

POSTMAN ŠKOLENÍ

OBSAH

<i>Obsah</i>	1
<i>Úvod</i>	3
<i>O přednášejícím</i>	3
<i>Rest API</i>	3
Co je to Rest API	3
předávání dat v REST API.....	3
<i>HTTP Requesty</i>	4
HTTP statusy	5
<i>Postman</i>	5
<i>Co je to postman</i>	5
<i>První spuštění Postman</i>	5
Registrace Postman.com	5
Přihlášení do Postman.....	5
<i>Základní orientace</i>	7
Přidání prvku.....	8
Filtr.....	9
Settings.....	9
Console	10
<i>Requesty</i>	10
<i>Orientace v collections</i>	11
Postup vytvoření kolekce	11
Import, export kolekcí.....	11
Orientace ve Složkách.....	14
<i>HTTP Requesty</i>	15
Vytvoření a provolání requestu.....	15
URL	24
Typ HTTP callu	25
Params.....	28
Auth.....	28
Headers	43
Settings.....	44
Data v Requestech.....	45
GraphQL	51
SOAP	53
<i>Environments a proměnné</i>	54
<i>Priority použití proměnných</i>	55
<i>Nastavení proměnných</i>	55
Initial value	55
Current value.....	55
Persist all/Reset All.....	55

<i>Globals</i>	56
Import globals	56
Export globals	58
<i>Collection</i>	59
<i>Environment</i>	59
Vytvoření prostředí	59
Import proměnných pro prostředí	61
Export proměnných pro prostředí	62
<i>Data</i>	63
<i>Local</i>	63
<i>Set as variable</i>	63
<i>Použití proměnných a prostředí</i>	63
<i>Testování v postmanu</i>	64
<i>Úvod do testování v postmanu</i>	64
Javascript	64
Snippets.....	64
Externí knihovny.....	64
<i>Typy testů</i>	64
Nastavení proměnných ve skriptech	64
Pre-request script	68
Tests	69
<i>Psaní testů v Postmanovi</i>	70
<i>Snippety</i>	70
Vysvětlení a použití	70
<i>Javascript testy</i>	72
Kontrola statusů	72
Kontrola typu	73
Parse JSON	73
Kontrola dat	73
<i>Skriptování v postmanovi</i>	75
Dynamické proměnné	75
Práce s Timestamp (BONUS).....	77
<i>Debugging</i>	78
Debugging requestu	78
Debugging testů	79
<i>Přepoužívání kódu (BONUS hack)</i>	80
<i>Collection Runner</i>	82
<i>Ovládání Runneru</i>	82
Konfigurace runneru	83
<i>Obecné spuštění</i>	84
ENV Proměnné pro rohlik.cz.....	87
Spuštění připravené složky/kolekce.....	88
Výsledek běhu.....	89

< TREDGATE >

Data driven testy	91
Užití v praxi	92
Sdílení dat v týmu.....	96
Verzování kolekcí.....	96
Workspace	96
Vytvoření workspace	97
Vytváření sdílených requestů.....	99
Správa workspace, pozvání nových členů ..	99
Changelog	100
Newman.....	101
Instalace	101
Spouštění testů.....	101
Základní Spuštění testu	101
Globals	102
EnvirOnment.....	102
Reporting	103
BATCH FILE.....	104
Komplexní příklad	105
MOCK servers v Postman	106
Příklad.....	106
Postman 9 vs 7 (BONUS).....	112
Přístup k detailu kolekce	112
Collection Runner	113
Proměnné.....	113
Zdroje, kam dál	115
Slovnik pojmu.....	120

ÚVOD

O PŘEDNÁŠEJÍCÍM

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

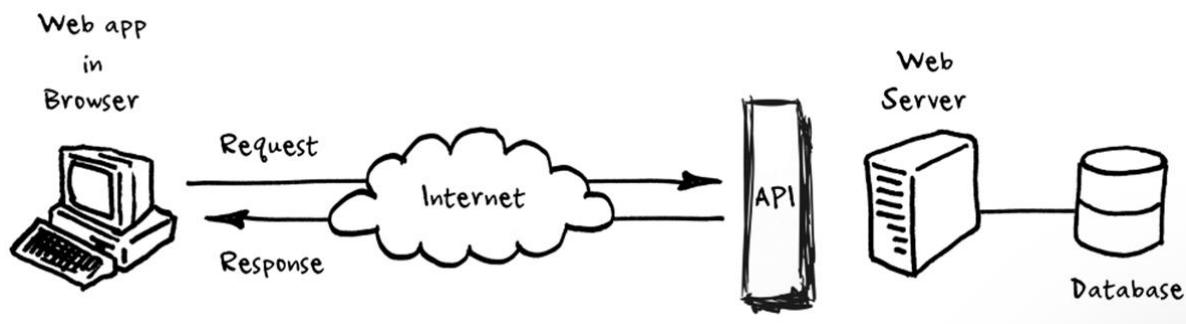
E-mail: petr.fifka@tredgate.cz

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 předat 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
String	Text, zapisuje se vždy do dvojitých uvozovek. Příklad: "Text"
Number	Číslo, zapisuje se bez uvozovek. Příklad: 3.21
Object	Objekt. Může obsahovat další json data. Zapisuje se vždy do složených závorek.
Array	Pole hodnot, zapisuje se vždy do hranatých závorek
Boolean	Logická hodnota. Zapisuje se: true nebo false
Null	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

```
{
  "jmeno": "Petr",
  "vek": 31,
  "majetek": ["mobil", "auto", "byt"]
}
```

HTTP REQUESTY

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

Proč se zmíňujeme 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?
200	OK	Vše ok, odpověď zpracována
201	Created	Vše ok, request zapsán
204	No Content	Vše ok, neposílám žádné tělo odpovědi
400	Bad Request	Request nezpracován, špatná struktura. Něco je špatně s daty nebo parametry
401	Unauthorized	Něco je špatně s autorizací nebo není vyplněna
404	Not Found	URL neexistuje nebo spadl celý server a není možné ho provolat
500	Internal Server Error	Ve většině případů znamená, že je API server nefunkční.
503	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 zjednodušuje 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:

< TREDGATE >

Postman

File Edit View Help

POSTMAN

Create an account or sign in

Create Free Account

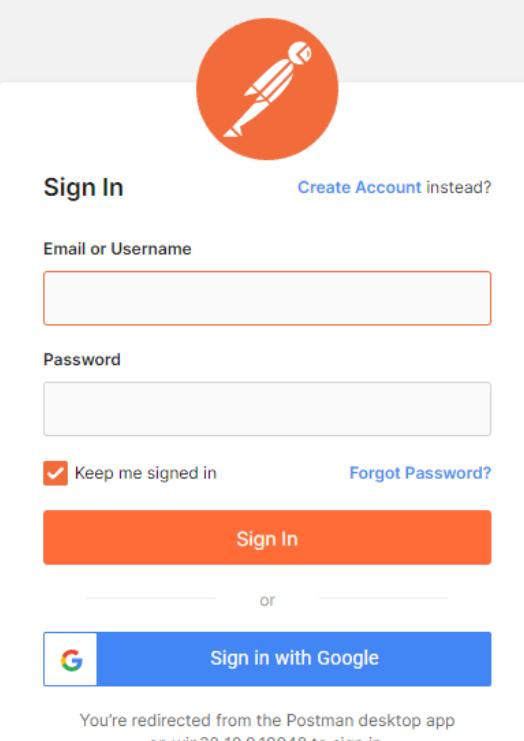
Sign in

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

< TREDGATE >

The screenshot shows the Postman application window. At the top, there's a navigation bar with 'Postman' logo, 'File', 'Edit', 'View', 'Help' menus, and a search bar 'Search Postman'. Below the menu is a toolbar with icons for 'Home', 'Workspaces', 'Reports', 'Explore', and 'Upgrade'. The main content area starts with a greeting 'Good afternoon, Petr Fifka!' followed by a message 'Pick up where you left off, catch up with your team's work.' On the left, there's a sidebar with a workspace icon containing a letter 'B', titled 'bold-rocket-601006' with a URL 'bold-rocket-601006.postman.co'. It includes links for 'Workspaces', 'Private API Network', and 'Integrations'. Below this is a 'Help' section with links to 'Learning Center', 'Support Center', and 'Bootcamp', accompanied by a cartoon character icon.

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	

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.

This screenshot shows the 'Workspaces' view in Postman. The 'Workspaces' tab in the navigation bar is highlighted with a red box. The main content area is identical to the dashboard in the previous screenshot, featuring the same greeting, workspace information, and 'Get started with Postman' sections. The 'Recent workspaces' table also remains the same.

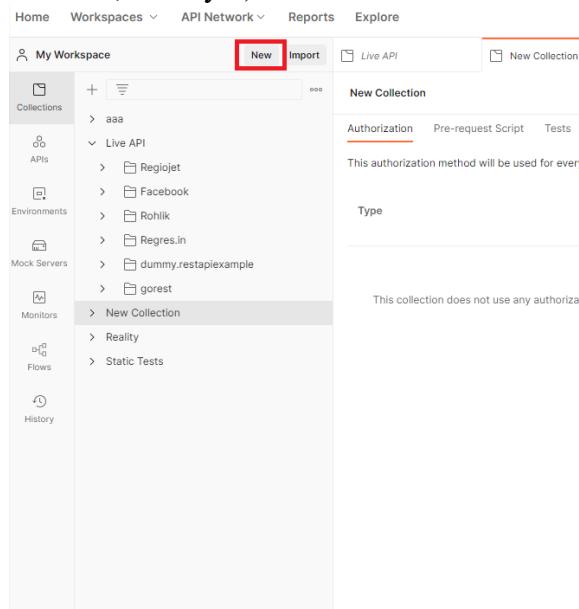
ZÁKLADNÍ ORIENTACE

< TREDGATE >

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.

A screenshot of the 'Create New' dialog box in Postman. It has a header 'Create New' and a close button 'X'. The dialog is divided into sections: 'Building Blocks' and 'Advanced'.
Building Blocks:

- HTTP Request** (GET icon): 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.

At the bottom of the dialog, there's a note: 'Not sure where to start? [Explore](#) featured APIs, collections, and workspaces published by the Postman community.' and a link '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 folder named 'gorest' with several sub-folders like 'Live API', 'Facebook', 'Rohlik', 'Regres.in', 'dummy.restapiexample', and another 'gorest' folder containing a POST request to 'https://qorest.co.in/public/v1/'. The right panel shows the 'Authorization' tab of the 'New Collection' dialog.

FILTR

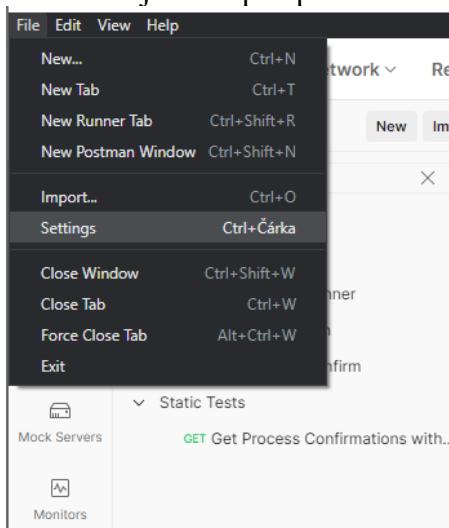
Filtr umožňuje fulltextově hledat v rámci jednotlivých sekcí.

Home Workspaces ▼ API Network ▼ Reports

The screenshot shows the Postman interface with a search filter applied. The search bar at the top contains the text 'CONFIRM'. A red box highlights the search bar. The left sidebar lists 'My Workspace' with sections for Collections, APIs, Environments, Mock Servers, and Monitors. The main workspace shows a list of collections: 'Live API', 'Rohlik' (with 'Collection Runner' and 'General Run' sub-folders), and 'Static Tests'. Under 'Static Tests', there is a GET request for 'POST Confirm'. The search results are filtered to show only items containing 'CONFIRM'.

SETTINGS

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



< TREDGATE >

The screenshot shows the Postman application window. On the left is the sidebar with options like Home, Workspaces, Reports, Explore, Collections, APIs, Environments, Mock Servers, Monitors, and History. In the center, there's a search bar and a 'No Environment' dropdown. A 'SETTINGS' dialog box is open over the main area, specifically the 'General' tab. The 'Request' section contains settings for trimming keys, SSL verification, opening requests in new tabs, asking about unsaved tabs, language detection, request timeout, and max response size. The 'Headers' section includes options for no-cache header, Postman Token header, retaining headers, automatically following redirects, anonymous usage data, and two-pane view. The 'User interface' section has settings for icons with tab names and variable autocomplete.

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

CONSOLE

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

The screenshot shows the Postman application window again. The sidebar is identical to the previous one. The main area now features a 'Console' tab at the bottom, which is highlighted with a red box. To its right is a 'RUN ORDER' section with a 'Deselect All' button, a 'Select All' button, and a 'Reset' button. It also includes fields for 'Iterations' (set to 1), 'Delay' (set to 0), and a 'Data' section with a 'Select File' button. There are several checkboxes on the right: 'Save responses', 'Keep variable values' (which is checked), 'Run collection without using stored cookies', and 'Save cookies after collection run'. At the bottom right is a large blue 'Start Run' button.

REQUESTY

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

< TREDGATE >

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

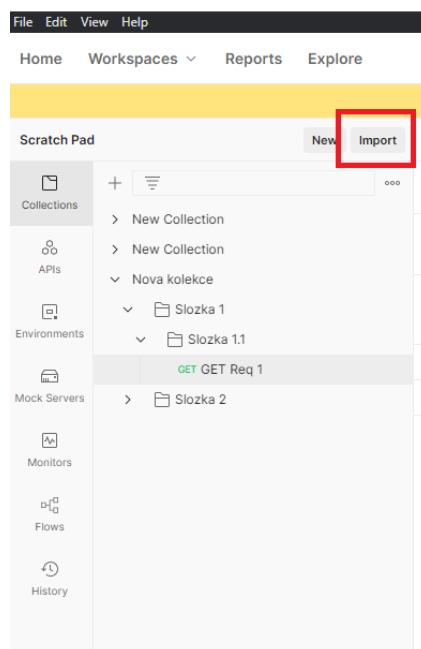
IMPORT, EXPORT KOLEKCÍ

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

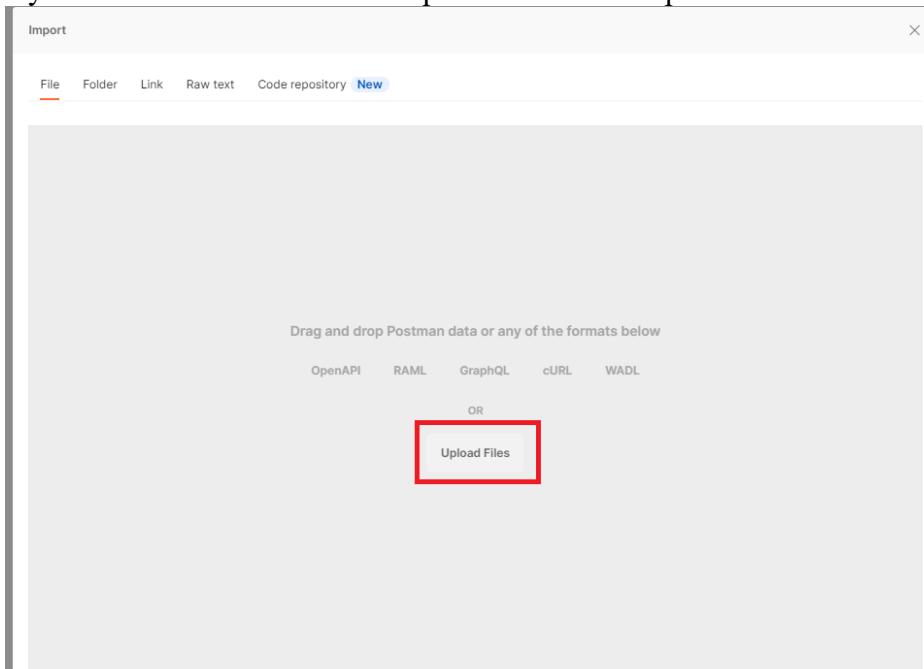
IMPORT KOLEKCE

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..."

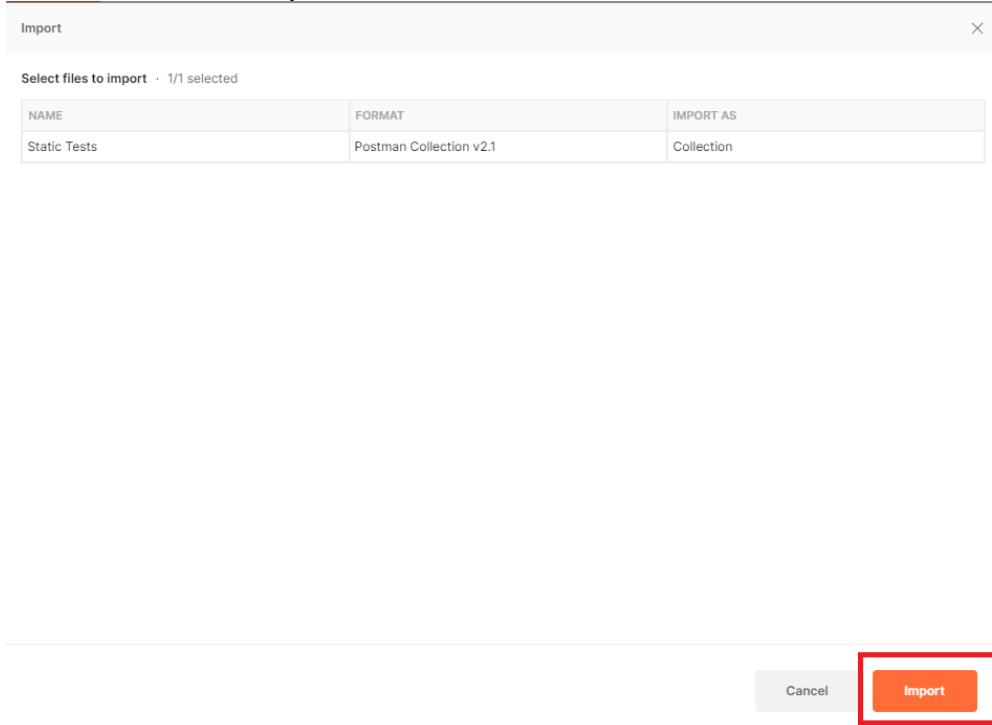
1. V záložce Collections kliknout na „Import“



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:

- Top Bar:** File, Edit, View, Help, Home, Workspaces, API Network, Reports, Explore, Search Postman.
- Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Central Area:**
 - Overview tab: Kolekce 1, Kolekce 2.
 - Selected Collection: Kolekce 2, sub-collection Kolekce pro import.
 - Authorization tab: Auth (highlighted), Pre-req., Tests, Variables.
 - Description: This authorization method will be used for every request in this collection. You can override this by specifying one in the request.
 - Type: No Auth.
 - Note: This collection does not use any authorization. Learn more about authorization.
- Right Panel:** Documentation (Make things easier for your teammates with a complete collection description).
- Bottom Status Bar:** Find and Replace, Console, Collection imported message (Collection Kolekce pro import imported), View complete collection documentation, Bootcamp, Runner, Trash.

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:

- Top Bar:** File, Edit, View, Help, Home, Workspaces, API Network, Reports, Explore, Search Postman.
- Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Central Area:**
 - Overview tab: Kolekce 1, Kolekce 2, Kolekce pro import.
 - Selected Collection: Kolekce pro import, sub-collection Kolekce pro import.
 - Authorization tab: Authorization (highlighted), Pre-request Script, Tests, Variables.
 - Description: This authorization method will be used for every request in this collection. You can override this by specifying one in the request.
 - Type: No Auth.
 - Note: This collection does not use any authorization. Learn more about authorization.
- Right Panel:** Watch, Fork, Run, Save, Share, icons for copy, edit, delete, etc.

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](#).

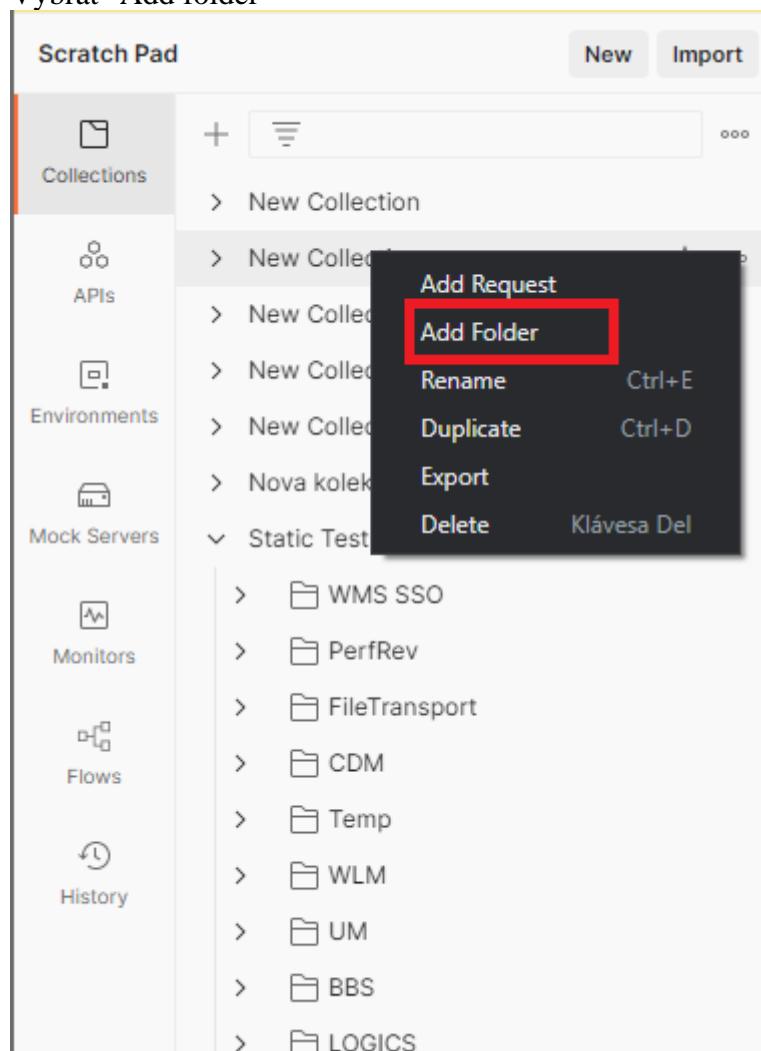
Run ikona slouží pro spuštění [Collection runneru](#), který probereme později.

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 Scratch Pad interface. On the left, there's a sidebar with categories: Collections, APIs, Environments, Mock Servers, Monitors, and Flows. The main area has tabs for 'New', 'Import', and file operations ('N.', 'S.', 'S.', 'GET G.', 'N.'). A red box highlights the input field 'New Folder'. Below it, the 'Authorization' tab is selected, showing a note about using the same authorization method for all requests in this folder. It also includes a 'Type' dropdown set to 'Inherit auth from parent' and a link to learn more about authorization. At the bottom, it states that the folder uses Basic Auth from its parent collection.

STRUKTURA SLOŽKY

Velice podobná struktuře Collection.

Obsahuje záložky:

- Authorization
- Pre-request Script
- Tests

A pro spuštění Collection Runneru tlačítko Run

The screenshot shows the Postman interface with a navigation bar at the top: Overview, Kolekce 1, Kolekce 2, Test (selected), Run, Save, and more. Below the navigation is a section titled 'Kolekce pro import / Test'. The 'Authorization' tab is selected, with a note that the same authorization method will be used for all requests in this folder. It includes a 'Type' dropdown set to 'Inherit auth from parent' and a link to learn more about authorization. At the bottom, it states that the folder uses No Auth from its parent collection.

HTTP REQUESTY

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

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

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

VYTVOŘENÍ A PROVOLÁNÍ REQUESTU

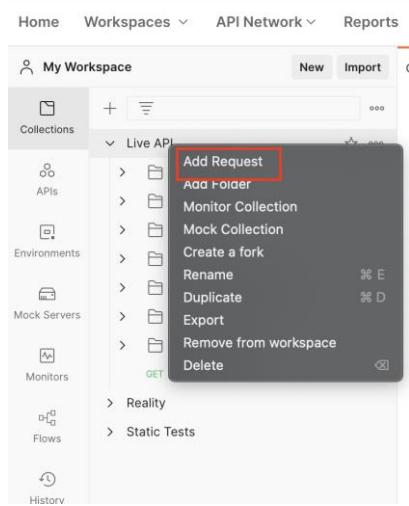
Vytvoříme složku "First calls" v kolekci Skoleni.

< TREDGATE >

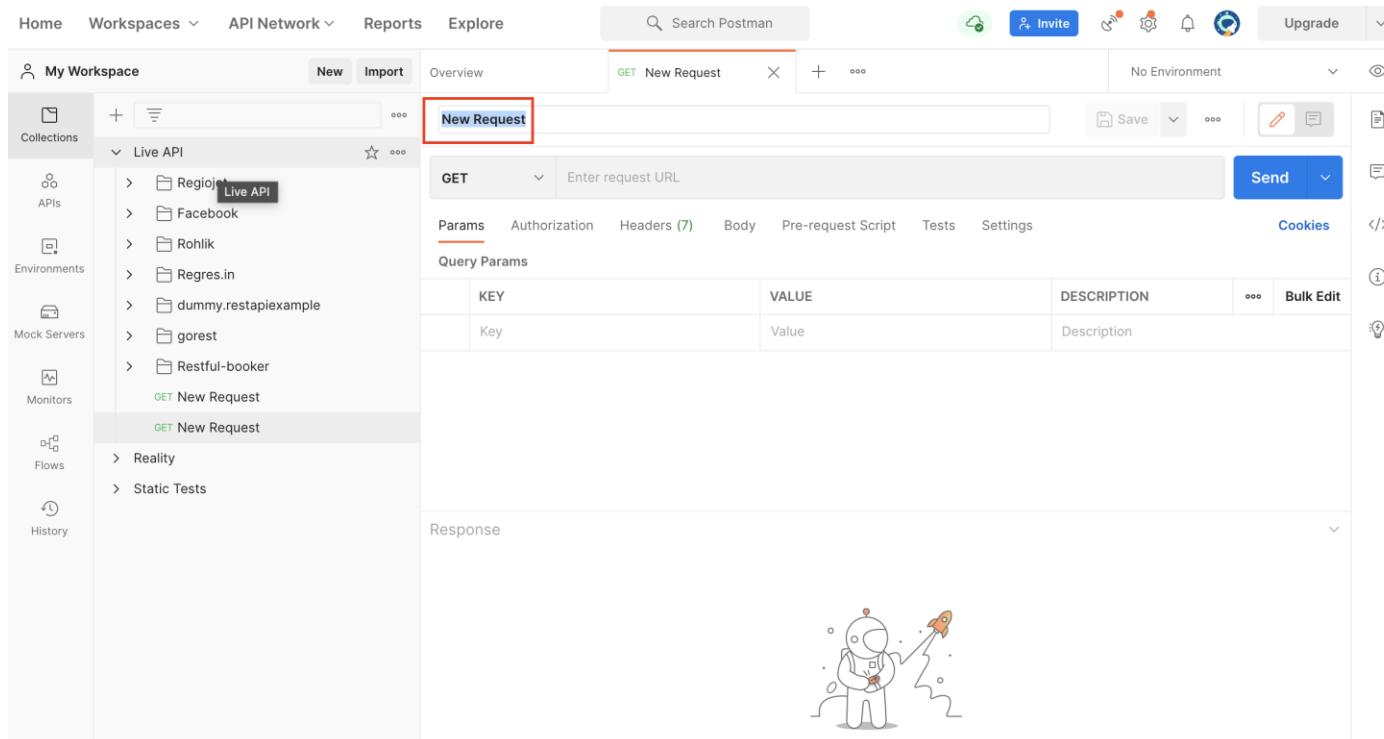
Existuje několik postupů, jak v Postman založit HTTP request.

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



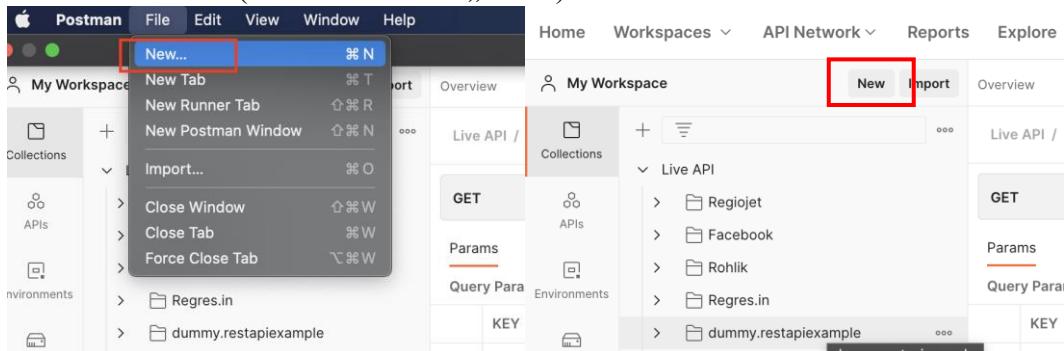
Úkol:

Vytvořte 2 requesty:

1. Vytvořte novou kolekci
2. Vytvořte složku v kolekci
3. V kolekci vytvořte request se jménem: HTTP Request 1
4. Ve složce "First calls": HTTP Request 2

POSTUP 2

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



2. Vybrat HTTP Request

The screenshot shows the 'Create New' dialog in Postman. It displays several 'Building Blocks' and 'Advanced' options:

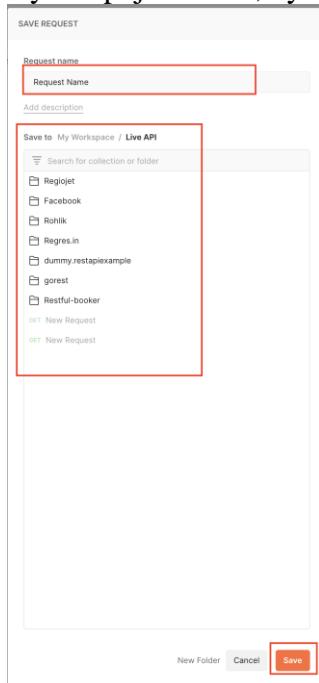
- HTTP Request** (highlighted with a red box): Create a basic HTTP request.
- WebSocket Request BETA**: Test and debug your WebSocket connections.
- Collection**: Save your requests in a collection for reuse and sharing.
- Environment**: Save values you frequently use in an environment.
- Workspace**: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

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".

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

The screenshot shows the 'Untitled Request' screen in Postman. At the top right, there is a 'Save' button with a dropdown arrow, which is highlighted by a red box. Below the header, there are tabs for 'Overview', 'GET New Request', 'Restful-booker', and 'GET Untitled Request'. The main area shows a 'Params' section with a 'Query Params' table and a 'Send' button at the bottom right.

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:

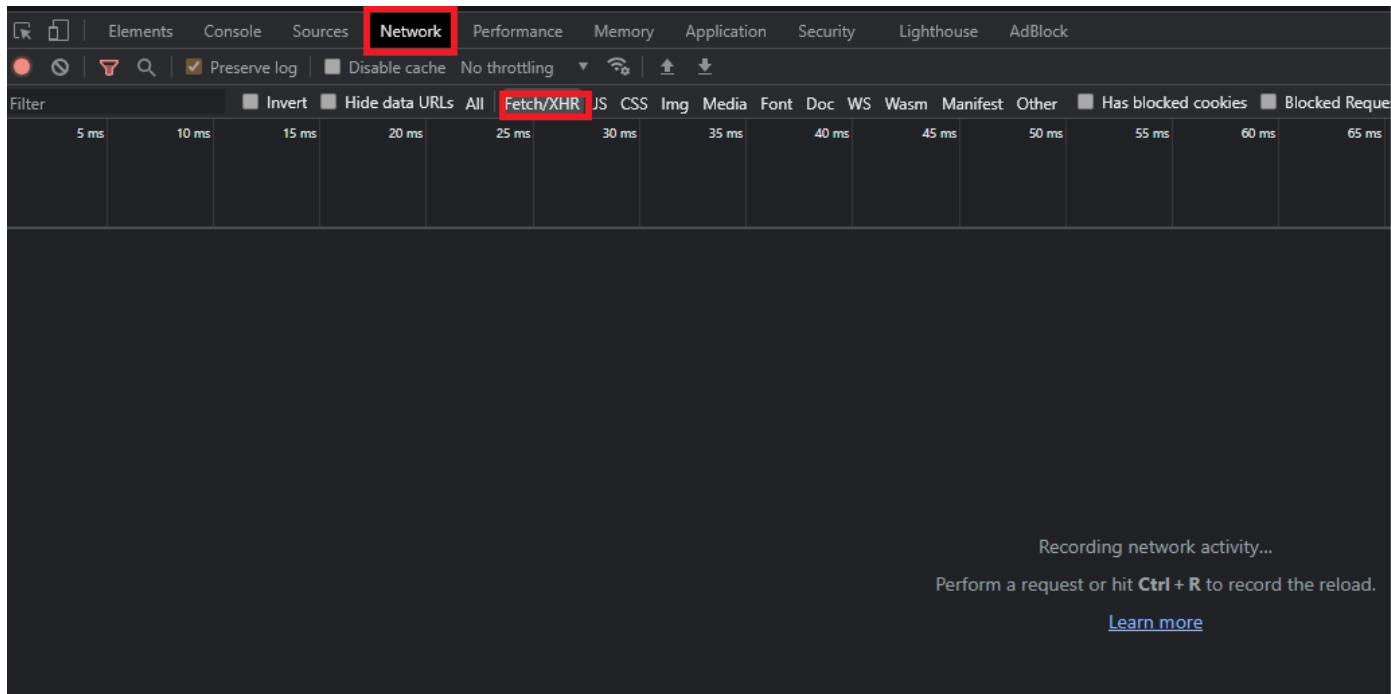
```
curl 'https://reqres.in/api/users?page=2' \
-H 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
-H 'Referer: https://reqres.in/' \
-H 'sec-ch-ua-mobile: ?0' \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/95.0.4638.54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

IMPORT CURL

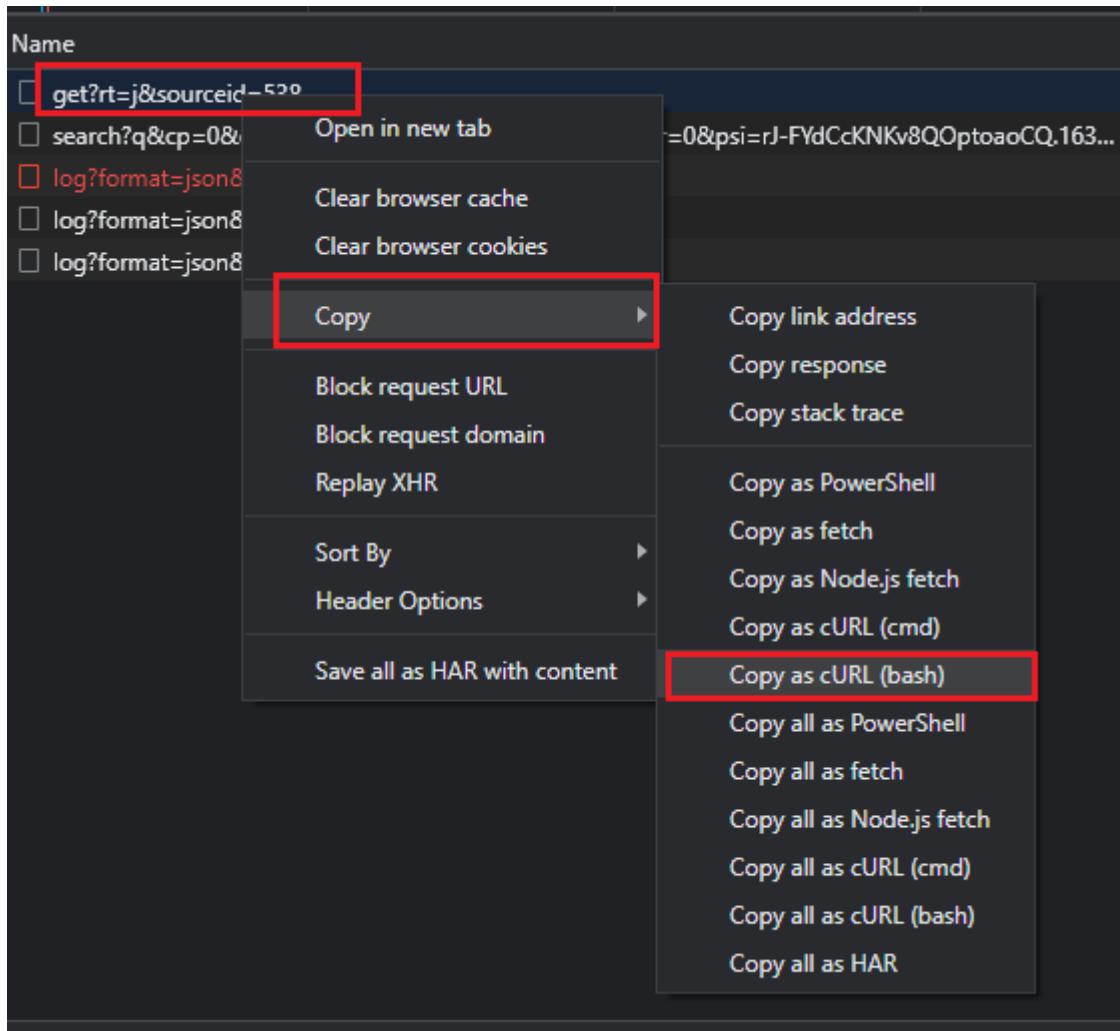
Jak fungují níže zmíněné DEV naleznete v [tomto tutoriu](#).

- Otevřeme prohlížeč a poté dev tooly (CTRL+SHIFT+I)

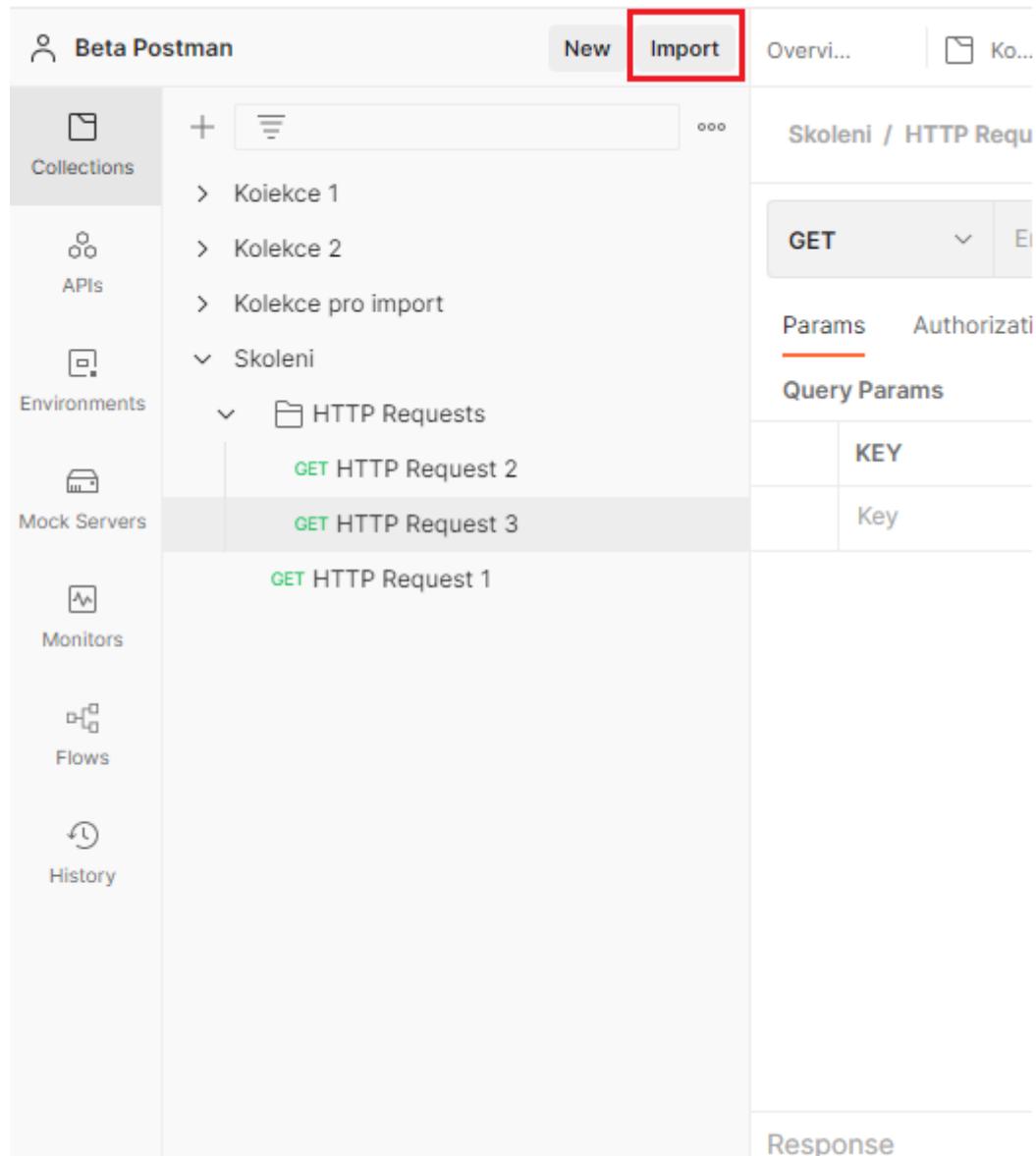
2. Otevřeme záložku Network a zvolíme Fetch/XHR



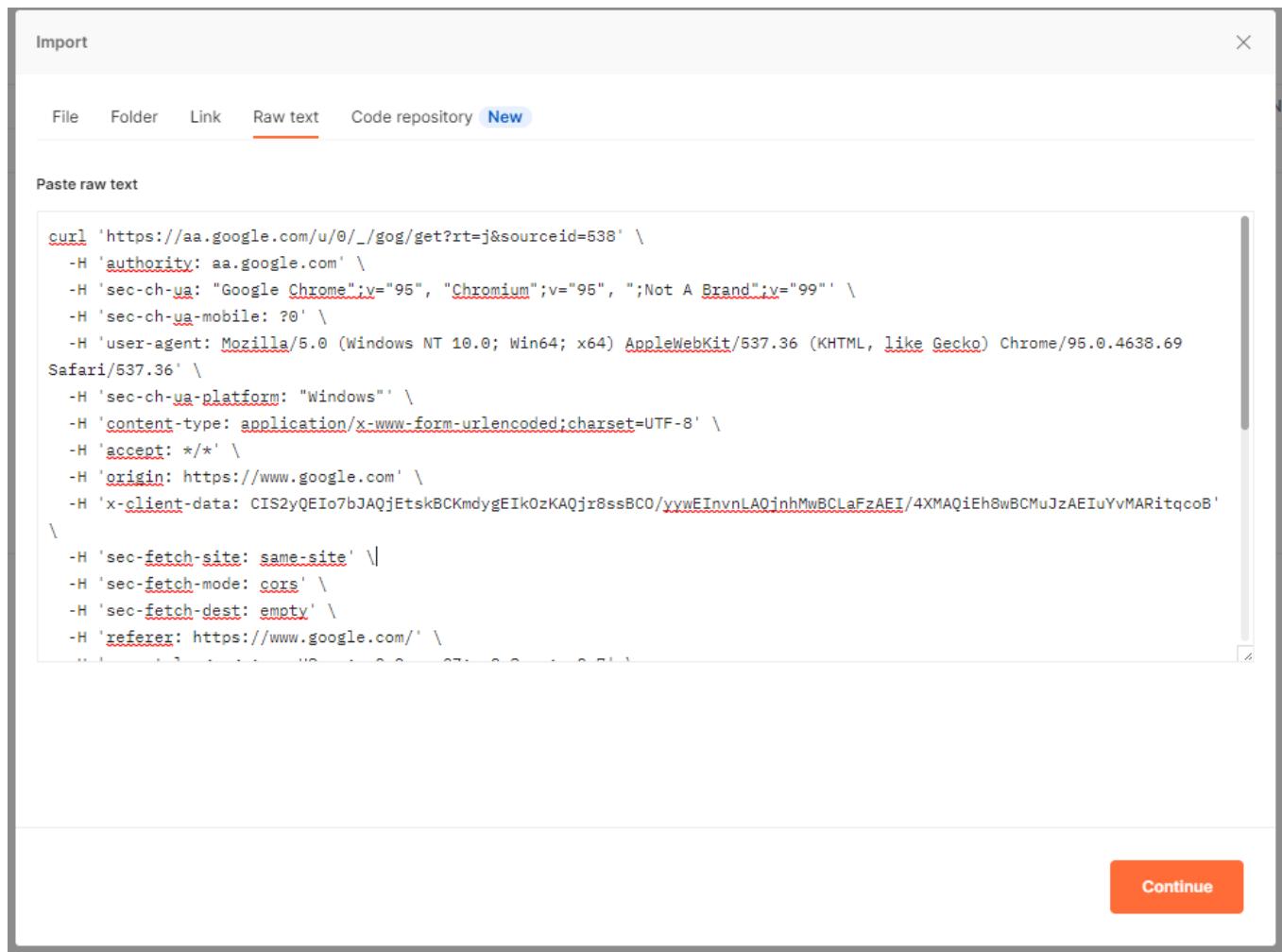
3. Otevřeme stránku google.com
4. V seznamu requestů bychom měli vidět request "get?...", klikneme na něj pravým tlačítkem, zvolíme "copy" a následně "Copy as cURL (bash)"



5. V Postman klikneme na tlačítko Import

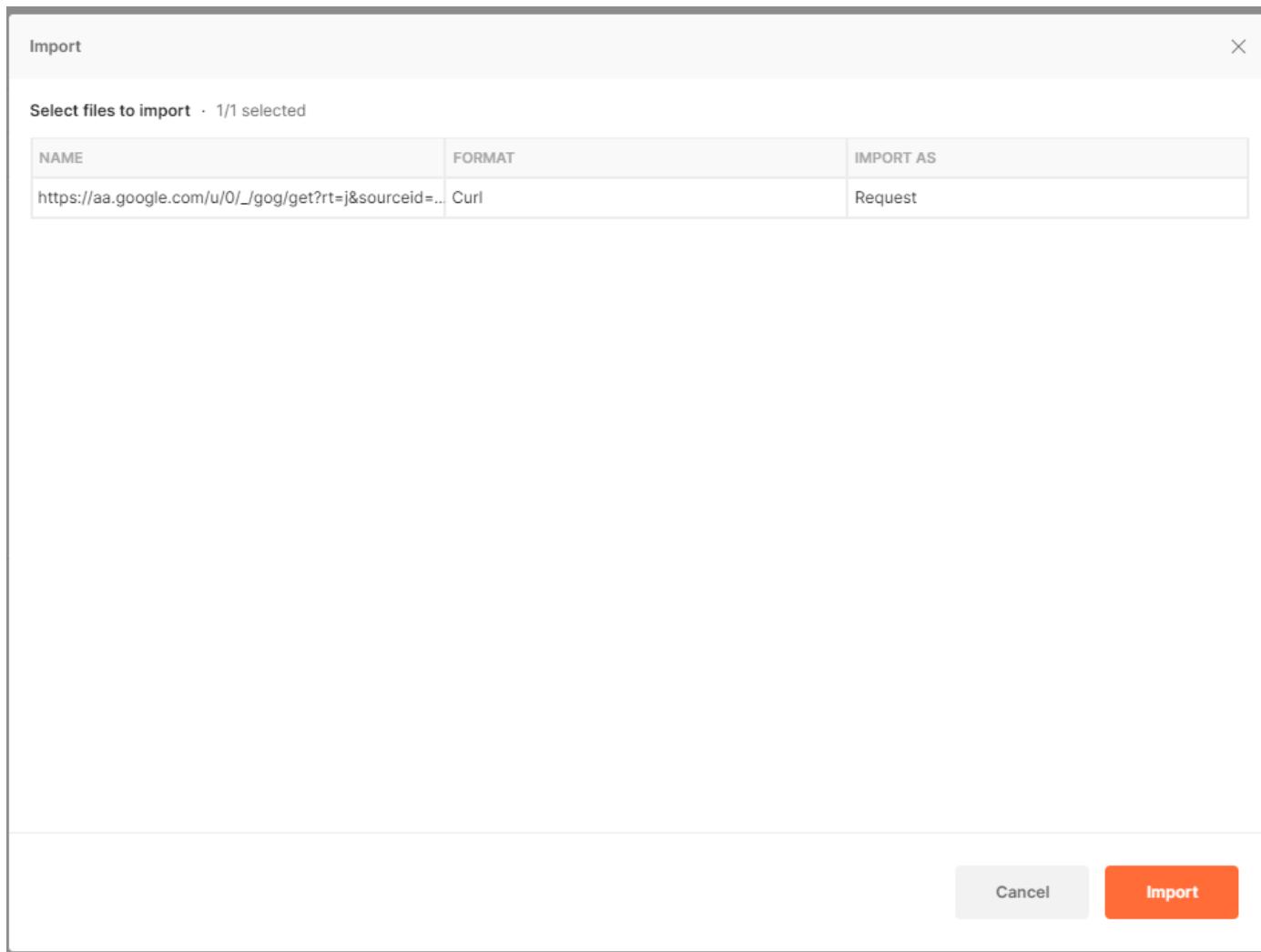


6. Vybereme záložku "Raw text" a vložíme do pole náš cURL, stiskneme tlačítko "Continue"



< TREDGATE >

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

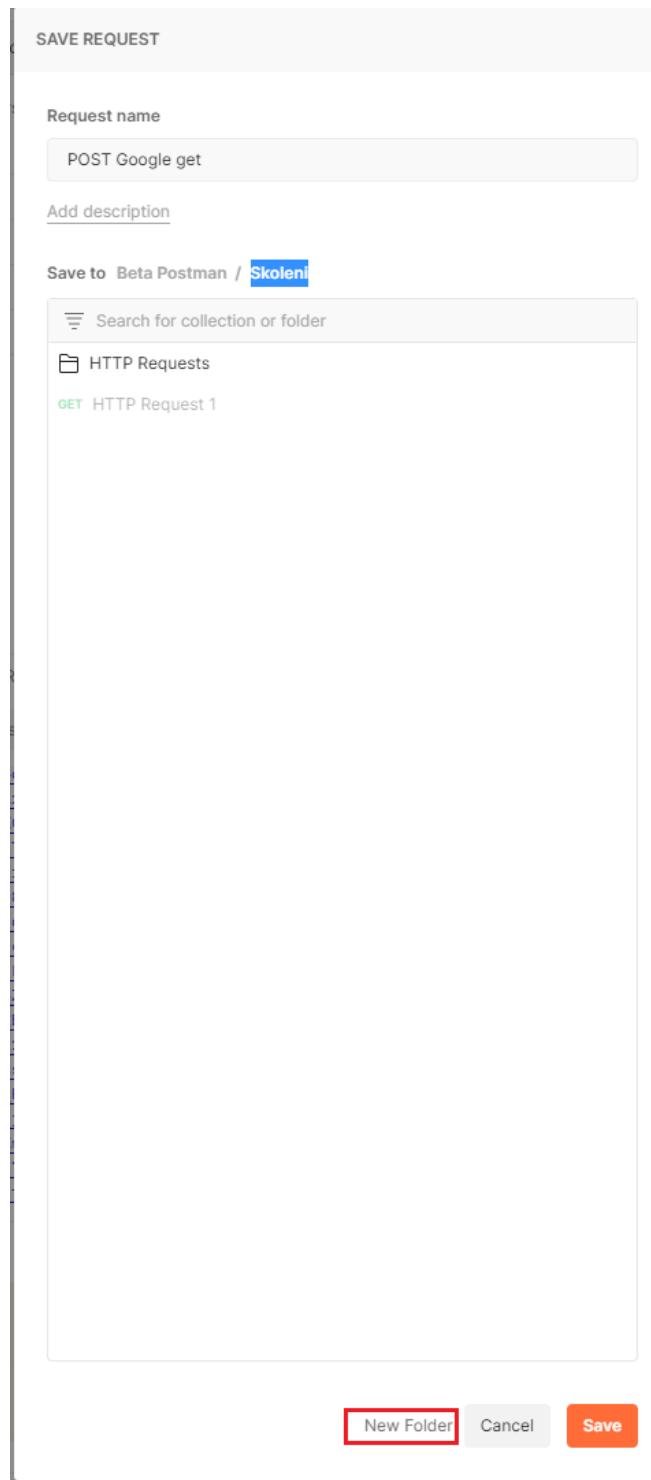


- Importovaný Request se nám zobrazí v novém Postman tabu (záložky v horní části). Oranžová tečka u našeho tabu znamená, že nemáme uložené změny.

The screenshot shows the Postman interface with a new tab open. The tab title is 'https://aa.google.com/u/0/_/gog/get?rt=j&sourceid=538'. The tab bar has an orange dot, indicating uncommitted changes. The tab content shows a POST request to the same URL. The 'Params' tab is selected, showing two parameters: 'rt' with value 'j' and 'sourceid' with value '538'. Other tabs include Authorization, Headers (24), Body, Pre-request Script, Tests, and Settings. A 'Send' button is visible at the top right of the request form.

- Request uložíme (CTRL + S)
- 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:

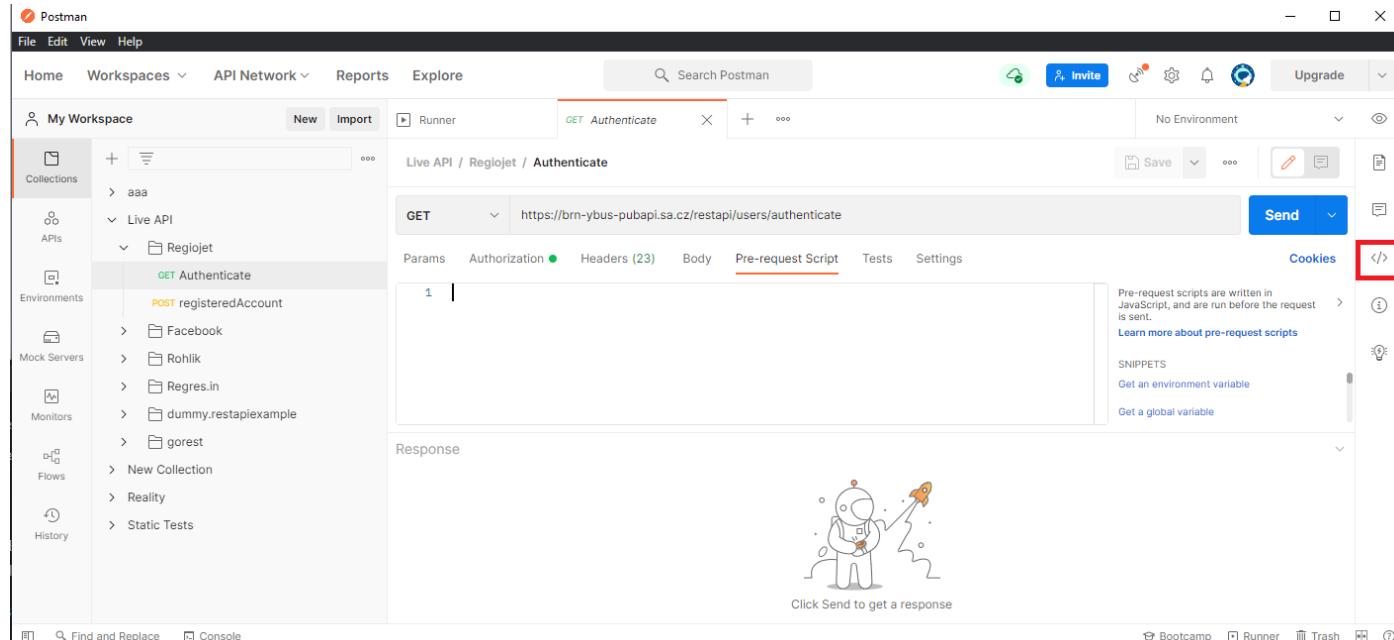
```
curl --location --request GET 'https://reqres.in/api/users' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
--header 'Referer: https://reqres.in/' \
--header 'sec-ch-ua-mobile: ?0'
```

```
--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'
```

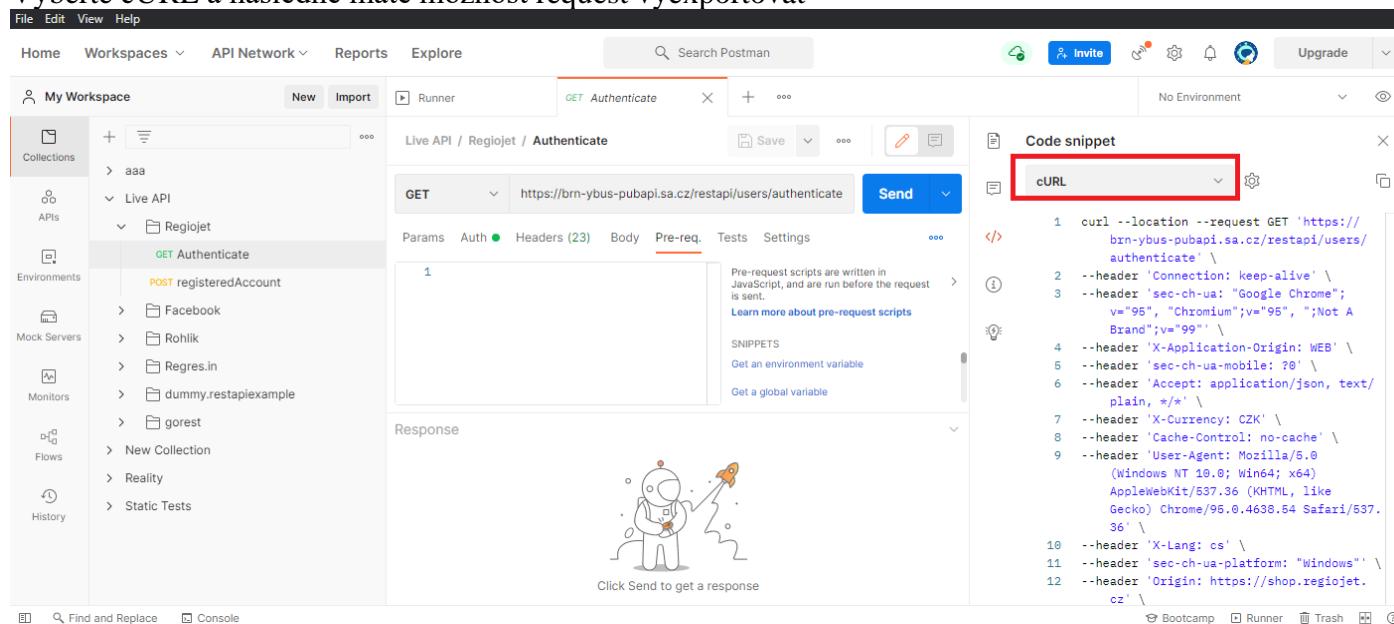
EXPORT REQUESTU V POSTMAN JAKO cURL

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



3. Vyberte cURL a následně máte možnost request vyexportovat



Úkol:

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

URL

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

File Edit View Help

The screenshot shows the Postman interface. On the left, there's a sidebar with sections for 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, with its URL highlighted in a red box: `https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate`. Below the URL, the 'Headers (23)' tab is selected. To the right of the request details, there's a 'Pre-request scripts' section with a note about JavaScript being run before the request is sent, and links to learn more or get snippets. At the bottom right, there's a cartoon character holding a rocket and the text 'Click Send to get a response'.

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.

The screenshot shows the Postman interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a collection named 'My Workspace' with a sub-collection 'Live API' containing a folder 'Regiojet' and a request 'GET Authenticate'. The request details show it's a 'GET' method pointing to <https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate>. A red box highlights the 'GET' method type. Below the request details is a 'Pre-request' section with a note about JavaScript scripts. At the bottom right, there's a cartoon illustration of an astronaut launching a rocket, with the text 'Click Send to get a response'.

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 'Content-Type: application/json'
```

```
--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

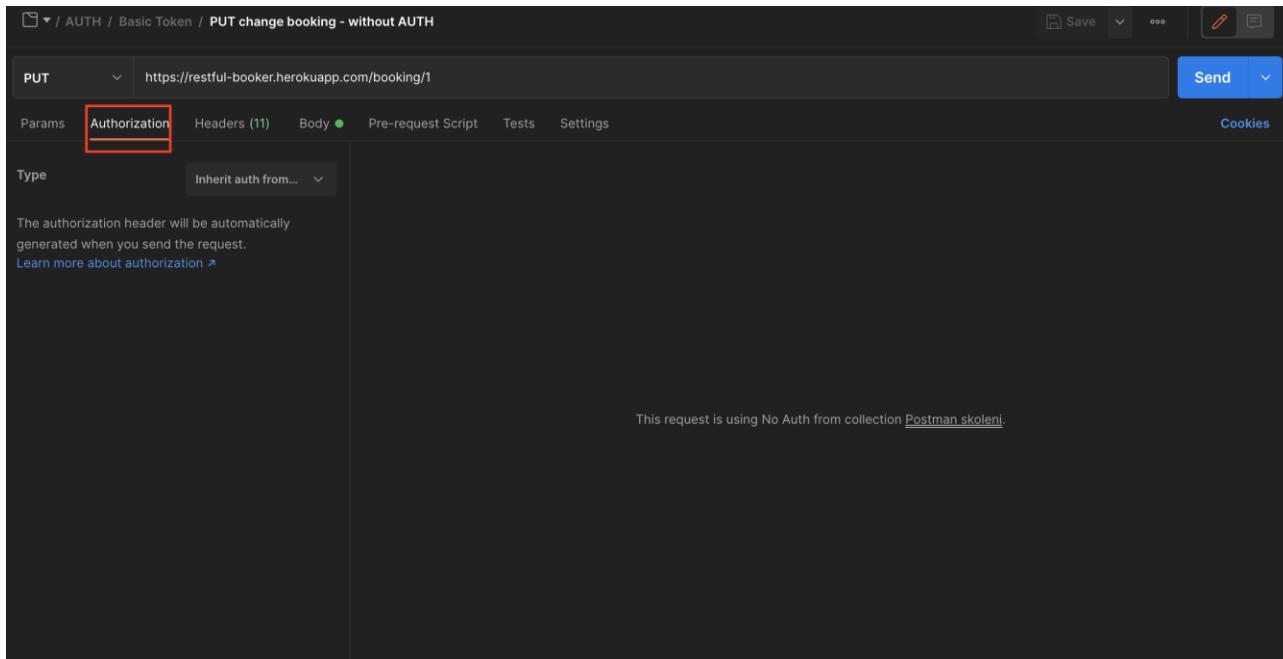
Autentikace je ověření volání requestu v případě, že to API vyžaduje (podobně jako přihlášení například do e-mailu). Existuje několik různých typů autentikace.

< TREDGATE >

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 PUT request to `https://restful-booker.herokuapp.com/booking/1`. The 'Authorization' tab is highlighted with a red box. The 'Type' dropdown is set to 'Inherit auth from...'. A note says: 'The authorization header will be automatically generated when you send the request.' Below it, a message states: 'This request is using No Auth from collection Postman skolení.'

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.

The screenshot shows the Postman interface with a GET request to `http://2qvvy.mocklab.io/json/1`. The 'Authorization' tab in the header section is highlighted with a red box. The status bar at the bottom right indicates "This request is using No Auth from collection Skoleni."

Vyberte Type: Basic Auth

The screenshot shows the Postman interface with a dropdown menu for selecting an authentication type. The 'Basic Auth' option is highlighted with a red box. The status bar at the bottom right indicates "This request is using No Auth from collection Skoleni."

Vložte:

- Username: userSkoleni
- Password: Test123

The screenshot shows the Postman interface with a request configuration for 'GET Mock JSON'. The 'Authorization' tab is selected, showing 'Basic Auth' as the type. The 'Username' field contains 'userSkoleni' and the 'Password' field contains 'Test123', both highlighted with a red box. A note at the top right of the form area says: '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](#)'.

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	http://2qvvy.mocklab.io/json
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" : {
```

< TREDGATE >

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

Key	Cookie
Value	token=2370cab9fc6706
Add 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"
  },
  "adults": 1
}
```

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

TOKEN AUTH

Vytvořte složku "Token" ve složce "Auth".

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

API Access Token

a69dd...

NOTE: This is your personal access token, do not share it with any one, do not publish this on any websites.

Regenerate Access Token

© Go Rest 2021 Contact Us Privacy Policy

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

< TREDGATE >

- 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 under 'Type' shows various authentication options, with 'Bearer Token' highlighted and surrounded by a red box. The status bar at the bottom right indicates '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 with the 'Token' field populated with the value 'a69d...'. A tooltip message at the top right of the interface reads: '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, 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:

[REDACTED]

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):

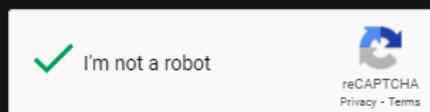
[REDACTED]

Email:

petrfifk@gmail.com

Description:

test



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	https://api.imgur.com/oauth2/authorize
Access Token URL	https://api.imgur.com/oauth2/token
Callback URL	https://imgur.com
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:

- Type:** OAuth 2.0
- Request URL:** https://api.imgur.com/oauth2/token
- Access Token:** Available Tokens dropdown contains "ee928db221212f9b8812d55de319de...". A yellow warning icon is present.
- Header Prefix:** Bearer
- Configure New Token:**
 - Configuration Options:** Selected
 - Advanced Options:** Unselected
 - 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
- Buttons:** Clear cookies, Get New Access Token

Stiskněte tlačítko Get New Access Token a přihlaste se do imguru (pokud Vám vyskočí okno s chybou, zkонтrolujte zadané 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.

The screenshot shows the 'Manage Access Tokens' section of an OAuth 2.0 interface. On the left, a sidebar lists 'All Tokens' and 'Imgur token'. The main area, titled 'Token Details', displays the following information for the selected token:

- Token Name: Imgur token
- Access Token: ee928db2...
- Token Type: bearer
- expires_in: 315360000
- scope: null
- refresh_token: e9e9e40b7af...
- account_id: 156602709
- account_username: petrfifk

A red button labeled 'Use Token' is visible in the top right corner.

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

The screenshot shows the 'Generate Access Token working' request in Postman. The 'Authorization' tab is selected. In the 'Current Token' section, the 'Type' dropdown is set to 'OAuth 2.0'. The 'Access Token' field has a dropdown menu labeled 'Available Tokens' which is open, showing the token 'Imgur token' selected. A red box highlights this selection. Below the dropdown, there is a 'Header Prefix' input field containing 'Bearer'.

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

The screenshot shows the 'Generate Access Token working' request in Postman. The 'Authorization' tab is selected. In the 'Current Token' section, the 'Type' dropdown is set to 'OAuth 2.0'. The 'Access Token' field has a dropdown menu labeled 'Available Tokens' which is open, showing the token 'Imgur token' selected. A red box highlights this selection. Below the dropdown, there is a 'Header Prefix' input field containing 'Bearer'.

V Body Requestu vyplňte:

- refresh_token

< TREDGATE >

- client_id
- client_secret

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

The screenshot shows the Postman interface with the following details:

- Request URL:** https://api.imgur.com/oauth2/token
- Method:** POST
- Body:** form-data (checkbox checked)
- 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	e179s[REDACTED]	The client secret obtained during application registration - ...
grant_type	refresh_token	As defined in the OAuth2 specification, this field must contain a val
- Response Headers:**
 - Status: 200 OK
 - Time: 480 ms
 - Size: 1.03 KB
- Response Body (Pretty JSON):**

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

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 following details:

- Left Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Beta Postman Tab:** New, Import.
- Current Collection:** Skoleni / Auth / OAuth.
- Authorization Tab:** Selected. Type: Inherit auth from parent. Note: The authorization header will be automatically generated when you send the request. Learn more about authorization.
- OAuth Sub-Collection:** OAuth (highlighted with a red box).
- Items in OAuth:**
 - GET GET imgur images
 - POST POST imgur auth
 - GET HTTP Request 1

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

Skoleni / 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 ⓘ

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

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

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

Type

- Inherit auth from ... ^
- Inherit auth from parent
- 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 [Skoleni](#).

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 "Skoleni", 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://regres.in/api/users?page=2' \
--header 'sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"' \
```

< TREDGATE >

```
--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, highlighted with a red box. Other tabs like 'Params', 'Authorization', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible but not selected. The 'Headers' table lists several key-value pairs:

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

V Requestu vidíme například tyto hlavičky:

Header	Význam
Referer	Slouží serveru k identifikaci odkud uživatel přichází
User-Agent	Slouží serveru k identifikaci aplikace, operačního systému...
Content-Type	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 'Live API / Regiojet / Authenticate' request. The 'Settings' tab is highlighted with a red box. The request details show a GET method and a URL: <https://brn-ybus-pubapi.sa.cz/restapi/users/authenticate>. Below the URL, there are several configuration options with toggle switches:

- Enable SSL certificate verification**: ON
- Automatically follow redirects**: ON
- 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)

At the bottom of the settings panel, there's a cartoon illustration of a character launching a rocket.

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.

The screenshot shows the Postman interface with a GET request to <https://reqres.in/api/users?page=2>. The 'Params' tab is selected, displaying the following headers:

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

Below the table, there are columns for 'Key', 'Value', and 'Description'. At the bottom of the interface, there are buttons for 'Save', 'Send', and 'Cookies'.

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 najeznlete klíče a hodnoty, které je potřeba vyplnit do parametrů.

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

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



Úkol:

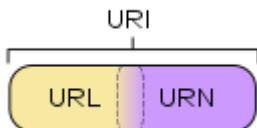
Upravte request následovně:

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"

POST https://brn-ybus-pubapi.sa.cz/restapi/users/login/registeredAccount/:something

Params Authorization Headers (25) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Path Variables

KEY	VALUE
something	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.

Skoleni / Params / GET regres.in users URI

GET https://reqres.in/api/users/:id

Save Cookies Send

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

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Path Variables

KEY	VALUE	DESCRIPTION	Bulk Edit
id	22	Description	

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 "Skoleni".

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"

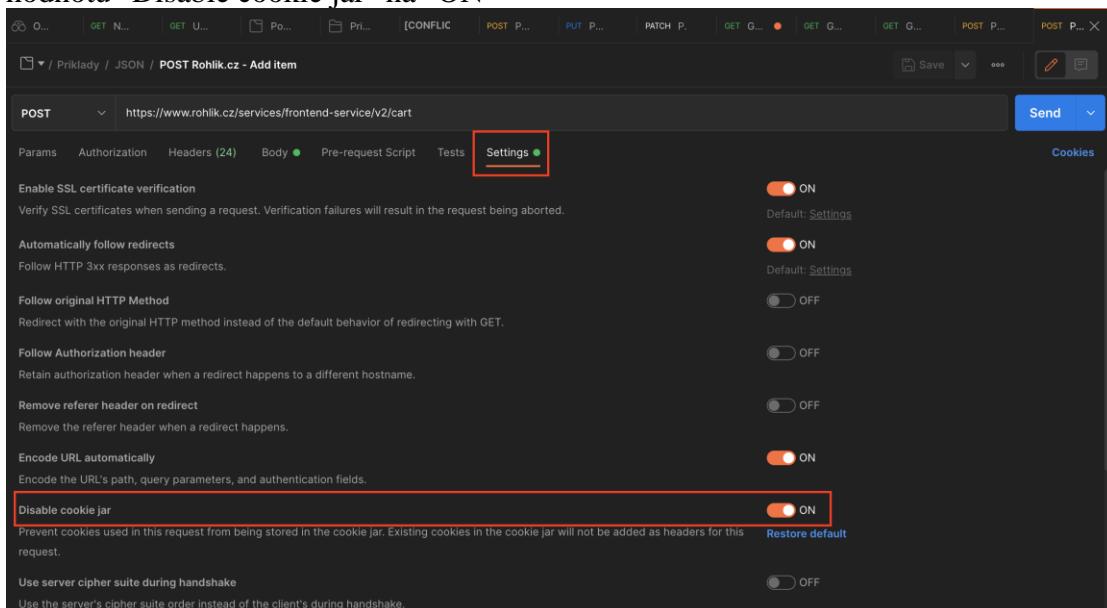
cURL:

```
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'
```

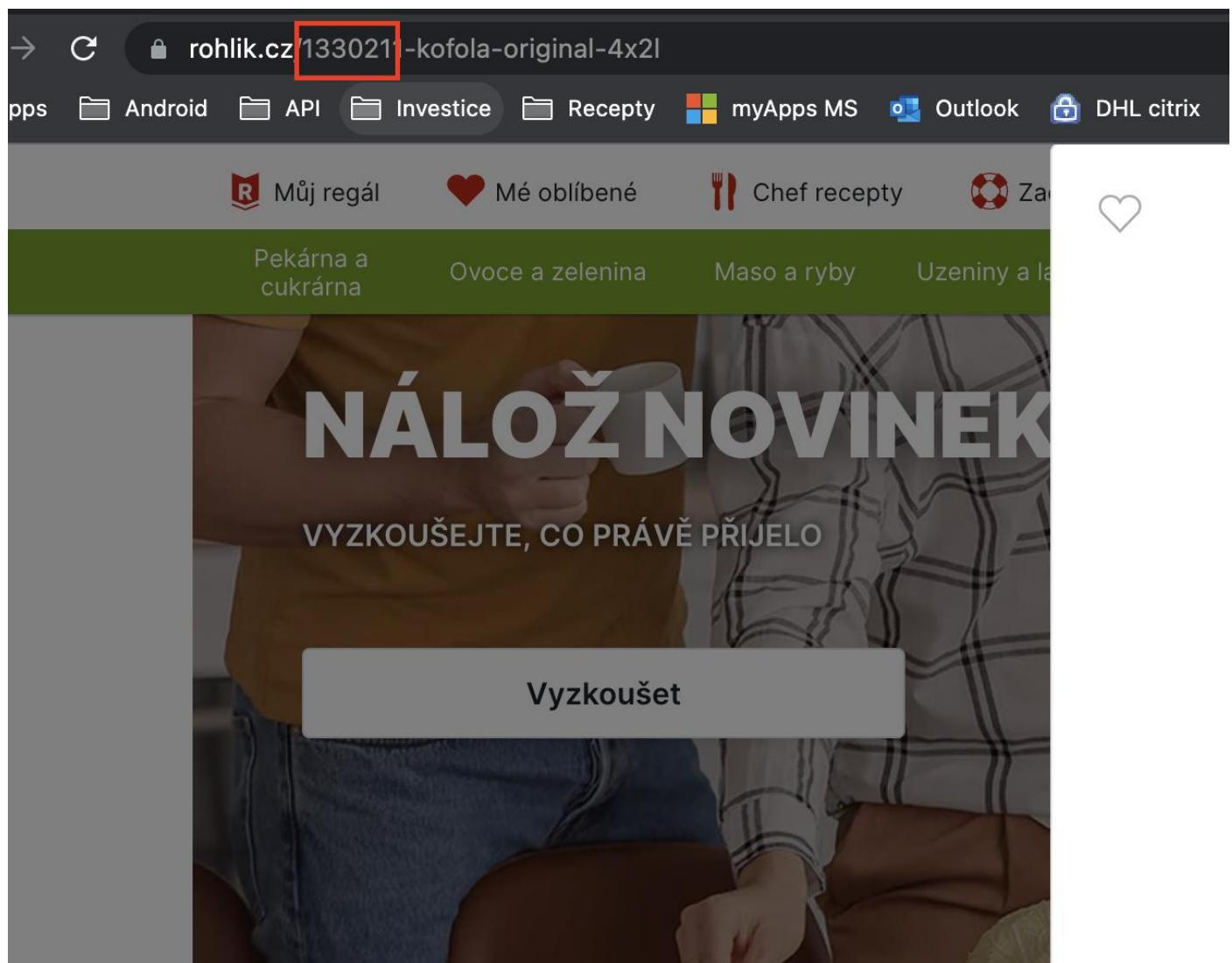
```
--header 'sec-ch-ua-platform: "Windows" \
--header 'origin: https://www.rohlik.cz' \
--header 'sec-fetch-site: same-origin' \
--header 'sec-fetch-mode: cors' \
--header 'sec-fetch-dest: empty' \
--header 'referer: https://www.rohlik.cz/c300102000-ovoce-a-zelenina' \
--header 'accept-language: en-US,en;q=0.9' \
--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"



Otevřete si stránku 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 Cookies Beautify

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

```

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

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"

< TREDGATE >

Create New X

Building Blocks

HTTP Request
 GET Create a basic HTTP request

Environment
 Save values you frequently use in an environment

WebSocket Request BETA
 Test and debug your WebSocket connections

Collection
 Save your requests in a collection for reuse and sharing

Advanced

API Documentation
 Create and publish beautiful documentation for your APIs

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

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

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

Monitor
 Schedule automated tests and check performance of your APIs

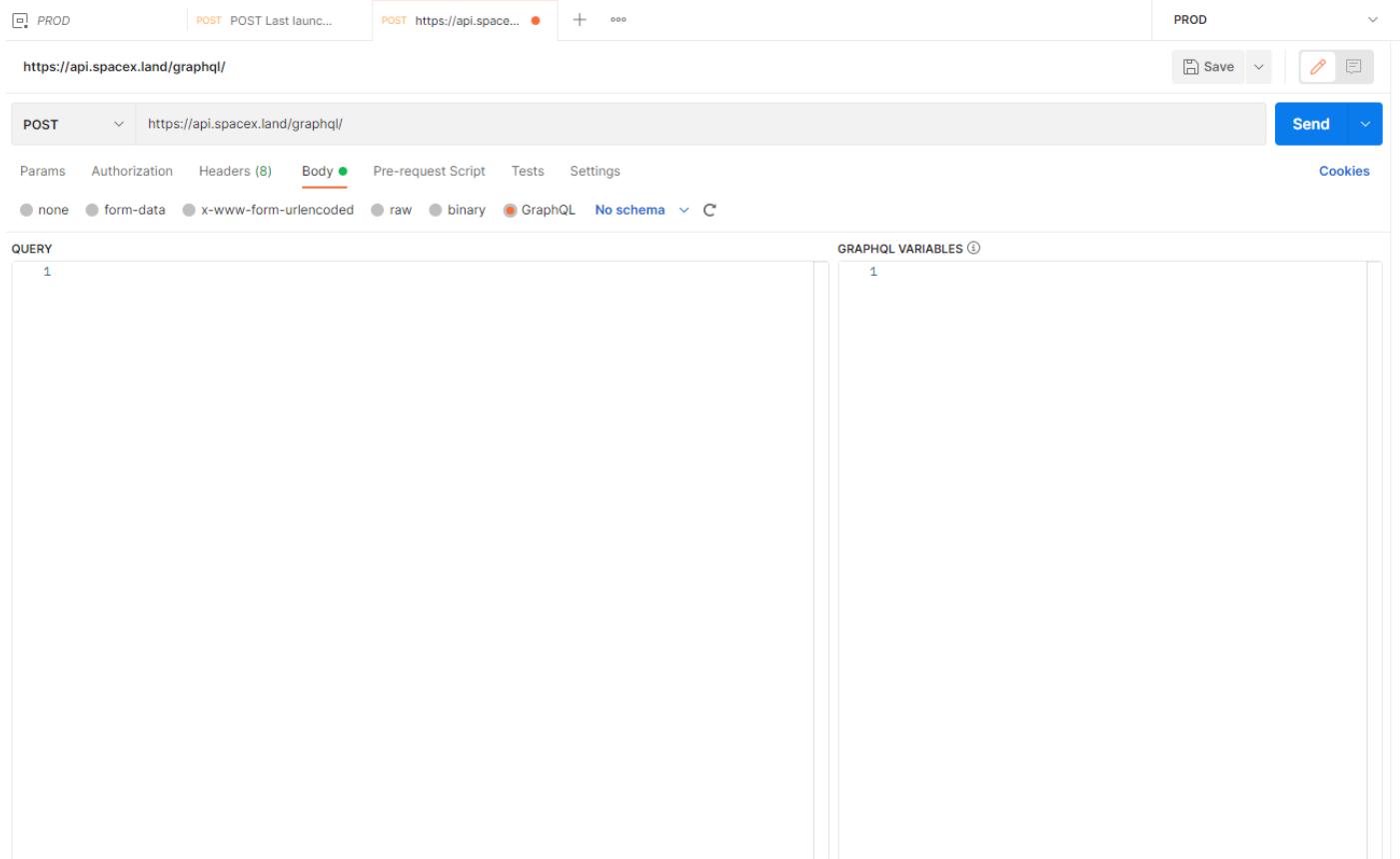
Not sure where to start? [Explore](#) featured APIs, collections, and workspaces published by the Postman community.

[Learn more on Postman Docs](#)

Typ Requestu: POST

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

Zvolte: Body/GraphQL



The screenshot shows the Postman application interface. At the top, there's a header bar with 'PROD' and a dropdown. Below it, the URL 'https://api.spacex.land/graphql/' is entered into the address bar. The main workspace shows a single request card for a POST method to the same URL. The 'Body' tab is selected under the 'Body' section of the request settings. The 'Params', 'Authorization', 'Headers (8)', 'Pre-request Script', 'Tests', and 'Settings' tabs are also visible. On the right side, there are sections for 'GRAPHQL VARIABLES' and 'C'. The bottom right corner has a 'Send' button.

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 (9) Test Results

Pretty Raw Preview Visualize JSON

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

```

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-measurements-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 }
```

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.wsdl>

Save Send

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

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

Status: 200 OK Time: 847 ms Size: 4.96 KB Save Response

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 & 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):

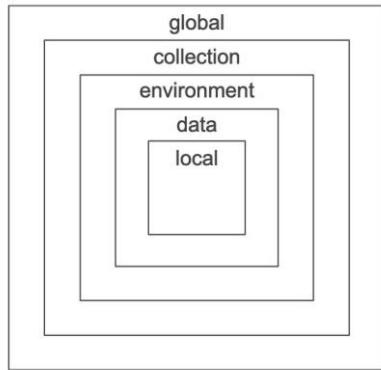
- Globals
- Environment

< TREDGATE >

- 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

The screenshot shows the Postman interface with the 'UAT' workspace selected. On the left, the sidebar shows 'Collections' (My Workspace, Global, APIs), 'Environments' (Mock Servers, Monitors, Flows, History), and 'Workspaces' (New, Import). In the center, under 'UAT', there is a table for variables. One row is selected, showing:

VARIABLE	INITIAL VALUE	CURRENT VALUE
abcd	aaa	aaa

The 'INITIAL VALUE' and 'CURRENT VALUE' columns are highlighted with red boxes.

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".

The screenshot shows the Postman interface with the 'UAT' environment selected. In the Globals section, there is one variable named 'abcd' with an initial value of 'aaa' and a current value of 'aaa'. The 'Persist All' button at the bottom right of the table is highlighted with a red box.

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

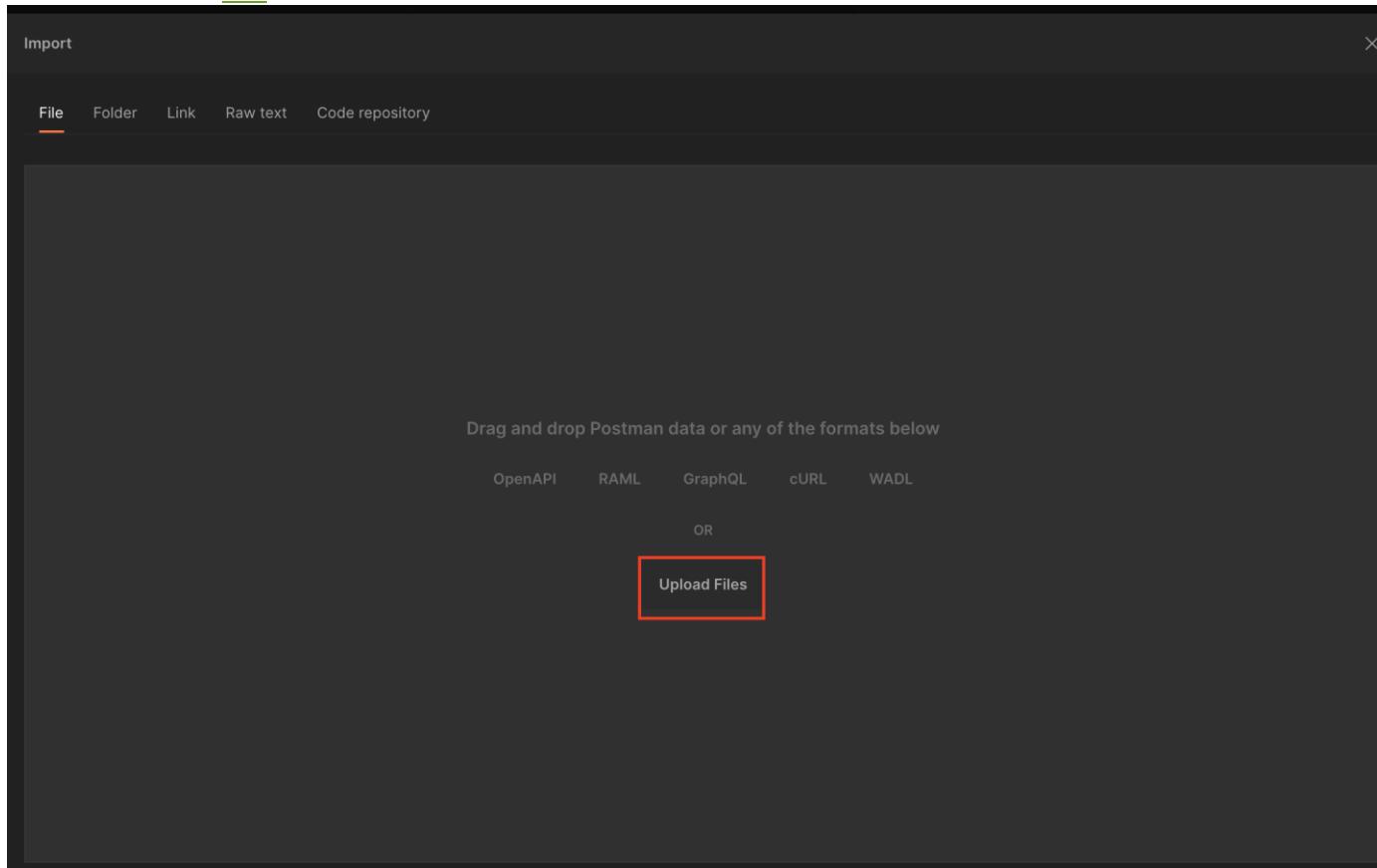
Globalní proměnné se importují pomocí import tlačítka. Musí se jednat o JSON, který je označený jako postman global variables.

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

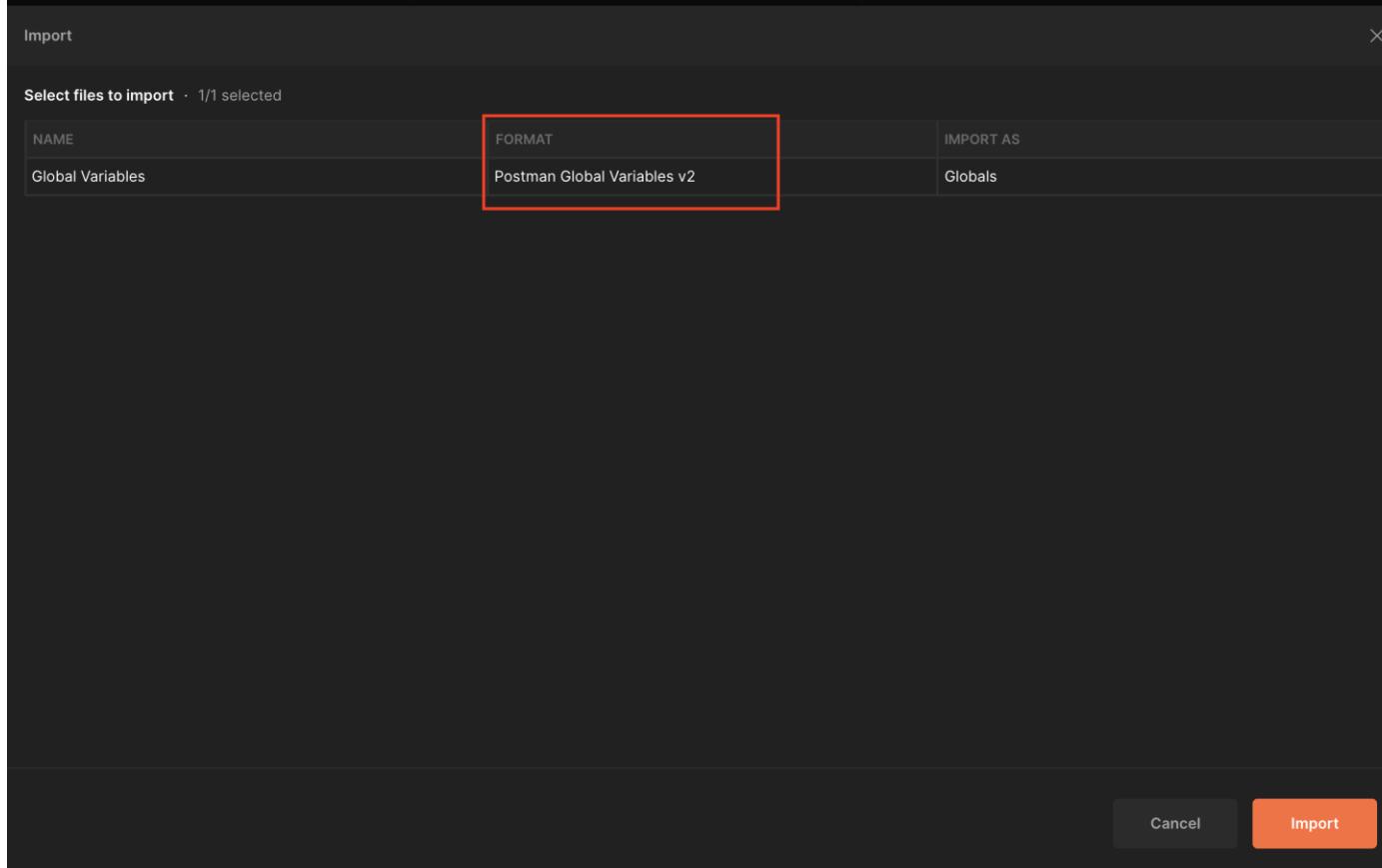
1. Klikněte na tlačítko na Import

The screenshot shows the Postman interface with the 'Import' tab highlighted in the top navigation bar. The left sidebar shows the 'Environments' section selected. The main area displays the 'Globals' section with variables 'aaa' and 'UAT'.

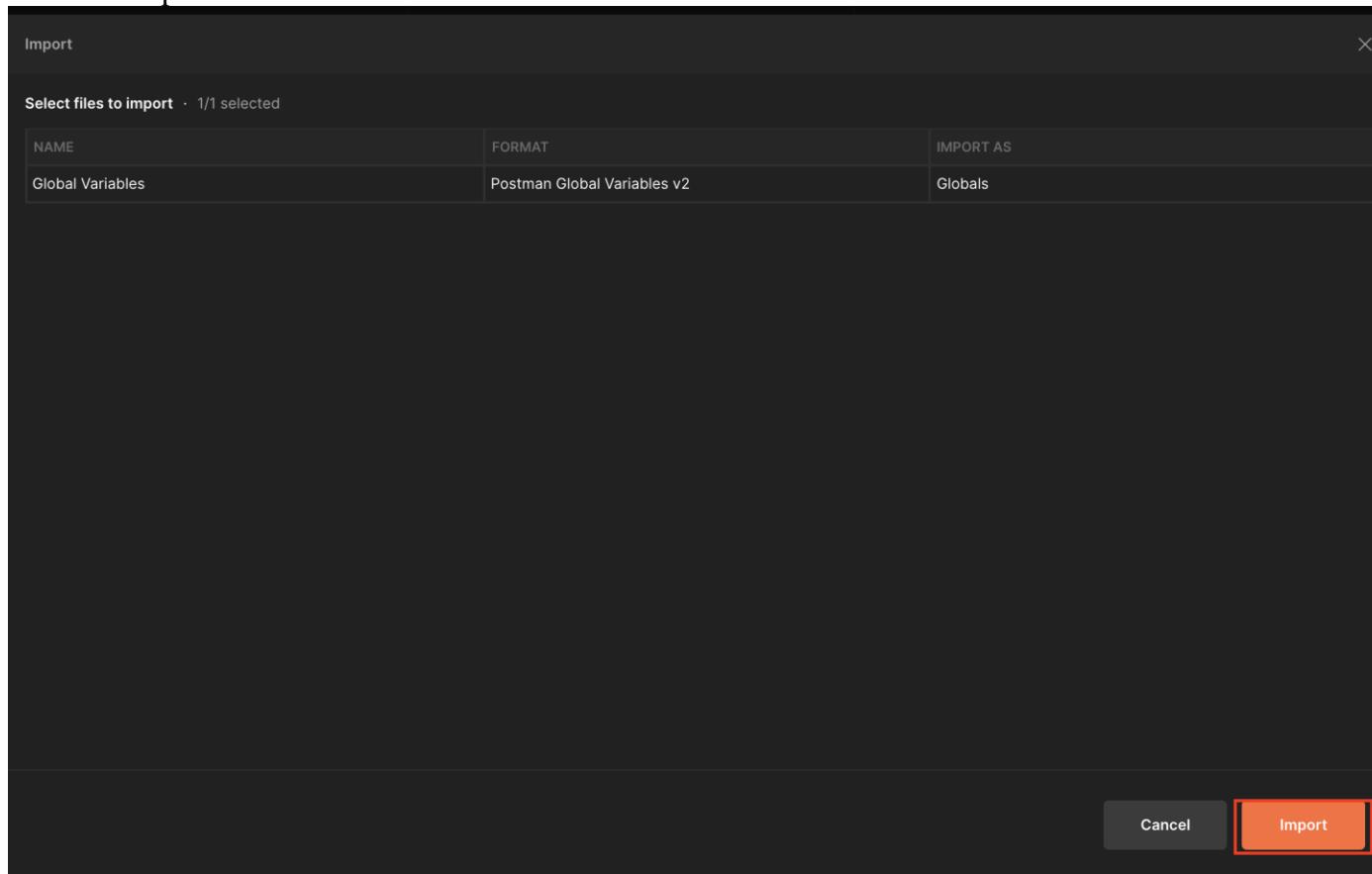
2. Klikněte na "Upload files" a vyberte JSON s globálními proměnnými. Soubor s globals proměnnými můžete stáhnout [zde](#).



3. Zkontrolujte, že se jedná o JSON s globals



4. Potvrďte import



EXPORT GLOBALS

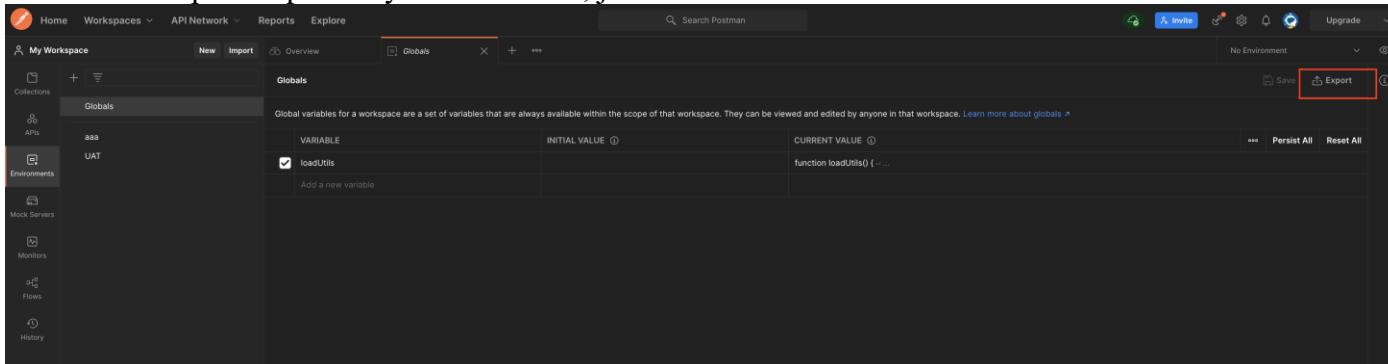
1. Vyberte Environments

The screenshot shows the Postman interface with the 'Environments' section selected in the sidebar. The main area displays a table of global variables. One variable, 'loadUtils', is selected, showing its initial value as 'function loadUtils()' and its current value as 'function loadUtils()'.

2. Vyberte Globals

The screenshot shows the Postman interface with the 'Globals' section selected in the sidebar. The main area displays a table of global variables. One variable, 'loadUtils', is selected, showing its initial value as 'function loadUtils()' and its current value as '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.

The screenshot shows the Postman interface with the title "Beta Postman". On the left sidebar, there are sections for "Collections", "JSON LS", "Environments" (which is highlighted with a red box), and "Mock Servers". The main area displays a list of environments: "Koiekce 1", "Kolekce 2", "Kolekce pro import", "Skoleni" (which is expanded to show "First calls", "Google", and "Regres.in"), and "Koiekce 3". On the right, there are tabs for "Globals" and "Params". Below the environment list, there is a search bar with "Skoleni / Var scripting" and a dropdown menu with "GET" and "http".

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

Klikněte na ikonu "+" vedle vyhledávače.

The screenshot shows the Postman interface with the title "Beta Postman". On the left sidebar, there are sections for "Collections", "APIs", and "Environments". The "Environments" section has a red box around the "+" icon next to the search bar. The main area shows a list of environments: "Koiekce 1", "Kolekce 2", "Kolekce pro import", "Skoleni" (expanded to show "First calls", "Google", and "Regres.in"), and "Koiekce 3". On the right, there is a search bar with "Skoleni / Var scripting" and a dropdown menu with "GET" and "http".

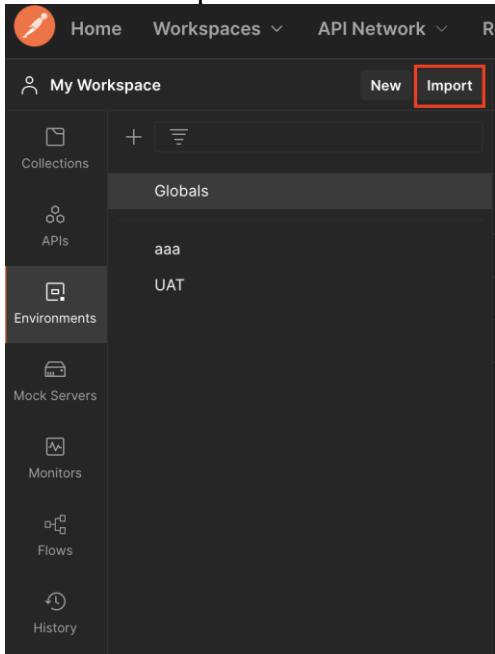
Prostředí pojmenujte PROD a uložte.

The screenshot shows the Postman interface with the title "Beta Postman". On the left sidebar, there are sections for "Collections", "APIs", "Environments" (highlighted with a red box), and "Flows". The "Environments" section has a red box around the "+" icon next to the search bar. The main area shows a list of environments: "Koiekce 1", "Kolekce 2", "Kolekce pro import", "Skoleni" (expanded to show "First calls", "Google", and "Regres.in"), and "Koiekce 3". A new environment named "PROD" is being created, with a red box around the input field. The interface includes tabs for "New", "Import", "Globals", "Graphs", "POST", "SOAP", "GET", "Parameters", "Variables", and "New ...".

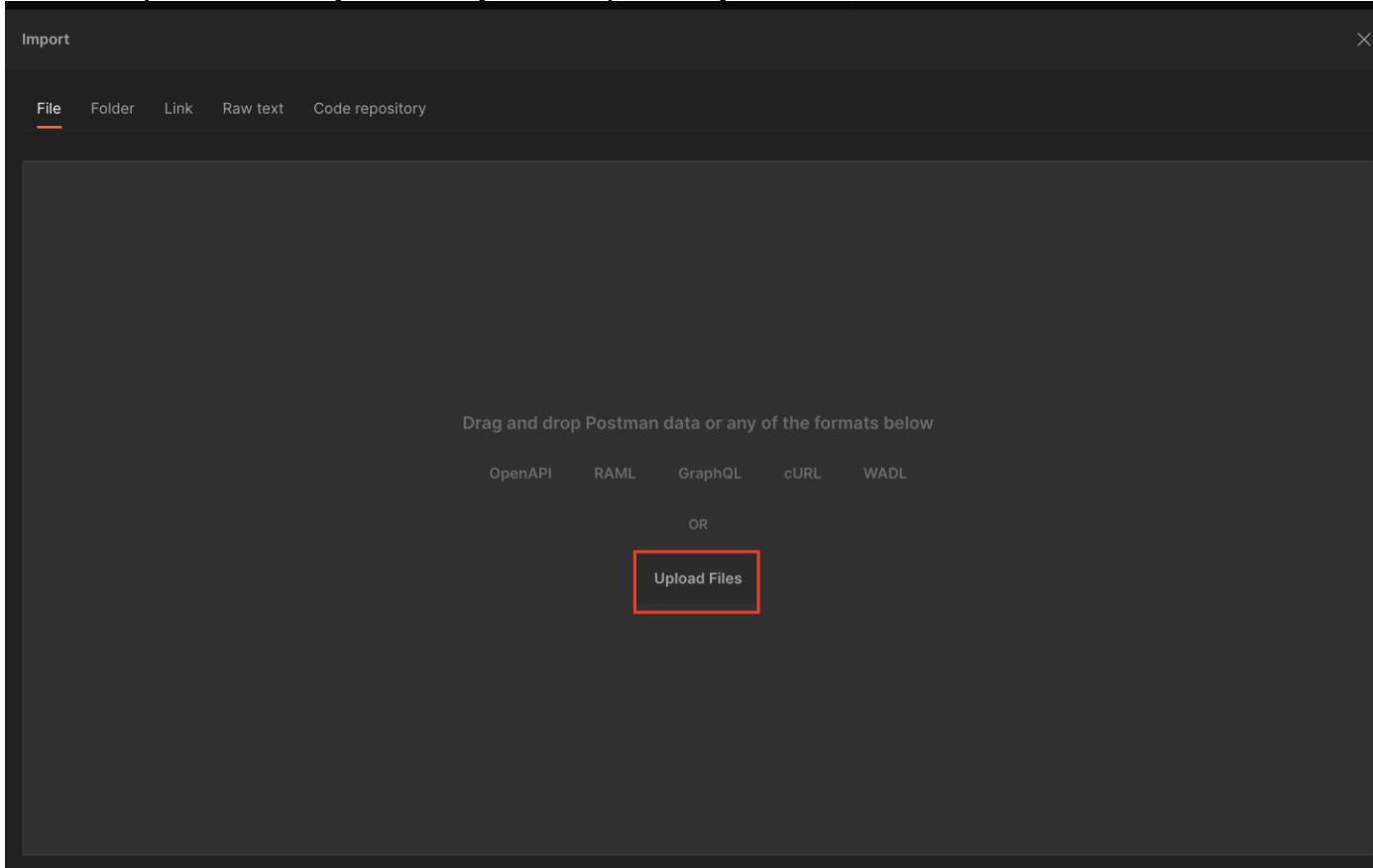
IMPORT PROMĚNNÝCH PRO PROSTŘEDÍ

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

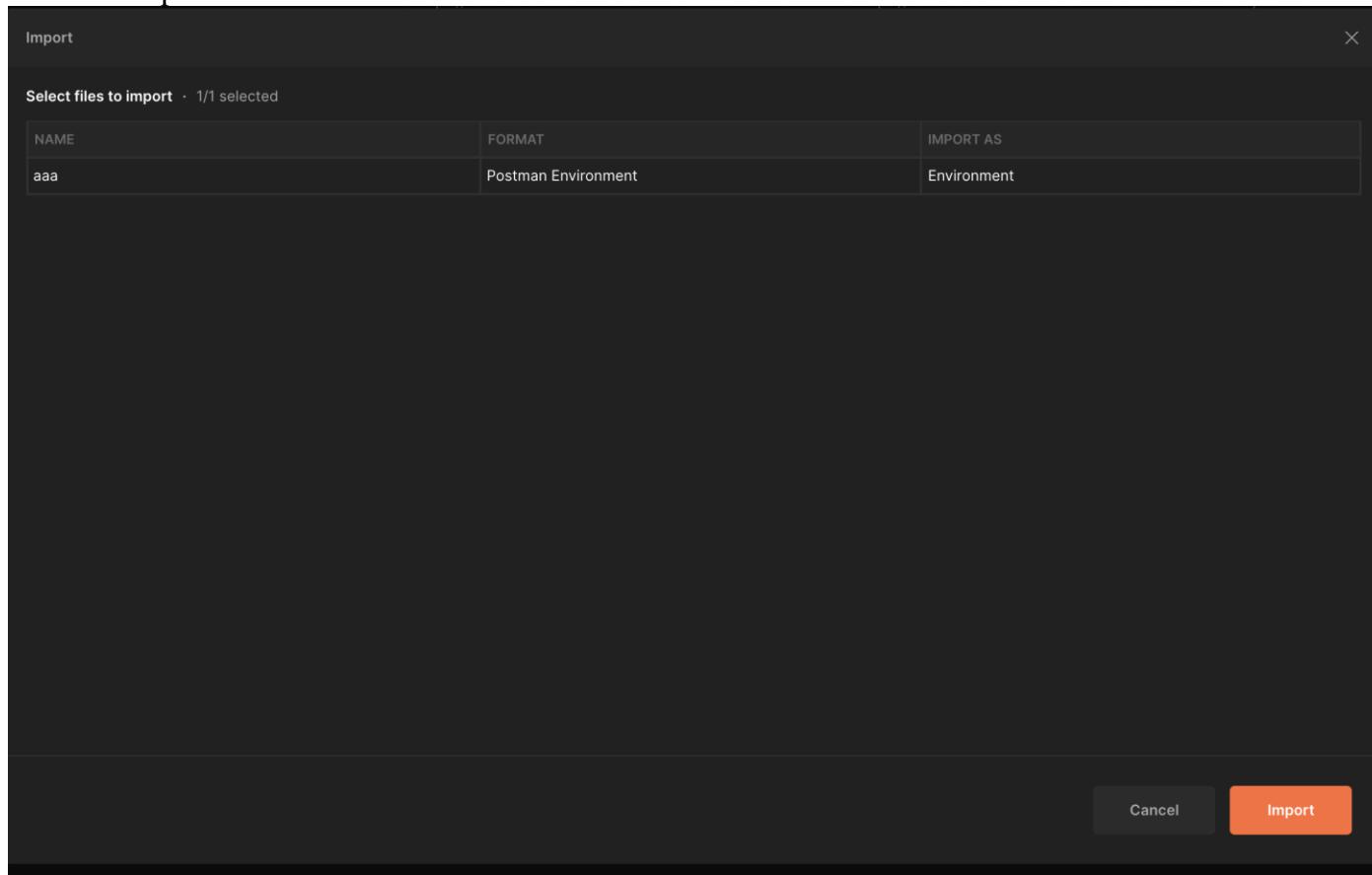
1. Klikněte na Import



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



3. Potvrďte import



EXPORT PROMĚNNÝCH PRO PROSTŘEDÍ

1. Vyberte Environments

VARIABLE	INITIAL VALUE	CURRENT VALUE
<input checked="" type="checkbox"/> loadUtils		function loadUtils() { ... }
Add a new variable		

2. Vyberte konkrétní prostředí

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
Add a new variable		

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
	Add a new variable		

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](#).

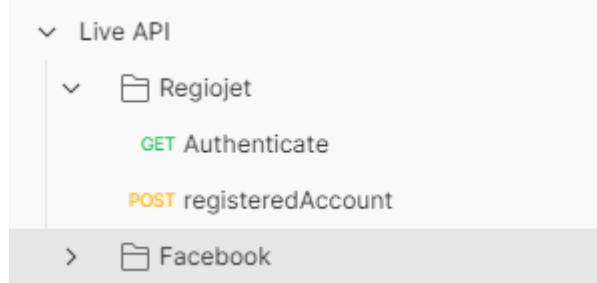
TYPY TESTŮ

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
Globals	pm.globals.get("variable_key");	pm.globals.set("variable_key", "variable_value");
Collection	pm.collectionVariables.get("variable_key");	pm.collectionVariables.set("variable_key", "variable_value");
Environment	pm.environment.get("variable_key");	pm.environment.set("variable_key", "variable_value");
Data	pm.iterationData.get("variable_key");	-
Local	pm.variables.get("variable_key");	pm.variables.set("variable_key", "variable_value");
Jakákoli vrstva	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 "Skoleni" a nazvete ji "Var scripting"

V ní vytvořte nový GET HTTP Request, URL: <https://google.com>, nazvete 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. In the main area, there is one line of code: '1'. On the right side, there is a sidebar with a list of snippets:

- Pre-request script: JavaScript, and a variable is sent.
- [Learn more about this snippet](#)
- 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](#)

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 a 'Beta Postman' workspace. On the left sidebar, 'Environments' is selected. In the main area, a collection named 'Skoleni / Var scripting / GET google' is shown. A GET request is configured to 'https://google.com'. 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.

The screenshot shows the Postman Globals panel for the 'PROD' environment. It lists three variables: 'Test1' (initial value: 'test', current value: 'testěěě'), 'Test2' (initial value: 'test2', current value: 'test2'), and 'globalsVar1' (initial value: '', current value: 'globální proměnná'). The 'PROD' environment is highlighted with a red box.

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 so 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= shp&hl=cs&src=1 200 | 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= shp&hl=cs&src=1 200 | 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

+ Global PROD

VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
Test1	test	testééé
Test2	test2	test2
globalsVar1		globální proměnná

Add a new variable

Save Export

*** Persist All Reset All

Otevřete prostředí PROD v enviroments a zkontrolujte vytvořenou proměnnou

The screenshot shows the Postman interface with the 'Beta Postman' tab selected. In the left sidebar, under 'Environments', the 'PROD' environment is selected. In the main workspace, there is a table for variables. A row for 'environmentVar1' is selected, showing its initial value as 'proměnná prostředí'. There are buttons for 'Fork', 'Save', 'Share', 'Persist All', and 'Reset All' at the top right of the table.

Otevřete kolekci Skoleni, její proměnné a zkontrolujte vytvořenou proměnnou.

The screenshot shows the Postman interface with the 'Beta Postman' tab selected. In the left sidebar, under 'Collections', the 'Skoleni' collection is selected. In the main workspace, there is a table for variables. A row for 'collectionVar1' is selected, showing its initial value as 'proměnná kolekce'. There are buttons for 'Share', 'Run', 'Persist All', and 'Reset All' at the top right of the table.

PRE-REQUEST SCRIPT

Se vyhodnuje 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.

< TREDGATE >

The screenshot shows the Postman interface with a GET request to `https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocat...`. The 'Params' tab is selected, showing the following query parameters:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
tariffs	REGULAR			
toLocationType	CITY			
toLocationId	2147875000			
fromLocationType	CITY			
fromLocationId	10202003			
departureDate	<code>{{depDate}}</code>			

Below the table, there's a 'Key' column, a 'Value' column, and a 'Description' column.

Response

Do Pre-Request skriptu vložte následující kód.

```
pm.variables.set("depDate", "2022-03-30");
```

Následně request provolejte.

Kontrola v konzoli:

```
curl https://brn-ybus-pubapi.sa.cz/restapi/routes/search/simple?tariffs=REGULAR&toLocationType=CITY&toLocationId=2147875000&fromLocationType=CITY&fromLocationId=10202003&departureDate=2021-11-30
```

The terminal output shows a curl command with a red box around the `departureDate` parameter value `2021-11-30`.

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

< TREDGATE >

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 'Tests' tab is selected, displaying a JavaScript test script:

```
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. The first test, 'Status code is 200', is marked as PASS. The second test, '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 tests listed include 'Content-Type is present' (PASS), 'Response time is less than 200ms' (PASS), 'Successful POST request | Assertion: expected 200 to be one of [201, 202]' (FAIL), and 'Status code name has string | Assertion: expected response to have status reason 'Created' but got 'OK'' (FAIL).

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, z jednoho uložíme ID do proměnné a následně použijeme v druhém requestu. V testu následně zkонтrolujeme, 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 a69dd3dc892e72c9b44fe0843a0e8171c7cfa618775dd8e96e8295895219f5fd'
```

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

Provoláme 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 a69dd3dc892e72c9b44fe0843a0e8171c7cfa618775dd8e96e8295895219f5fd' \
--data-raw '{
    "name": "Dixie Effertz",
    "gender": "male",
    "email": "Deja75@example.com",
    "status": "active"
}'
```

< TREDGATE >

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.

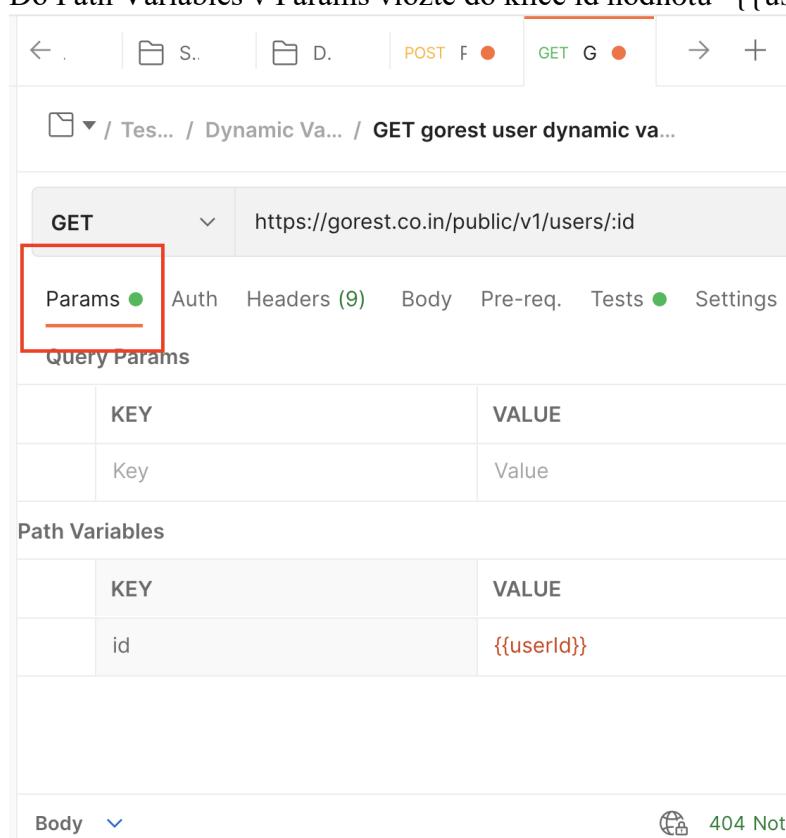
Složka: Skoleni/Tests/Dynamic Vars

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, showing a table with a single row: 'Key' (id) and 'Value' ({{userId}}). The 'Path Variables' section below it also shows a table with a single row: 'Key' (id) and 'Value' ({{userId}}). The status bar at the bottom right shows '404 Not'.

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
years	y
quarters	Q
months	M
weeks	w
days	d
hours	h
minutes	m
seconds	s
milliseconds	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 POST request to `https://gorest.co.in/public/v1/users`. The request body contains a JSON object with fields: name, gender, email, and status. The response body shows the created user record with id 6417 and the same fields as the request body.

Request Headers:

- Accept: "application/json"
- Content-Type: "application/json"
- Authorization: "Bearer a69dd3dc892e72c9b44fe0843a0e8171c7cfa618775dd8e96e8295895219f5fd"
- User-Agent: "PostmanRuntime/7.28.4"
- Cache-Control: "no-cache"
- Postman-Token: "bc0462b3-332f-444e-9832-e4b5d68f4e9"
- Host: "gorest.co.in"
- Accept-Encoding: "gzip, deflate, br"
- Connection: "keep-alive"
- Content-Length: "137"

Request Body

```
{
  "name": "Tenali Ramakrishna22a",
  "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/\"2efc934781cc672fd0877e0e189608fe"
- Cache-Control: "max-age=0, private, must-revalidate"
- X-Request-Id: "946e2621-0784-4970-9969d49344ab"
- X-Runtime: "0.019217"
- Strict-Transport-Security: "max-age=63072800; includeSubDomains"
- Vary: "Origin"

Response Body

```
{"meta":null,"data":[{"id":6417,"name":"Tenali Ramakrishna22a","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 a69dd3dc892e72c9b44fe0843a0e8171c7cfa618775dd8e96e8295895219f5fd"
    User-Agent: "PostmanRuntime/7.28.4"
    Cache-Control: "no-cache"
    Postman-Token: "bc0462b3-332f-444e-9832-e4b5d68f44e9"
```

201 946 ms

Show raw log

Příklad Raw logu:

```
POST /public/v1/users HTTP/1.1
Accept: application/json
Content-Type: application/json
Authorization: Bearer a69dd3dc892e72c9b44fe0843a0e8171c7cfa618775dd8e96e8295895219f5fd
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: 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/"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:

< TREDGATE >

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

⚠ "Toto je warning"

"Standardní log"

ⓘ "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 interface with a search bar at the top left and a dropdown menu titled 'All Logs' with four options: Log, Info, Warning, and Error. The 'Warning' option is selected. Below the dropdown, there is a list of logs from a GET request to https://reqres.in/api/users. The logs include standard network and request headers, followed by response headers. At the bottom left, there are error and warning counts: 3 errors and 1 warning. At the bottom right, there are checkboxes for 'Show timestamps' and 'Hide network'.

PŘEPOUŽÍVÁNÍ KÓDU (BONUS HACK)

Postman bohužel oficiálně neumožňuje 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.

< TREDGATE >

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

The screenshot shows a collection tree in Postman. The root node 'aaa' has a red box around it. Under 'aaa', there is a node 'Live API' which is also highlighted with a red box. This node has three sub-folders: 'Regiojet', 'Facebook', and 'Rohlik'. 'Regiojet' contains two requests: 'GET Authenticate' and 'POST registeredAccount'. 'Facebook' is described as 'This folder is empty' and has a link 'Add a request' to start working. 'Rohlik' is also empty.

2. Otevřít záložku *Pre-request Script*

The screenshot shows the Postman interface with the 'Live API' collection selected. The 'Pre-request Script' tab is highlighted with a red box. The left panel shows a code editor with a single line of code: '1'. The right panel contains documentation for pre-request scripts, including a note that they are written in JavaScript and run before each request, and a list of available snippets:

- 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

< TREDGATE >

Následující kód umožní vytvářet vlastní přepoužitelné funkce:

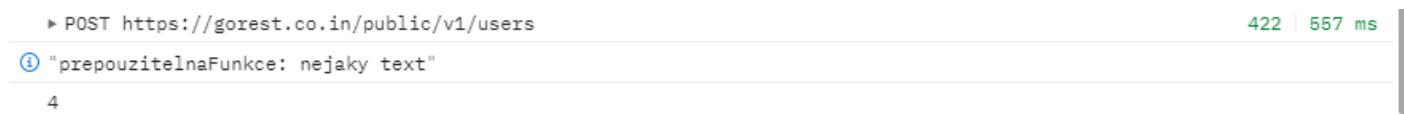
```
pm.globals.set('loadUtils', function loadUtils() {
    let utils = {};
    utils.prepouzitelnaFunkce = function prepouzitelnaFunkce(text) {
        console.info("prepouzitelnaFunkce: " + text)
    };
    utils.sectiDveCisla = function sectiDveCisla(prvniCislo, druheCislo) {
        vysledek = prvniCislo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

```
1 var data = pm.response.json();
2
3 var rohlikPriceSum = pm.variables.get("rohlikPriceSum")
4
5 pm.test("Total price is " + rohlikPriceSum, () => {
6     pm.expect(data.data.totalPrice).to.equal(rohlikPriceSum);
7 })
```

< TREDGATE >

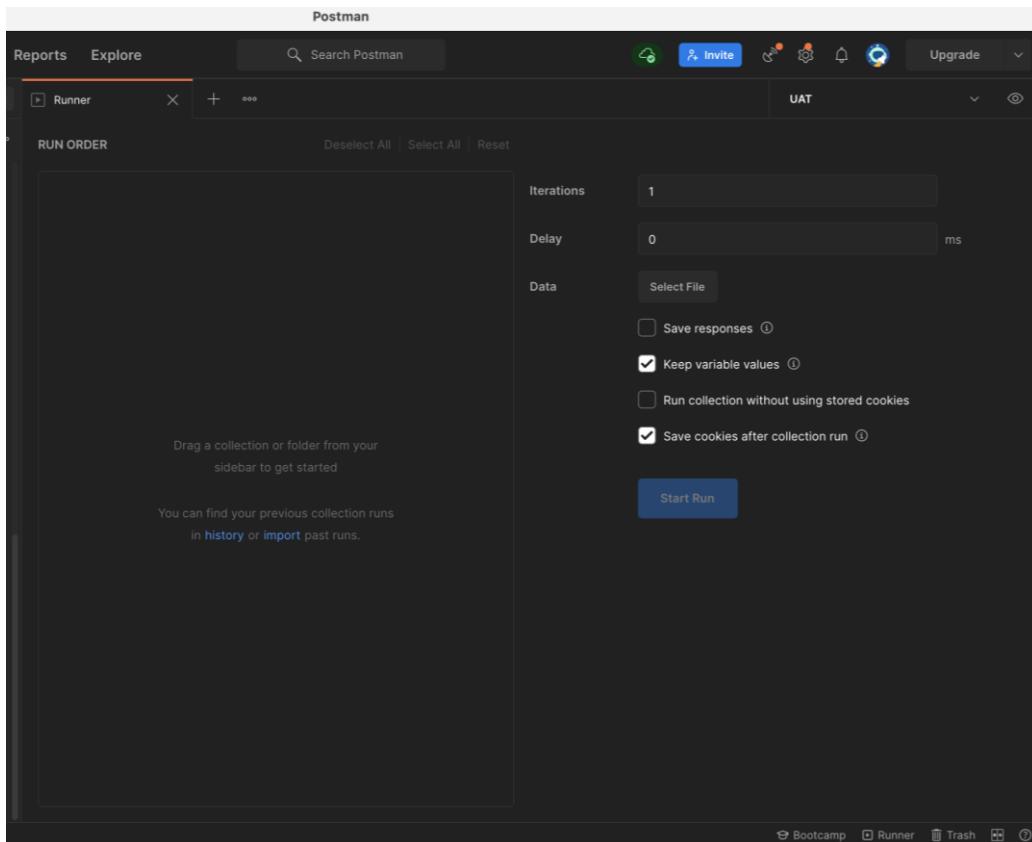
Otevře se okno v tabech, odkud následně spouštíme requesty.

The screenshot shows the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', 'API Network', 'Reports', 'Explore', a search bar 'Search Postman', and various icons for 'Invite', 'Settings', 'Notifications', and 'Upgrade'. Below the navigation is a toolbar with 'New', 'Import', 'Runner' (which is highlighted with a red box), and other options. On the left sidebar, there are sections for 'My Workspace' (Collections, APIs, Environments, Mock Servers, Monitors, Flows, History), 'Find and Replace', and 'Console'. The main area is titled 'RUN ORDER' and contains a list of API requests under a collection named 'Runner'. The requests include: 'Předávání variabls' (GET, GET Users - save variable, GET, GET User - ID as var), 'DYNAMIC VARS' (POST, POST USER - DYN VAR, GET, GET User - ID as var), 'GENERAL RUN' (POST, POST Confirm, POST, POST Check, POST, SET set-and-delete, POST, POST Cart, GET, GET Check cart), and two entries for 'Rohlik add item to cart' (POST, POST Confirm, POST, POST Check, POST, SET set-and-delete, POST, POST Cart 1, POST, POST Cart 2, POST, POST Cart 3, GET, GET Check cart). To the right of the list are configuration options: 'Iterations' (set to 1), 'Delay' (set to 0 ms), 'Data' (button to 'Select File'), and 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 located at the bottom right of the configuration area. At the very bottom of the interface, there are links for 'Bootcamp', 'Runner', 'Trash', and a refresh icon.

KONFIGURACE RUNNERU

V runneru nastavujeme

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

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>

< TREDGATE >

Následně nahraďte začátek URL touto proměnnou ve všech requestech.

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

Spustíme 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 'Runner' tab. On the left side of the runner, under 'My Workspace', there's a collection named 'Postman skolení'. This collection contains several requests: 'POST Confirm', 'POST Check', 'POST SET set-and-delete', 'POST POST Cart', and 'GET GET Check cart'. Below these are sections for 'Předávání variálek' (variables) and 'DYNAMIC VARS'. To the right of the collection list are settings for 'Iterations' (set to 1), 'Delay' (0 ms), and 'Data' (with a 'Select File' button). There are also 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 runner area is a large blue button with white text that says 'Run Postman skolení', which is also highlighted with a red box.

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 'Test Results' window from Postman. It displays the results of the 'Postman skolení' collection run. The window has tabs for 'View Summary', 'Run Again', 'New', and 'Export Results'. The summary table shows 1 iteration, 1 test passed, and 1 failed. The detailed view shows five requests:

- POST POST Cart**: Status 200 OK, 115 ms, 1.307 KB. Result: Failed (1)
- POST POST Confirm**: Status 200 OK, 115 ms, 1.307 KB. Result: Passed (0)
- POST POST Check**: Status 200 OK, 113 ms, 1.456 KB. Result: Failed (1)
- POST SET set-and-delete**: Status 200 OK, 80 ms, 1.511 KB. Result: Failed (1)
- GET GET Check cart**: Status 200 OK, 144 ms, 3.861 KB. Result: Failed (1)

For the failed GET request, the details show: 'Fail Total price is 6.97 | Assertion Error: expected 84.99 to equal 6.97'. The entire results table is highlighted with a red box.

- Exekuované requesty a výsledky jejich testů

The screenshot shows the Postman interface with the following details:

- Environment:** Postman skolení UAT, just now
- Iteration:** Iteration 1
- Test Status:** Failed (1)
- Requests:**
 - POST POST Confirm**: https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm / Priklady / GENERAL RUN / Rohlik add item to cart / POST Confirm - Status: 200 OK, 115 ms, 1.307 KB. Note: 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 - Status: 200 OK, 113 ms, 1.456 KB. Note: 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 - Status: 200 OK, 80 ms, 1.511 KB. Note: 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 - Status: 200 OK, 104 ms, 2.701 KB. Note: 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 - Status: 200 OK, 144 ms, 3.861 KB. Note: 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:

- Environment:** Postman skolení UAT, 18 hrs ago
- Iteration:** Iteration 1
- Test Status:** Failed (1)
- Requests:**
 - POST POST Confirm**: https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm / Priklady / GENERAL RUN / Rohlik add item to cart / POST Confirm - Status: 200 OK, 115 ms, 1.307 KB. Note: 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 - Status: 200 OK, 113 ms, 1.456 KB. Note: 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 - Status: 200 OK, 80 ms, 1.511 KB. Note: 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 - Status: 200 OK, 104 ms, 2.701 KB. Note: 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 - Status: 200 OK, 144 ms, 3.861 KB. Note: 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:

- Environment:** Postman skolení UAT
- Iterations:** Iteration 1, Iteration 2, Iteration 3
- Test Status:** Passed (3) / Failed (3)
- Requests:**
 - POST POST Confirm**: https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm / Priklady / GENERAL RUN / Rohlik add item to cart / POST Confirm - Status: 200 OK, 78 ms, 1.307 KB. Note: 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 - Status: 200 OK, 81 ms, 1.456 KB. Note: 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 - Status: 200 OK, 67 ms, 1.511 KB. Note: 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 - Status: 200 OK, 82 ms, 2.701 KB. Note: 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 - Status: 200 OK, 91 ms, 3.861 KB. Note: Fail - Total price is 6.97 | Assertion: expected 84.99 to equal 6.97

The screenshot shows a Postman Collection Runner interface. At the top, there are tabs for 'Postman skolení', 'POST POST Cart', 'GET GET Check cart', and 'Rohlik add item to ...'. Below the tabs, it says 'Postman skolení UAT, just now'. On the right, there are buttons for 'View Results' (orange), 'Run Again' (blue), 'New' (grey), and 'Export Results' (grey). The main area is titled 'RUN SUMMARY' and contains a table of test results:

	1	2	3
POST POST Confirm	0 0		
POST POST Check	0 0		
POST SET set-and-delete	0 0		
POST POST Cart	0 0		
▼ GET GET Check cart	0 3	X	X
	Fail	Total price is 6.97	

- Přepínáč mezi jednotlivými iteracemi

This screenshot shows the same Postman Collection Runner interface as above, but with three iterations labeled 1, 2, and 3 on the right. The 'Iteration 1' section is expanded, showing the details of each test run. The 'Iteration 2' section is collapsed.

Iteration	Test	URL	Method	Status	Time	Size
1	POST POST Confirm	https://www.rohlik.cz/services/frontend-service/address/v1/autocomplete/confirm	POST	200 OK	78 ms	1.307 KB
1	POST POST Check	https://www.rohlik.cz/services/frontend-service/delivery-address/check	POST	200 OK	81 ms	1.456 KB
1	SET set-and-delete	https://www.rohlik.cz/services/frontend-service/delivery-address/set-and-delete	POST	200 OK	67 ms	1.511 KB
1	POST POST Cart	https://www.rohlik.cz/services/frontend-service/v2/cart	POST	200 OK	82 ms	2.701 KB
1	GET GET Check cart	https://www.rohlik.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false	GET	200 OK	91 ms	3.861 KB
1				Fail	Total price is 6.97 Assertion Error: expected 84.99 to equal 6.97	
2						
3						

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:

< TREDGATE >

- 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:

```
collapse Rohlík add items to Cart - d...  
  POST POST Confirm  
  POST POST Check  
  POST SET set-and-delete  
  GET GET Check cart  
  POST POST Cart 1  
  POST POST Cart 2  
  GET GET Check cart
```

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.dat+++++a.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. Most requests are checked, except for the last one which is a GET request. The requests listed are:

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

In the center, there are several configuration options:

- Deselect All | Select All | Reset**
- Iterations:** 2
- Delay:** 0 ms
- Data:** Select File (button) rohlik_data.json (button)
- Data File Type:** application/json (dropdown) Preview (button) (highlighted)
- Checkboxes:**
 - Save responses (checked)
 - Keep variable values (unchecked)
 - Run collection without using stored cookies (checked)
 - Save cookies after collection run (unchecked)

At the bottom right is a blue button labeled **Run Postman skolení**.

PREVIEW DATA		
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.
- Data File Type:** 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".

< TREDGATE >

Spustíme runner a počkáme na výsledky

The screenshot shows the Postman Runner interface with two iterations of tests. Iteration 1 contains 10 tests, and Iteration 2 contains 10 tests. Each test row includes the method, URL, description, status code, response time, and file size. Test details like status code assertions are shown in the rows below each test.

Iteration	Test Description	Method	URL	Description	Status	Time	Size
Iteration 1	POST Confirm https://www.rohlík.cz/services/frontend-service/address/v1/autocomplete/confirm / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Confirm	POST	https://www.rohlík.cz/services/frontend-service/address/v1/autocomplete/confirm	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Confirm	200 OK	115 ms	1.307 KB
	Pass Status code is 200						
	POST Check https://www.rohlík.cz/services/frontend-service/delivery-address/check / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Check	POST	https://www.rohlík.cz/services/frontend-service/delivery-address/check	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Check	200 OK	56 ms	1.456 KB
	Pass Status code is 200						
	POST SET set-and-delete https://www.rohlík.cz/services/frontend-service/delivery-address/set-and-delete / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / SET set-and-del...	POST	https://www.rohlík.cz/services/frontend-service/delivery-address/set-and-delete	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / SET set-and-del...	200 OK	60 ms	1.511 KB
	Pass Status code is 200						
	POST POST Cart 1 https://www.rohlík.cz/services/frontend-service/v2/cart / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 1	POST	https://www.rohlík.cz/services/frontend-service/v2/cart	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 1	200 OK	54 ms	2.679 KB
	Pass Status code is 200						
	Pass Data.data.totalprice is a number						
	POST POST Cart 2 https://www.rohlík.cz/services/frontend-service/v2/cart / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 2	POST	https://www.rohlík.cz/services/frontend-service/v2/cart	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 2	200 OK	59 ms	3.696 KB
Pass Status code is 200							
Pass Data.data.totalprice is a number							
GET GET Check cart https://www.rohlík.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / GE...	GET	https://www.rohlík.cz/services/frontend-service/v2/cart-review/check-cart?blockingValidation=false	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / GE...	200 OK	83 ms	3.915 KB	
Pass Status code is 200							
Pass Total price is 42.45							
Iteration 2	POST POST Confirm https://www.rohlík.cz/services/frontend-service/address/v1/autocomplete/confirm / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Confirm	POST	https://www.rohlík.cz/services/frontend-service/address/v1/autocomplete/confirm	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Confirm	200 OK	50 ms	1.307 KB
	Pass Status code is 200						
	POST POST Check https://www.rohlík.cz/services/frontend-service/delivery-address/check / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Check	POST	https://www.rohlík.cz/services/frontend-service/delivery-address/check	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Check	200 OK	59 ms	1.456 KB
	Pass Status code is 200						
	POST SET set-and-delete https://www.rohlík.cz/services/frontend-service/delivery-address/set-and-delete / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / SET set-and-del...	POST	https://www.rohlík.cz/services/frontend-service/delivery-address/set-and-delete	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / SET set-and-del...	200 OK	55 ms	1.511 KB
	Pass Status code is 200						
	POST POST Cart 1 https://www.rohlík.cz/services/frontend-service/v2/cart / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 1	POST	https://www.rohlík.cz/services/frontend-service/v2/cart	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 1	200 OK	65 ms	4.844 KB
	Pass Status code is 200						
	Pass Data.data.totalprice is a number						
	POST POST Cart 2 https://www.rohlík.cz/services/frontend-service/v2/cart / Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 2	POST	https://www.rohlík.cz/services/frontend-service/v2/cart	/ Příklady / GENERAL RUN / Rohlík add items to Cart - data file / POST Cart 2	200 OK	60 ms	6.091 KB
Pass Status code is 200							
Pass Data.data.totalprice is a number							

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

< TREDGATE >

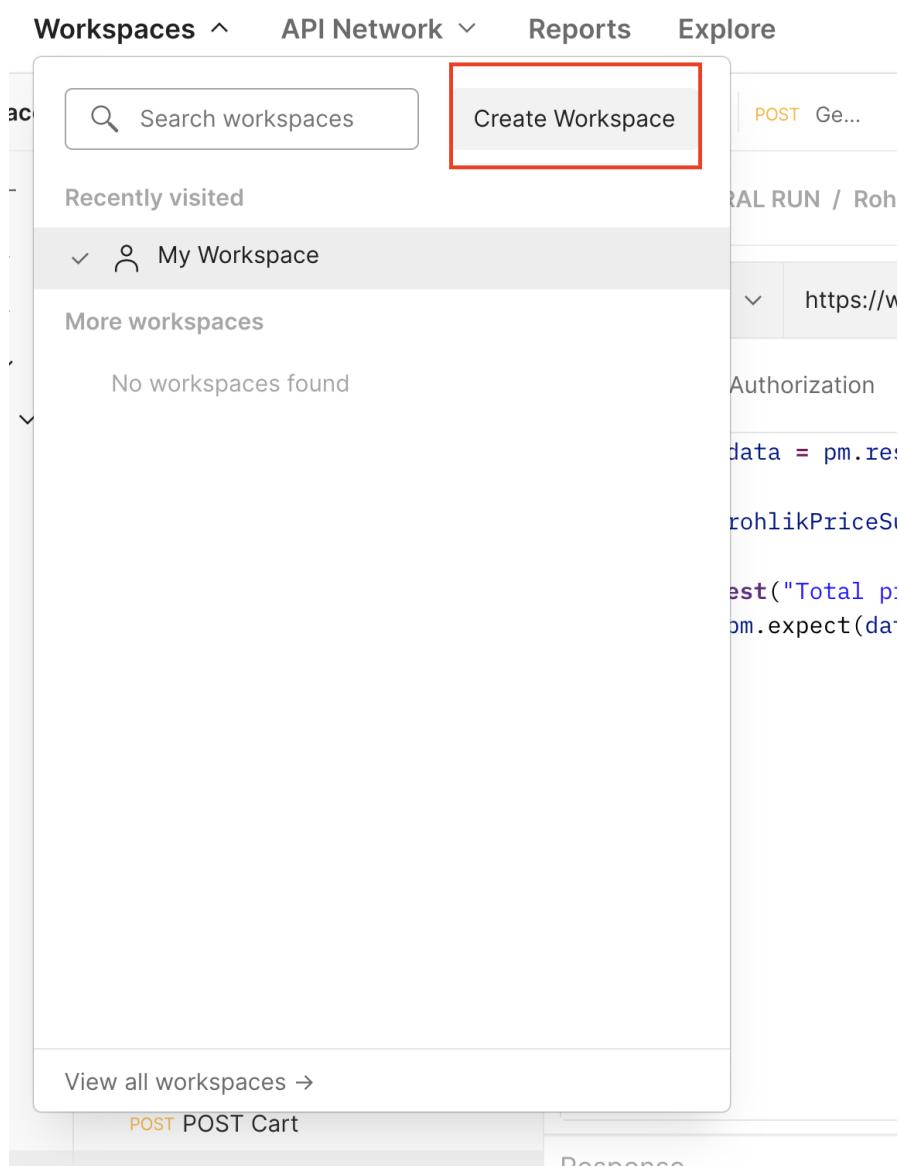
Workspace je funkcionalita Postman, která umožňuje sdílet data mezi více uživateli nebo zálohovat data uživatele. Tato funkcionalita je zdarma v limitu 3 uživatelů.

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' with a single entry: 'My Workspace'. Below this, there is a section titled 'More workspaces' with the message 'No workspaces found'. On the right side, there is a preview area showing a POST request to 'https://www.simplilearn.com/api/v1/carts'. The request body contains JSON data: 'data = pm.replace("\$totalPrice", pm.response.json().r...'). The response body shows a JSON object with 'r...': 'rohlikPriceS...' and 'est("Total p:...'): 'pm.expect(da...'. At the bottom of the sidebar, there is a link 'View all workspaces →' and a 'POST Cart' button.

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 TeamCancel

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 profileGo to Workspace

V Postmanu se Vám zobrazí nový Workspace

VYTVAŘENÍ SDÍLENÝCH REQUESTŮ

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

PŘÍKLAD

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.

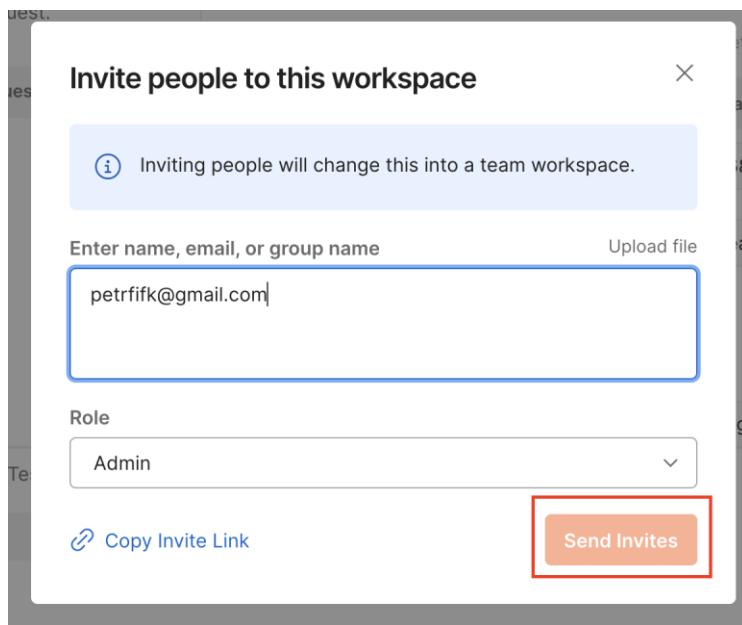
The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows collections, APIs, environments, mock servers, monitors, flows, and history.
- Top Bar:** Home, Workspaces, API Network, Reports, Explore, Search Postman, Invite (highlighted), Upgrade.
- Request Details:**
 - Method: GET
 - URL: https://api.imgur.com/3/account/me/images
 - Authorization: OAuth 2.0 (selected)
 - Headers: (9)
 - Body: Pre-request Script, Tests, Settings
 - Current Token: Available Tokens (0680e39f4d9f1166ec667d9dbb315cda), Header Prefix: Bearer
 - Configure New Token: Token Name: Imgur token
- Response Body:**

```

1  "data": {
2    "error": "Authentication required",
3    "request": "/3/account/me/images",
4    "method": "GET"
5  },
6  "success": false,
7  "status": 401
8
9
  
```
- Bottom Status:** Status: 401 Unauthorized, Time: 113 ms, Size: 114 KB, Save Response.

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 sections like "Team Workspace", "Collections", "APIs", "Environments", "Mock Servers", "Monitors", "Flows", and "History". The main area shows a "New Collection" card with a "GET Request A" and a "New Collection" link. The right side of the screen has a "Overview" tab, a "Changelog" section, and a "No Environment" message. The "Changelog" section lists recent activities:

- 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

< TREDGATE >

1. klikněte na název workspace

The screenshot shows the Postman interface with the 'Team Workspace' tab selected. The left sidebar has a red border around the 'Team Workspace' tab. The main area displays a summary message: 'Summary is yet to be added.' Below it, a section titled 'This is where the collaboration happens. Use this space to share and collaborate on APIs, collections, environments, monitors, and mocks.' The 'Activity' section shows recent changes made by Petr Fifka, such as editing a collection and creating the workspace. On the right, there are sections for 'In this workspace' (Requests, Collections, APIs, Environments, Mock Servers, Monitors) and 'Recent contributors' (You).

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:

< TREDGATE >

```
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, 166ms]

└ 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, 187ms]

└ 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 Rohlik.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
```

< TREDGATE >

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:

< TREDGATE >

```
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
  □ Příklady / 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's a sidebar with various sections like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The 'Collections' section is expanded, showing a tree structure with 'Beta Postman skolení', 'Imgur API', 'Kolekce pro import Copy', 'Live API', 'Postman skolení' (which is expanded to show 'Příklady', 'REQ Params', and 'HTTP Requests'), and other categories like 'JSON', 'AUTH', etc. In the main area, a specific request titled 'GET regres.in mock example' is selected. A context menu is open over this request, with the 'Add example' option highlighted. The main workspace shows a GET request configuration for 'https://reqres.in/api/use' with 'Query Params' and a JSON response body.

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

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

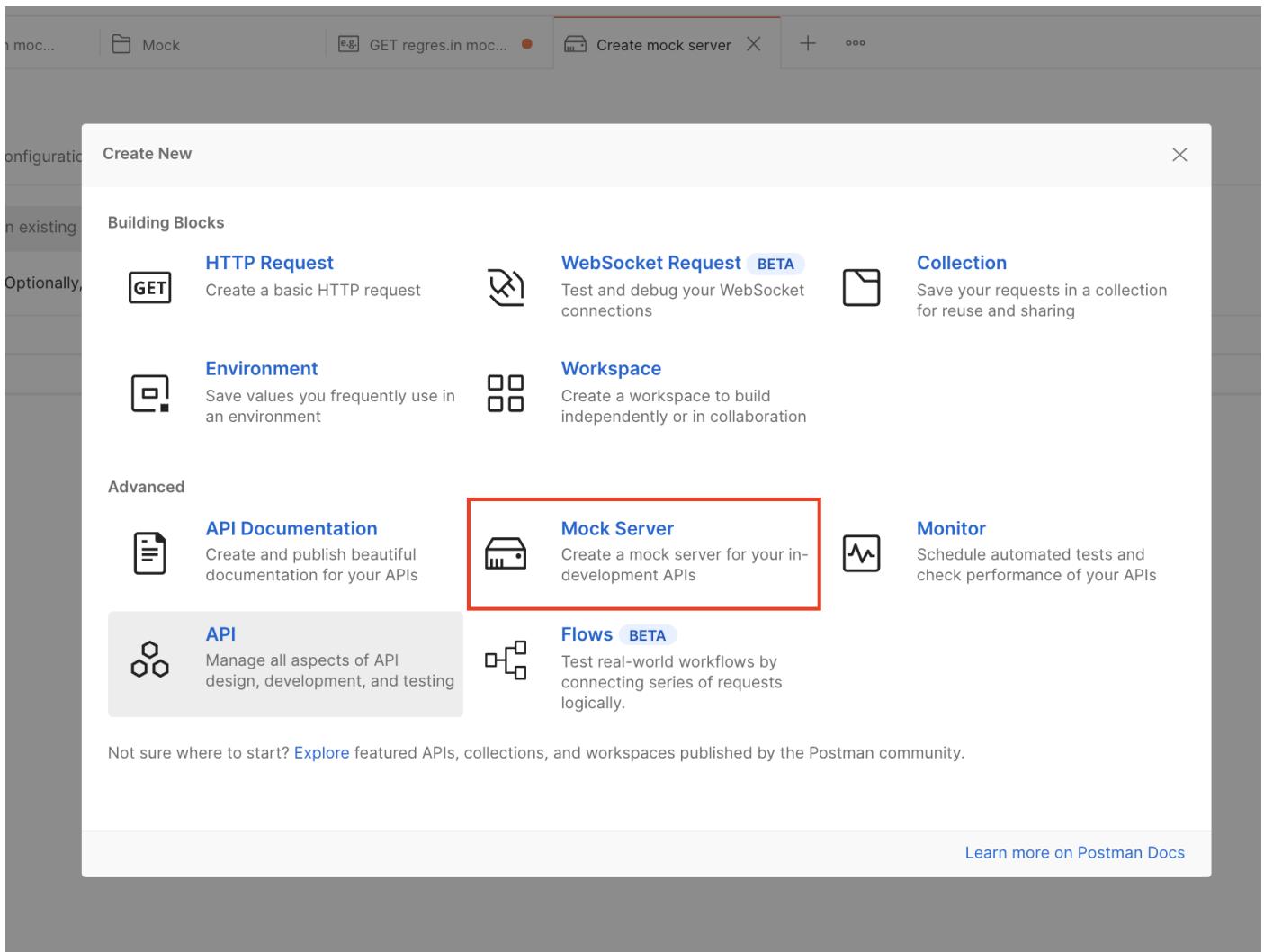
< TREDGATE >

The screenshot shows the Postman application interface. On the left, there's a sidebar with various collections like 'Beta Postman skolení', 'Imgur API', etc. The main area shows a request for 'GET regres.in mock example' with a status code of 200 OK. The response body is JSON, containing the message: "{'info': 'právě jsi provolal mock a dostal odpověď'}".

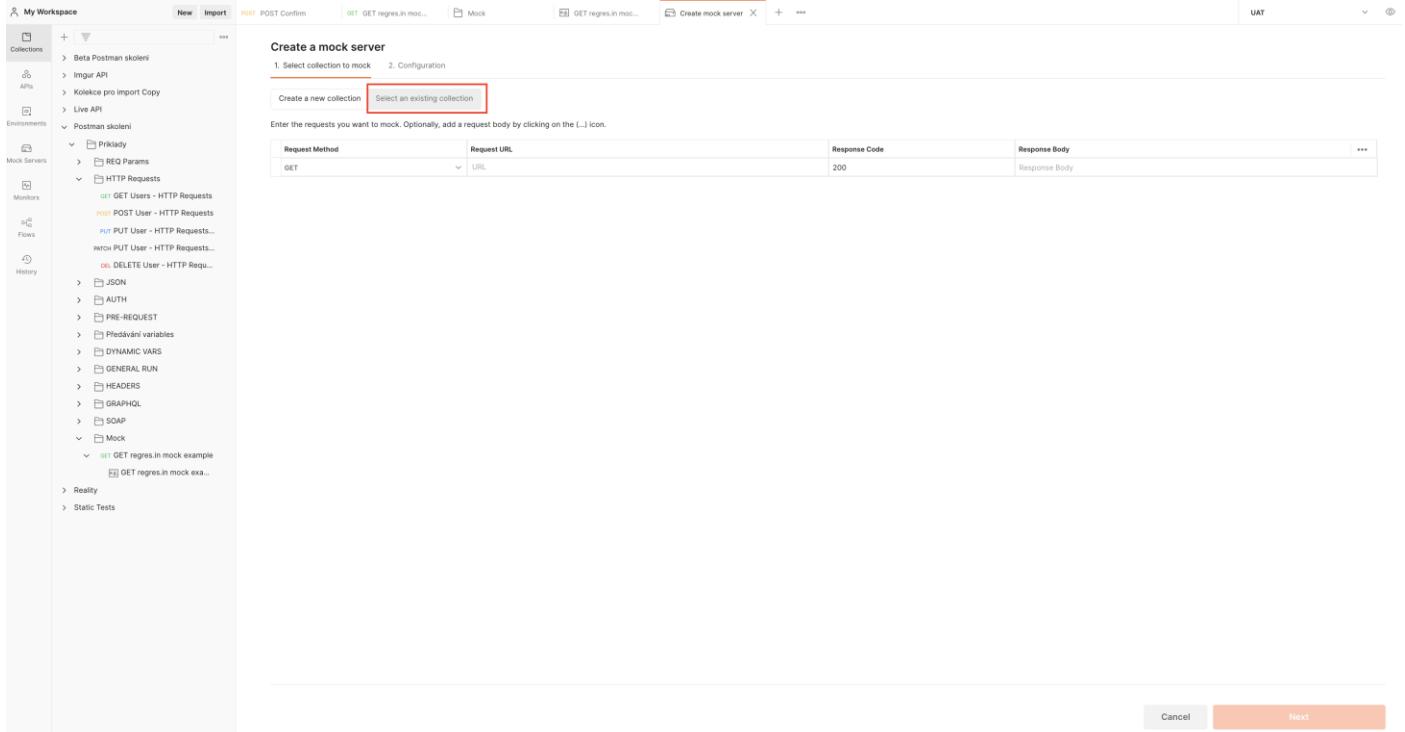
3. Do Status Code zadej 200 OK

The screenshot shows the Postman application interface. On the left, there's a sidebar with various collections like 'Beta Postman skolení', 'Imgur API', etc. The main area shows a request for 'GET regres.in mock example' with a status code of 200 OK. The response body is JSON, containing the message: "{'info': 'právě jsi provolal mock a dostal odpověď'}".

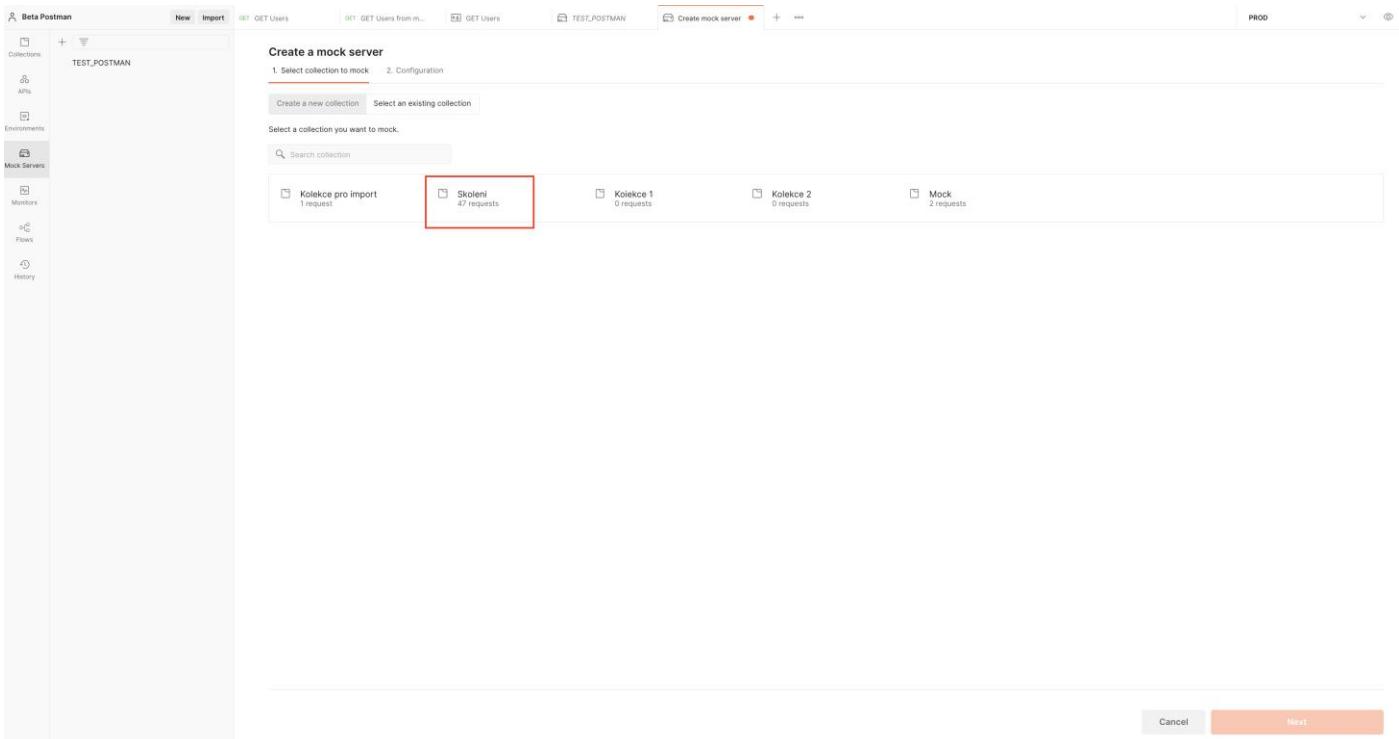
4. Klikni na tlačítko "New" a vyber "Mock Server"



5. Klikni na tlačítko "Select an existing collection"



6. Vyber kolekci Skoleni a stiskni tlačítko Next



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

Create a mock server

1. Select collection to mock 2. Configuration

Mock server name

Skoleni Mock

Collection

Skoleni

Tag

CURRENT

Environment

PROD

Save the mock server URL as an environment variable

Make mock server private

Simulate fixed network delay

Note: This will create a new environment containing the URL.
Note: To call a private mock server, you'll need to add an `x-api-key` header to your requests. See how to generate a Postman API key.

Close **Create Mock Server**

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". A new tab titled "Skoleni Mock" is open. The main area features a small cartoon character holding a briefcase and a folder. Below the character, the text "No Mock server calls yet" is displayed. A note below it says "To call the mock server, follow these 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 "Kolekce 1", "Kolekce 2", "Kolekce pro import", and "Mock"), "APIs", and "Environments". The main workspace shows a collection named "Beta Postman" with a "GET Regres.in users mock" request. The request details show a "GET" method and the URL "https://657e5f4c-cd77-4abf-8060-3fd19dc5ac71.mock.pstmn.io/api/users". A red box highlights the URL field.

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

< TREDGATE >

The screenshot shows the Postman 9 interface. On the left is a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area shows a collection named 'Beta Postman' with several items: 'Koikce 1', 'Kolekce 2', 'Kolekce pro import', 'Mock' (which contains 'GET Users' and 'GET Users from mock'), 'Skoleni' (which contains 'First calls', 'Google', 'Regres.in', and 'HTTP Requests' which further contains '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'), 'Params', 'Auth', 'Headers', 'JSON', 'GraphQL', and 'SOAP'. At the top, there are tabs for 'New' and 'Import', and a search bar. Below the search bar are two requests: 'GET GET Users' and 'GET GET Regres.in user...'. The 'Body' tab is selected for the first request, which has a URL of 'https://657e5f4c-cd77-4abf-8060-3fd19dc5ac71.mock.pstmn.io/api/users'. The 'Headers' tab shows '(13)' headers. The 'Body' tab shows the response body as JSON:

```
1 "info": "pr\u00e1v\u00e9 jsi provolal mock a dostal odpov\u00e9d"
```

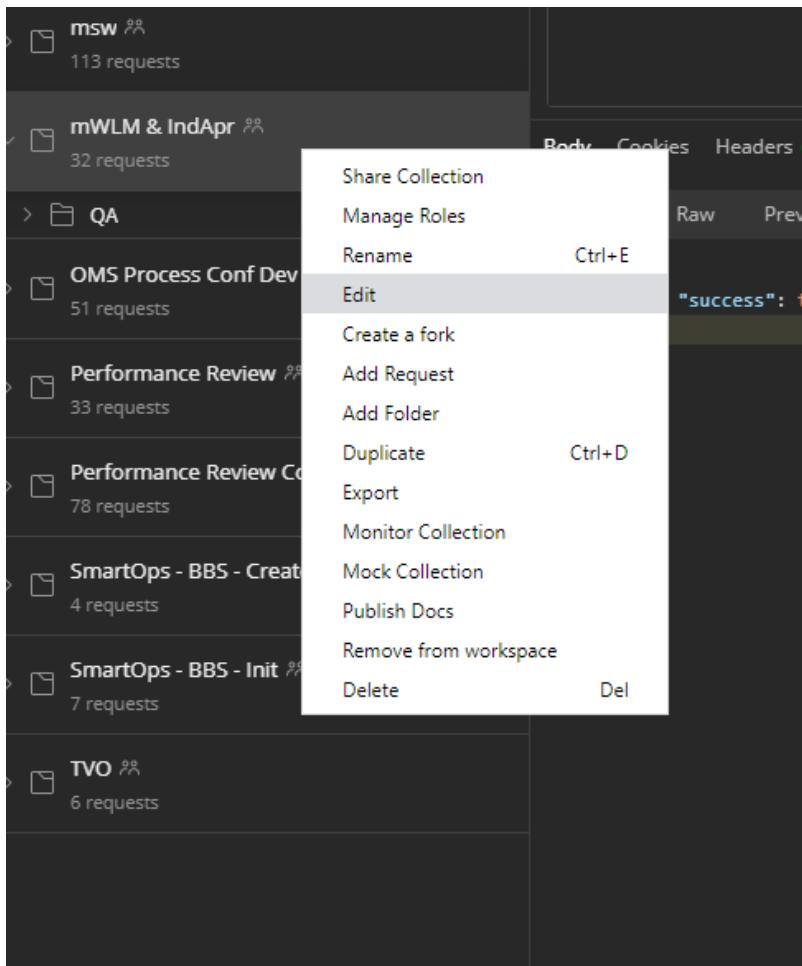
POSTMAN 9 VS 7 (BONUS)

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

PŘÍSTUP K DETAILU KOLEKCE

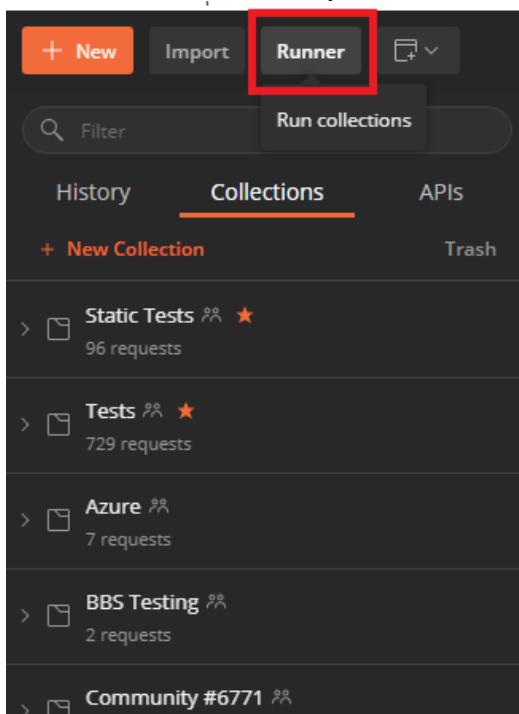
Ve starší verzi Postman se kolekce neotvírá v okně jako request, ale je potřeba k ní přistoupit přes:

1. Pravý klik na kolekci
2. Vybrat Edit



COLLECTION RUNNER

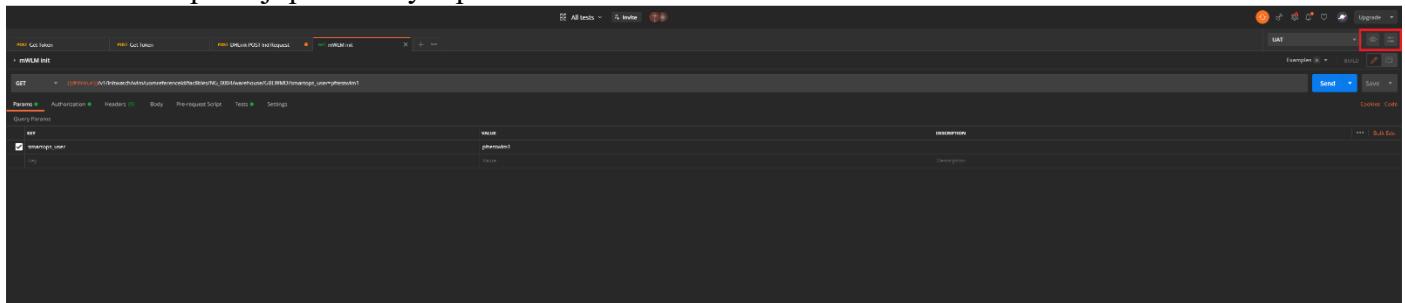
Collection Runner se nespouští z kolekce, ale má svoje tlačítko v Postman UI



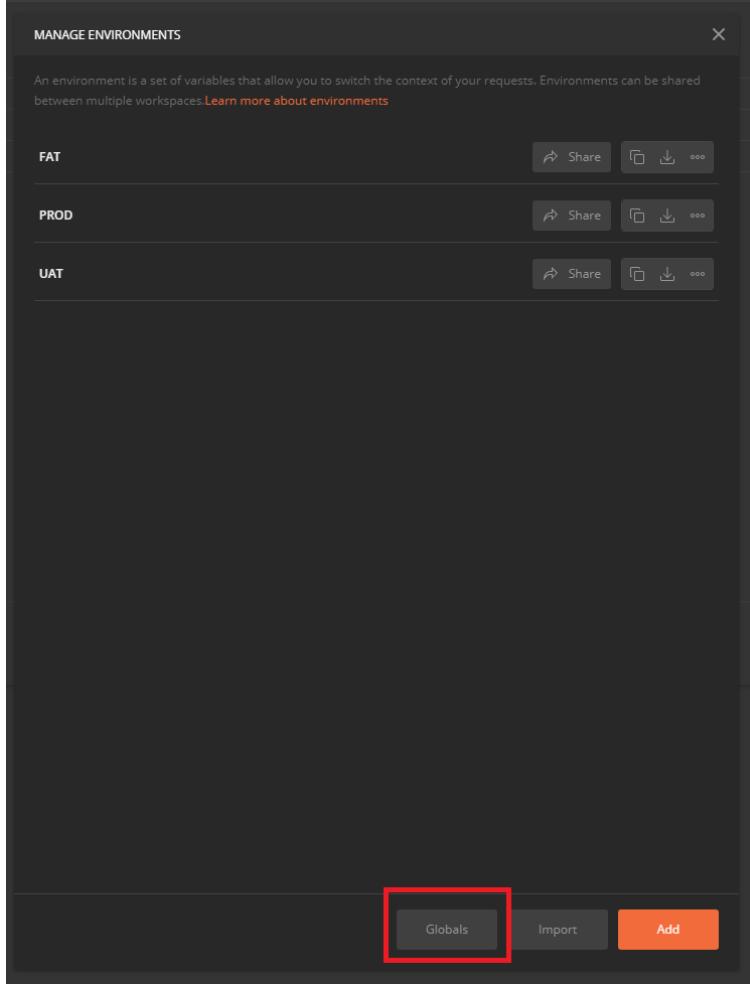
PROMĚNNÉ

< TREDGATE >

Proměnné se spravují přes ikony v pravé horní části Postman UI

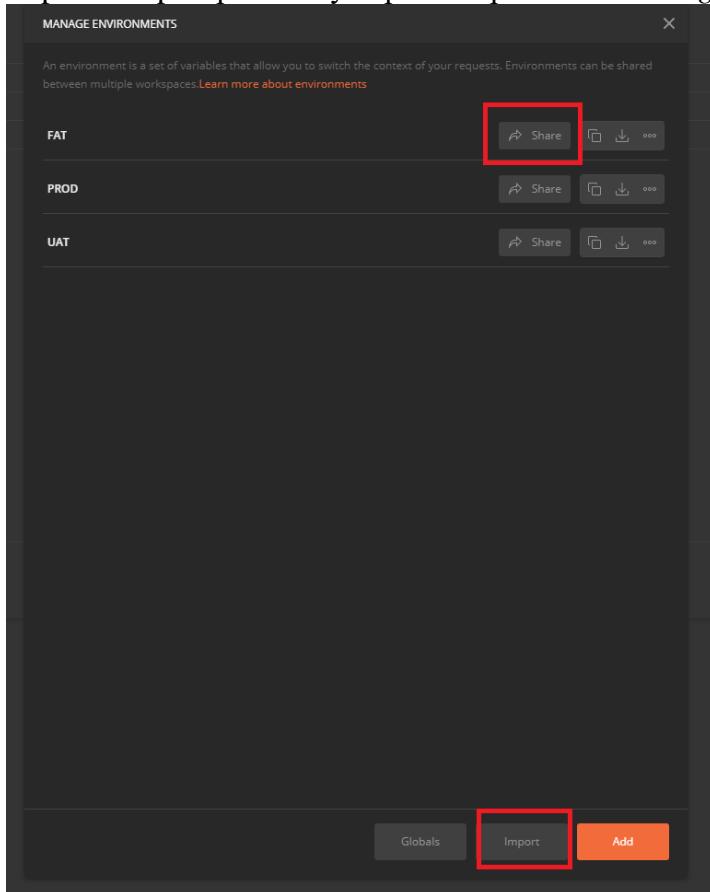


Globals se spravují z okna pro *Manage Environments* (viz ikony výše)



< TREDGATE >

Export a import proměnných probíhá přes okno *Manage Environments*



ZDROJE, KAM DÁL

Zdroj	URL	QR
ChaiJS knihovna	https://www.chaijs.com/	
Co je to API	https://en.wikipedia.org/wiki/API	
Co je to REST	https://cs.wikipedia.org/wiki/Representational_State_Transfer	

< TREDGATE >

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

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

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

< TREDGATE >

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

Wikipedia URN	https://cs.wikipedia.org/wiki/Uniform_Resource_Name	
Wikipedia HTTP	https://cs.wikipedia.org/wiki/Hypertext_Transfer_Proto	

SLOVNÍK POJMŮ

Pojem	Vysvětlení
API	Application programming interfaces
REST	Representational State Transfer
Interface	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.
HTTP	Hypertext Transfer Protocol - internetový protokol určený pro komunikaci s WWW servery. Slouží pro přenos hypertextových dokumentů ve formátu HTML, XML, i jiných typů souborů.
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
URN	Uniform Resource Name
URI	Uniform Resource Identifier
SOAP	Simple Object Access Protocol
GUI	Grafické uživatelské rozhraní (Graphic User Interface)
OS	Operační systém (Operating System). Jedná se o prostředí, ve kterém běží všechny procesy, aplikace... Nejznámější OS: Windows, MacOS, Android, iOS, Linux
WWW	World Wide Web
Cookies	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.
OOP	Objected Oriented Programming
MOCK	Simulované objekty, které imitují chování reálných objektů kontrolovaným způsobem