Projects Portfolio

Matej Rástocký

www.rastocky.sk

Batch Processing - Improving the speed an efficiency of a distributed system
Meway - Admin Panel
SpotQ - Third-party API integration
GetSH - Bash script for REST API testing

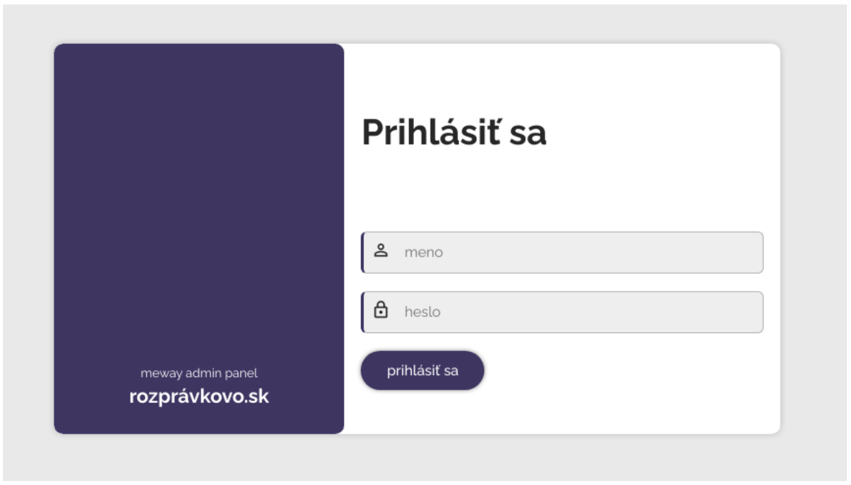
Batch Processing

NodeJs | Kafka | SalesForce

In my recent project, I worked on optimizing the process of moving data from a Kafka topic to Salesforce. Unfortunately, I cannot go into detail about the specifics of the project. However, I can say that I implemented a batching mechanism where messages coming from other part of the system are grouped together and submitted to SF as a whole. If there are not enough messages to submit, the mechanism will wait for more, until either there are enough messages or a certain timeout is reached, after which it will submit the messages that are present. This approach has solved multiple problems, including the fact that the Salesforce API has a limit on the number of requests that can be made (we were reaching this limit before). By batching messages, I have significantly reduced the number of requests required. The mechanism is designed to be resilient - even if some part of the system fails no messages will be lost.

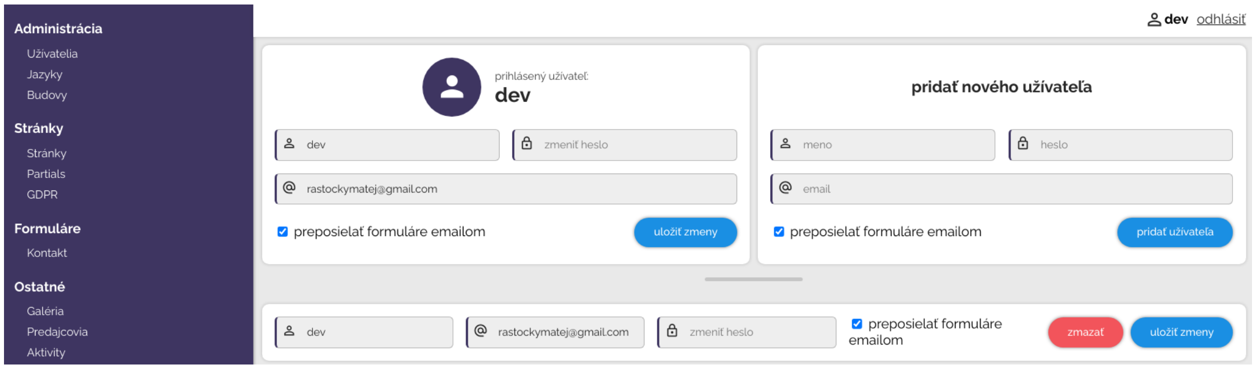
Meway Admin Panel

Node.js | Express.js | ejs | MongoDB | Bcrypt.js | Mailgun | Passport



Meway is an administration panel for managing website content. Basically it is a collection of functional parts that I build according to the needs of the project I am currently working on. For other projects, I then use components that I have already created, adapt them to current requirements or develop new ones. Currently includes:

User management:
- Login
- New user creation
- Change password
- Delete user



dev

rastockymatej@gmail.com

☑ preposielať formuláre emailom

☑ preposielať formuláre emailom

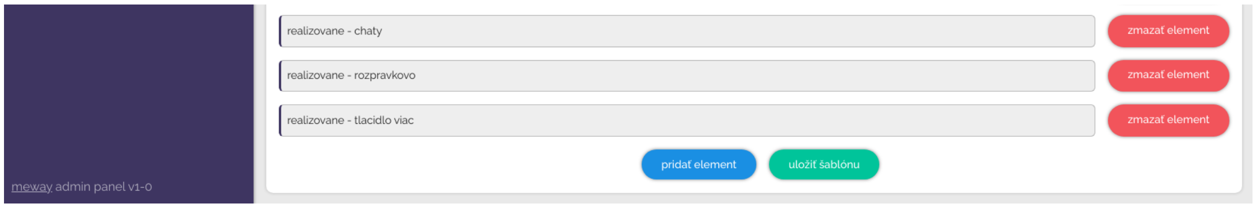
dev | rastockymatej@gmail.com | ☑ preposielať formuláre emailom

Internationalization:

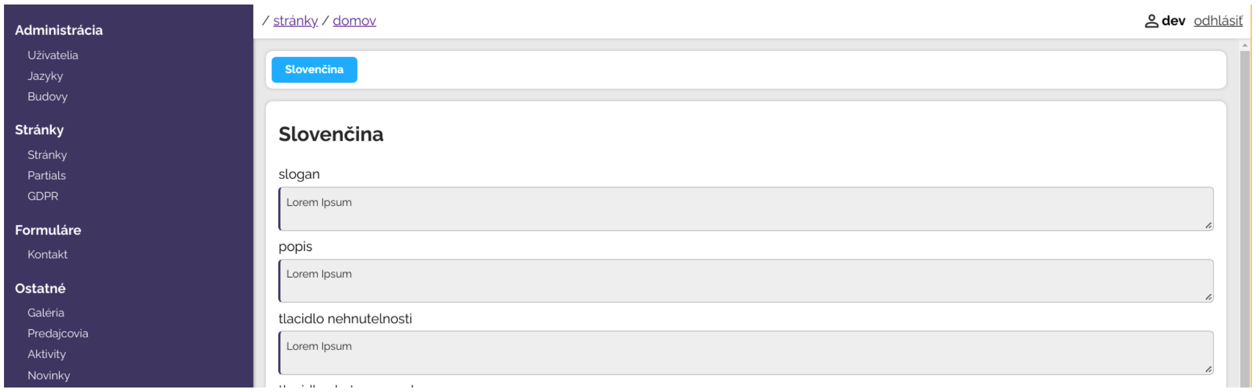
- Add new language
- Fill in translations (if a translation is not filled in, default lang will show up instead)

Page builder:

- Simple way to define structure of pages
- Primitive template builder / editor
- Partials



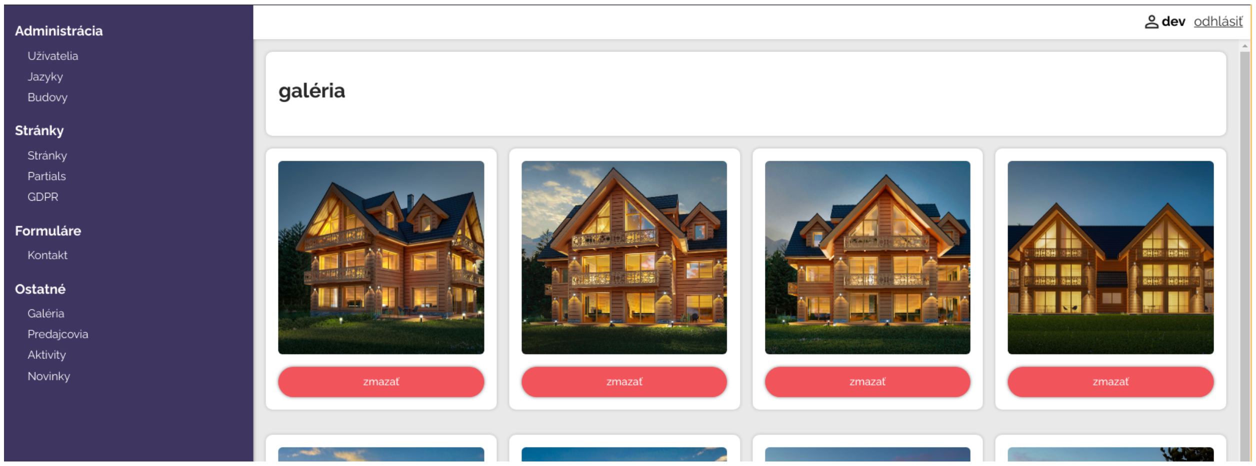
Define page template



Edit texts and translations

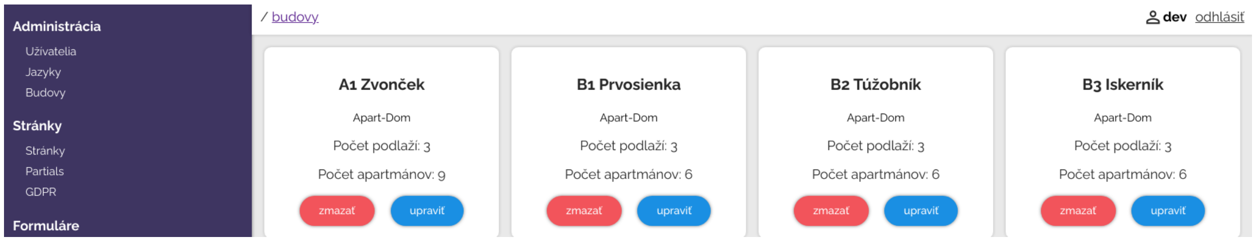
Components:

- Gallery
- News feed
- Sponsors, Activities...



Forms:
- Contact
- Newsletter

System for managing buildings (developer projects):
- Building
- Floor
- Apartment
- Rooms and floor-area



Administrácia
- Uživatelia
- Jazyky
- Budovy

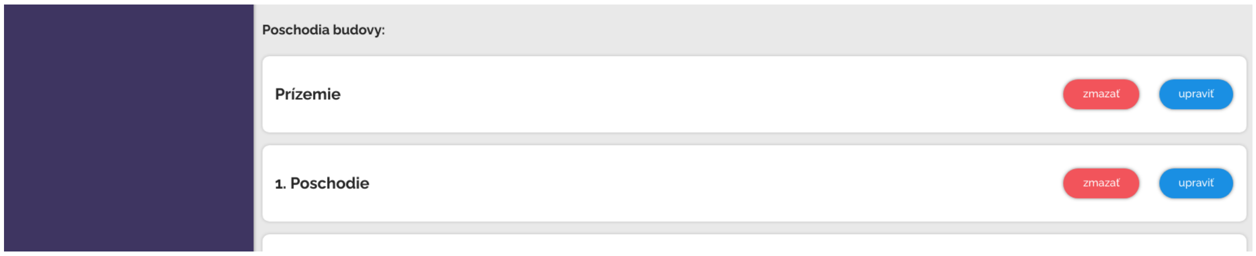
Stránky
- Stránky
- Partials
- GDPR

Formuláre

/ budovy

dev odhlásiť

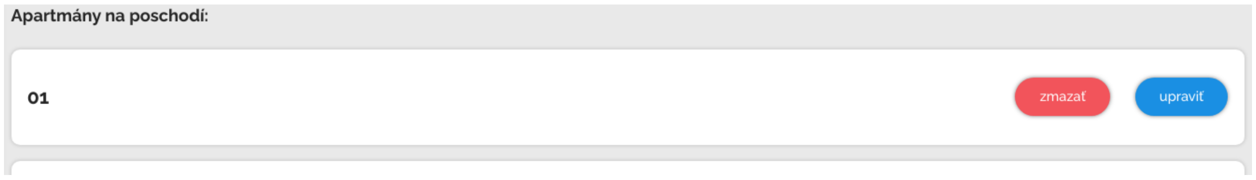
A1 Zvonček	B1 Prvosienka	B2 Túžobník	B3 Iskerník
Apart-Dom	Apart-Dom	Apart-Dom	Apart-Dom
Počet podlaží: 3	Počet podlaží: 3	Počet podlaží: 3	Počet podlaží: 3
Počet apartmánov: 9	Počet apartmánov: 6	Počet apartmánov: 6	Počet apartmánov: 6
zmazať upraviť	zmazať upraviť	zmazať upraviť	zmazať upraviť



Poschodia budovy:

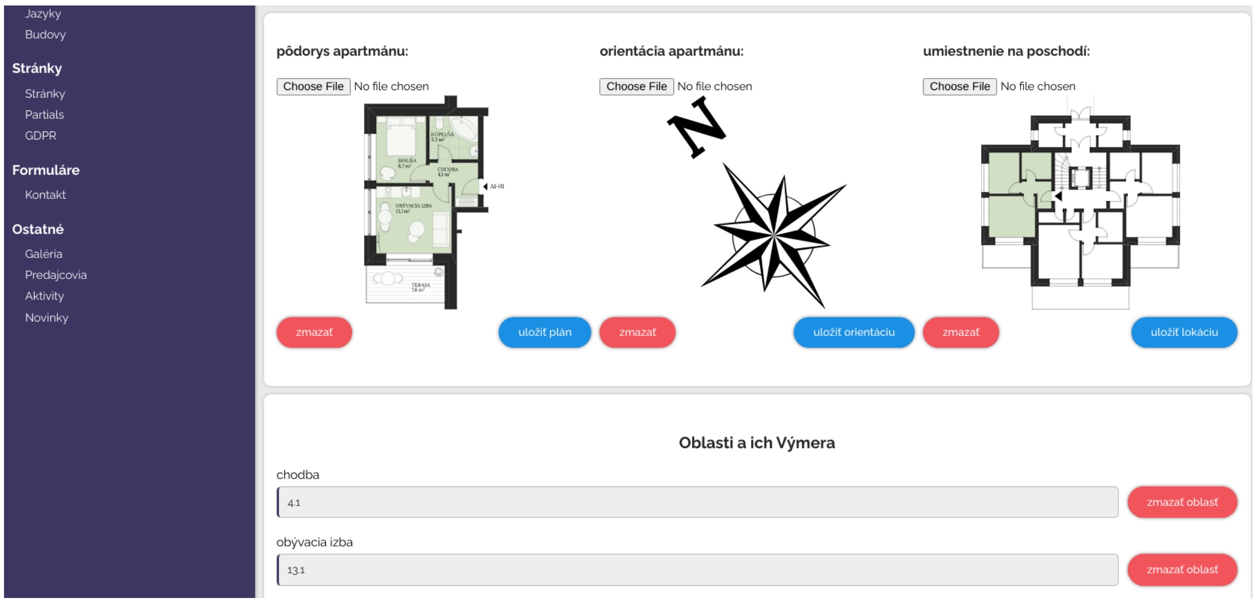
Prízemie — zmazať upraviť

1. Poschodie — zmazať upraviť



Apartmány na poschodí:

01 — zmazať upraviť



- Jazyky
- Budovy

Stránky
- Stránky
- Partials
- GDPR

Formuláre
- Kontakt

Ostatné
- Galéria
- Predajcovia
- Aktivity
- Novinky

pôdorys apartmánu: Choose File No file chosen — zmazať uložiť plán

orientácia apartmánu: Choose File No file chosen — zmazať uložiť orientáciu

umiestnenie na poschodí: Choose File No file chosen — zmazať uložiť lokáciu

Oblasti a ich Výmera

chodba: 4.1 — zmazať oblasť

obývacia izba: 13.1 — zmazať oblasť

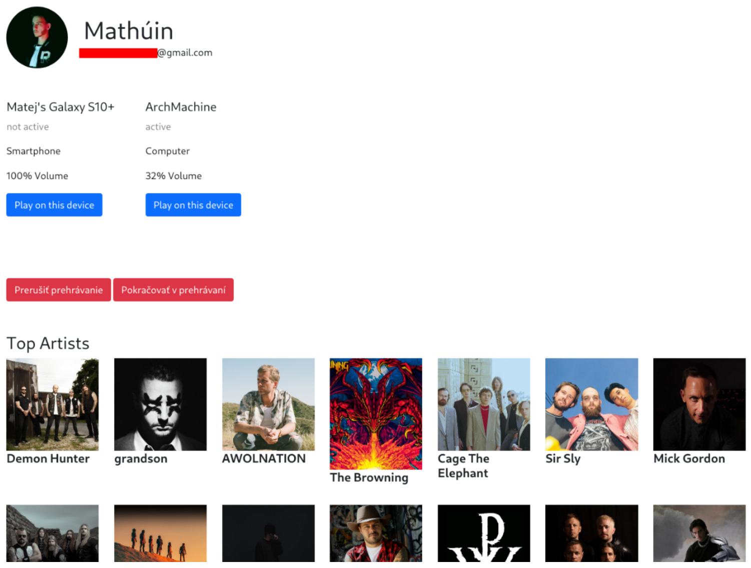
SpotQ

Node.js | Express.js | ejs | Spotify API

SpotQ is a solution to the problem that arises when several people listen to music together - they want to play their songs as soon as the previous one ends. Using the Spotify API, I developed an application into which the "master" logs in with his Spotify Premium account. Subsequently, other users open the application, search for songs and add them to the queue - hence SpotQ (Spotify Queue).

App is just a proof-of-concept:

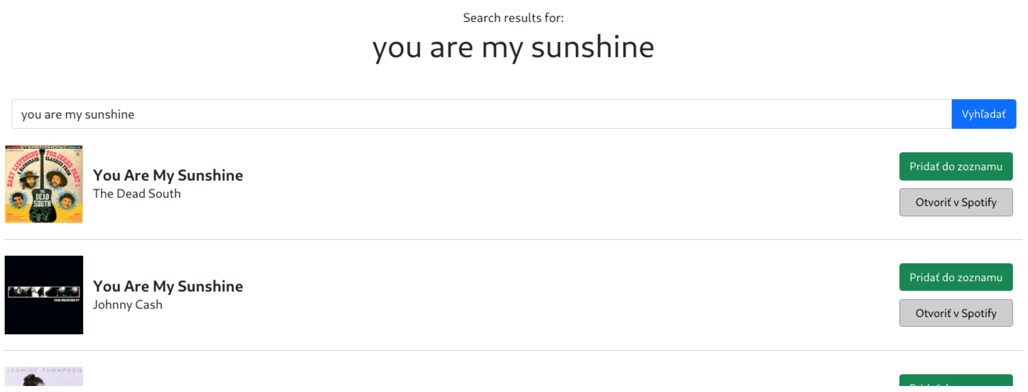
- Auth to Spotify API
- User info from Spotify account (photo, name, email, devices, top songs, top producents, song that's playing right now...)
- Commands for "master" device
    - Add song to queue
    - Skip song
    - Pause / play
    - Change device that plays songs
- Functionality:
    - Search for songs
    - Add song to queue
    - Open song in Spotify app



Mathúin
[redacted]@gmail.com

Matej's Galaxy S10+	ArchMachine
not active	active
Smartphone	Computer
100% Volume	32% Volume

User profile after logging in - user ingo (name, email, devices, switch device, play/pause, top producents, top songs)


Search results for:

you are my sunshine

you are my sunshine | Vyhľadať

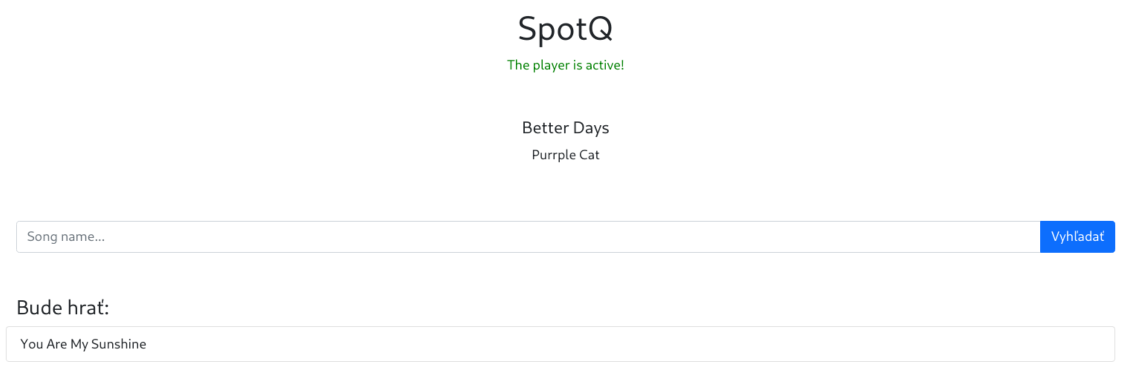
You Are My Sunshine
The Dead South

Pridať do zoznamu | Otvoriť v Spotify

You Are My Sunshine
Johnny Cash

Pridať do zoznamu | Otvoriť v Spotify

search for songs in Spotify library


SpotQ

The player is active!

Better Days

Purrple Cat

Song name... | Vyhľadať

Bude hrať:

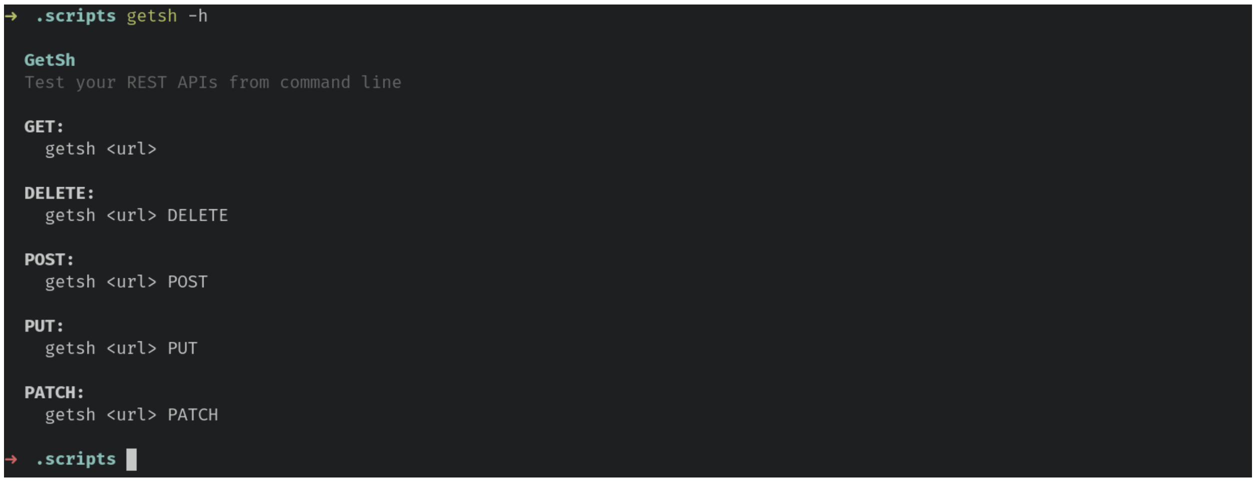
You Are My Sunshine

Landing page, shows status of app (whether or not we are able to play songs), song that is playing, search and finally queue of songs

GetSH

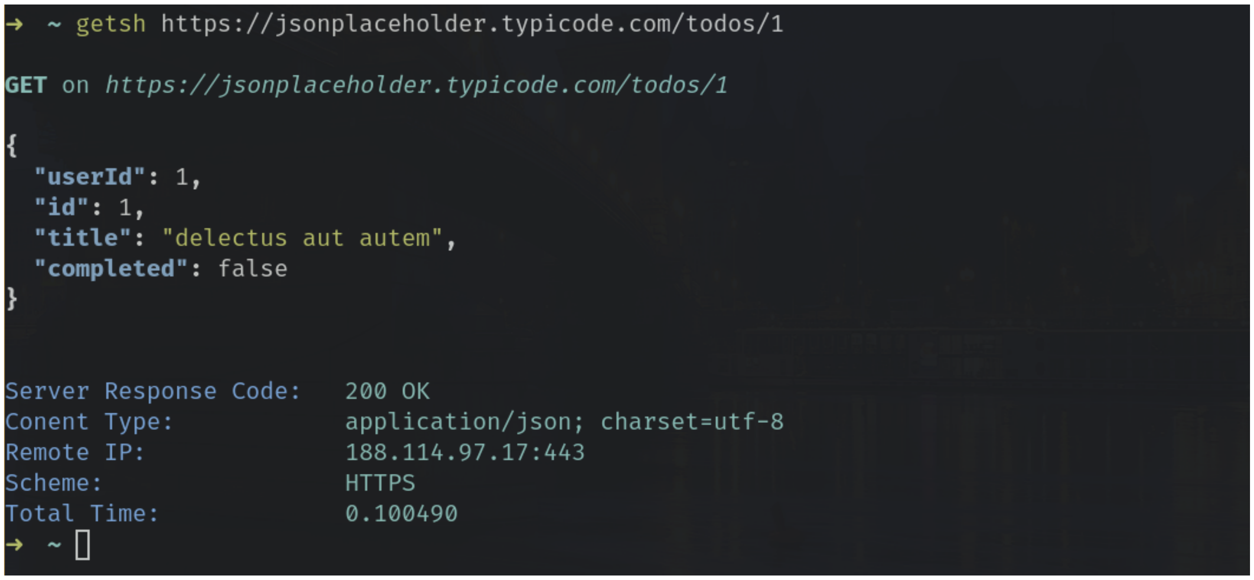
Bash | Curl | JQ | REST API

github.com/uyohn/getsh



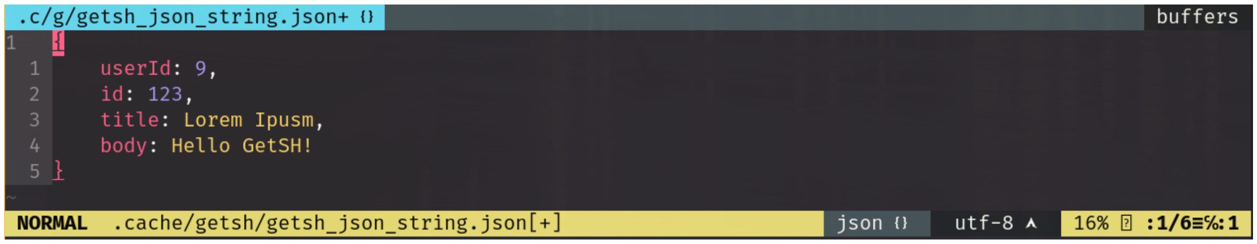
```
➜  .scripts getsh -h

  GetSh
  Test your REST APIs from command line

  GET:
    getsh <url>

  DELETE:
    getsh <url> DELETE

  POST:
    getsh <url> POST

  PUT:
    getsh <url> PUT

  PATCH:
    getsh <url> PATCH

➜  .scripts █
```

GetSH is a simple tool for testing REST APIs, ideal for simple workflows. Basically, it is a curl wrapper prepared for basic requests (GET, POST...). After a successful request, it displays a nicely formatted JSON response (using the *jq* utility) and other information about the request.

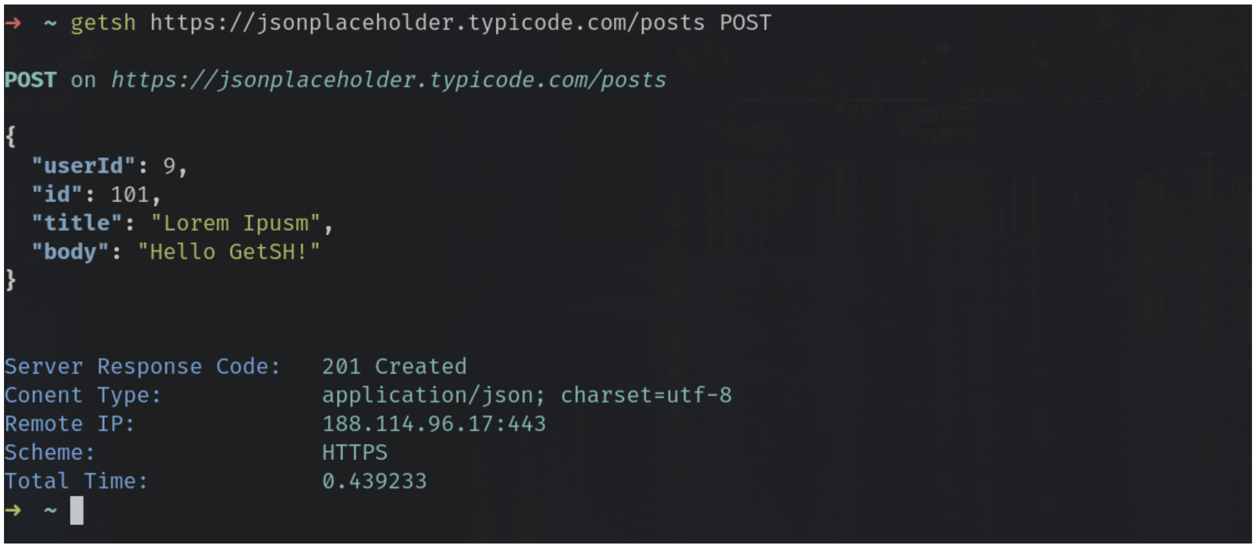


```
➜  ~ getsh https://jsonplaceholder.typicode.com/todos/1

GET on https://jsonplaceholder.typicode.com/todos/1

{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}


Server Response Code:   200 OK
Conent Type:            application/json; charset=utf-8
Remote IP:              188.114.97.17:443
Scheme:                 HTTPS
Total Time:             0.100490
➜  ~ ▯
```

For **POST** requests, an editor will open (according to the $EDITOR env. variable), in which we can fill in the request payload (this file remains saved, we do not need to fill it in again for other requests, just make the necessary changes).



```
 .c/g/getsh_json_string.json+ {}                                                              buffers
{
    userId: 9,
    id: 123,
    title: Lorem Ipusm,
    body: Hello GetSH!
}
~
 NORMAL   .cache/getsh/getsh_json_string.json[+]                     json {}   utf-8   16%  :1/6≡%:1
```



```
➜ ~ getsh https://jsonplaceholder.typicode.com/posts POST

POST on https://jsonplaceholder.typicode.com/posts

{
  "userId": 9,
  "id": 101,
  "title": "Lorem Ipusm",
  "body": "Hello GetSH!"
}


Server Response Code:   201 Created
Conent Type:            application/json; charset=utf-8
Remote IP:              188.114.96.17:443
Scheme:                 HTTPS
Total Time:             0.439233
➜ ~
```

Another feature that I would like to implement is support for simple authentication and auth using JWT.