# Postman

Automation with Postman

Aleksandr Elmekeev

1. Postman Overview
2. Postman Glossary
3. Automation With Postman
4. Useful Links

# Postman Overview

- Testing
- Collaboration
- Documentation
- Mocks
- Monitoring

# Postman Overview: What If Won’t Use Postman Cloud?

- Testing
- ~~Collaboration~~
- ~~Documentation~~
- ~~Mocks~~
- ~~Monitoring~~

# Postman Glossary

- Collection - executable description of API. Includes:
  - API specification
  - API documentation (description of each method, examples)
  - Tests
- Variables:
  - Global - variable initialized in the code in a global scope or from special file
  - Collection - collection level variable, can be initialized only from UI
  - Environment - variable initialized with value from environment definition / special file
  - Data - variable initialized with value from data file (see Control flow)
  - Local - variable initialized in the code in a local scope of pre-request script / test / function

# Automation With Postman

- Code Review
- Pipeline Integration
- Debugging
- Sandbox
- Authorization
- Control Flow
- Shared Functions
- Error Handling
- Connection To Database?

# Code Review

```
"name": "UI endpoint",
"event": [
    {
        "listen": "test",
        "script": {
            "id": "94b2a259-fb8e-4c6e-97fd-aa5975ce6536",
            "exec": [
                "pm.test(request.name + ' - Status code is 200', function () {",
                "    pm.response.to.have.status(200);",
                "});",
                "",
                "pm.test(request.name + ' - Response is valid', function() {",
                "    let expectedResponse = {",
                "        message: 'UI endpoint'",
                "    }",
                "",
                "    pm.expect(pm.response.json()).to.deep.equal(expectedResponse);",
                "});"
            ],
            "type": "text/javascript"
        }
    }
],
"request": {
    "method": "GET",
    "header": [],
    "body": {
        "mode": "raw",
        "raw": ""
    },
    "url": {
        "raw": "{{baseUrl}}/",
        "host": [
        "path": [
    }
},
```

- Collection is stored as a single JSON file.
- Environment specific variables are stored as separate JSON files.

Examples can be found in [Postman API Network]

# Pipeline Integration

```
newman run /collection.json
    -e /environment.json
    --timeout-script 1000
    ~r html
    ~~reporter-html-export report.html
```

- command line test runner
- contains environment specific variables
- fails test if no response after 1000 ms
- allows to generate html report, also available:
  - CLI report
  - JUnit report
  - JSON report

# Debugging During Development

- Debugging using Request and Response body in Collection Runner
- Postman Console
  - View requests and responses
  - Log whatever you want with `console.log()`

# Debugging Of Tests Running Through Pipeline

- Reporters:
  - CLI - allows to see in job console logs whatever you print with `console.log()`
  - JSON - provides full information about run including actually sent request and received response:
    - Note: response is stored as a stream, so you will need to decode it
  - HTML  with custom template - allows to include everything you need to html report. Examples:
    - Colored report for newman reporter with request/response body by tegomass
    - awesome-newman-html-template by Marcos Ellys
- Exporting state:
  - `--export-environment` - allows to save environment variables file before completing a run
  - `--export-globals` - allows to save global variables file before completing a run

# Sandbox

- chai - assertion library
- lodash - dozens of methods to simplify work with arrays, objects, strings, etc
- xml2js - xml to json converter
- crypto-js - encoding/decoding with popular crypto algorithms
- moment - work with date and time
- tv4 - allows to validate json object against json schema
- uuid - uuid generation

More info: Postman Sandbox and Postman Sandbox API reference

# Authorization

- Bearer Token
- Basic auth
- Digest Auth
- OAuth 1.0
- OAuth 2.0
- Hawk Authentication
- AWS Signature
- NTLM Authentication

More details here

# Control Flow

Control flow is managed by

- collection structure (folders, order of scripts in folders) - linear execution
- `postman.setNextRequest()` function - branching and looping
  - `.setNextRequest('request_name')` - sets next request to execute
  - `.setNextRequest(null)` - stops workflow
- data files - multiple executions

Note: no parallel execution available.

# Control Flow: pm.sendRequest Anti-pattern

The `pm.sendRequest` function allows sending HTTP/HTTPS requests asynchronously. Simply put, with asynchronous scripts, you can now execute logic in the background if you have a heavy computational task or are sending multiple requests.

Some things to know about pm.sendRequest():

1. It will break your test report - see newman issue #1303
2. ...

More info on Postman Sandbox API reference.

# Shared Functions

Pre-request script of the first request:

```
pm.globals.clear();

pm.globals.set('loadUtils', function loadUtils() {
    let utils = {};

    utils.yourFunction = function yourFunction() {
        console.log('foobar');
    }

    return utils;
} + '; loadUtils();');
```

Pre-request script or test of any request:

```
const utils = eval(pm.globals.get('loadUtils'));
utils.yourFunction();
```

Note: according to documentation “Global variables are always be stored as strings. If you’re storing objects or arrays, be sure to JSON.stringify() / JSON.parse() them.”

But if you don’t want to export your globals in files it’s totally fine to save objects in them.

# Error Handling

To avoid interruption of collection execution all the code that can fail should be located either in:

- `pm.test`
- or `try ... catch`

```
try {
    const res = pm.response.json();
} catch (e) {
    console.log('Unable to parse response');
}

pm.test(request.name + ' - Status code is 422', function() {
    pm.response.to.have.status(422);
});

if (res && res.id) {
    try {
        const utils = eval(pm.globals.get('loadUtils'));
        utils.deleteObject(res.id);
    } catch (e) {
        console.error(`Not able to delete obj ${res.id}`);
    }
}
```

# Connection To Database?

Not supported. Possible workaround - REST-enabled database, e.g.:

- PostgREST or pREST for Postgres
- Eve, RESTHeart and others for MongoDB

# Useful Links

Videos:

- POST/CON 2018 workshops
- Postman & Exploratory testing by Amber Race

Repositories:

- Postman by Examples by Danny Dainton

Other:

- Postman Blog

# Q&A