies

Query Params

| KEY  | VALUE                    | DESCRIPTION | ... | Bulk Edit |
|--|--------------------------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> tariffs          | REGULAR                  |             | ... |           |
| <input checked="" type="checkbox"/> toLocationType   | CITY                     |             | ... |           |
| <input checked="" type="checkbox"/> toLocationId     | 2147875000               |             | ... |           |
| <input checked="" type="checkbox"/> fromLocationType | CITY                     |             | ... |           |
| <input checked="" type="checkbox"/> fromLocationId   | 10202003                 |             | ... |           |
| <input checked="" type="checkbox"/> departureDate    | <code>{{depDate}}</code> |             | ... |           |
| Key  | Value                    | Description | ... |           |

Response

Do Pre-Request skriptu vložte následující kód.

```
pm.variables.set("depDate", "2021-11-30");
```

Následně request provolejte.

### Kontrola v konzoli:

```
Find and Replace Console
GET https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocationId=10202003&departureDate=2021-11-30
+ Network
+ Request Headers
Connection: "keep-alive"
sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"
X-Application-Origin: "WEB"
sec-ch-ua-mobile: "?0"
User-Agent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36"
Accept: "application/json, text/plain, */*"
X-Currency: "CZK"
Cache-Control: "no-cache"
X-Lang: "cs"
sec-ch-ua-platform: "macOS"
```

### TESTS

Testy se provádí vždy po doručení response v requestu. Mohou však sloužit i ke skriptování. Běžně se používá pro ukládání dat pro další testy.

### PŘÍKLAD

Vytvořte novou složku v kolekci "Skolení", nazvěte ji "Tests".

Importujte následující cURL do nově vytvořené složky, pojmenujte ho "GET regres.in users tests" cURL:

```
curl --location --request GET 'https://reqres.in/api/users?page=2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

Následující kód vložte do Tests:

```
console.log("Request status: " + pm.response.status);
```

## Kontrola v Konzoli:

```
> GET https://reqres.in/api/users?page=2
"Request status: OK"
```

## PSANÍ TESTŮ V POSTMANOVÍ

Testy v Postman se píšou v programovacím jazyku Javascript. Dokumentaci naleznete [zde](#). Můžete také využít tzv. snippetů, které pomohou testy vytvořit i těm, kteří nejsou seznámení s Javascriptem

## SNIPPETY

Snippety jsou malé kousky kódu uložené v Postman, které můžete využít pro zefektivnění psaní testů.

## VYSVĚTLENÍ A POUŽITÍ

Ve školení si ukážeme použití několika typů snippetů

| Snippet  | Popis   |
|--|---|
| <b>Get/Set/clear variable (global, collection, variable, environment)</b>  | Získání (Get), nastavení (Set) a vyčištění (Clear) proměnné v testech                 |
| <b>Send request</b>  | Poslání jiného requestu. Kvůli nízké přepoužitelnosti toto ale nedoporučují používat. |
| <b>Status code: code is 200</b>  | Kontrola HTTP statusu response  |
| <b>Response body:</b> <ul style="list-style-type: none"> <li><b>Contains string</b></li> <li><b>JSON value check</b></li> <li><b>Is equal to a string</b></li> <li><b>Content-type header check</b></li> <li><b>Convert XML body to a JSON object</b></li> </ul> | Testy response body, které se běžně používají   |
| <b>Response time</b>   | Kontrola času odpovědi  |
| <b>Status calls:</b> <ul style="list-style-type: none"> <li><b>Successful POST request</b></li> <li><b>Code name has string</b></li> </ul>   | Kontroly statusů  |
| <b>Use Tiny Validator for JSON data</b>  | Kontrola dle schématu   |

## CVIČENÍ

Vyzkoušíme některé snippety a zkusíme si provolat request.

Tests po použití budou vypadat nějak takto:

```
console.log("Request status: " + pm.response.status);

pm.environment.set("environmentVar2", "variable_value");
pm.globals.set("environmentVar2", "variable_value");
pm.collectionVariables.set("environmentVar2", "variable_value");

//postman si vytáhne proměnné, ale nic s nimi neudělá, proto byly následně zabaleny do console.log()
console.log(pm.environment.get("environmentVar2"));
console.log(pm.globals.get("globalsVar2"));
console.log(pm.variables.get("globalsVar2"));
console.log(pm.collectionVariables.get("collectionVar2"));

pm.environment.unset("environmentVar2");
pm.globals.unset("globalsVar2");
pm.collectionVariables.unset("collectionVar2");
```

```
pm.sendRequest("https://postman-echo.com/get", function (err, response) {
    console.log(response.json());
});

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("string_you_want_to_search");
});

pm.test("Your test name", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.page).to.eql(2);
});

pm.test("Body is correct", function () {
    pm.response.to.have.body("response_body_string");
});

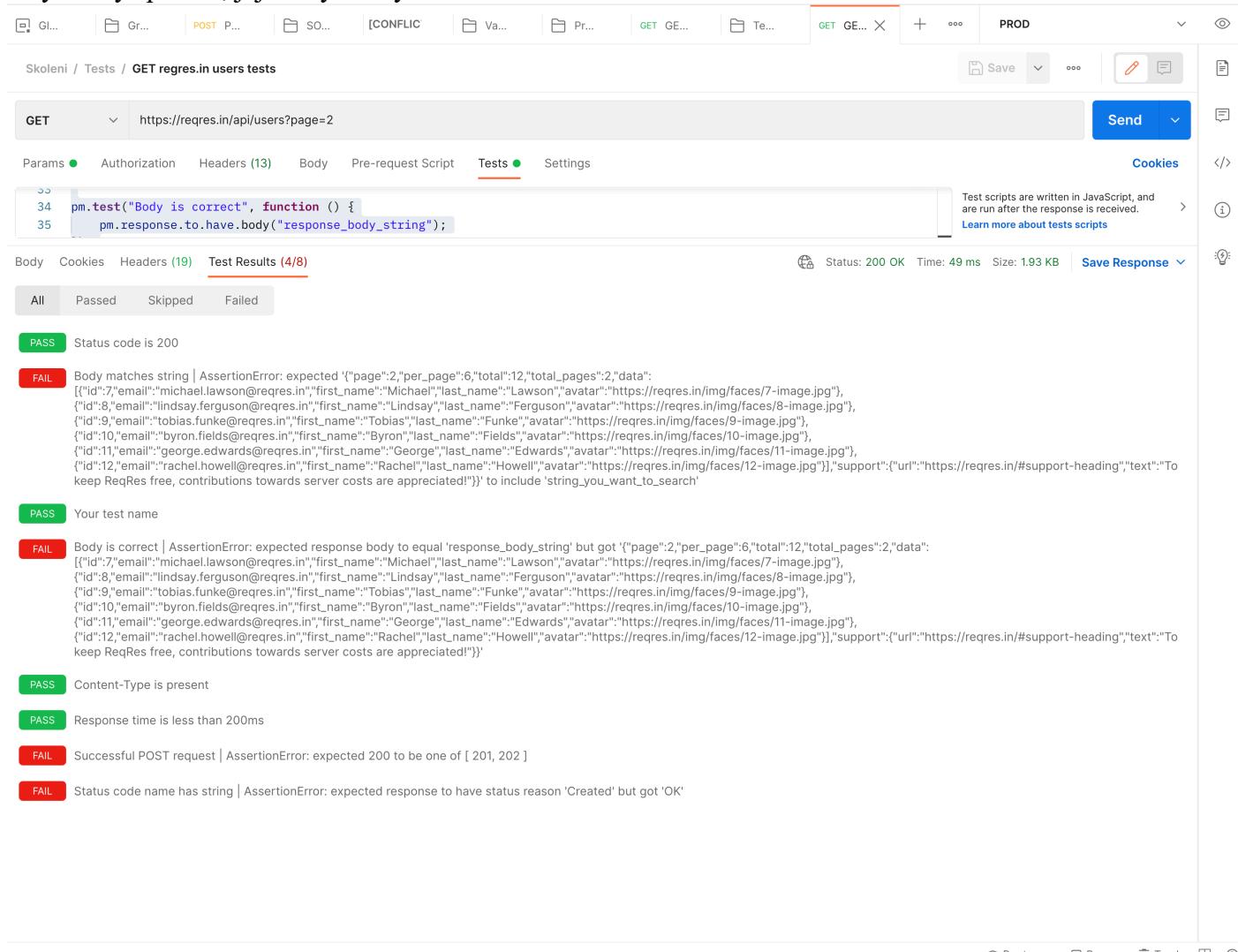
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type");
});

pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Successful POST request", function () {
    pm.expect(pm.response.code).to.be.oneOf([201, 202]);
});

pm.test("Status code name has string", function () {
    pm.response.to.have.status("Created");
});
```

Když testy spustíte, jejich výsledky uvidíte v záložce "Test Results"



The screenshot shows the Postman interface with a successful GET request to <https://reqres.in/api/users?page=2>. The test script in the 'Tests' tab contains:

```

33
34 pm.test("Body is correct", function () {
35   pm.response.to.have.body("response_body_string");

```

The 'Test Results' section shows 4/8 tests passed, with one failing assertion:

- PASS** Status code is 200
- FAIL** Body matches string | Assertion: expected '({"page":2,"per\_page":6,"total":12,"total\_pages":2,"data": [{"id":7,"email":"michael.lawson@reqres.in","first\_name":"Michael","last\_name":"Lawson","avatar":"https://reqres.in/img/faces/7-image.jpg"}, {"id":8,"email":"lindsay.ferguson@reqres.in","first\_name":"Lindsay","last\_name":"Ferguson","avatar":"https://reqres.in/img/faces/8-image.jpg"}, {"id":9,"email":"tobias.funke@reqres.in","first\_name":"Tobias","last\_name":"Funke","avatar":"https://reqres.in/img/faces/9-image.jpg"}, {"id":10,"email":"byron.fields@reqres.in","first\_name":"Byron","last\_name":"Fields","avatar":"https://reqres.in/img/faces/10-image.jpg"}, {"id":11,"email":"george.edwards@reqres.in","first\_name":"George","last\_name":"Edwards","avatar":"https://reqres.in/img/faces/11-image.jpg"}, {"id":12,"email":"rachel.howell@reqres.in","first\_name":"Rachel","last\_name":"Howell","avatar":"https://reqres.in/img/faces/12-image.jpg"}],"support": {"url": "https://reqres.in/#support-heading","text": "To keep ReqRes free, contributions towards server costs are appreciated!"}})' to include 'string you want to search'

Other test results listed:

- PASS** Your test name
- FAIL** Body is correct | Assertion: expected response body to equal 'response\_body\_string' but got ...
- PASS** Content-Type is present
- PASS** Response time is less than 200ms
- FAIL** Successful POST request | Assertion: expected 200 to be one of [ 201, 202 ]
- FAIL** Status code name has string | Assertion: expected response to have status reason 'Created' but got 'OK'

## JAVASCRIPT TESTY

### Struktura JS testu:

```
pm.test("Název testu", () => {
  boolean _check;
});
```

### KONTROLA STATUSŮ

#### STATUS OK 200

##### CURL:

```
curl --location --request GET 'https://reqres.in/api/users?page=2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

##### Tests:

```
pm.test("Check status 200", () => {
  pm.response.to.have.status(200);
});
```

#### STATUS NOT OK 400

## cURL:

```
curl --location --request POST 'https://reqres.in/api/register' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json' \
--data-raw '{

}'
```

## Tests:

```
pm.test("Check status 400", () => {
    pm.response.to.have.status(400);
});
```

---

## KONTROLA TYPU

## cURL:

```
curl --location --request POST 'https://reqres.in/api/register' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json' \
--data-raw '{

}'
```

## Tests:

```
pm.test("Check json type", function() {
    pm.response.to.be.json();
});
```

---

## PARSE JSON

Pro přístup k JSONu potřebujeme JSON nejdříve připravit pro Javascript:

```
var data = Pm.response.json();
```

---

## KONTROLA DAT

V rámci kontroly dat můžeme kontrolovat:

- Zda-li element existuje a je definovaný
- Typ elementu
- Hodnotu elementu

Budeme používat funkci `pm.expect`, která [využívá ChaiJS BDD syntaxi](#).

### Příklad expect syntaxe:

```
pm.expect(data.id).to.be.a(number);
```

Zkopírujte request "GET regres.in users 200" do stejné složky a přejmenujte jej na " GET regres.in users tests"

---

## EXISTENCE ELEMENTU V JSON

Následující kontrola provede kontrolu, zdali je prvek definován a nabývá ne null hodnot:

```
var data = pm.response.json();
pm.test("Check data.per_page is ok", () => {
    pm.expect(data.per_page).to.exist;
})
```

```
//toto je vždy fail  
pm.test("Check data.blabla is ok", () => {  
    pm.expect(data.blabla).to.exist;  
})
```

## KONTROLA TYPU ELEMENTU

Skript provede kontrolu, zda-li je element *page* typ number a element *data[0].email* typ boolean (nepravda, vyhodí fail, simulujeme chybu):

```
pm.test("Check data.page is a number", () => {  
    pm.expect(data.page).to.be.a('number');  
})  
  
//toto je vždy fail  
pm.test("Check data.data[0].email is a boolean", () => {  
    pm.expect(data.data[0].email).to.be.a('boolean');  
})
```

## KONTROLA HODNOTY ELEMENTU

Pro systémové a akceptační testy může být dobré kontrolovat přímo hodnoty elementů.

```
pm.test("Check data.page is 2", () => {  
    pm.expect(data.page).to.be.equal(2);  
})  
  
pm.test("Check data.data[0].email is michael.lawson@reqres.in", () => {  
    pm.expect(data.data[0].email).to.be.equal('michael.lawson@reqres.in');  
})  
  
//fail  
pm.test("Check data.data[0].first_name is Petr", () => {  
    pm.expect(data.data[0].first_name).to.be.equal('Petr');  
})
```

## PŘEDÁVÁNÍ DAT V RÁMCI REQUESTŮ

V případě volání více requestů za sebou můžeme zkombinovat nastavení proměnných v testu, následně využít v dalším requestu. Pokud toto potřebujete pro automatizované testy, doporučuji využít spíše [Collection Runneru](#) a Local Variables. Využitím environment proměnných si můžete velice rychle udělat ve variables velký neporádeček.

## PŘÍKLAD

Provoleme 2 requesty v [gorest.co.in](https://gorest.co.in), z jednoho uložíme ID do proměnné a následně použijeme v druhém requestu. V testu následně zkontrolujeme, že ID je totožné s tím, co máme uložené.

Pro requesty vytvoříme novou složku ve složce "Tests", nazveme ji "Data transfer"

### PRVNÍ REQUEST (GET USERS)

#### Nová složka: Skoleni/Tests/Data transfer

##### Curl:

```
curl --location --request GET 'https://gorest.co.in/public/v1/users' \  
--header 'Accept: application/json' \  
--header 'Content-Type: application/json' \  
--header 'Authorization: Bearer a69dd3dc892e72c9b44fe0843a0e8171c7cfaf18775dd8e96e8295895219f5fd'
```

##### Tests:

```
var data = pm.response.json();
```

```
pm.environment.set("userId", data.data[0].id);
```

## DRUHÝ REQUEST (GET USER)

Po importu je potřeba vložit proměnnou userId do URI.

Curl:

```
curl --location -g --request GET 'https://gorest.co.in/public/v1/users/:id' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer a69dd3dc892e72c9b44fe0843a0e8171c7efa618775dd8e96e8295895219f5fd'
```

Tests:

```
var data = pm.response.json();
var id = pm.environment.get("userId")

pm.test("data.data.id is " + id, () => {
    pm.expect(data.data.id).to.equal(id);
});
```

## SKRIPTOVÁNÍ V POSTMANOVÍ

### DYNAMICKÉ PROMĚNNÉ

Dynamické proměnné jsou vhodné pro použití jako dummy data. Generují se vždy při zavolání dynamické proměnné Dyn, což znamená, že je možné je použít i několikrát v jednom requestu. Dokumentaci proměnných najdete [zde](#).

Použití dynamických proměnných:

- Ve standardních oknech:  
    {{\${jmenoDynamickePromenne}}}
- Ve skriptovacích oknech:  
    pm.variables.replaceIn('{{\${jmenoDynamickePromenne}}}).

V následujícím příkladu použijeme tyto dynamické proměnné:

- \$randomFullName
- \$randomExampleEmail

## PŘÍKLAD

Provoleme POST users a vytvoříme uživatele v API: gorest.co.in

Pro post potřebujeme bearer token, který máme vytvořený z Authorizací.

**Nová složka: Skoleni/Tests/Dynamic Vars**

**Název Req: POST gorest users dynamic vars**

Curl:

```
curl --location --request POST 'https://gorest.co.in/public/v1/users' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer a69dd3dc892e72c9b44fe0843a0e8171c7efa618775dd8e96e8295895219f5fd' \
--data-raw '{
    "name": "Dixie Effertz",
    "gender": "male",
    "email": "Deja75@example.com",
    "status": "active"
}'
```

V Body nahradíme jméno a email dynamickými proměnnými.

```
{
  "name": "{$randomFullName}",
  "gender": "male",
  "email": "{$randomEmail}",
  "status": "active"
}
```

V Tests uložíme lokální proměnnou pro ID, jméno a e-mail, který následně zkontrolujeme v dalším requestu.

### Tests:

```
var data = pm.response.json();

pm.environment.set("userId", data.data.id);
pm.environment.set("name", data.data.name);
pm.environment.set("email", data.data.email);
```

Následně vytvoříme další request, GET User, kde použijeme uložené ID a v testech zkontrolujeme i ostatní hodnoty.

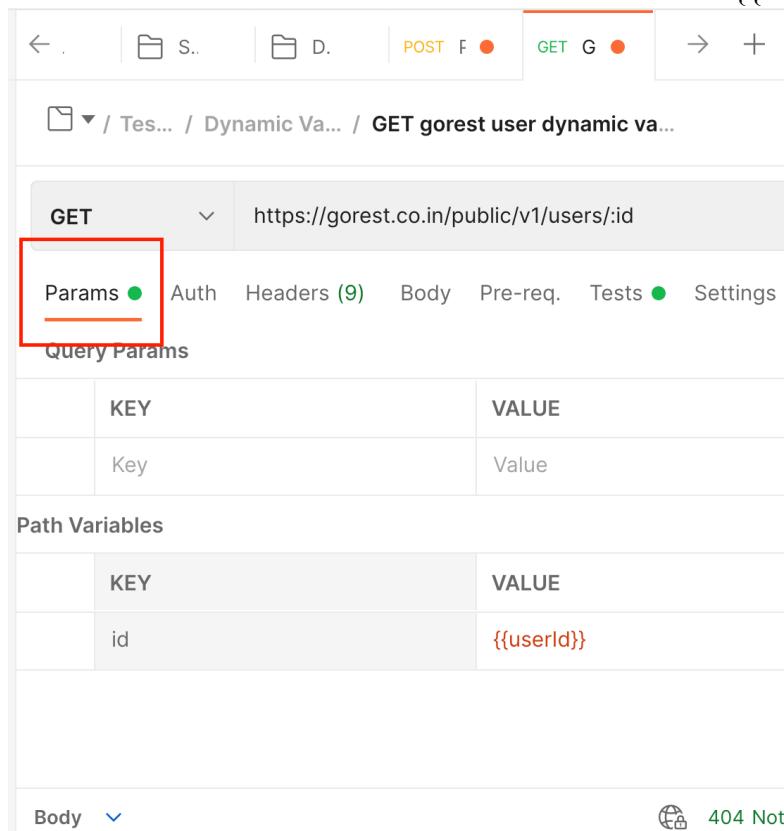
### Složka: Skoleni/Tests/Dynamic Vars

#### Název Req: GET gorest user dynamic vars

### Curl:

```
curl --location -g --request GET 'https://gorest.co.in/public/v1/users/:id' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json'
```

Do Path Variables v Params vložte do klíče id hodnotu "{{userId}}".



The screenshot shows the Postman interface with a GET request to `https://gorest.co.in/public/v1/users/:id`. The 'Params' tab is selected, and the 'id' parameter is set to `value: {{userId}}`. The 'Path Variables' section also shows 'id' with value `value: {{userId}}`.

|  | KEY | VALUE |
|--|-----|-------|
|  | Key | Value |

|  | KEY | VALUE                          |
|--|-----|--------------------------------|
|  | id  | <code>value: {{userId}}</code> |

### Tests:

```

var data = pm.response.json();
var id = pm.environment.get("userId");
var name = pm.environment.get("name");
var email = pm.environment.get("email")

pm.test("data.data.id is " + id, () => {
    pm.expect(data.data.id).to.equal(id);
});

pm.test("data.data.name is " + name, () => {
    pm.expect(data.data.name).to.equal(name);
});

pm.test("data.data.email is " + email, () => {
    pm.expect(data.data.email).to.equal('email');
});

```

## PRÁCE S TIMESTAMP (BONUS)

Může se Vám stát, že do requestu budete potřebovat přidat čas. Proto je dobré vědět, jak je možné timestamp v požadovaném formátu vygenerovat.

Pro práci s časem a datem používám v Postman knihovnu [momentjs](#). Tuto knihovnu je nutné nejdříve importovat do skriptu, následně je možné začít s ní pracovat.

### AKTUÁLNÍ TIMESTAMP V POŽADOVANÉM FORMÁTU

Následující příklad vrátí timestampu ve formátu: YYYY-MM-DD[T]HH:mm:ss[Z], například: 2021-10-22T21:16:00Z

```

const moment = require('moment');
Console.log(moment().format("YYYY-MM-DD[T]HH:mm:ss[Z]"));

```

### ADD, SUBTRACT

*Add* a *subtract* funkce můžete použít, pokud potřebujete k aktuální timestamp přidat nebo odebrat čas.

#### Syntaxe:

```

moment().add(cislo, klic);
moment().subtract(cislo, klic);

```

### HODNOTY PRO MANIPULACI S ČASEM

| Klíč                | Zkratka |
|---------------------|---------|
| <b>years</b>        | y       |
| <b>quarters</b>     | Q       |
| <b>months</b>       | M       |
| <b>weeks</b>        | w       |
| <b>days</b>         | d       |
| <b>hours</b>        | h       |
| <b>minutes</b>      | m       |
| <b>seconds</b>      | s       |
| <b>milliseconds</b> | ms      |

### PŘÍKLADY PRÁCE S MOMENT()

```

const moment = require('moment');
//Přidá 10 minut k aktuální timestamp
addTenMinutes = moment().add(10, 'minutes').format("YYYY-MM-DD[T]HH:mm:ss[Z]");

//Přidá 2 dny k aktuální timestamp
addTwoDays = moment().add(2, 'd').format("YYYY-MM-DD[T]HH:mm:ss[Z]");

//Odebere 1 měsíc z aktuální timestamp
subtractMonth = moment().subtract(2, 'M').format("YYYY-MM-DD[T]HH:mm:ss[Z]");

//Odebere 10 let z aktuální timestamp
subtractTenYears = moment().subtract(10, 'years').format("YYYY-MM-DD[T]HH:mm:ss[Z]");

console.log(addTenMinutes)
console.log(addTwoDays)
console.log(subtractMonth)
console.log(subtractTenYears)

```

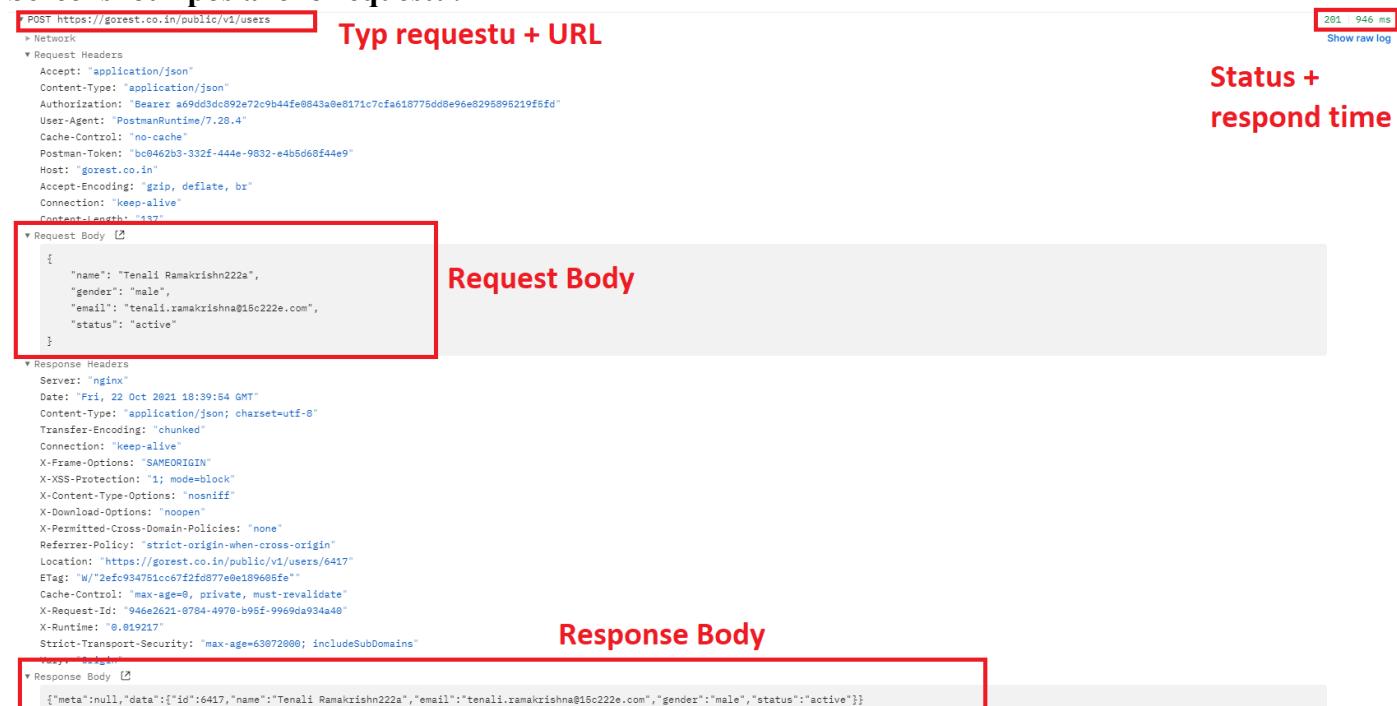
## DEBUGGING

Konzoli v Postman již znáte z předchozích příkladů. Všechny requesty, které pošlete se zapíšou do konzole. Do testů si můžete napsat vlastní logování, které se následně zobrazí v konzoli. Také se jedná o účinného pomocníka, když používáte proměnné a nejste si jistí, jaká hodnota se poslala.

## DEBUGGING REQUESTU

Consoli naleznete všechny informace, které se poslali z requestu. Jsou zde reálné hodnoty, pokud používáte proměnné nebo dynamické proměnné, tak právě zde naleznete poslané hodnoty.

### Screenshot z posланého requestu:



The screenshot shows a Postman request response. The URL is <https://gorest.co.in/public/v1/users>. The status code is 201 with a response time of 946 ms. The response body is shown in red boxes.

**Request Headers**

- Accept: "application/json"
- Content-Type: "application/json"
- Authorization: "Bearer a69dd3dc892e72c9b44fe8843a0e8171c7cfa618775dd8e96e8295895219f5fd"
- User-Agent: "PostmanRuntime/7.28.4"
- Cache-Control: "no-cache"
- Postman-Token: "bc0462b3-332f-444e-9832-e4b5d68f44e9"
- Host: "gorest.co.in"
- Accept-Encoding: "gzip, deflate, br"
- Connection: "keep-alive"
- Content-Length: "137"

**Request Body**

```
{
  "name": "Tenali Ramakrishn22a",
  "gender": "male",
  "email": "tenali.ramakrishna@15c222e.com",
  "status": "active"
}
```

**Status + respond time**

201 946 ms  
Show raw log

**Response Headers**

- Server: "nginx"
- Date: "Fri, 22 Oct 2021 18:39:54 GMT"
- Content-Type: "application/json; charset=utf-8"
- Transfer-Encoding: "chunked"
- Connection: "keep-alive"
- X-Frame-Options: "SAMEORIGIN"
- X-XSS-Protection: "1; mode=block"
- X-Content-Type-Options: "nosniff"
- X-Download-Options: "noopen"
- X-Permitted-Cross-Domain-Policies: "none"
- Referrer-Policy: "strict-origin-when-cross-origin"
- Location: "https://gorest.co.in/public/v1/users/6417"
- ETag: "W/\"2ef0934751c67df087e0e189606fe"
- Cache-Control: "max-age=0, private, must-revalidate"
- X-Request-ID: "946e2621-0784-4970-b95f-9969da934a0"
- X-Runtime: "0.019217"
- Strict-Transport-Security: "max-age=63072000; includeSubDomains"
- Vary: "Origin"

**Response Body**

```
{"meta":null,"data":[{"id":6417,"name":"Tenali Ramakrishn22a","email":"tenali.ramakrishna@15c222e.com","gender":"male","status":"active"}]}
```

Je také možné zobrazit raw log (například po zkopirování do defectu) a to pomocí tlačítka "Show raw log":

```
▼ POST https://gorest.co.in/public/v1/users
  ▶ Network
  ▷ Request Headers
    Accept: "application/json"
    Content-Type: "application/json"
    Authorization: "Bearer a69dd3dc892e72c9b44fe0843a0e8171c7cfab18775dd8e96e8295895219f5fd"
    User-Agent: "PostmanRuntime/7.28.4"
    Cache-Control: "no-cache"
    Postman-Token: "bc0462b3-332f-444e-9832-e4b5d68f44e9"
```

281 946 ms  
Show raw log

## Příklad Raw logu:

```
POST /public/v1/users HTTP/1.1
Accept: application/json
Content-Type: application/json
Authorization: Bearer a69dd3dc892e72c9b44fe0843a0e8171c7cfab18775dd8e96e8295895219f5fd
User-Agent: PostmanRuntime/7.28.4
Cache-Control: no-cache
Postman-Token: bc0462b3-332f-444e-9832-e4b5d68f44e9
Host: gorest.co.in
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 137
```

```
{
  "name": "Tenali Ramakrishn222a",
  "gender": "male",
  "email": "tenali.ramakrishna@15c222e.com",
  "status": "active"
}
```

```
HTTP/1.1 201 Created
Server: nginx
Date: Fri, 22 Oct 2021 18:39:54 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Download-Options: noopener
X-Permitted-Cross-Domain-Policies: none
Referrer-Policy: strict-origin-when-cross-origin
Location: https://gorest.co.in/public/v1/users/6417
ETag: W/"2efc934751cc67f2fd877e0e189605fe"
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: 946e2621-0784-4970-b95f-9969da934a40
X-Runtime: 0.019217
Strict-Transport-Security: max-age=63072000; includeSubDomains
Vary: Origin

{"meta":null,"data":{"id":6417,"name":"Tenali
Ramakrishn222a","email":"tenali.ramakrishna@15c222e.com","gender":"male","status":"active"}}
```

---

## DEBUGGING TESTŮ

V rámci testů můžeme logovat následujícím způsobem:

```
console.warn("Toto je warning")
console.log("Standardní log")
console.info('O něčem informuju')
console.error('ajejej, chyba')
```

Vložte tento kus kódu do Tests jakéhokoliv requestu a provolejte ho.

V Konzoli tyto logy vypadají takto:

▶ POST <https://gorest.co.in/public/v1/users>

⚠ "Toto je warning"

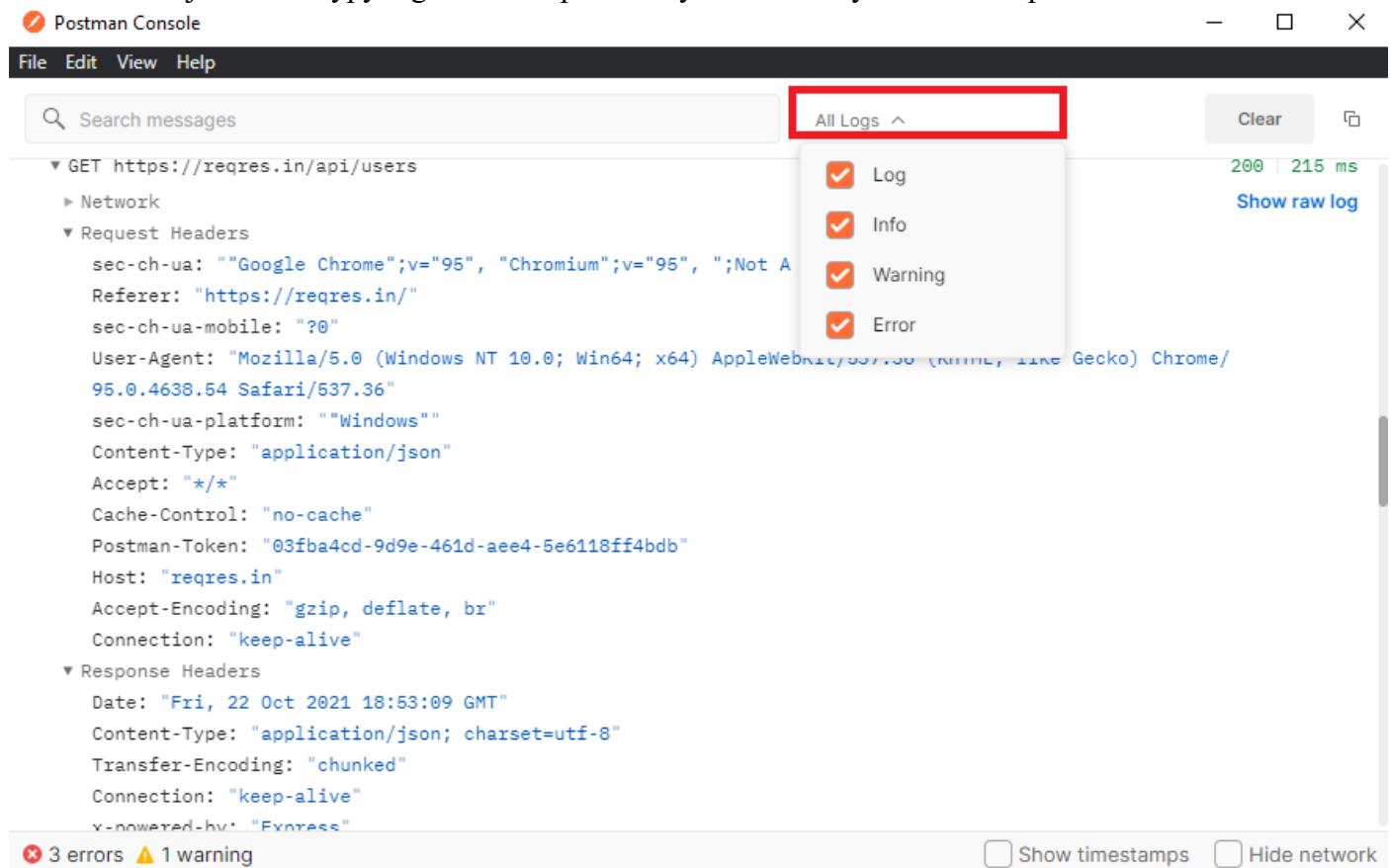
"Standardní log"

ℹ "O něčem informuji"

⚠ "ajejej, chyba"

▶ POST <https://gorest.co.in/public/v1/users>

Můžeme také jednotlivé typy logů filtrovat pomocí výběru vedle vyhledávacího pole:



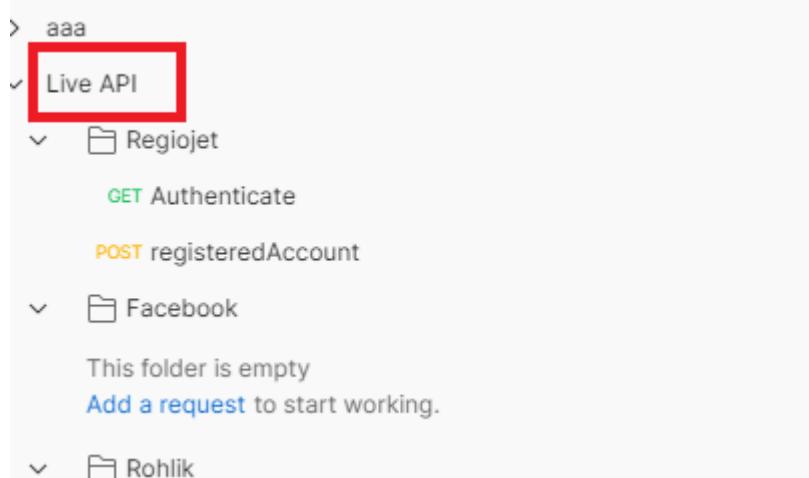
The screenshot shows the Postman Console interface. At the top, there's a search bar labeled "Search messages". To its right is a dropdown menu titled "All Logs" which is currently expanded, showing four filter options: "Log", "Info", "Warning", and "Error", each with a checked checkbox. Below this dropdown, the main content area displays a request and response for a GET request to "https://reqres.in/api/users". The request headers include "sec-ch-ua", "Referer", "sec-ch-ua-mobile", "User-Agent", "sec-ch-ua-platform", "Content-Type", "Accept", "Cache-Control", "Postman-Token", "Host", "Accept-Encoding", and "Connection". The response headers include "Date", "Content-Type", "Transfer-Encoding", "Connection", and "x-powered-by". At the bottom left, there are error and warning counts: "3 errors" and "1 warning". At the bottom right, there are two checkboxes: "Show timestamps" and "Hide network".

## PŘEPOUŽÍVÁNÍ KÓDU (BONUS HACK)

Postman bohužel oficiálně neumožnuje vytvářet funkce, které by šly jednodušše sdílet mezi requesty. Proto obzvláště ve složitějších testech dochází k duplikaci kódu. Jak se tomuto vyvarovat? Existuje možnost napsat si vlastní funkce a vložit je do *pre-request script* kolekce a následně je volat z testů.

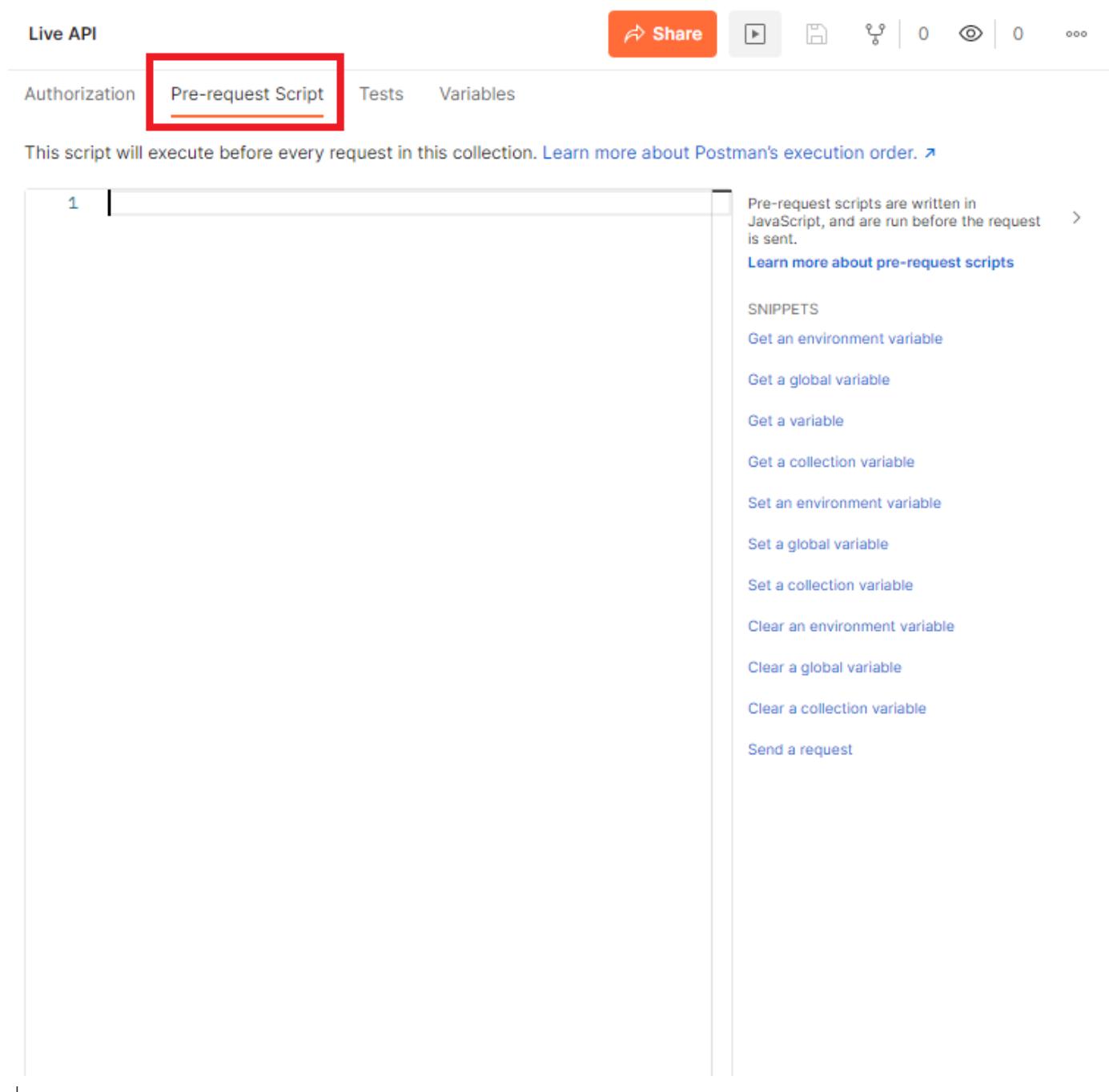
**Pozor! Toto je tak trochu hackování Postmanu a je potřeba tyto skripty tvorit se zvýšenou pozorností.**

1. Otevřít detail kolekce (levý klik na její název)



The screenshot shows the left sidebar of the Postman interface. A collection named 'Live API' is selected, indicated by a red box around its name. Other collections visible include 'aaa' and 'Rohlik'. Under 'Live API', there are two requests: 'GET Authenticate' and 'POST registeredAccount'. Below 'Live API' is a folder named 'Facebook' which is described as 'This folder is empty' and has a link to 'Add a request'.

2. Otevřít záložku *Pre-request Script*



The screenshot shows the main interface of Postman with the 'Pre-request Script' tab selected, highlighted by a red box. The tab bar also includes 'Authorization', 'Tests', and 'Variables'. At the top right, there are various icons for sharing, saving, and other actions. The main area contains a code editor with a single line of code: '1'. To the right of the editor is a sidebar with documentation for pre-request scripts, including sections on snippets and environment variables.

This script will execute before every request in this collection. [Learn more about Postman's execution order.](#)

Pre-request scripts are written in JavaScript, and are run before the request is sent.  
[Learn more about pre-request scripts](#)

**SNIPPETS**

[Get an environment variable](#)  
[Get a global variable](#)  
[Get a variable](#)  
[Get a collection variable](#)  
[Set an environment variable](#)  
[Set a global variable](#)  
[Set a collection variable](#)  
[Clear an environment variable](#)  
[Clear a global variable](#)  
[Clear a collection variable](#)  
[Send a request](#)

Následující kód umožní vytvářet vlastní přepoužitelné funkce:

```
pm.globals.set('loadUtils', function loadUtils() {
    let utils = {};
    utils.prepouzitelnaFunkce = function prepouzitelnaFunkce(text) {
        console.info("prepouzitelnaFunkce: " + text)
    };
    utils.sectiDveCisla = function sectiDveCisla(prvniCislo, druheCislo) {
        vysledek = prvniCislo + druheCislo
        return vysledek;
    }
    return utils;
} + ';\nloadUtils();');
```

V testech je pak potreba nas kod zavolat:

```
const utils=eval(globals.loadUtils);

utils.prepouzitelnaFunkce("nejaky text");
vysledek = sectiDveCisla(1, 3)
console.log(vysledek)
```

V konzoli pak vidíme, že se funkce definované na úrovni kolekce skutečně provedly:



## COLLECTION RUNNER

Collection Runner slouží ke spouštění requestů v dané kolekci/složce. Requesty se spouští v pořadí, v jakém jsou umístěny v kolekci.

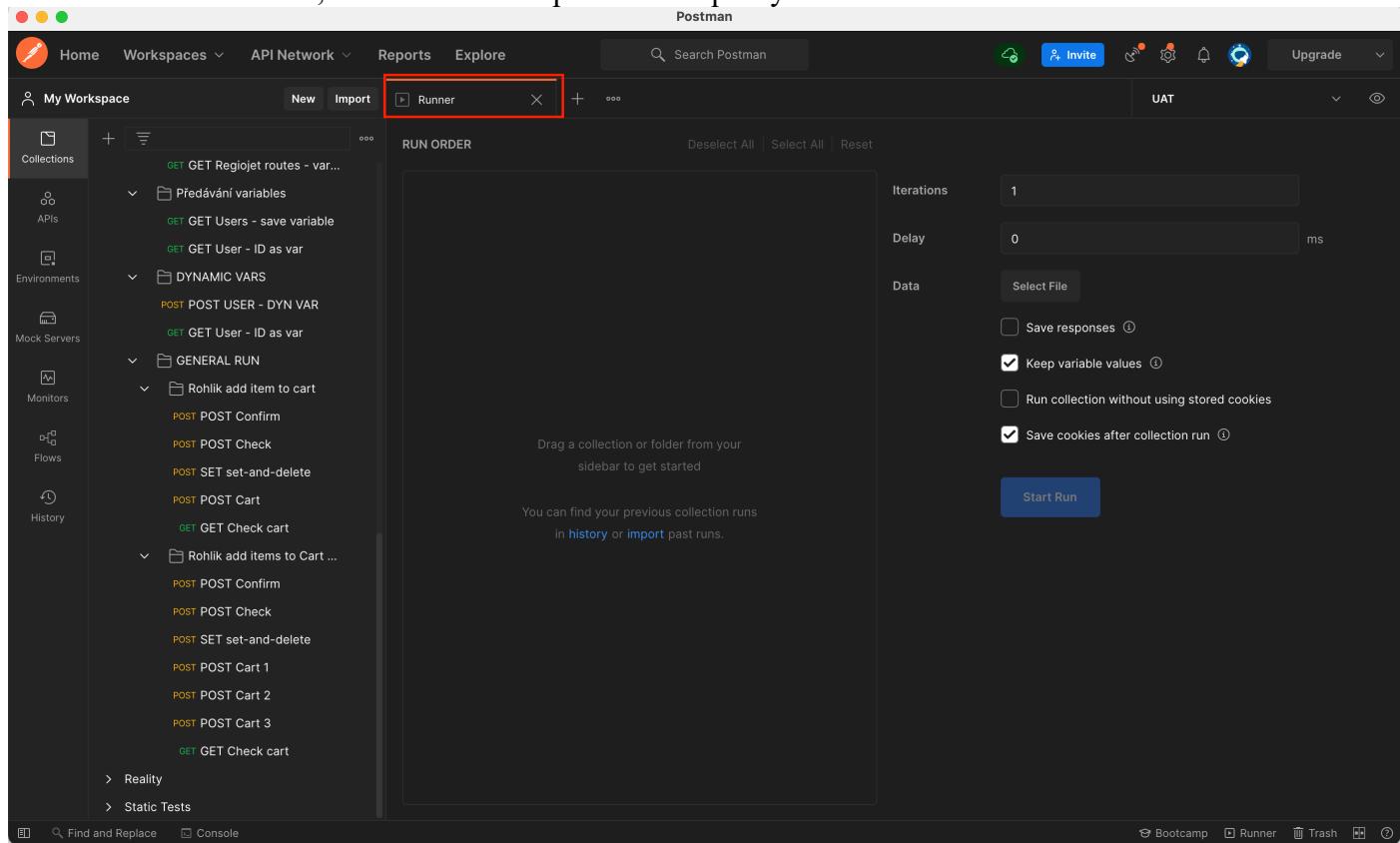
V rámci běhu v Collection Runneru můžeme použít local variables (po dokončení běhu se smažou) a také iterationData, které postmanu můžeme předat externím JSON souborem.

## OVLÁDÁNÍ RUNNERU

Runner můžeme spustit pomocí tlačítka v dolní části Postman UI

The screenshot shows the Postman interface with the 'Runner' button highlighted in red at the bottom right of the toolbar. The main area displays a collection named 'GENERAL...' containing several requests. One request is selected, showing its details in the center panel, including the URL, method, and a snippet of a JavaScript test script. The test script checks if the total price matches a variable 'rohlikPriceSum'. The response panel shows a successful 200 OK status with a response body containing a JSON object with fields like 'status', 'messages', and 'data'. The left sidebar shows the 'Collections' section with various projects and environments listed.

Otevře se okno v tabech, odkud následně spouštíme requesty.

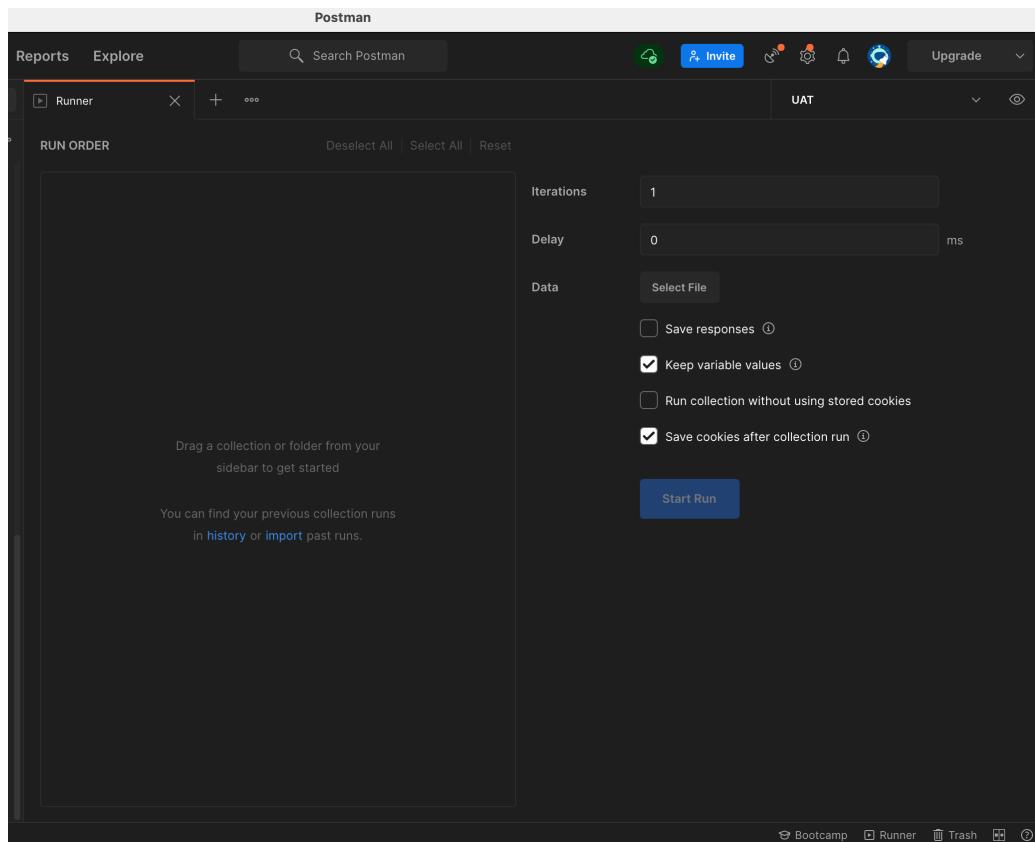


The screenshot shows the Postman application interface. The top navigation bar includes Home, Workspaces, API Network, Reports, Explore, a search bar, and various icons for invite, upgrade, and help. Below the navigation is a toolbar with New, Import, and a 'Runner' button highlighted with a red box. The main area is titled 'My Workspace' and contains a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A 'RUN ORDER' section displays a list of requests under categories like 'Predávání variabls', 'DYNAMIC VARS', 'GENERAL RUN', and 'Rohlik add item to cart'. On the right, configuration options for 'Iterations' (set to 1), 'Delay' (set to 0 ms), and 'Data' (with a 'Select File' button) are shown. Below these are checkboxes for 'Save responses', 'Keep variable values' (which is checked), 'Run collection without using stored cookies', and 'Save cookies after collection run'. A large blue 'Start Run' button is at the bottom right. The bottom of the interface has tabs for Bootcamp, Runner, Trash, and a help icon.

## KONFIGURACE RUNNERU

V runneru nastavujeme

| Hodnota  | Vysvětlení   |
|--|--|
| <b>Iterations</b>                                  | Počet opakování, pouští se sériově za sebou. Možné použít například pro monitorování response time.  |
| <b>Delay</b>                                       | Zpoždění mezi jednotlivými iteracemi   |
| <b>Data</b>  | Místo pro datový soubor viz <a href="#">Data driven testy</a>  |
| <b>Save Responses</b>                              | Runner neukládá responses, může to zpomalovat běh runneru. Bez nich je ale obtížnější debugging. Pro testování kolekcí většinou ponechávám zapnuté |
| <b>Keep variable values</b>                        | Je možné vypnout ukládání proměnných. Pokud je tento checkbox neaktivní, pak běh runneru neovlivní ani globals ani environment proměnné.           |
| <b>Run collection without using stored cookies</b> | Pokud je checkbox aktivní, pak se nepoužijí uložené cookies  |
| <b>Save cookies after collection run</b>           | Uloží cookies po dokončení runneru   |



## OBECNÉ SPUŠTĚNÍ

Připravíme si 5 requestů na API rohlik.cz. Nejdříve zvolíme adresu, následně přidáme 2 položky do košíku a zkontrolujeme celkovou cenu.

Provádíme address autocomplete confirm pro získání ID adresy, to následně uložíme do proměnné. Nezapomeňte nastavit prostředí na PROD.

**Nová složka: Skoleni/Collection runner**

**Nová složka: Skoleni/Collection runner/General**

**Název req: POST rohlik.cz autocomplete confirm**

### cURL:

```
curl --location --request POST 'https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm' \
--header 'authority: www.rohlik.cz' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'x-custom-sessionid: 7dbe32f2-2c5b-4871-9f99-60a3df16f95c' \
--header 'content-type: application/json' \
--header 'x-origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'accept: */*' \
--header 'origin: https://www.rohlik.cz' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'referer: https://www.rohlik.cz/regal' \
--header 'accept-language: en-US,en;q=0.9' \
--data-raw '{
  "suggestText": "Na Hřebenech II 1718/8",
  "suggestHint": "Adresa, Praha 4 - Nusle, Česko",
  "country": "Česko",
  "countryCode": "CZ",
```

```

"region": "Hlavní město Praha",
"city": "Praha",
"street": "Na Hřebenech II",
"houseNumber": "1718",
"streetNumber": "8",
"evidenceNumber": "",
"longitude": 14.430146454451089,
"latitude": 50.05145454486651,
"importance": 0.09419794408814568,
"postalNumber": "14000",
"quarter": "Praha 4",
"valid": true,
"serviceType": "SEZNAM",
"evidenceNumberResult": false
}'

```

### Tests:

```

var data = pm.response.json();

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.environment.set("rohlikAddressId", data.data.id)

```

Následně provoláme delivery-address/check s uloženou addressId pro získání store ID, které následně také uložíme do proměnné.

### Složka: Skoleni/Collection runner/General

#### Název req: POST rohlik.cz delivery address check

##### cURL:

```

curl --location --request POST 'https://www.rohlik.cz/services/frontend-service/delivery-address/check' \
--header 'authority: www.rohlik.cz' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'accept: application/json' \
--header 'content-type: application/json' \
--header 'x-origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'origin: https://www.rohlik.cz' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'referer: https://www.rohlik.cz/c300102000-ovoce-a-zelenina' \
--header 'accept-language: en-US,en;q=0.9' \
--data-raw '{"addressId": {"rohlikAddressId": "14000"} }'

```

### Tests:

```

var data = pm.response.json();

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.environment.set("rohlikStoreId", data.data.availableStores[0].storeId)
pm.environment.set("rohlikFullAddress", data.data.address.fullAddress)
pm.environment.set("rohlikStreet", data.data.address.street)
pm.environment.set("rohlikPsc", data.data.address.postalCode)
pm.environment.set("rohlikCity", data.data.address.city)

pm.test("Data.data is object", function () {
    pm.expect(data.data).to.be.an('object');
});

```

Další request nám uloží adresu do cookies, abychom mohli začít pracovat s košíkem.

### Složka: Skoleni/Collection runner/General

#### Název req: POST rohlik.cz delivery address set and delete

##### cURL:

```
curl --location --request POST 'https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete' \
--header 'authority: www.rohlik.cz' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'content-type: application/json' \
--header 'x-origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'accept: */*' \
--header 'origin: https://www.rohlik.cz' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'accept-language: en-US,en;q=0.9' \
--header 'Cookie: PHPSESSION=fdMHwRzqCMwTRph6twvS0MLWDlQL5iht' \
--data-raw '{
    "warehouseId": {{rohlikStoreId}},
    "streetWithNumber": "{{rohlikStreet}}3",
    "city": "{{rohlikCity}}",
    "postalCode": "{{rohlikPsc}}",
    "isGeocodeResult": false,
    "id": 2644841,
    "flatDetails": {
        "floor": "",
        "door": ""
    }
}'
```

##### Tests:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Přidáme položku do košíku a uložíme hodnotu celé objednávky

### Složka: Skoleni/Collection runner/General

#### Název req: POST rohlik.cz cart

##### cURL:

```
curl --location --request POST 'https://www.rohlik.cz/services/frontend-service/v2/cart' \
--header 'authority: www.rohlik.cz' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'accept: application/json' \
--header 'content-type: application/json' \
--header 'x-origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'origin: https://www.rohlik.cz' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'referer: https://www.rohlik.cz/c300102000-ovoce-a-zelenina' \
--header 'accept-language: en-US,en;q=0.9' \
--data-raw '{
    "productId": 1410881,
```

```

    "quantity": 1,
    "source": ":ProductCategory:300102000",
    "actionId": 2972455,
    "recipeId": null
}'

```

### Tests:

```

var data = pm.response.json();

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Data.data.totalprice is a number", () => {
    pm.expect(data.data.totalPrice).to.be.a('number');
    pm.environment.set("rohlikPriceSum", data.data.totalPrice);
})

```

Jako poslední krok zavoláme kontrolu košíku a ověříme celkovou hodnotu objednávky.

### Složka: Skoleni/Collection runner/General

#### Název req: GET rohlik.cz cart review

### curl:

```

curl --location --request GET 'https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-
cart?blockingValidation=false' \
--header 'authority: www.rohlik.cz' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'x-origin: WEB' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sentry-trace: e1c874ed50da42baa14ae9443422105a-9ecc6dbbd758c56c-1' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'accept: */*' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'referer: https://www.rohlik.cz/objednavka/prehled-kosiku' \
--header 'accept-language: en-US,en;q=0.9' \
--data-raw "

```

### Tests:

```

var data = pm.response.json();

var rohlikPriceSum = pm.variables.get("rohlikPriceSum")

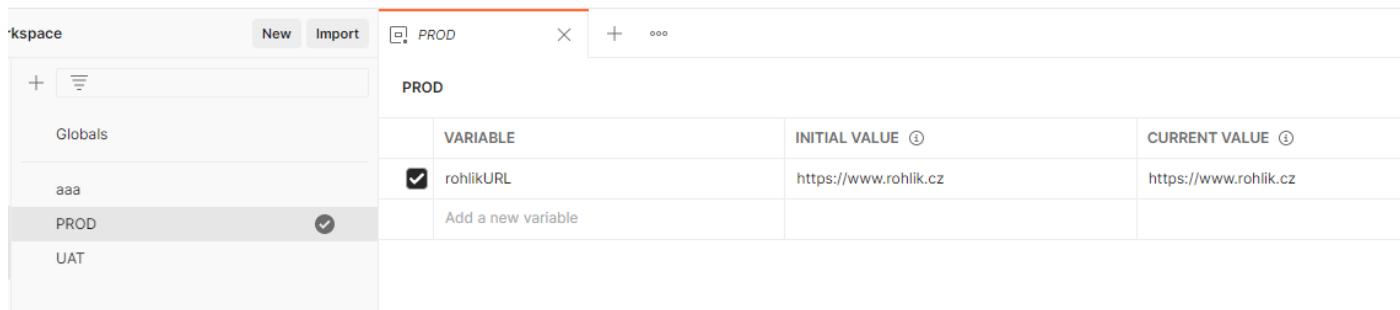
pm.test("Total price is " + rohlikPriceSum, () => {
    pm.expect(data.data.totalPrice).to.equal(rohlikPriceSum);
})

```

## ENV PROMĚNNÉ PRO ROHLIK.CZ

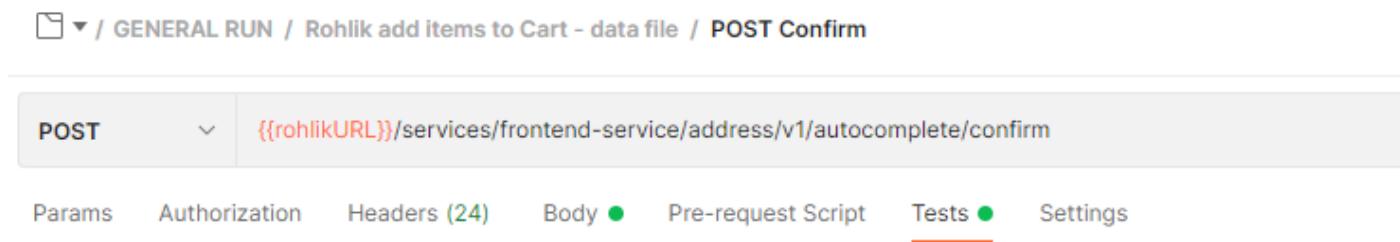
URL rohlíku můžeme uložit do environment variables.

1. Otevřete environments
2. Otevřete prostředí "PROD"
3. Vložte novou proměnnou rohlikURL (pozor na mezery za názvem)a vložte do ní:  
<https://www.rohlik.cz>



The screenshot shows the Postman interface with a workspace named 'PROD'. On the left, there's a sidebar with sections for 'Globals', 'aaa', 'PROD' (which is selected), and 'UAT'. In the main area, under 'PROD', there's a table for variables. It has columns for 'VARIABLE', 'INITIAL VALUE', and 'CURRENT VALUE'. A row for 'rohlikURL' is shown with a checked checkbox, an initial value of 'https://www.rohlik.cz', and a current value of 'https://www.rohlik.cz'.

Následně nahraďte začátek URL touto proměnnou ve všech requestech.



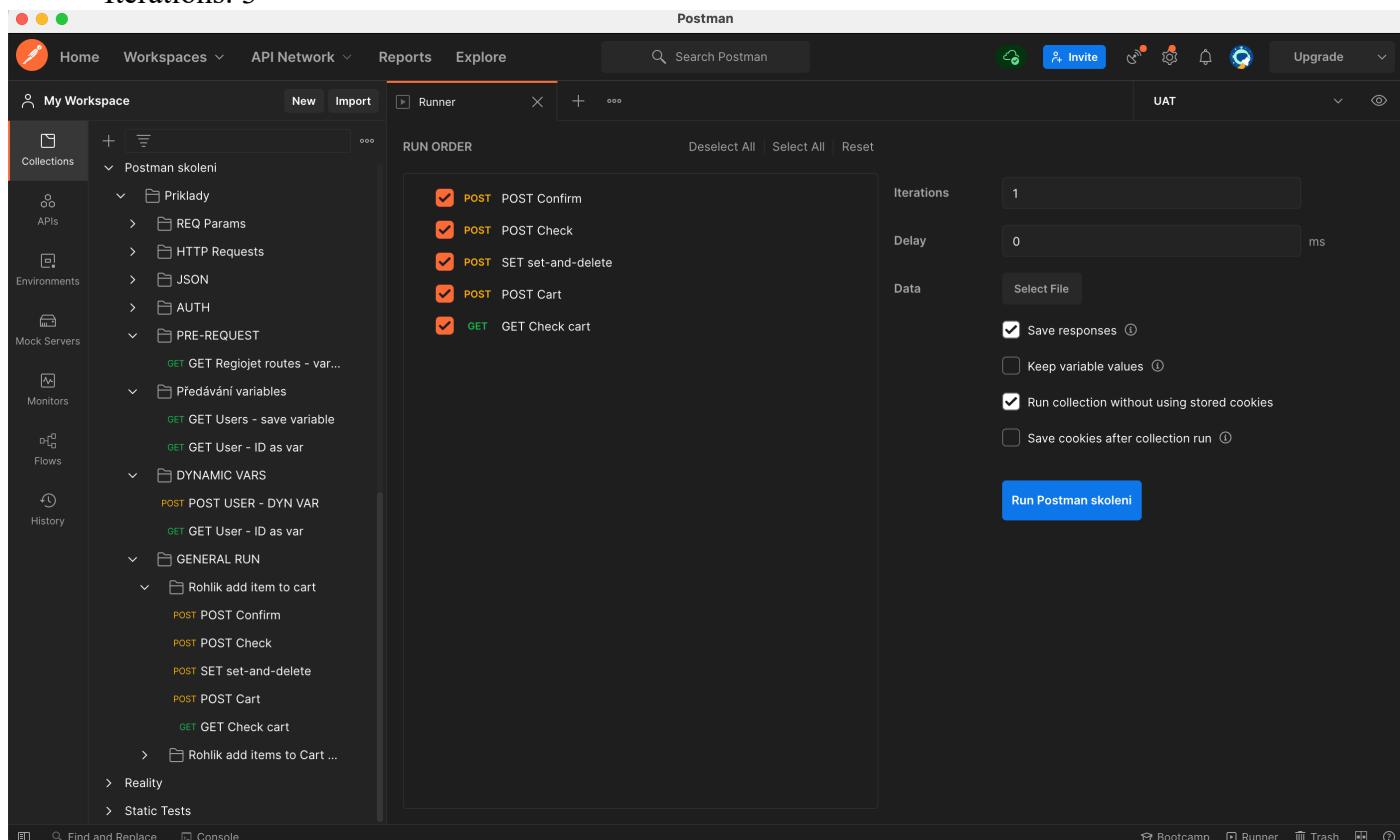
The screenshot shows a single POST request in the Postman interface. The URL field contains the placeholder `{{rohlikURL}}/services/frontend-service/address/v1/autocomplete/confirm`. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (24)', 'Body', 'Pre-request Script', 'Tests' (which is selected), and 'Settings'.

## SPUŠTĚNÍ PŘIPRAVENÉ SLOŽKY/KOLEKCE

Složku nebo kolekci přetáhneme levým tlačítkem myši do okna s runnerem. V našem případě složku s připravenými requesty na rohlik.cz

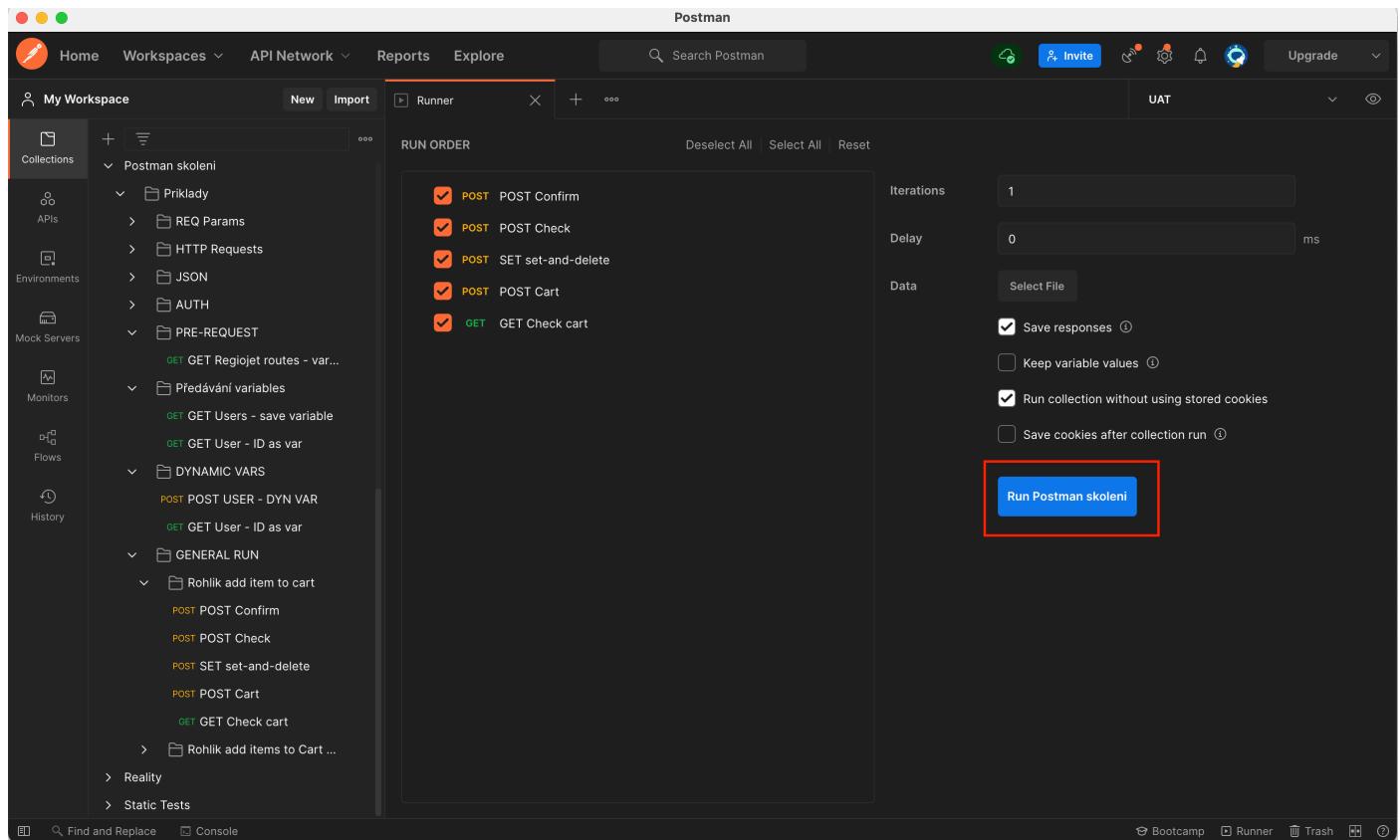
Pro náš run nastavíme následující konfiguraci:

- Save Responses
- Run Collection without using stored cookies
- Iterations: 3



The screenshot shows the Postman Runner interface. On the left, the 'Collections' section lists a collection named 'Postman skolení' which contains sub-folders like 'Priklady', 'REQ Params', 'HTTP Requests', 'JSON', 'AUTH', 'PRE-REQUEST', 'GENERAL RUN', and 'Reality'. Under 'GENERAL RUN', there are several requests: 'POST Confirm', 'POST Check', 'POST SET set-and-delete', 'POST POST Cart', and 'GET GET Check cart'. These requests are all selected with checkboxes. To the right, there are configuration options for 'Iterations' (set to 1), 'Delay' (set to 0 ms), and checkboxes for 'Save responses', 'Run collection without using stored cookies', and 'Keep variable values'. At the bottom right is a large blue button labeled 'Run Postman skolení'.

Spuštěme pomocí tlačítka *Run*

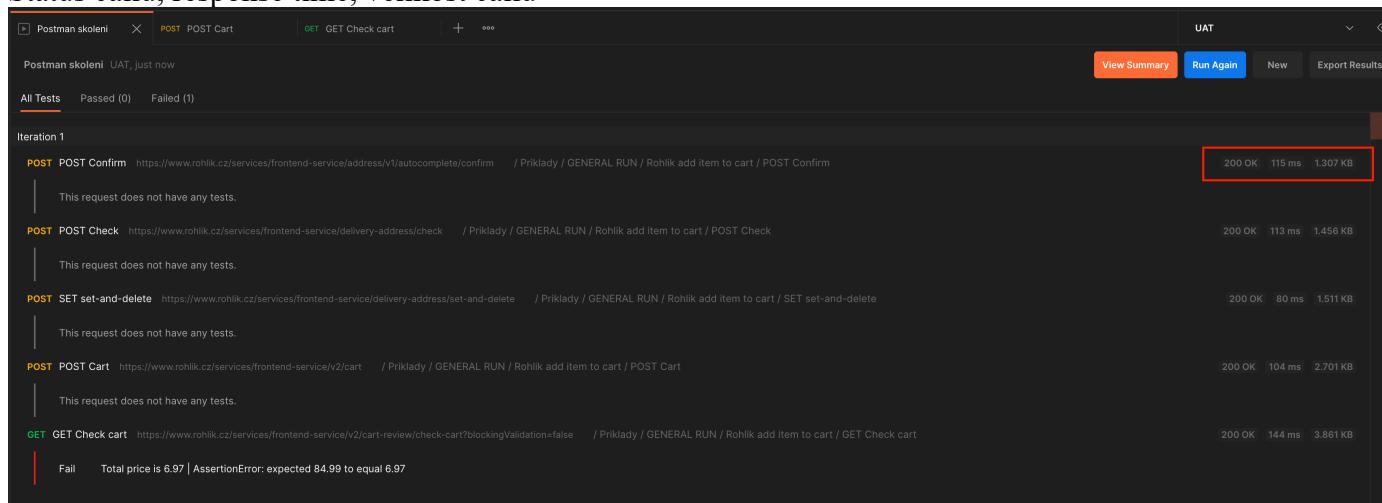


The screenshot shows the Postman application interface. On the left, there's a sidebar with navigation links like Home, Workspaces, API Network, Reports, and Explore. The main area is titled 'Postman' and shows a 'RUN ORDER' list. The collection 'Postman skolení' is expanded, revealing several requests: POST Confirm, POST Check, POST SET set-and-delete, POST POST Cart, and GET GET Check cart. To the right of the run order, there are settings for Iterations (set to 1), Delay (0 ms), Data (Select File), and checkboxes for Save responses, Keep variable values, Run collection without using stored cookies, and Save cookies after collection run. At the bottom right of the run order section is a prominent blue button labeled 'Run Postman skolení' with a red border.

## VÝSLEDEK BĚHU

Po spuštění runu se zobrazí okno s výsledky. V tomto okně můžeme vidět informace, jako jsou:

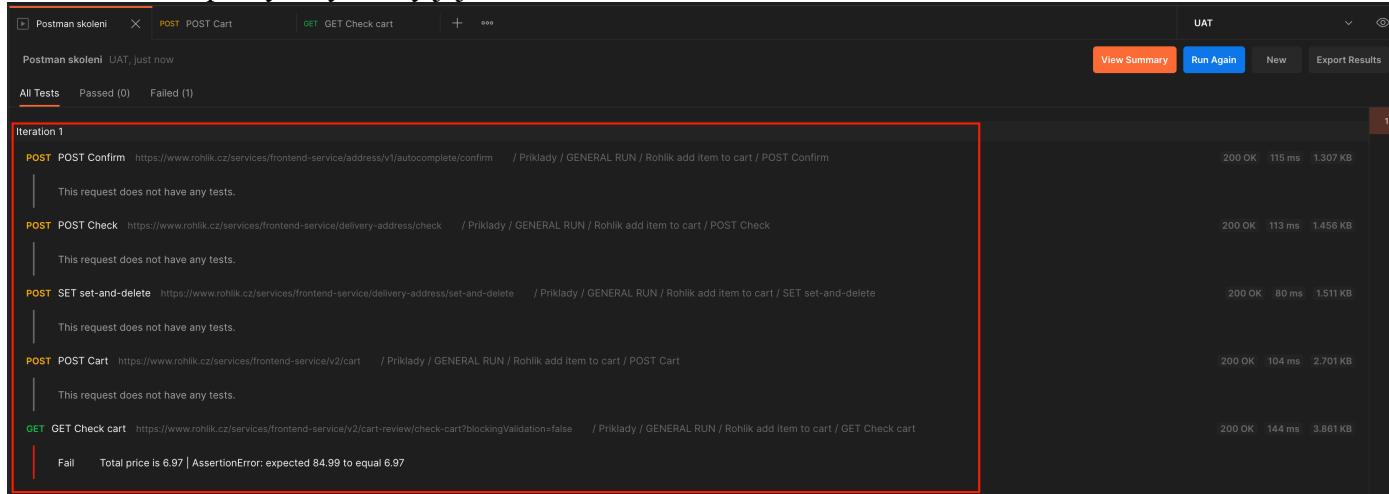
- Status callu, response time, velikost callu



The screenshot shows the Postman Test Results window for the collection 'Postman skolení'. It displays the results of an iteration with one failed test. The failed test is 'POST POST Cart' with the message 'Fail Total price is 6.97 | Assertion: expected 84.99 to equal 6.97'. Other tests listed include 'POST POST Confirm', 'POST POST Check', 'POST SET set-and-delete', and 'GET GET Check cart', all of which passed. The results table shows columns for Method, URL, Path, Response Time, and Size. A red box highlights the failed test entry.

| Method      | URL  | Path   | Response Time | Size     |
|-------------|--|--|---------------|----------|
| POST        | https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm                    | /Příklady / GENERAL RUN / Rohlik add item to cart / POST Confirm       | 200 OK 115 ms | 1.307 KB |
| POST        | https://www.rohlik.cz/services/frontend-service/delivery-address/check                             | /Příklady / GENERAL RUN / Rohlik add item to cart / POST Check         | 200 OK 113 ms | 1.456 KB |
| POST        | https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete                    | /Příklady / GENERAL RUN / Rohlik add item to cart / SET set-and-delete | 200 OK 80 ms  | 1.511 KB |
| POST        | https://www.rohlik.cz/services/frontend-service/v2/cart  | /Příklady / GENERAL RUN / Rohlik add item to cart / POST Cart          | 200 OK 104 ms | 2.701 KB |
| GET         | https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false | /Příklady / GENERAL RUN / Rohlik add item to cart / GET Check cart     | 200 OK 144 ms | 3.861 KB |
| <b>Fail</b> | Total price is 6.97   Assertion: expected 84.99 to equal 6.97                                      |  |               |          |

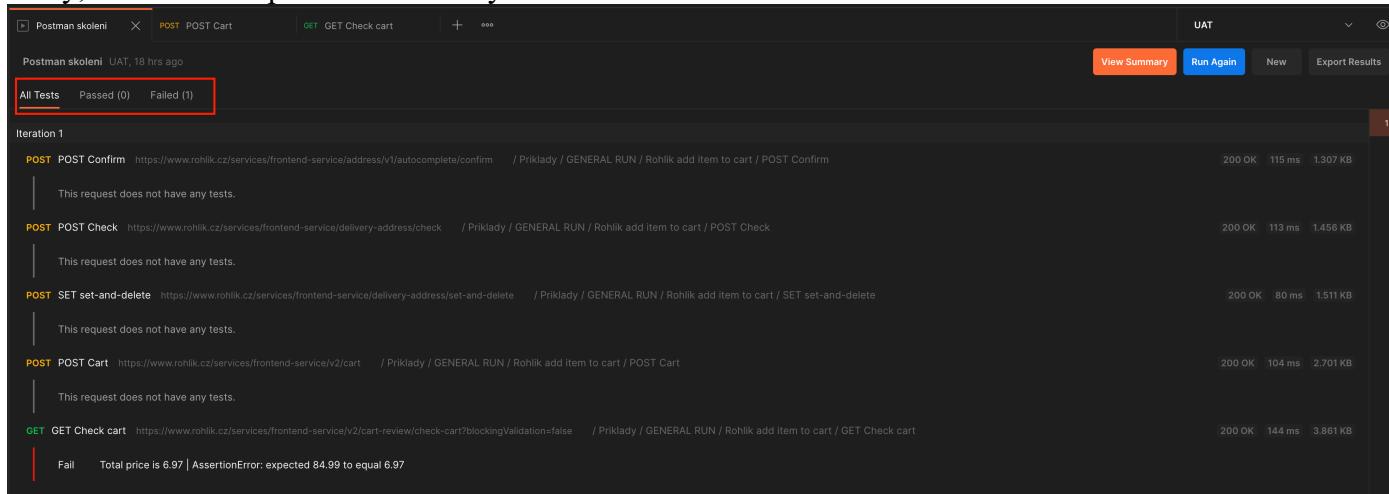
- Exekuované requesty a výsledky jejich testů



The screenshot shows the Postman interface with the following details:

- Iteration 1**
- POST POST Confirm**: https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm / Priklady / GENERAL RUN / Rohlik add item to cart / POST Confirm  
This request does not have any tests.
- POST POST Check**: https://www.rohlik.cz/services/frontend-service/delivery-address/check / Priklady / GENERAL RUN / Rohlik add item to cart / POST Check  
This request does not have any tests.
- POST SET set-and-delete**: https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete / Priklady / GENERAL RUN / Rohlik add item to cart / SET set-and-delete  
This request does not have any tests.
- POST POST Cart**: https://www.rohlik.cz/services/frontend-service/v2/cart / Priklady / GENERAL RUN / Rohlik add item to cart / POST Cart  
This request does not have any tests.
- GET GET Check cart**: https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false / Priklady / GENERAL RUN / Rohlik add item to cart / GET Check cart  
Fail Total price is 6.97 | Assertion: expected 84.99 to equal 6.97

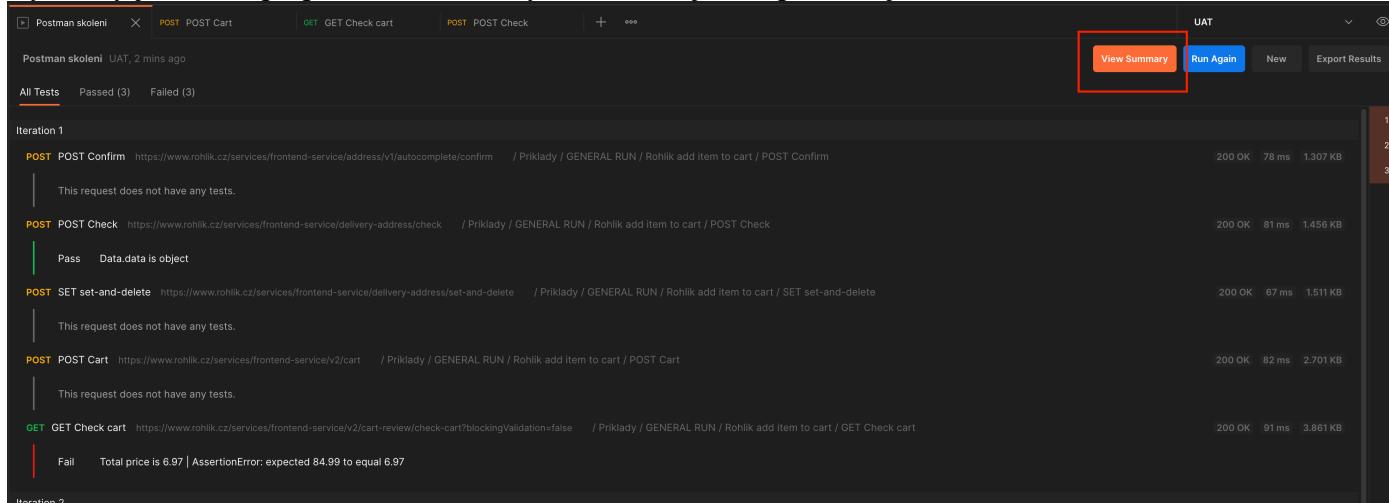
- Filtry, které zobrazí passed/failed testy



The screenshot shows the Postman interface with the following details:

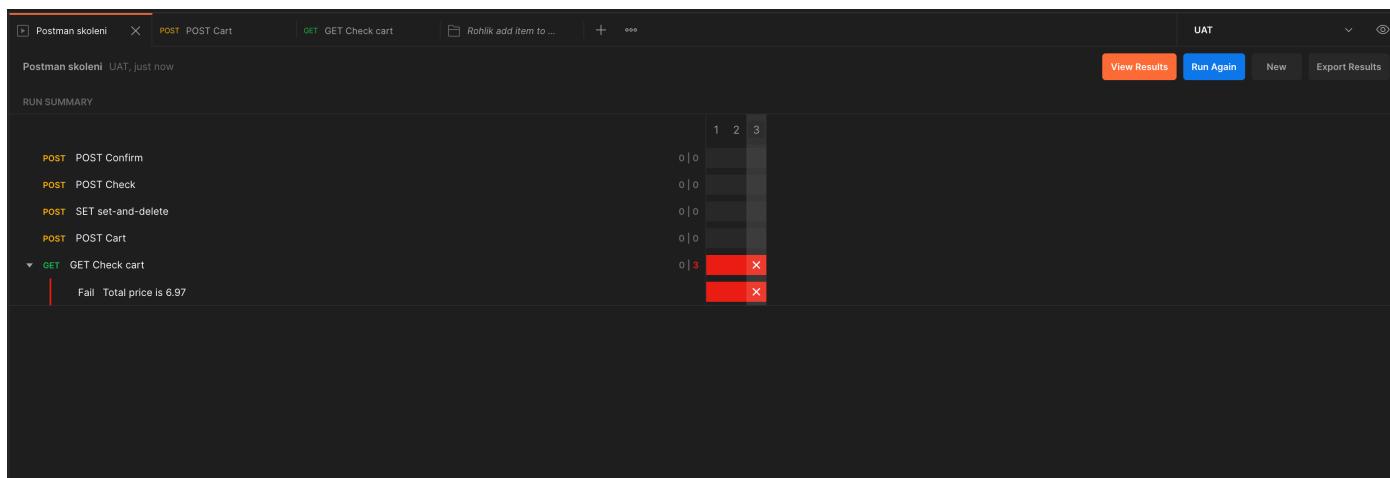
- All Tests** Passed (0) Failed (1)
- Iteration 1**
- POST POST Confirm**: https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm / Priklady / GENERAL RUN / Rohlik add item to cart / POST Confirm  
This request does not have any tests.
- POST POST Check**: https://www.rohlik.cz/services/frontend-service/delivery-address/check / Priklady / GENERAL RUN / Rohlik add item to cart / POST Check  
This request does not have any tests.
- POST SET set-and-delete**: https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete / Priklady / GENERAL RUN / Rohlik add item to cart / SET set-and-delete  
This request does not have any tests.
- POST POST Cart**: https://www.rohlik.cz/services/frontend-service/v2/cart / Priklady / GENERAL RUN / Rohlik add item to cart / POST Cart  
This request does not have any tests.
- GET GET Check cart**: https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false / Priklady / GENERAL RUN / Rohlik add item to cart / GET Check cart  
Fail Total price is 6.97 | Assertion: expected 84.99 to equal 6.97

- Výsledky je možné přepnout do summary modu, kde jsou lépe vidět jednotlivé iterace

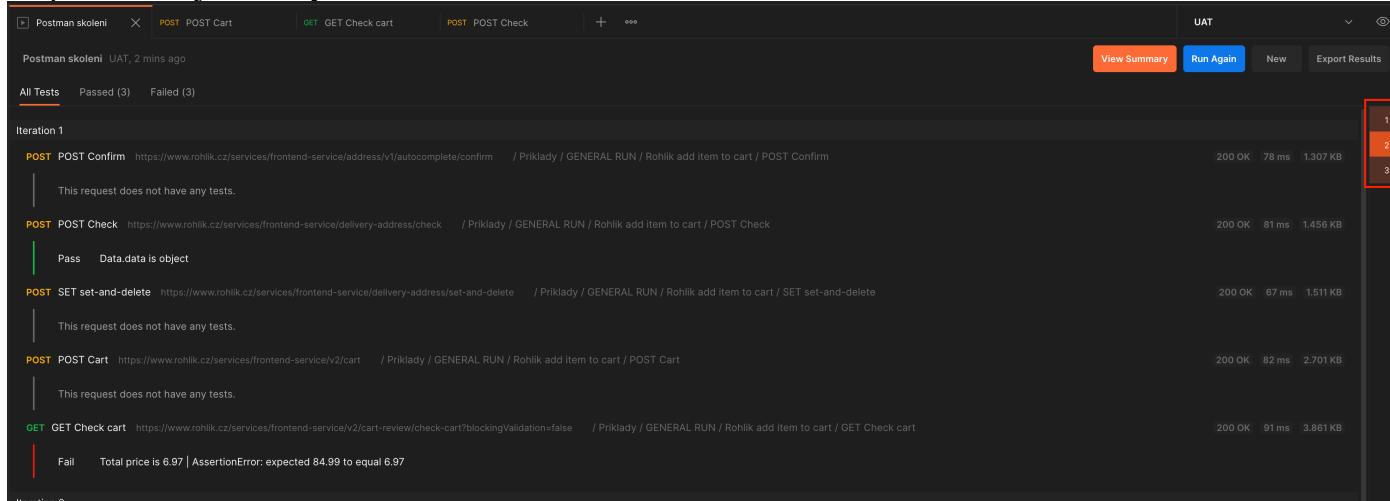


The screenshot shows the Postman interface with the following details:

- View Summary** (button highlighted with a red box)
- All Tests** Passed (3) Failed (3)
- Iteration 1**
- POST POST Confirm**: https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm / Priklady / GENERAL RUN / Rohlik add item to cart / POST Confirm  
This request does not have any tests.
- POST POST Check**: https://www.rohlik.cz/services/frontend-service/delivery-address/check / Priklady / GENERAL RUN / Rohlik add item to cart / POST Check  
Pass Data data is object
- POST SET set-and-delete**: https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete / Priklady / GENERAL RUN / Rohlik add item to cart / SET set-and-delete  
This request does not have any tests.
- POST POST Cart**: https://www.rohlik.cz/services/frontend-service/v2/cart / Priklady / GENERAL RUN / Rohlik add item to cart / POST Cart  
This request does not have any tests.
- GET GET Check cart**: https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false / Priklady / GENERAL RUN / Rohlik add item to cart / GET Check cart  
Fail Total price is 6.97 | Assertion: expected 84.99 to equal 6.97
- Iterations 2**



- Přepínáč mezi jednotlivými iteracemi



## DATA DRIVEN TESTY

Postman Collection Runner umožňuje předávat data pomocí externího JSON souboru. To nám umožňuje vytvářet kombinace requestů, které bychom jinak museli duplikovat či ručně měnit v kolekcích.

### Obecná syntaxe data JSON:

```
[  
  {  
    path: "post",  
    value: "1"  
  },  
  {  
    path: "post",  
    value: "2"  
  },  
  {  
    path: "post",  
    value: "3"  
  },  
  {  
    path: "post",  
    value: "4"  
  }]
```

V datovém JSON zmíněném výše jsou 4 objekty v array, to znamená, že budou vytvořeny 4 iterace testů. Každý objekt má 2 elementy:

- field1
- field2

Tyto elementy se uloží do proměnných:

- {{field1}}
- {{field2}}

Následně je můžeme využít jako standardní proměnné v postman. Tyto proměnné se nikam neukládají a z paměti se po doběhnutí testu mažou.

---

## UŽITÍ V PRAXI

Naším cílem je sestavit přidání 2 položek do košíku na rohlik.cz

Uděláme 2 iterace, proto budeme potřebovat 4 produkty.

Vytvoříme datový JSON s vybranými produkty (nebo ho můžete stáhnout [zde](#))

```
[  
  {  
    "firstItemId": 1407899,  
    "secondItemId": 1315797  
  },  
  {  
    "firstItemId": 1377729,  
    "secondItemId": 1324587  
  }  
]
```

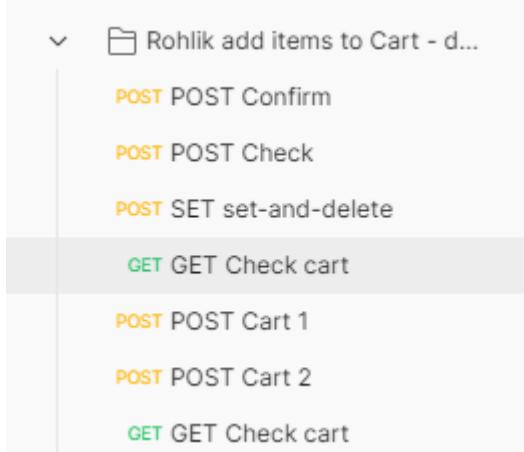
**Nová složka: Skoleni/Collection runner/Data files (vytvořte ji zkopirováním a přejmenováním složky General)**

Sestavíme requesty. Využijeme cally z předchozího příkladu. Seskládáme je následovně:

1. POST rohlik.cz autocomplete confirm
2. POST rohlik.cz delivery address check
3. POST rohlik.cz delivery address set and delete
4. POST rohlik.cz cart 1
5. POST rohlik.cz cart 2
6. GET rohlik.cz cart review

Do testů přidáme kontrolu na status 200.

Struktura složky:



The screenshot shows a Postman collection structure. At the top level, there is a folder named "Rohlik add items to Cart - d...". Inside this folder, there are several requests listed vertically:

- POST POST Confirm (Status: 200)
- POST POST Check (Status: 200)
- POST SET set-and-delete (Status: 200)
- GET GET Check cart** (Status: 200) - This request is highlighted with a grey background.
- POST POST Cart 1 (Status: 200)
- POST POST Cart 2 (Status: 200)
- GET GET Check cart (Status: 200)

---

## NASTAVENÍ REQUESTŮ

## 1. POST ROHLIK.CZ AUTOCOMPLETE CONFIRM

---

### Tests

```
var data = pm.response.json();

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.variables.set("rohlikAddressId", data.data.id)
```

## 2. POST ROHLIK.CZ DELIVERY ADDRESS CHECK

---

### Tests

```
var data = pm.response.json();

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.variables.set("rohlikstoreId", data.data.availableStores[0].storeId)
pm.variables.set("rohlikFullAddress", data.data.address.fullAddress)
pm.variables.set("rohlikStreet", data.data.address.street)
pm.variables.set("rohlikPsc", data.data.address.postalCode)
pm.variables.set("rohlikCity", data.data.address.city)

pm.test("Data.data is object", function () {
    pm.expect(data.data).to.be.an('object');
});
```

## 3. POST ROHLIK.CZ DELIVERY ADDRESS SET AND DELETE

---

### Tests

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

## 4. POST ROHLIK.CZ CART 1

---

### Body

```
{
    "productId": {{firstItemId}},
    "quantity": 1,
    "source": ": ProductCategory:300102000",
    "actionId": null,
    "recipeId": null
}
```

### Tests

```
var data = pm.response.json();

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Data.data.totalprice is a number", () => {
    pm.expect(data.data.totalPrice).to.be.a('number');
    pm.variables.set("rohlikPriceSum", data.data.totalPrice);
})
```

## 5. POST ROHLIK.CZ CART 2

---

## Body

```
{
  "productId": {{secondItemId}},
  "quantity": 1,
  "source": ": ProductCategory:300102000",
  "actionId": null,
  "recipeId": null
}
```

## Tests

```
var data = pm.response.json();

pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Data.data.totalprice is a number", () => {
  pm.expect(data.data.totalPrice).to.be.a('number');
  pm.variables.set("rohlikPriceSum", data.data.totalPrice);
})
```

## 6. GET ROHLIK.CZ CART REVIEW

---

### Tests

```
var data = pm.response.json();

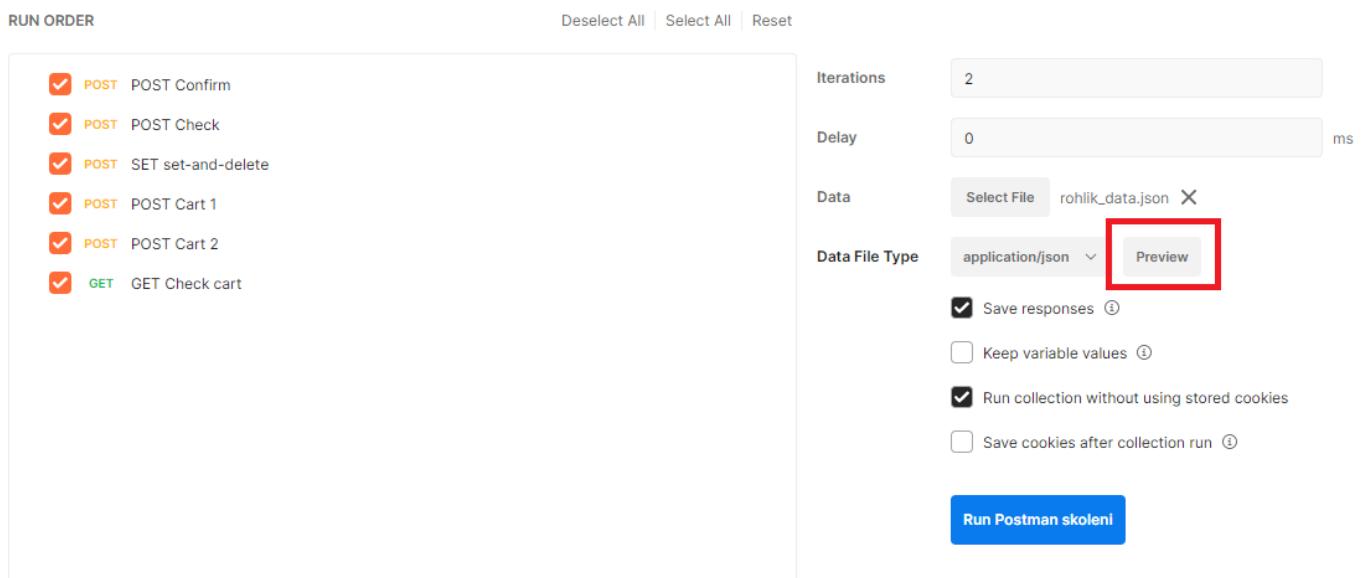
var rohlikPriceSum = pm.variables.get("rohlikPriceSum")

pm.test("Total price is " + rohlikPriceSum, () => {
  pm.expect(data.data.totalPrice).to.equal(rohlikPriceSum);
});
```

## RUN SLOŽKY

---

Vytvoříme nový runner, vložíme do něj předpřipravený JSON datový soubor. Zkontrolujeme hodnoty kliknutím na tlačítko preview:



The screenshot shows the Postman 'Run Order' interface. On the left, a list of requests is shown with checkboxes next to each one. The requests are:

- POST POST Confirm
- POST POST Check
- POST SET set-and-delete
- POST POST Cart 1
- POST POST Cart 2
- GET GET Check cart

On the right, there are several configuration options:

- Iterations:** A text input field containing '2'.
- Delay:** A text input field containing '0' followed by 'ms'.
- Data:** A 'Select File' button with 'rohlik\_data.json' selected, and a 'Preview' button which is highlighted with a red box.
- Data File Type:** A dropdown menu set to 'application/json'.
- Run Options:**
  - Save responses
  - Keep variable values
  - Run collection without using stored cookies
  - Save cookies after collection run
- Run Button:** A blue button labeled 'Run Postman skolení'.

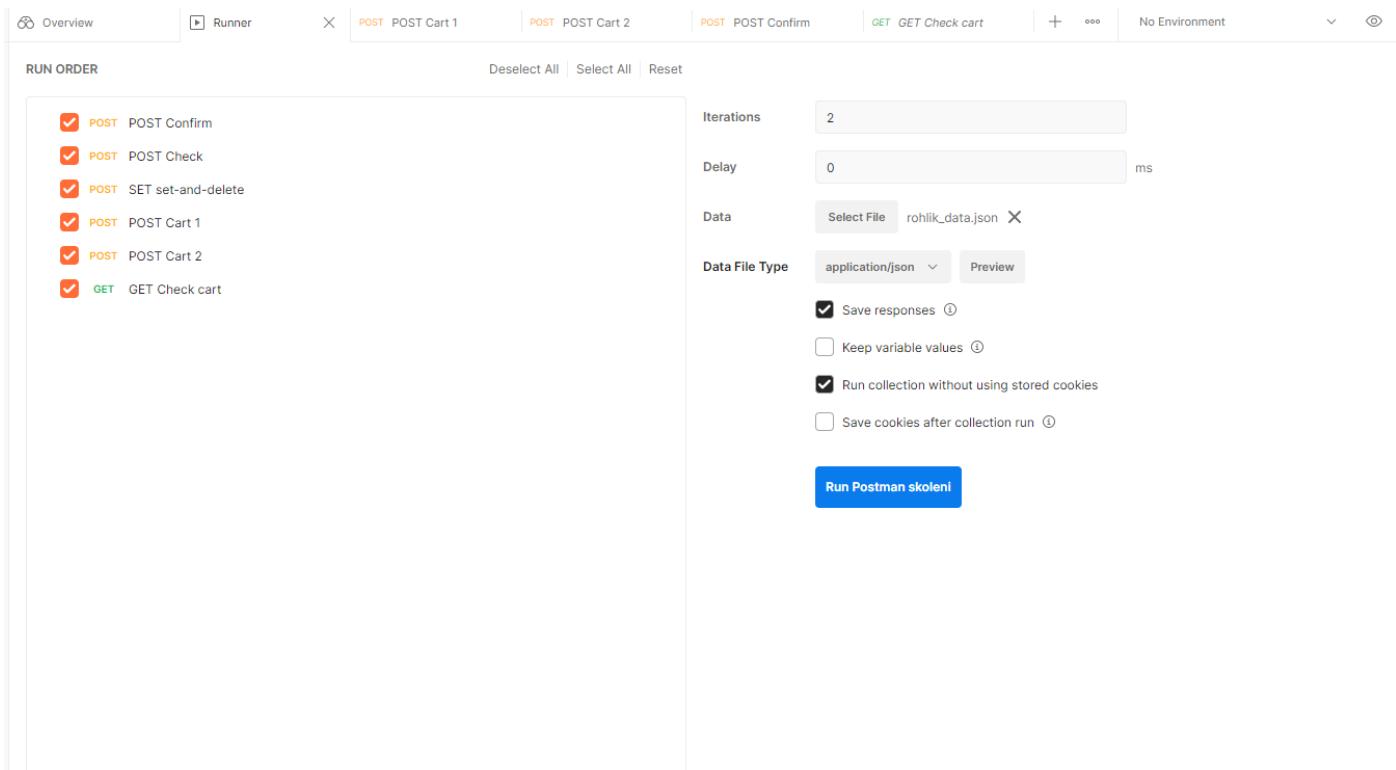
## PREVIEW DATA

X

| Iteration | firstItemId | secondItemId |
|-----------|-------------|--------------|
| 1         | 1286477     | 1350165      |
| 2         | 1377729     | 1324587      |

V Runneru nastavíme následující parametry:

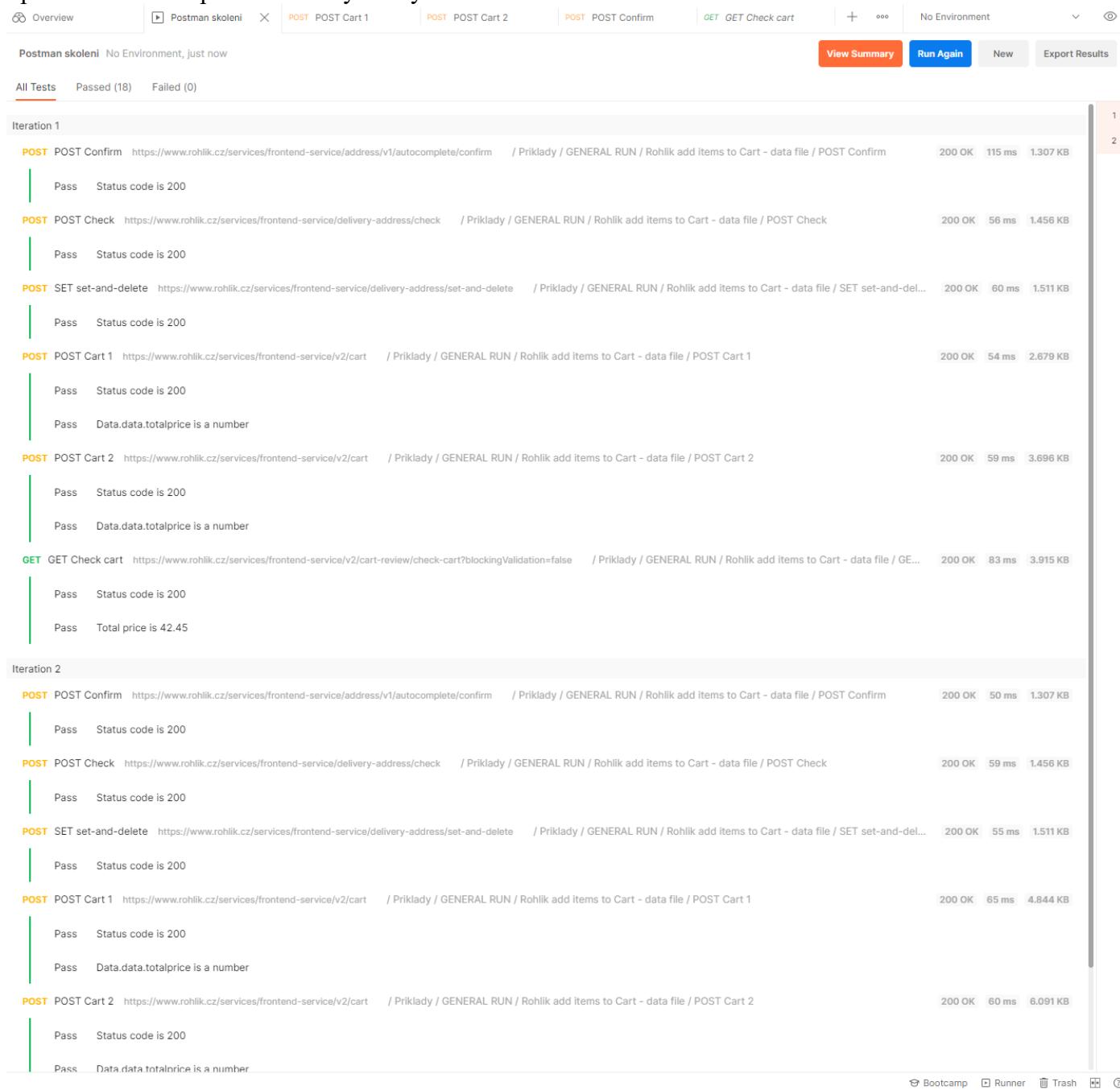
- Save response
- Run collection without using stored cookies



The screenshot shows the Postman Runner interface with the following configuration:

- RUN ORDER:** A list of selected requests: POST POST Confirm, POST POST Check, POST SET set-and-delete, POST POST Cart 1, POST POST Cart 2, and GET GET Check cart.
- Iterations:** Set to 2.
- Delay:** Set to 0 ms.
- Data:** A file named "rohlik\_data.json" is selected, and the Data File Type is set to application/json.
- Run Options:**
  - Save responses
  - Keep variable values
  - Run collection without using stored cookies
  - Save cookies after collection run
- Buttons:** A blue button labeled "Run Postman skoleni".

## Spustíme runner a počkáme na výsledky



The screenshot shows the Postman interface with the following details:

- Collection:** Postman skoleni
- Tests:**
  - POST POST Confirm**: Status code is 200 (Pass)
  - POST POST Check**: Status code is 200 (Pass)
  - POST SET set-and-delete**: Status code is 200 (Pass)
  - POST POST Cart 1**: Status code is 200 (Pass), Data.data.totalprice is a number
  - POST POST Cart 2**: Status code is 200 (Pass), Data.data.totalprice is a number
  - GET GET Check cart**: Status code is 200 (Pass), Total price is 42.45
- Iterations:**
  - Iteration 1** (1 test): POST POST Confirm, Status code is 200 (Pass)
  - Iteration 2** (1 test): GET GET Check cart, Status code is 200 (Pass), Total price is 42.45
- Environment:** No Environment
- Buttons:** View Summary, Run Again, New, Export Results

## SDÍLENÍ DAT V TÝMU

Existuje několik možných způsobů sdílení dat v Postman.

Jsou to:

- Verzování kolekcí
- Workspace

## VERZOVÁNÍ KOLEKcí

Postman umožňuje integraci do GitHub. Jelikož se však jedná o placenou funkcionalitu, ukazovat si ji ve školení nebudeme.

Návod jak na to, naleznete [zde](#).

## WORKSPACE

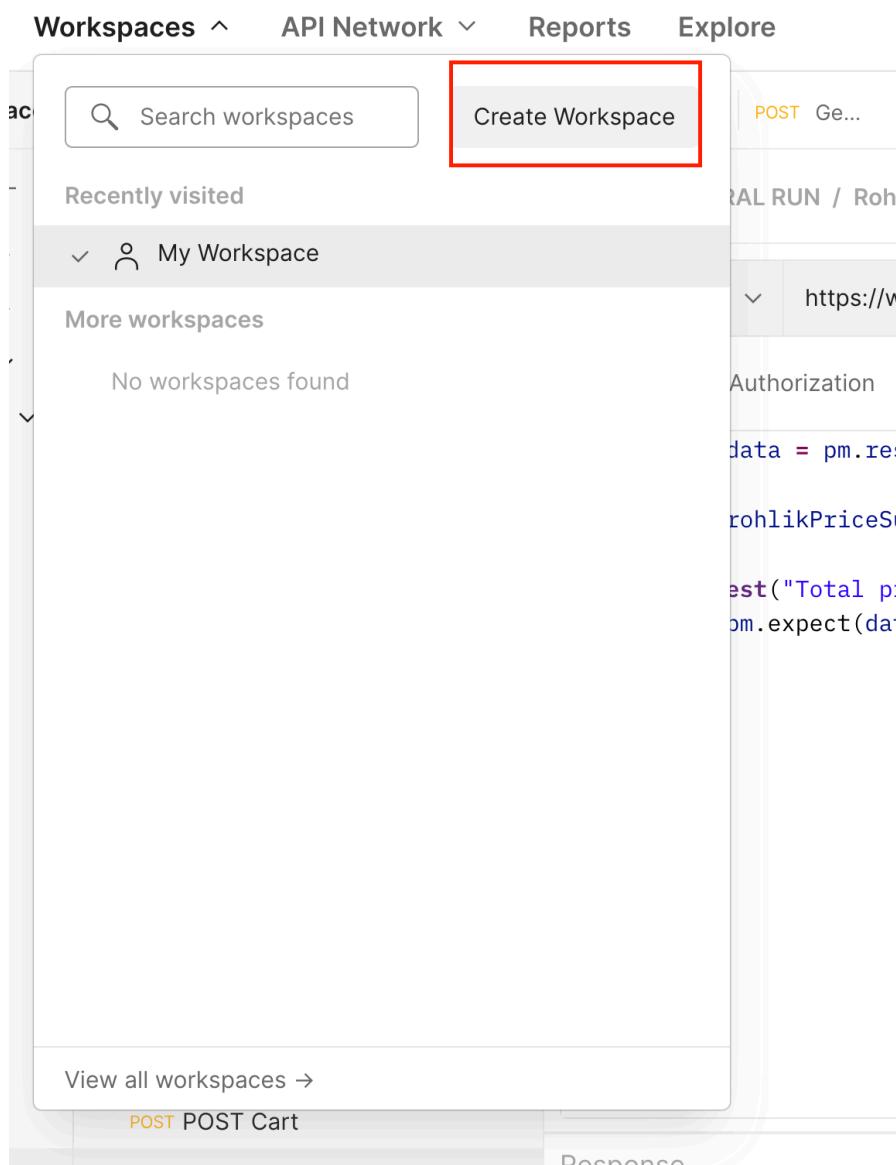
Workspace je funkcionalita Postman, která umožňuje sdílet data mezi více uživateli nebo zálohovat data uživatele. Tato funkcionalita je zdarma v limitu 3 uživatelů.

Následně je již zpoplatněna dle zvoleného Postman Plánu. Více informací o tom, jak Workspaces fungují naleznete [zde](#).

Výhodou workspace je to, že nemusíte složitě exportovat a sdílet kolekce a proměnné. Postman si toto deluguje sám.

## VYTVOŘENÍ WORKSPACE

Klikněte na workspaces a následně create Workspace v Postman



The screenshot shows the Postman application interface. At the top, there is a navigation bar with tabs: 'Workspaces ^', 'API Network ^', 'Reports', and 'Explore'. Below the navigation bar, there is a search bar labeled 'Search workspaces' and a button labeled 'Create Workspace' which is highlighted with a red box. On the left side, there is a sidebar titled 'Recently visited' which shows a single entry: 'My Workspace'. Below this, there is a section titled 'More workspaces' with the message 'No workspaces found'. At the bottom of the sidebar, there is a link 'View all workspaces →'. To the right of the sidebar, there is a main content area showing a POST request to 'https://...'. The request body contains JSON data: 'data = pm.replace("\$totalPrice", pm.response.json().rohlikPriceSum)'. The response tab is visible at the bottom.

Vyplňte údaje o workspace a stiskněte "Create Workspace and Team"

## Create workspace

Name

### Summary

Add a brief summary about this workspace.

### Visibility

Determines who can access this workspace.

 Personal

Only you can access

 Private

Only invited team members can access

 Team

All team members can access

 Public

Everyone can view

 A team of your own will be created since you don't have one now.

[Create Workspace and Team](#)[Cancel](#)

Zobrazí se vám stránka "Team workspace created". Klikněte "Go to Workspace"



### Team workspace created

Customize your team's profile to make it easy for your future team members to find.

[Edit Team profile](#)[Go to Workspace](#)

V Postmanu se Vám zobrazí nový Workspace

## VYTВAŘENÍ SDÍLENÝCH REQUESTŮ

Jakmile nový request uložíte ve workspace, uloží se také online do workspace.

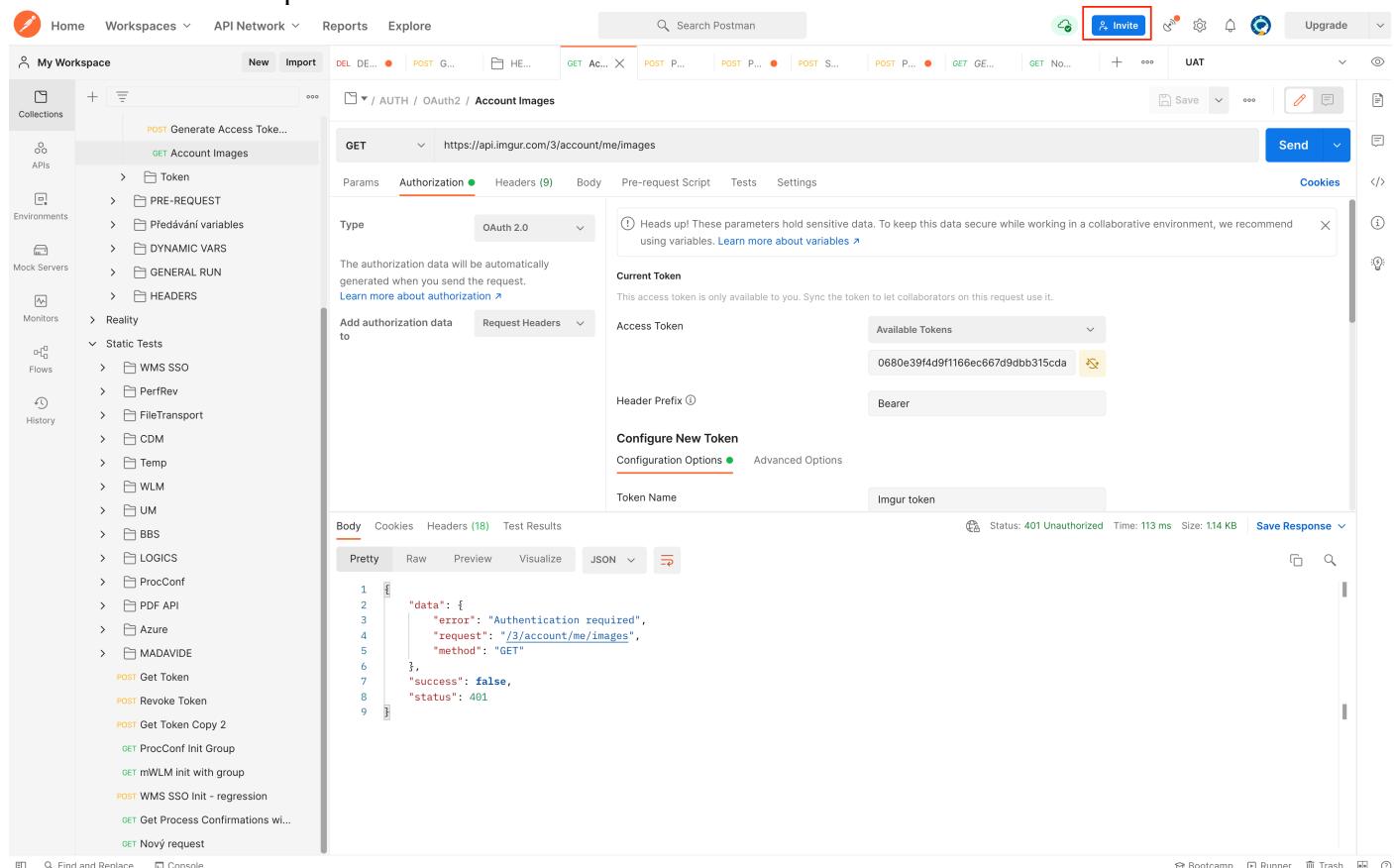
### PРÍKLAD

Vyzkoušíme si vytvořit request a následně ho zkontrolujeme na webové aplikaci Postmana.

1. Vytvořte prázdný request v kolekci, pojmenujte ho "Workspace test"
2. Vyplňte URL: test
3. Uložte request
4. Otevřete webový postman na adresu: <https://www.postman.com>
5. Najděte vytvořený request
6. Změňte URL na webu
7. Zkontrolujte změny v aplikaci

## SPRÁVA WORKSPACE, POZVÁNÍ NOVÝCH ČLENŮ

Klikneme na invite v pravé horní části Postman.



The screenshot shows the Postman interface with the following details:

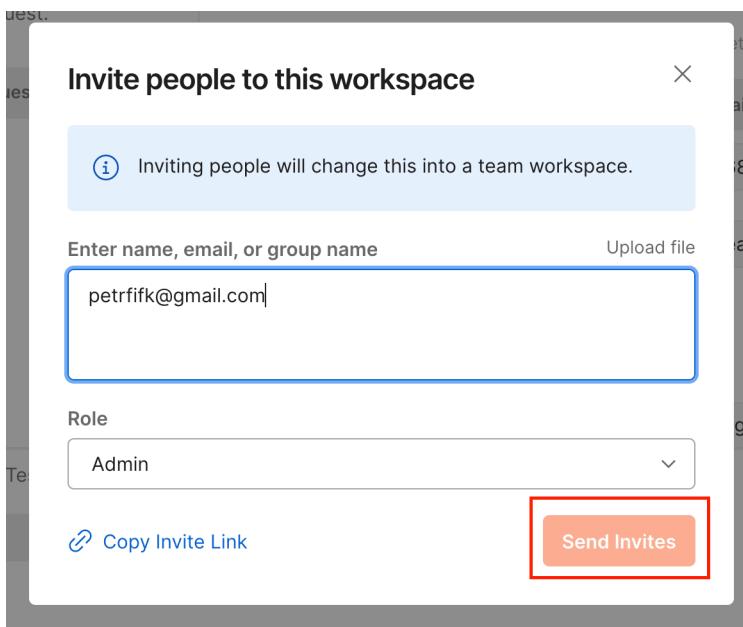
- Left Sidebar:** My Workspace, Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Request Editor:**
  - Method: GET
  - URL: https://api.imgur.com/3/account/me/images
  - Authorization tab is selected, showing OAuth 2.0 configuration. The 'Type' dropdown is set to OAuth 2.0. The 'Current Token' section shows a token: 0680e39f4d9f1166ec667d9dbb315cda. The 'Header Prefix' is set to Bearer.
  - Body tab shows a JSON response:

```

1
2   "data": {
3     "error": "Authentication required",
4     "request": "/3/account/me/images",
5     "method": "GET"
6   },
7   "success": false,
8   "status": 401
9

```
- Top Bar:** Home, Workspaces, API Network, Reports, Explore, Search Postman, Invite (highlighted with a red box), Save, Cookies, Upgrade.

Vyplníme údaje o člověku a stiskneme "Send Invites"



## CHANGELOG

V changelogu vidíte změny provedené jednotlivými uživateli ve Workspace.

## KOLEKCE

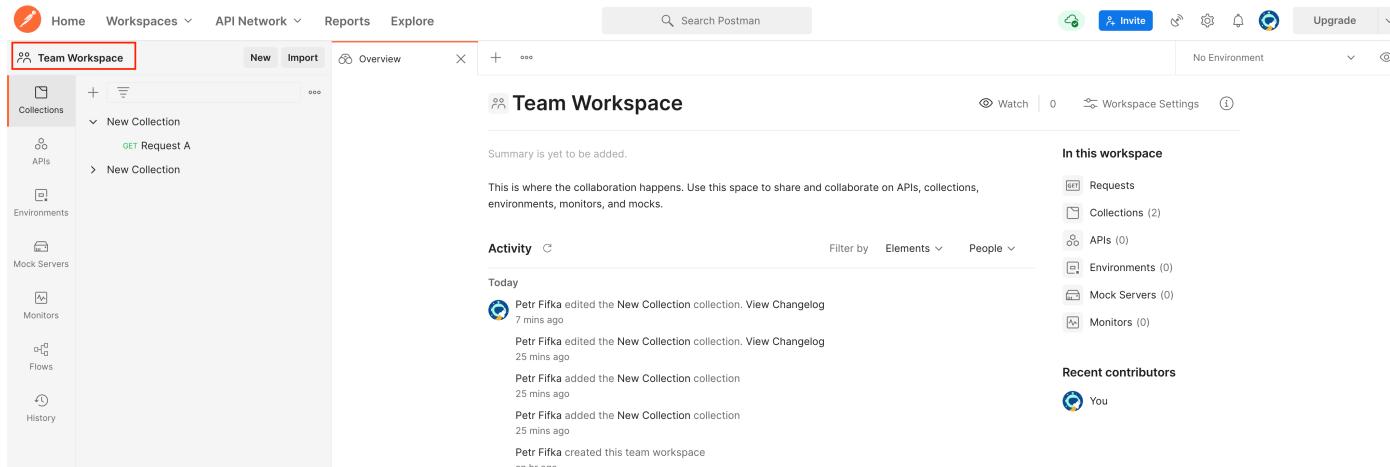
1. Otevřete detail kolekce
2. Klikněte na ikonku historie

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like 'Team Workspace', 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. The main area shows a 'New Collection' overview with tabs for 'Overview', 'New Collection', 'GET Request A', 'Share', 'Run', and others. The 'Authorization' tab is currently active. On the right, there's a 'Changelog' section titled 'Today' which lists several recent actions:

- just now You modified the request Request A
- 18 mins ago You modified this collection
- 18 mins ago You added the request Request A
- 18 mins ago You created this collection

## WORKSPACE

1. klikněte na název workspace



The screenshot shows the Postman interface with the 'Team Workspace' tab selected in the top navigation bar. The left sidebar contains links for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays the 'Team Workspace' overview, which includes a summary message, a brief description of the workspace's purpose, and an activity log. The activity log shows several entries from a user named Petr Fifka, including edits to a 'New Collection' collection at different times ago and the creation of the workspace itself.

2. Otevře se overview a v něm vidíte záložku Activity, kde vidíte provedené změny v celém Workspace

## NEWMAN

Newman je nástroj pro spouštění Postman requestů z konzole. Je vhodný například do CI/CD pipelines (například Jenkins)

Pro použití Newmana je potřeba mít nainstalovaný [Node.js](#). Dokumentaci, jak používat Newmana naleznete [zde](#).

## INSTALACE

Instalace Newmana přes Node.js

```
npm install -g newman
```

## SPOUŠTĚNÍ TESTŮ

### ZÁKLADNÍ SPUŠTĚNÍ TESTU

Vyexportujte stávající kolekci.

Syntaxe spouštění newmana:

```
newman run $PATH
```

Výběr testu, zadání do cmd:

```
newman run principal.tests.json
```

Newman projede všechny requesty, které jsou v dané kolekci. Výsledek vypadá v konzoli takto:

```

Postman skoleni

□ Příklady / REQ Params
└ GET Regiojet routes - URL Params with params
    GET https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocationId=10202003&departureDate=2021-10-26 [400 400, 628B, 172ms]

└ GET Regiojet Routes - URL Params without params
    GET https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&departureDate=2021-10-27&toLocationId=2147875000&fromLocationType=CITY&fromLocationId=10202003 [400 400, 628B, 106ms]

└ GET Regres.in User - With URI
    GET https://regres.in/api/users/2 [200 OK, 1.22kB, 103ms]

└ GET Regres.in User - without URI
    GET https://regres.in/api/users/2 [200 OK, 1.23kB, 36ms]

□ Příklady / HTTP Requests
└ GET Users - HTTP Requests
    GET https://regres.in/api/users [200 OK, 1.94kB, 36ms]

└ POST User - HTTP Requests
    POST https://regres.in/api/users [201 Created, 946B, 86ms]

└ PUT User - HTTP Requests Copy
    PUT https://regres.in/api/users/2 [200 OK, 975B, 107ms]

└ PUT User - HTTP Requests Copy 2
    PATCH https://regres.in/api/users/22 [200 OK, 953B, 84ms]

└ DELETE User - HTTP Request
    DELETE https://regres.in/api/users/2 [204 No Content, 822B, 79ms]

□ Příklady / JSON
└ POST Rohlík.cz - Add item
    POST https://www.rohlik.cz/services/frontend-service/v2/cart [400 Bad Request, 1.32kB, 115ms]

□ Příklady / AUTH / API KEY
└ PUT change booking - without AUTH
    PUT https://restful-booker.herokuapp.com/booking/1 [403 Forbidden, 252B, 508ms]

└ POST - booking - create token
    POST https://restful-booker.herokuapp.com/auth [200 OK, 271B, 113ms]

└ PUT change booking - with AUTH
    PUT https://restful-booker.herokuapp.com/booking/1 [403 Forbidden, 252B, 108ms]

└ Příklady / AUTH / OAuth2
    └ Generate Access Token without secret
        POST https://api.imgur.com/oauth2/token [400 Bad Request, 966B, 234ms]
        1. Returns 40 char hex access token
        2. Returns 40 char hex refresh token

Attempting to set next request to Account Base



|                    | executed | failed |
|--------------------|----------|--------|
| iterations         | 1        | 0      |
| requests           | 14       | 0      |
| test-scripts       | 1        | 0      |
| prerequest-scripts | 1        | 0      |
| assertions         | 2        | 2      |



total run duration: 3.1s
total data received: 2.41kB (approx)
average response time: 134ms [min: 36ms, max: 508ms, s.d.: 114ms]

# failure                               detail
1. AssertionError                         Returns 40 char hex access token
                                         expected false to be truthy
                                         at assertion:0 in test-script
                                         inside "Příklady / AUTH / OAuth2 / Generate Access Token without secret"

```

Můžeme spustit jednotlivé složky.

#### Syntaxe:

```
newman run $PATH --folder $FOLDER_PATH
```

#### Příklad:

```
newman run $postman_skoleni.json --folder "HTTP Requests"
```

Použít také můžeme proměnné. Je nutné je ale mít exportované z Postman. Syntaxe:

#### GLOBALS

```
newman run $PATH -g $GLOBALS_PATH
```

#### Příklad:

```
newman run $postman_skoleni.json -g workspace.postman_globals.json
```

#### ENVIRONMENT

```
newman run $PATH -e $ENV_PATH
```

#### Příklad:

```
newman run postman_skoleni.json -e UAT.postman_environment.json
```

Spouštění pomocí iteration dat. Podobně jako v Collection Runneru umí Newman využít externí data.  
Syntaxe:

```
newman run $PATH -d $DATA_PATH
```

### Příklad

```
newman run postman_skoleni.json -d rohlik_data.json
```

## REPORTING

Newman reportuje v základu pouze do konzole. Pro detailnější logy můžeme využít reportera.  
Instalace:

```
npm install -g newman-reporter-htmlextra
```

### Syntaxe

```
newman run $PATH --reporters cli,json,htmlextra
```

### Příklad

```
newman run "Postman_skoleni.postman_collection.json" --reporters cli,json,htmlextra
```

Výsledek vypadá následovně:

Light 
[Summary](#)
[Total Requests 35](#)
[Failed Tests 9](#)
[Skipped Tests 0](#)

# Newman Run Dashboard

Friday, 05 November 2021 12:00:39

**TOTAL ITERATIONS**
**1**
**TOTAL ASSERTIONS**
**20**
**TOTAL FAILED TESTS**
**9**
**TOTAL SKIPPED TESTS**
**0**

## FILE INFORMATION

 **Collection:** Postman skolení

## TIMINGS AND DATA

 **Total run duration:** 9.1s

 **Total data received:** 22.36KB

 **Average response time:** 174ms

| SUMMARY ITEM       | TOTAL | FAILED |
|--------------------|-------|--------|
| Requests           | 35    | 0      |
| Prerequest Scripts | 4     | 0      |
| Test Scripts       | 19    | 0      |
| Assertions         | 20    | 9      |
| Skipped Tests      | 0     | -      |

## BATCH FILE

Spouštění newman lze udělat i přes batch file. Máme potom možnost využít skriptovacích nástrojů, jako jsou například proměnné.

Proměnné v batch:

```
set varName = value
```

Pokud potřebujeme aktuální timestamp, zapíšeme ji v batch filu takto:

```
set Timestamp = %date:~4,2%%date:~7,2%%date:~10,4%_%time:~0,2%%time:~3,2%%time:~6,2%
echo %Timestamp%
```

Příklad Batche:

```
set envVar = UAT.postman_environment.json
set tests = Tests.postman_collection.json

set Timestamp = %date:~4,2%%date:~7,2%%date:~10,4% %time:~0,2%%time:~3,2%%time:~6,2%

newman run %tests% --folder PerfRev -e %envVar% -d Data\PerfRevCreateReview.json --reporters cli,json,htmlextra --
reporter-summary-json-export PerfRev_%Timestamp%
```

## KOMPLEXNÍ PŘÍKLAD

Připravíme náš rohlík scénář v Newman pomocí batch file.

Co potřebujeme?

Vyexportovat z Postman:

- Vytvořenou kolekci
- Prostředí PROD

Připravený datový file z [Collection Runner](#) kapitoly.

Následně seskládáme batch file:

```
set env = PROD.postman_environment.json
set tests = "Postman skoleni.postman_collection.json"
set data = rohlik_data.json

newman run %tests% --folder "Rohlik add items to Cart - data file" -e %env% -d %data% --reporters cli,json,htmlextra
```

Přes CMD následně file spustíme batch file, co jsme vytvořili:

```
C:\Users\tiash\OneDrive - Petr Fifka\Projekty\Skoleni\Postman>newmanComplexRun.bat
C:\Users\tiash\OneDrive - Petr Fifka\Projekty\Skoleni\Postman>set env=PROD.postman_environment.json
C:\Users\tiash\OneDrive - Petr Fifka\Projekty\Skoleni\Postman>set tests="Postman skoleni.postman_collection.json"
C:\Users\tiash\OneDrive - Petr Fifka\Projekty\Skoleni\Postman>set data="rohlik_data.json"
C:\Users\tiash\OneDrive - Petr Fifka\Projekty\Skoleni\Postman>newman run "Postman skoleni.postman_collection.json" --folder "Rohlik add items to Cart - data file" -e PROD.postman_environment.json -d rohlik_data.json --reporters cli,json,htmlextra
newman
Postman skoleni
Using htmlextra version 1.22.3
Iteration 1/2
  □ Priklady / GENERAL RUN / Rohlik add items to Cart - data file
    L POST Confirm
      POST https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm [200 OK, 1.31kB, 17ms]
        ✓ Status code is 200
    L POST Check
      POST https://www.rohlik.cz/services/frontend-service/delivery-address/check [200 OK, 1.46kB, 51ms]
        ✓ Status code is 200
    L SET set-and-delete
      POST https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete [200 OK, 1.51kB, 45ms]
        ✓ Status code is 200
    L POST Cart 1
      POST https://www.rohlik.cz/services/frontend-service/v2/cart [200 OK, 2.68kB, 66ms]
        ✓ Status code is 200
        ✓ Data.data.totalprice is a number
    L POST Cart 2
      POST https://www.rohlik.cz/services/frontend-service/v2/cart [200 OK, 3.7kB, 58ms]
        ✓ Status code is 200
        ✓ Data.data.totalprice is a number
    L GET Check cart
      GET https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false [200 OK, 3.92kB, 115ms]
        ✓ Status code is 200
        ✓ Total price is 42.49
  Iteration 2/2
    L POST Confirm
      POST https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm [200 OK, 1.31kB, 45ms]
        ✓ Status code is 200
    L POST Check
      POST https://www.rohlik.cz/services/frontend-service/delivery-address/check [200 OK, 1.46kB, 62ms]
        ✓ Status code is 200
    L SET set-and-delete
      POST https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete [200 OK, 1.51kB, 42ms]
        ✓ Status code is 200
    L POST Cart 1
      POST https://www.rohlik.cz/services/frontend-service/v2/cart [200 OK, 4.84kB, 49ms]
        ✓ Status code is 200
        ✓ Data.data.totalprice is a number
    L POST Cart 2
      POST https://www.rohlik.cz/services/frontend-service/v2/cart [200 OK, 6.09kB, 66ms]
        ✓ Status code is 200
        ✓ Data.data.totalprice is a number
    L GET Check cart
      GET https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false [200 OK, 6.44kB, 74ms]
```

## MOCK SERVERS V POSTMAN

Postman umožňuje vytvoření Mock serveru, který slouží od toho, aby nám vytvořil API, která nám bude vracet požadované výsledky. Mock API se běžně používá na projektech, kde ještě nemáme hotovou integraci, ale potřebujeme vyzkoušet, jak se chová již vyvinutá aplikace, když dostane předpokládanou odpověď.

Dokumentaci Mock serverů v Postman najeznete [zde](#).

V rámci školení vytvoříme jednoduchý Mock server.

### PŘÍKLAD

**Vytvořte novou složku:** Skoleni/Mock

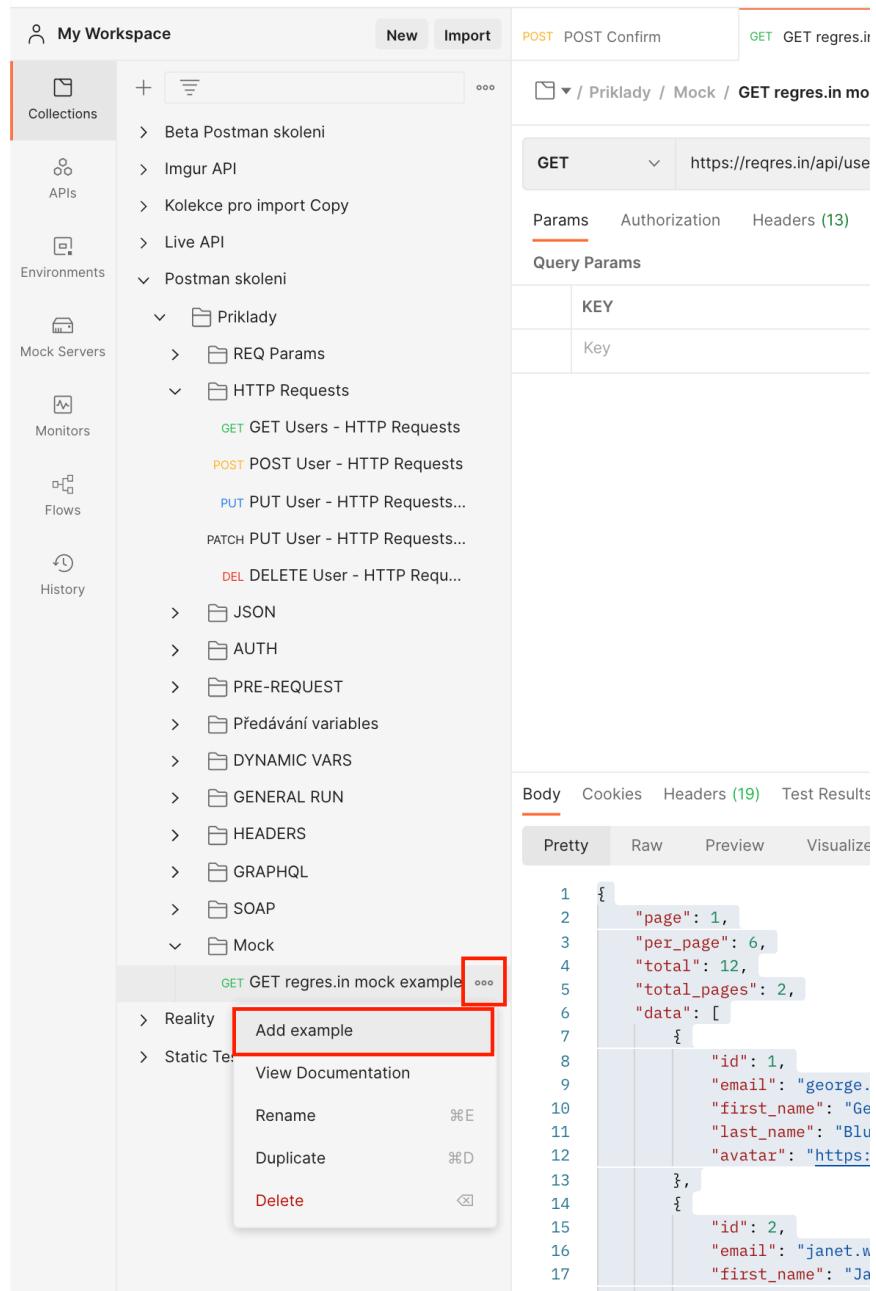
**Název requestu:** GET regres mock example

### cURL:

```
curl --location --request GET 'https://reqres.in/api/users' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0' \
--header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54 Safari/537.36' \
--header 'sec-ch-ua-platform: "Windows"' \
--header 'Content-Type: application/json'
```

Vytvořte nový example v requestu:

1. Vyberte request v levém menu, klikněte na více menu a následně vyberte Add Example



The screenshot shows the Postman application interface. On the left, there is a sidebar with various sections: Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. Under the Collections section, there is a tree view of collections and environments. A context menu is open over a specific item in the tree, with the "Add example" option highlighted.

The main workspace on the right shows a "POST Confirm" tab and a "GET regres.in mo" request. The request details show a GET method, URL https://reqres.in/api/use, and a "Params" tab selected. Below the request details, the "Body" tab is active, showing a JSON response with two user objects. The JSON content is:

```

1 {
2   "page": 1,
3   "per_page": 6,
4   "total": 12,
5   "total_pages": 2,
6   "data": [
7     {
8       "id": 1,
9       "email": "george.bluth@reqres.in",
10      "first_name": "George",
11      "last_name": "Bluth",
12      "avatar": "https://reqres.in/img/faces/1.jpg"
13    },
14    {
15       "id": 2,
16       "email": "janet.weaver@reqres.in",
17       "first_name": "Janet",
18       "last_name": "Weaver",
19       "avatar": "https://reqres.in/img/faces/2.jpg"
20     }
21   ]
22 }
  
```

2. V zobrazeném body vyberte JSON typ a zkopírujte následující JSON:

```
{
  "info": "právě jsi provolal mock a dostal odpověď"
}
```

My Workspace    New Import    POST POST Confirm    GET GET regres.in.moc...    Mock    GET regres.in.moc...    +    ...    UAT

Collections    +    ...    Beta Postman skoleni    Imgur API    Kolekce pro import Copy    Live API    Postman skoleni    Prikady    REQ Params    HTTP Requests    GET GET Users - HTTP Requests    POST POST User - HTTP Requests    PUT PUT User - HTTP Requests...    PATCH PUT User - HTTP Requests...    DELETE DELETE User - HTTP Requ...    JSON    AUTH    PRE-REQUEST    Predavani variables    DYNAMIC VARS    GENERAL RUN    HEADERS    GRAPHQL    SOAP    Mock    GET GET regres.in.mock.example    GET GET regres.in.mock.example...    Reality    Static Tests

APIS    Environments    Mock Servers    Monitors    Flows    History

GET / Mock / GET regres.in.mock.example / GET regres.in.mock.example

Params Headers (6) Body

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Headers

Pretty Raw Preview JSON

```

1 "info": "pravě jsi provolal mock a dostal odpověď"
2
3

```

Status Code 200 OK

### 3. Do Status Code zadej 200 OK

My Workspace    New Import    POST POST Confirm    GET GET regres.in.moc...    Mock    GET regres.in.moc...    +    ...    UAT

Collections    +    ...    Beta Postman skoleni    Imgur API    Kolekce pro import Copy    Live API    Postman skoleni    Prikady    REQ Params    HTTP Requests    GET GET Users - HTTP Requests    POST POST User - HTTP Requests    PUT PUT User - HTTP Requests...    PATCH PUT User - HTTP Requests...    DELETE DELETE User - HTTP Requ...    JSON    AUTH    PRE-REQUEST    Predavani variables    DYNAMIC VARS    GENERAL RUN    HEADERS    GRAPHQL    SOAP    Mock    GET GET regres.in.mock.example    GET GET regres.in.mock.example...    Reality    Static Tests

APIS    Environments    Mock Servers    Monitors    Flows    History

GET / Mock / GET regres.in.mock.example / GET regres.in.mock.example

Params Headers (6) Body

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Headers

Pretty Raw Preview JSON

```

1 "info": "pravě jsi provolal mock a dostal odpověď"
2
3

```

Status Code 200 OK

### 4. Klikni na tlačítko "New" a vyber "Mock Server"

Create New

**Building Blocks**

- HTTP Request** Create a basic HTTP request
- WebSocket Request** BETA Test and debug your WebSocket connections
- Collection** Save your requests in a collection for reuse and sharing
- Environment** Save values you frequently use in an environment
- Workspace** Create a workspace to build independently or in collaboration

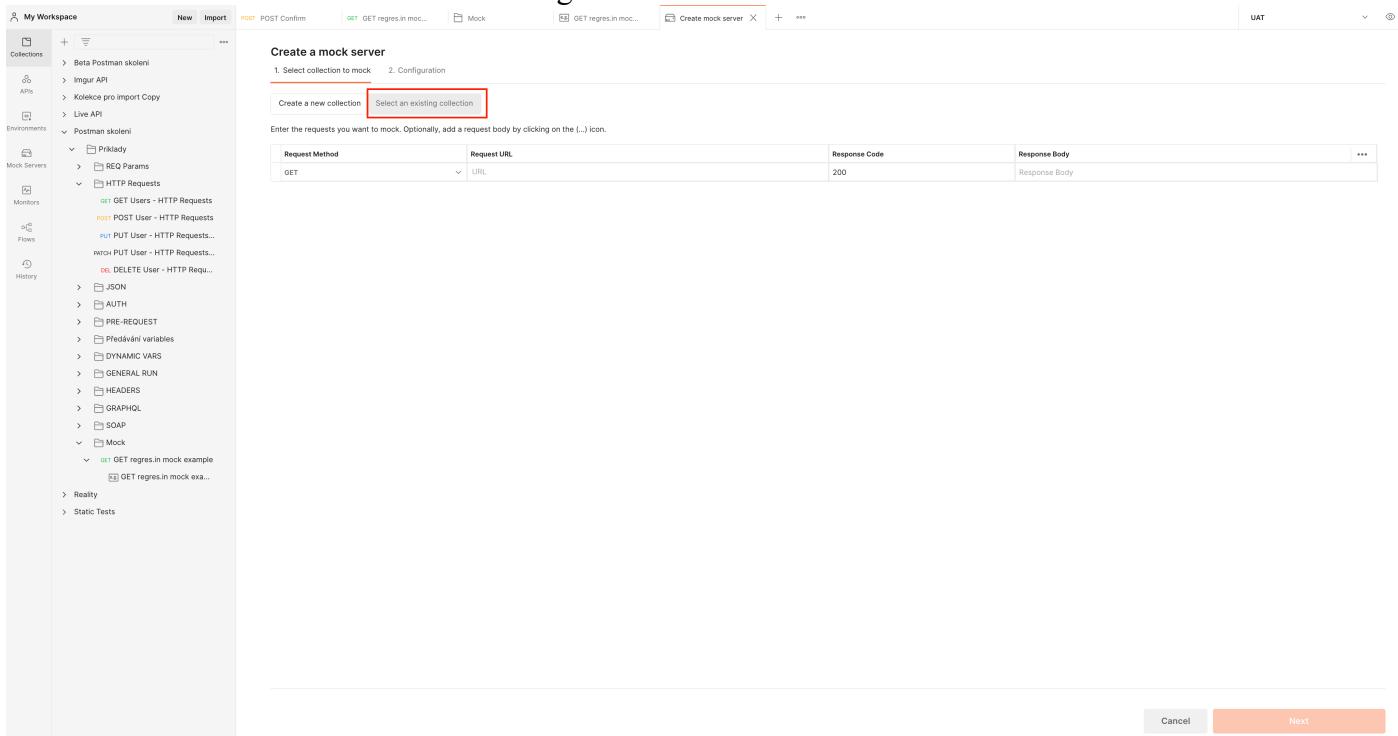
**Advanced**

- API Documentation** Create and publish beautiful documentation for your APIs
- Mock Server** Create a mock server for your in-development APIs
- Monitor** Schedule automated tests and check performance of your APIs
- API** Manage all aspects of API design, development, and testing
- Flows** BETA Test real-world workflows by connecting series of requests logically.

Not sure where to start? [Explore](#) featured APIs, collections, and workspaces published by the Postman community.

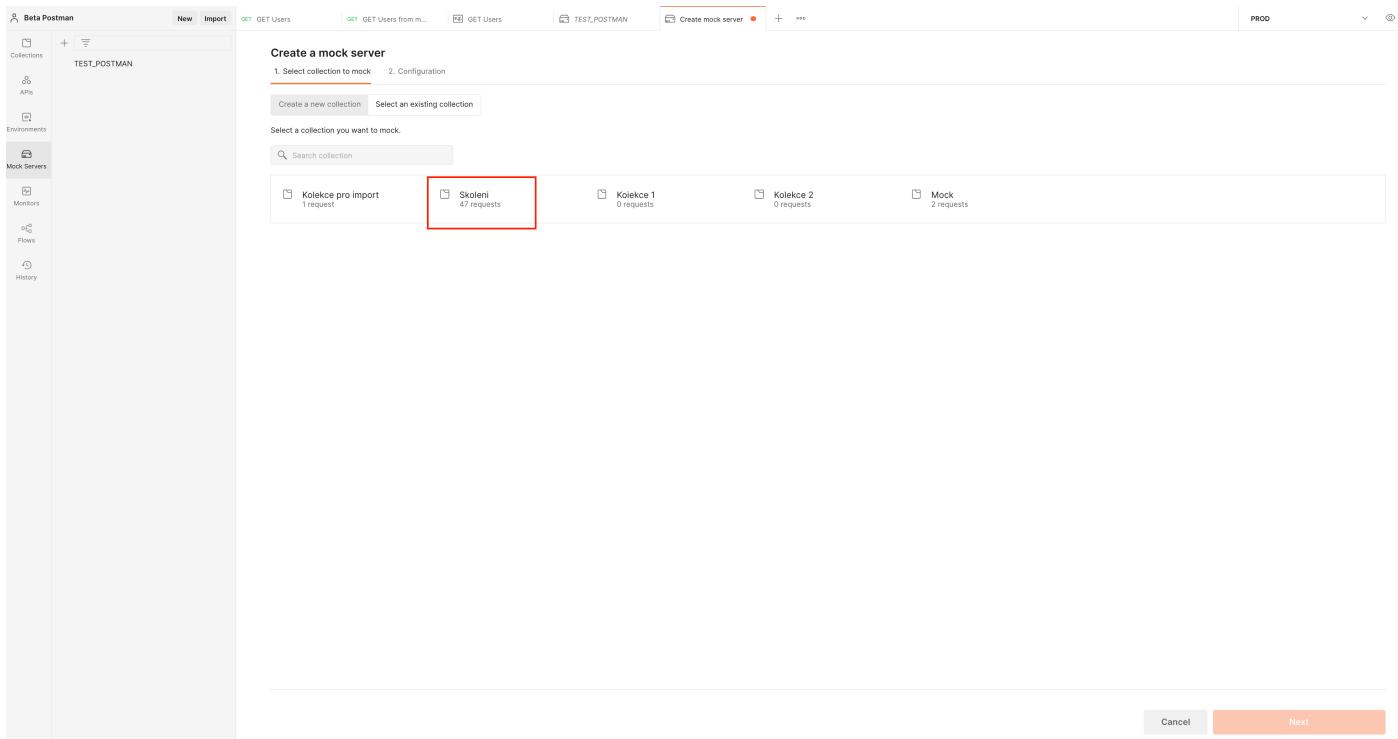
[Learn more on Postman Docs](#)

## 5. Klikni na tlačítko "Select an existing collection"



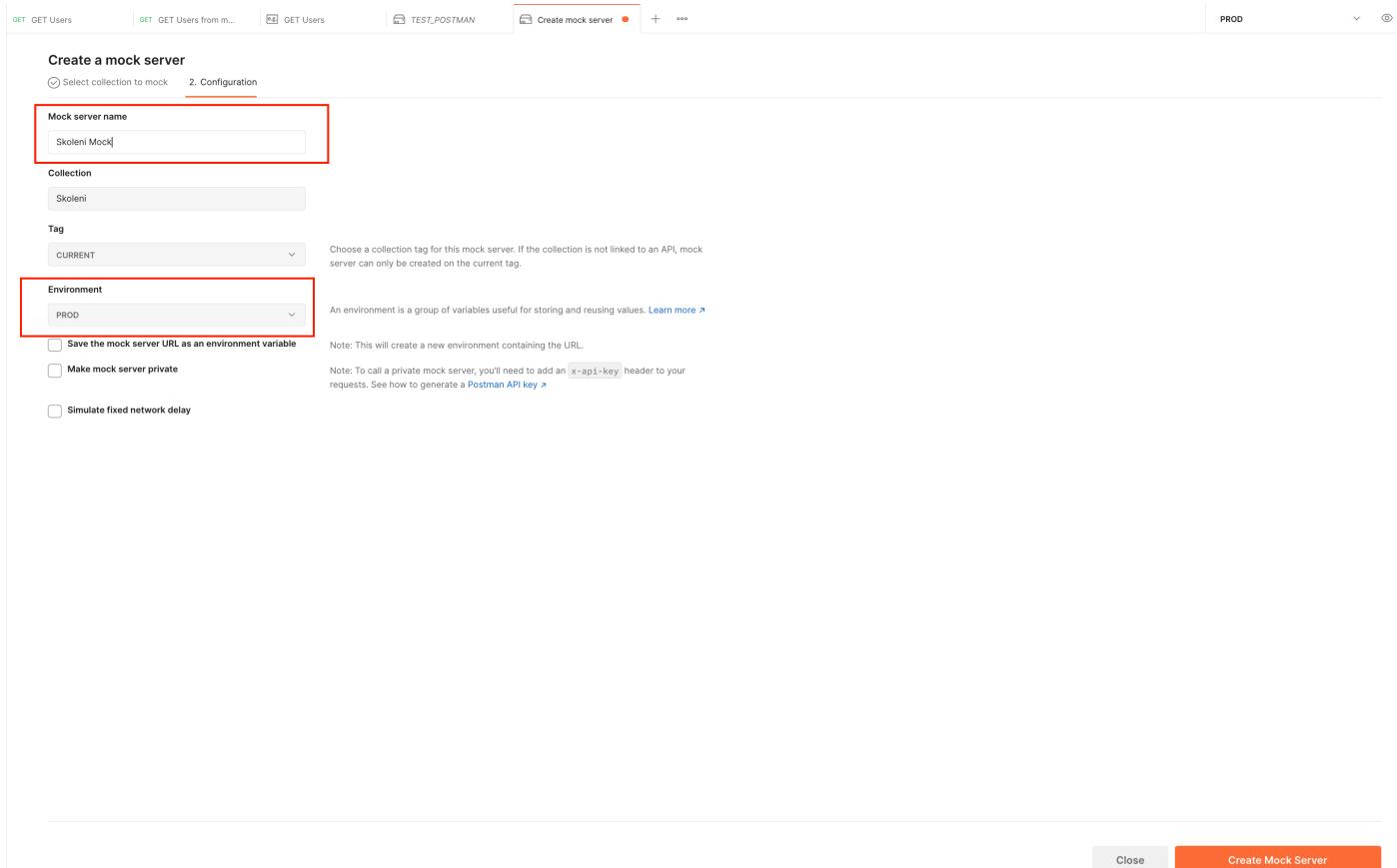
The screenshot shows the Postman interface with the 'Create a mock server' dialog open. On the left, there's a sidebar with various collections and environments listed. The main area shows the 'Create a mock server' steps: '1. Select collection to mock' and '2. Configuration'. A red box highlights the 'Select an existing collection' button under step 1. Below it is a text input field for entering requests and a table for defining the mock configuration.

## 6. Vyber kolekci Skoleni a stiskni tlačítko Next



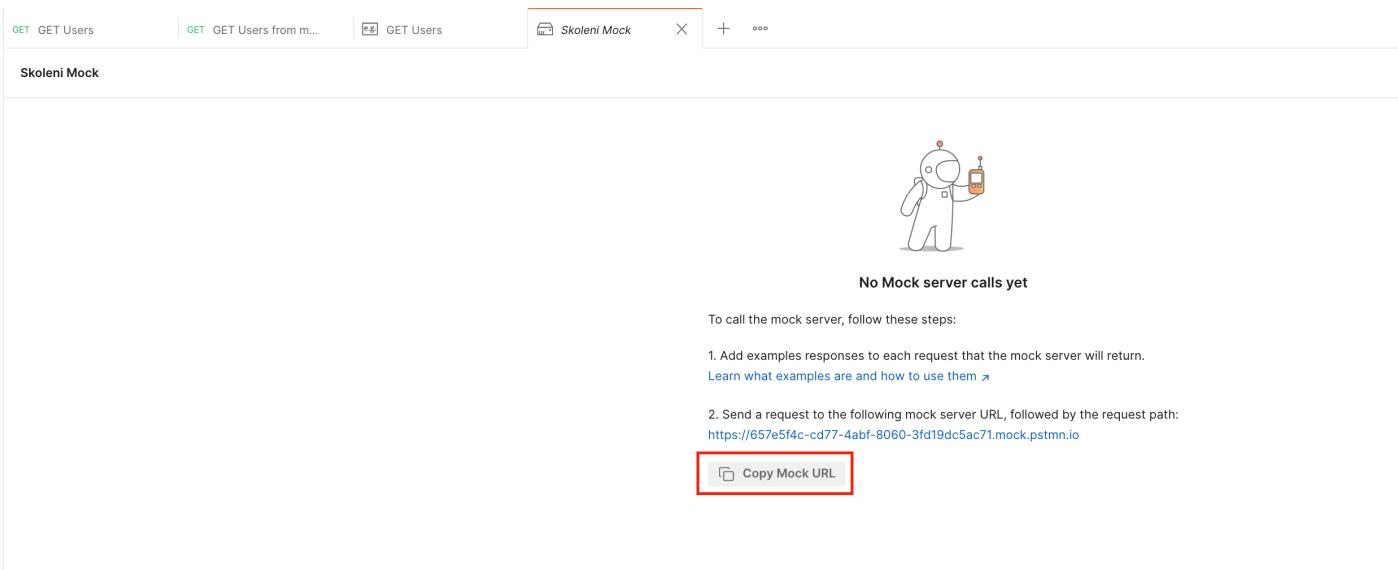
The screenshot shows the Postman interface with the 'Create a mock server' dialog open. On the left, there's a sidebar with various sections like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The 'Collections' section is expanded, showing several collections: 'Kolekce pro import' (1 request), 'Skoleni' (47 requests, highlighted with a red box), 'Kolekce 1' (0 requests), 'Kolekce 2' (0 requests), and 'Mock' (2 requests). The main area shows the 'Create a mock server' dialog with two tabs: '1. Select collection to mock' (selected) and '2. Configuration'. Below the tabs, there's a search bar and a list of collections. At the bottom right of the dialog are 'Cancel' and 'Next' buttons.

7. Zadej název Mock serveru "Skoleni Mock", vyber prostředí "PROD" a stiskni tlačítko "Create Mock Server"



The screenshot shows the 'Create a mock server' dialog in Postman. The 'Mock server name' field contains 'Skoleni Mock' (highlighted with a red box). The 'Collection' field is set to 'Skoleni'. The 'Tag' field is set to 'CURRENT'. The 'Environment' dropdown is set to 'PROD' (highlighted with a red box). There are three checkboxes at the bottom: 'Save the mock server URL as an environment variable', 'Make mock server private', and 'Simulate fixed network delay'. The 'Create Mock Server' button is located at the bottom right.

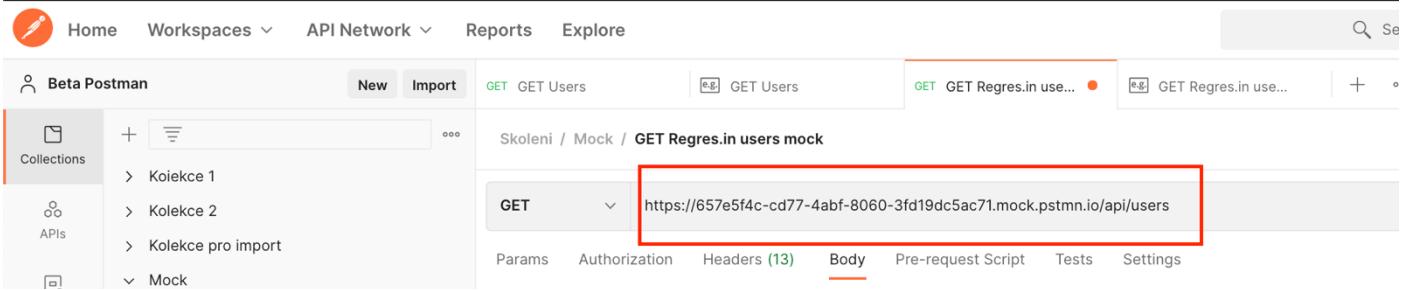
8. Zobrazí se ti okno s Mock URL, to vykopíruj



The screenshot shows the Skoleni Mock interface. At the top, there are three tabs: "GET GET Users", "GET GET Users from m...", and "GET GET Users". The central window is titled "Skoleni Mock" and contains a small illustration of a person holding a smartphone. Below the illustration, the text "No Mock server calls yet" is displayed. A message below it says "To call the mock server, follow these steps:" followed by two numbered steps. Step 1: "Add examples responses to each request that the mock server will return." with a link "Learn what examples are and how to use them". Step 2: "Send a request to the following mock server URL, followed by the request path: https://657e5f4c-cd77-4abf-8060-3fd19dc5ac71.mock.pstmn.io". A red box highlights the "Copy Mock URL" button.

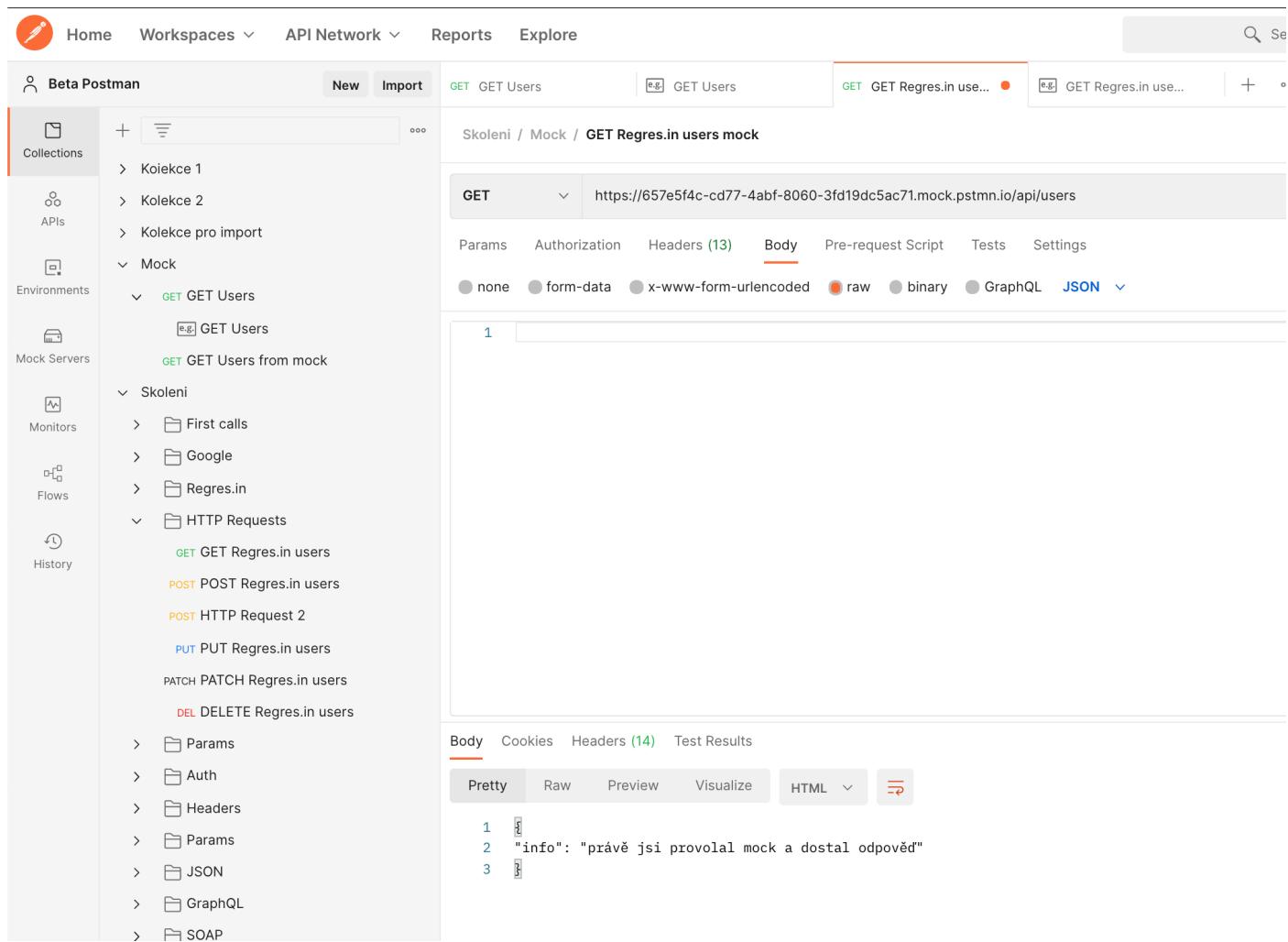
9. Otevři zpět request, který jsme vytvořili

10. Změň část URL: <https://reqres.in> za MOCK URL



The screenshot shows the Postman interface. The left sidebar has sections for "Collections" (with items "Koiekce 1", "Kolekce 2", "Kolekce pro import", and "Mock"), "APIs", and "Mocks". The main area shows a request configuration for a "GET Regres.in users mock" endpoint. The "Body" tab is selected, showing the URL "https://657e5f4c-cd77-4abf-8060-3fd19dc5ac71.mock.pstmn.io/api/users". A red box highlights this URL. Other tabs include "Params", "Authorization", "Headers (13)", "Tests", and "Settings".

11. Provolej Request, vrátí se ti response, kterou jsme nastavili v example.



The screenshot shows the Postman interface. On the left, there's a sidebar with navigation links: Home, Workspaces, API Network, Reports, Explore, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a collection named "Beta Postman". It contains several requests under different categories: "Kolekce 1", "Kolekce 2", "Kolekce pro import", "Mock", "Skoleni", and "History". Under "Mock", there are requests for "GET Users" and "GET Regres.in users". Under "Skoleni", there are requests for "GET Regres.in users", "POST Regres.in users", "POST HTTP Request 2", "PUT PUT Regres.in users", "PATCH PATCH Regres.in users", and "DEL DELETE Regres.in users". The "Body" tab is selected, showing a JSON response with the following content:

```

1
2 "info": "právě jsi provolal mock a dostal odpověď"
3

```

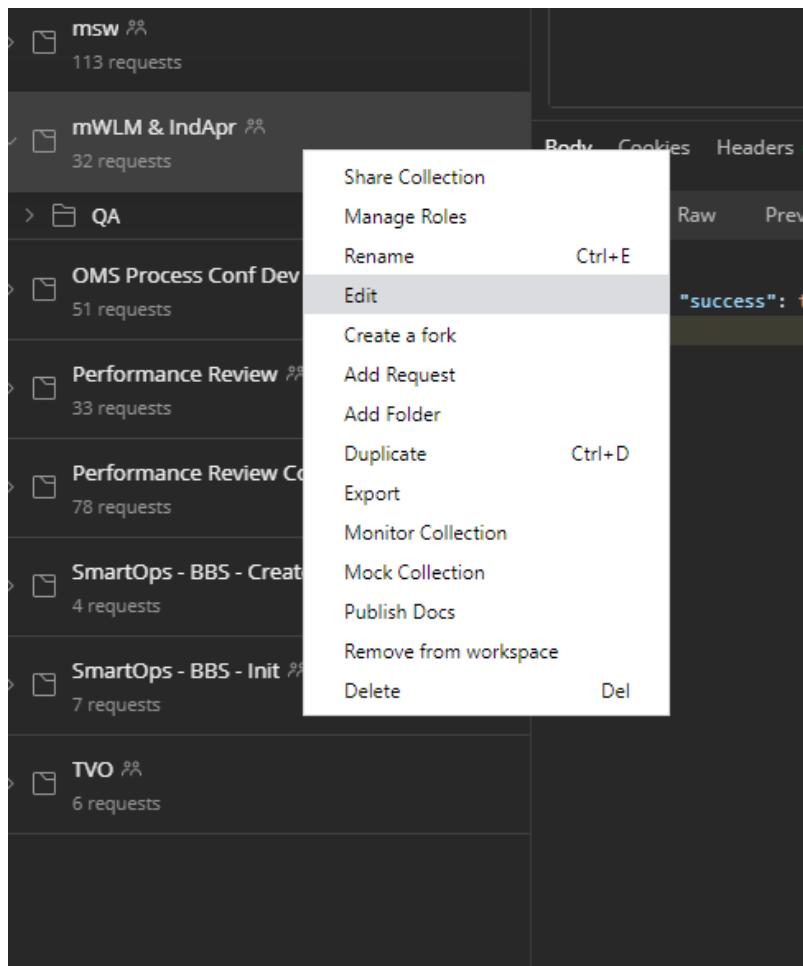
## POSTMAN 9 VS 7 (BONUS)

Některé projekty stále používají Postman 7, který má trochu jiné UI a ovládání.

### PŘÍSTUP K DETAILU KOLEKCE

Ve starší verzi Postman se kolekce neotvírá v okně jako request, ale je potřeba k ní přistoupit přes:

1. Pravý klik na kolekci
2. Vybrat Edit

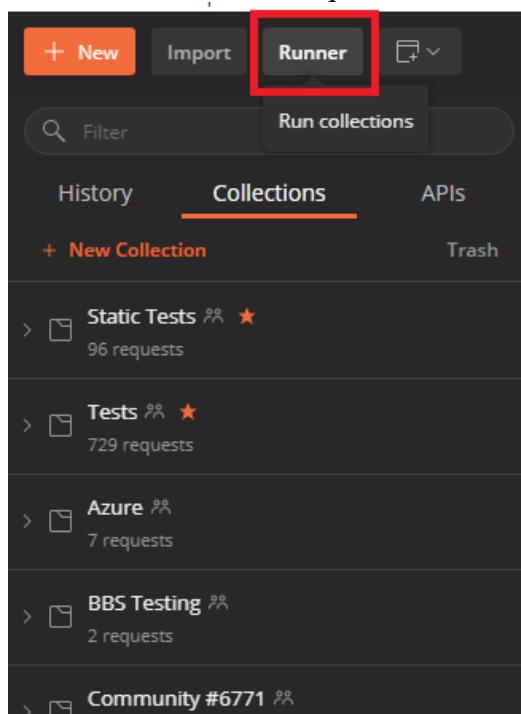


The screenshot shows the Postman interface with a sidebar listing various collections. A context menu is open over the 'QA' collection, which contains the following options:

- Share Collection
- Manage Roles
- Rename Ctrl+E
- Edit** "success": t (This option is highlighted)
- Create a fork
- Add Request
- Add Folder
- Duplicate Ctrl+D
- Export
- Monitor Collection
- Mock Collection
- Publish Docs
- Remove from workspace
- Delete Del

## COLLECTION RUNNER

Collection Runner se nespouští z kolekce, ale má svoje tlačítko v Postman UI



The screenshot shows the Postman UI with the 'Collections' tab selected. At the top, there is a toolbar with buttons for '+ New', 'Import', 'Runner' (which is highlighted with a red box), and 'Run collections'. Below the toolbar, there is a search bar labeled 'Filter' and a 'Run collections' button. The main area displays a list of collections:

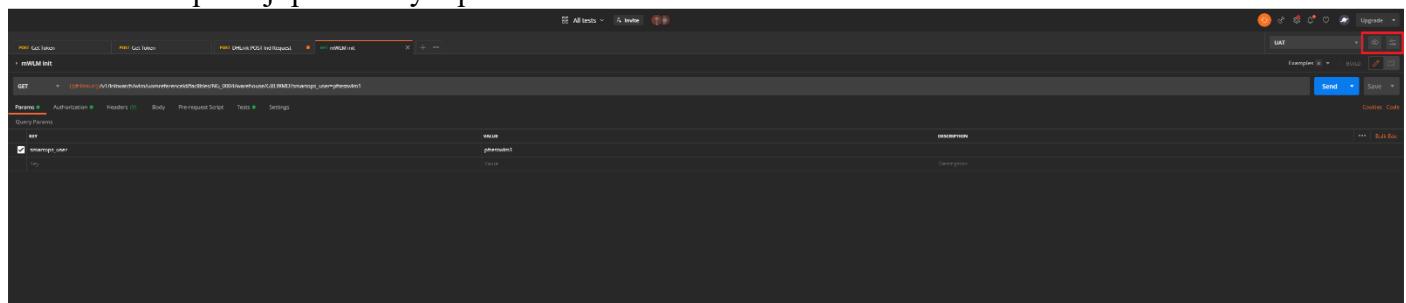
- History
- Collections** (selected)
- APIs
- + New Collection
- Trash

The collections listed are:

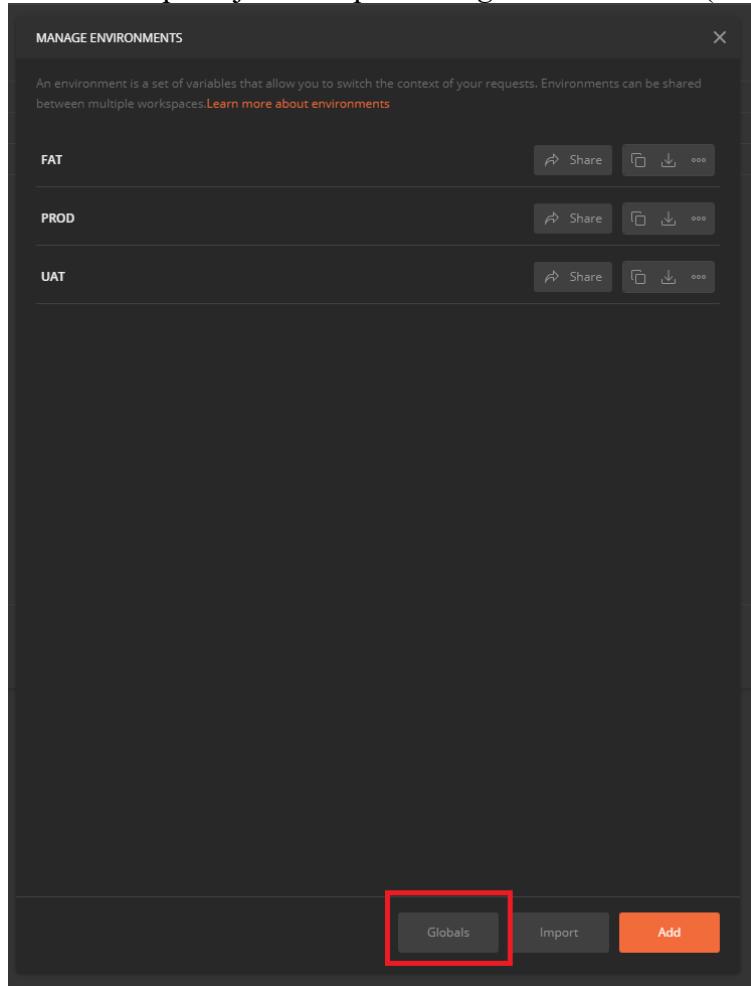
- Static Tests (96 requests)
- Tests (729 requests)
- Azure (7 requests)
- BBS Testing (2 requests)
- Community #6771

## PROMĚNNÉ

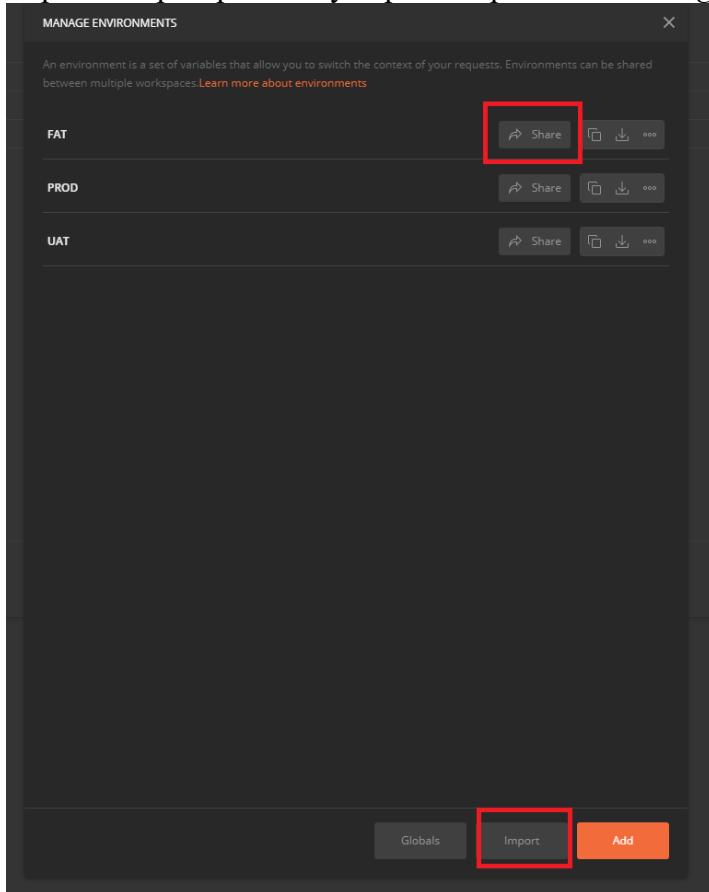
Proměnné se spravují přes ikony v pravé horní části Postman UI



Globals se spravují z okna pro *Manage Environments* (viz ikony výše)



Export a import proměnných probíhá přes okno *Manage Environments*



## ZDROJE, KAM DÁL

| Zdroj           | URL   | QR  |
|-----------------|---|---|
| ChaiJS knihovna | <a href="https://www.chaijs.com/">https://www.chaijs.com/</a>   |  |
| Co je to API    | <a href="https://en.wikipedia.org/wiki/API">https://en.wikipedia.org/wiki/API</a>   |  |
| Co je to REST   | <a href="https://cs.wikipedia.org/wiki/Representational_State_Transfer">https://cs.wikipedia.org/wiki/Representational_State_Transfer</a> |  |

|                                  |   |   |  |
|----------------------------------|---|---|--|
| <b>Co je to REST API</b>         | <a href="https://www.redhat.com/en/topics/api/what-is-a-rest-api">https://www.redhat.com/en/topics/api/what-is-a-rest-api</a>     |    |  |
| <b>Gorest API</b>                | <a href="https://gorest.co.in/">https://gorest.co.in/</a>   |    |  |
| <b>GraphQL dokumentace</b>       | <a href="https://graphql.org/learn/">https://graphql.org/learn/</a>   |    |  |
| <b>HTTP statusy</b>              | <a href="https://httpstatuses.com/">https://httpstatuses.com/</a>   |   |  |
| <b>Imgur apidocs</b>             | <a href="https://apidocs.imgur.com/">https://apidocs.imgur.com/</a>   |  |  |
| <b>Informace o HTTP Headerch</b> | <a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers">https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers</a> |  |  |
| <b>JSON</b>                      | <a href="https://www.json.org/json-en.html">https://www.json.org/json-en.html</a>   |  |  |

|                                   |   |   |  |
|-----------------------------------|---|---|--|
| <b>Newman dokumentace</b>         | <a href="https://learning.postman.com/docs/running-collections/using-newman-cli/command-line-integration-with-newman/">https://learning.postman.com/docs/running-collections/using-newman-cli/command-line-integration-with-newman/</a> |    |  |
| <b>Newman run možnosti</b>        | <a href="https://github.com/postmanlabs/newman/#newman-run-collection-file-source-options">https://github.com/postmanlabs/newman/#newman-run-collection-file-source-options</a>   |    |  |
| <b>NodeJS stažení</b>             | <a href="https://nodejs.org/en/download/current/">https://nodejs.org/en/download/current/</a>   |   |  |
| <b>Postman building request</b>   | <a href="https://learning.postman.com/docs/sending-requests/requests/">https://learning.postman.com/docs/sending-requests/requests/</a>   |  |  |
| <b>Postman creating API</b>       | <a href="https://learning.postman.com/docs/designing-and-developing-your-api/creating-an-api/">https://learning.postman.com/docs/designing-and-developing-your-api/creating-an-api/</a>   |  |  |
| <b>Postman documentation</b>      | <a href="https://learning.postman.com/docs/getting-started/introduction/">https://learning.postman.com/docs/getting-started/introduction/</a>   |  |  |
| <b>Postman dynamické proměnné</b> | <a href="https://learning.postman.com/docs/writing-scripts/script-references/variables-list/">https://learning.postman.com/docs/writing-scripts/script-references/variables-list/</a>   |  |  |

|  |   |   |
|--|---|---|
| <b>Postman external libraries in scripting</b> | <a href="https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/#using-external-libraries">https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/#using-external-libraries</a> |    |
| <b>Postman Github integrace</b>                | <a href="https://blog.postman.com/backup-and-sync-your-postman-collections-on-github/">https://blog.postman.com/backup-and-sync-your-postman-collections-on-github/</a>   |    |
| <b>Postman SOAP</b>                            | <a href="https://learning.postman.com/docs/sending-requests/supported-api-frameworks/making-soap-requests/">https://learning.postman.com/docs/sending-requests/supported-api-frameworks/making-soap-requests/</a>   |    |
| <b>Postman scripting documentation</b>         | <a href="https://learning.postman.com/docs/writing-scripts/intro-to-scripts/">https://learning.postman.com/docs/writing-scripts/intro-to-scripts/</a>   |  |
| <b>Postman Verzování ve workspace</b>          | <a href="https://learning.postman.com/docs/collaborating-in-postman/version-control-for-collections/#forking-a-collection">https://learning.postman.com/docs/collaborating-in-postman/version-control-for-collections/#forking-a-collection</a>                       |  |
| <b>Postman workspaces</b>                      | <a href="https://www.postman.com/product/workspaces/">https://www.postman.com/product/workspaces/</a>   |  |
| <b>Reqres.in API</b>                           | <a href="https://reqres.in/">https://reqres.in/</a>   |  |

|                                   |   |   |  |
|-----------------------------------|---|---|--|
| <b>Restful booker API</b>         | <a href="https://restful-booker.herokuapp.com/apidoc">https://restful-booker.herokuapp.com/apidoc</a>                             |    |  |
| <b>SOAPUI</b>                     | <a href="https://www.soapui.org/">https://www.soapui.org/</a>   |    |  |
| <b>Tutorial k Chrome Dev tool</b> | <a href="https://nira.com/chrome-developer-tools/">https://nira.com/chrome-developer-tools/</a>                                   |   |  |
| <b>W3SCHOOL JSON</b>              | <a href="https://www.w3schools.com/js/js_json_intro.asp">https://www.w3schools.com/js/js_json_intro.asp</a>                       |  |  |
| <b>W3SCHOOL HTTP methods</b>      | <a href="https://www.w3schools.com/tags/ref_httpmethods.asp">https://www.w3schools.com/tags/ref_httpmethods.asp</a>               |  |  |
| <b>Wikipedia URI</b>              | <a href="https://cs.wikipedia.org/wiki/Uniform_Resource_Identifier">https://cs.wikipedia.org/wiki/Uniform_Resource_Identifier</a> |  |  |
| <b>Wikipedia URL</b>              | <a href="https://cs.wikipedia.org/wiki/Uniform_Resource_Locator">https://cs.wikipedia.org/wiki/Uniform_Resource_Locator</a>       |  |  |

|                       |   |   |
|-----------------------|---|---|
| <b>Wikipedia URN</b>  | <a href="https://cs.wikipedia.org/wiki/Uniform_Resource_Name">https://cs.wikipedia.org/wiki/Uniform_Resource_Name</a>               |  |
| <b>Wikipedia HTTP</b> | <a href="https://cs.wikipedia.org/wiki/Hypertext_Transfer_Proto_col">https://cs.wikipedia.org/wiki/Hypertext_Transfer_Proto_col</a> |  |

## SLOVNÍK POJMŮ

| Pojem            | Vysvětlení   |
|------------------|--|
| <b>API</b>       | Application programming interfaces   |
| <b>REST</b>      | Representational State Transfer  |
| <b>Interface</b> | Software interfaces (programming interfaces) jsou jazyky, kódy a zprávy, které programy využívají ke vzájemné komunikaci a také pro komunikaci s Hardware. Příklady jsou: Windows, Mac a Linux operační systémy, SMTP e-maily, IP síťové protokoly a softwarové ovladače, které aktivují periferní zařízení. |
| <b>HTTP</b>      | Hypertext Transfer Protocol - internetový protokol určený pro komunikaci s WWW servery. Slouží pro přenos hypertextových dokumentů ve formátu HTML, XML, i jiných typů souborů.  |
| <b>JSON</b>      | JavaScript Object Notation   |
| <b>URL</b>       | Uniform Resource Locator   |
| <b>URN</b>       | Uniform Resource Name  |
| <b>URI</b>       | Uniform Resource Identifier  |
| <b>SOAP</b>      | Simple Object Access Protocol  |
| <b>Fulltext</b>  |  |
| <b>GUI</b>       | Grafické uživatelské rozhraní (Graphic User Interface)   |
| <b>OS</b>        | Operační systém (Operating System). Jedná se o prostředí, ve kterém běží všechny procesy, aplikace... Nejznámější OS: Windows, MacOS, Android, iOS, Linux  |
| <b>WWW</b>       | World Wide Web   |
| <b>Cookies</b>   | Jako cookie (anglicky koláček, oplatka, sušenka) se v protokolu HTTP označuje malé množství dat, která WWW server pošle prohlížeči, který je uloží na počítači uživatele. Při  |

|             |  |
|-------------|--|
|             | každé další návštěvě téhož serveru pak prohlížeč tato data posílá zpět serveru. Cookies běžně slouží k rozlišování jednotlivých uživatelů, ukládají se do nich uživatelské předvolby apod. Myšlenku cookies navrhl v 90. letech Lou Montulli, tehdy pracující u firmy Netscape Communications. |
| <b>OOP</b>  | Objected Oriented Programming  |
| <b>MOCK</b> | Simulované objekty, které imitují chování reálných objektů kontrolovaným způsobem  |