

• • • • • • • • • •

• **Introduction to API testing using Postman** •

• **tools or how to create 1258 tests in 27** •

• **minutes** •

• • • • • • • • • •

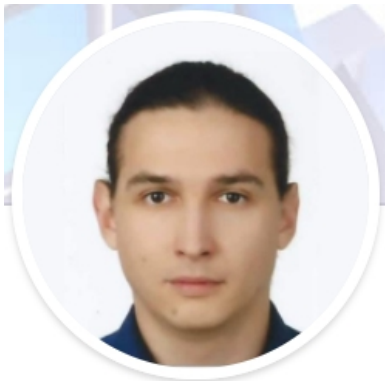
Who are we?

Svitlana Samko

<https://www.linkedin.com/in/svitlana-samko/>

Senior Developer in Test
over 14 years of web development practice
over 10 delivered software projects for middle and large business

My most beneficial skill: *I like to learn the business from the inside. Only so one can be sure that we build the right product in the right way at any stage of the development process.*



Andrii Stepura

<https://www.linkedin.com/in/andriistepura/>

Test Automation Lead, Senior Quality Assurance Automation Engineer
over 15 years of web development practice
over 300 delivered web projects as PO / Dev / Analyst / QA

My most beneficial skill: *Imagination to think like a stakeholder. Every piece of software starts from an idea. The first written code lines are just half of the delivery of that idea.*



Definition of done

1

Theory:

- What is API
- What is HTTP
- CRUD by HTTP methods

2

Analysis:

- Example project
- Familiarizing with business needs
- Business requirements
- Develop system requirements

3

Configuration:

- Install tools: Postman, Postman Interceptor, Node.js with NPM, Newman, Notepad++, Git Bash
- Deploy "Demo API v.1.0"

4

Test:

- Develop system requirements
- Create tests
- Prioritize tests
- Exploratory testing with Postman
- Postman environment variables
- Report bugs with Postman code (cURL)
- Re-testing by Postman Runner with smoke tests set example
- Verification tests with Postman Runner at DDT (data driven tests) example

5

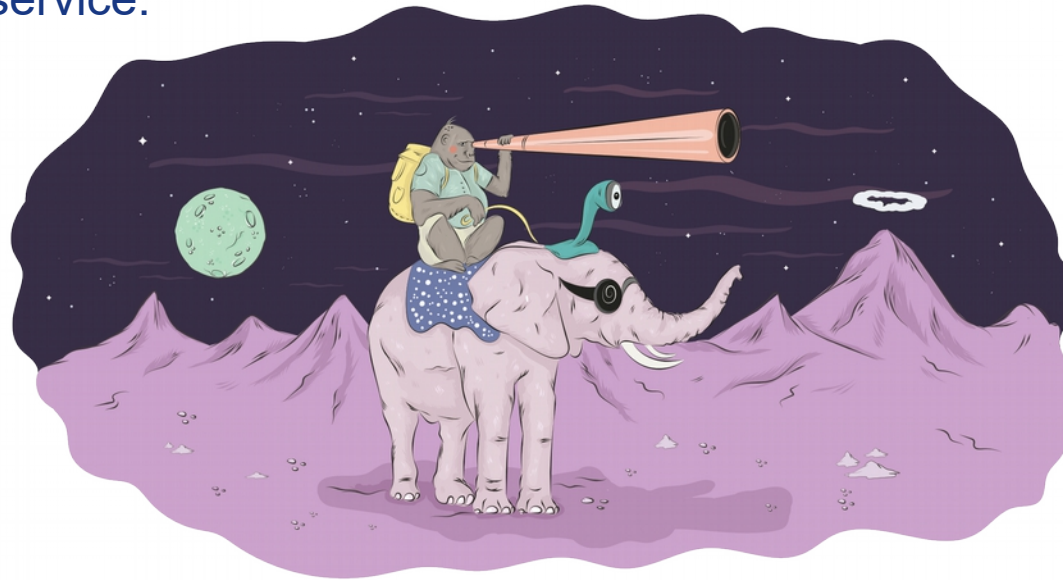
Continuous integration:

- Regression testing with Postman CLI tool Newman
- Verification of fixed API v.1.x ... 2.0

What is API?

API - Application Programming Interface

a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.



Purpose:

Just as a graphical user interface makes it easier for people to use programs, application programming interfaces make it easier for developers to use certain technologies in building applications.

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

What is API? (simplification for web rest APIs)

Just as a graphical user interface makes it easier for people:
open https://en.wikipedia.org/wiki/Application_programming_interface
Your browser sends requests to website, receives/sends packages
(example in networks tab you can see this), then provides responses to
UI model and displays it for you.

The screenshot shows a web browser window displaying the Wikipedia page for "Application programming interface". The page content includes a definition of an API and a list of related topics. The browser's developer tools are open, showing the "Network" tab with a list of requests. The selected request is "https://en.wikipedia.org/load.php?debug=false&lang=en&modules=script&only=scripts&skin=vector", which is a JSON response. The response body is visible in the "Response" pane, showing a large JSON object containing various metadata and content details for the article.

API makes it easier for developers to use certain technologies in building applications.

API just sends a request and receives a response

`curl get https://en.wikipedia.org/wiki/Application_programming_interface`

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

What is HTTP?

HTTP means Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative and hypermedia information systems.

HTTP is the foundation of data communication for the World Wide Web.

The HTTP/1.0 specification defined the **GET**, **POST** and **HEAD** methods and the HTTP/1.1 specification added 5 new methods: **OPTIONS**, **PUT**, **DELETE**, **TRACE** and **CONNECT**.

```
josh@blackbox:~$ telnet en.wikipedia.org 80
Trying 208.80.152.2...
Connected to rr.pmtpa.wikimedia.org.
Escape character is '^]'.
GET /wiki/Main_Page http/1.1
Host: en.wikipedia.org

HTTP/1.0 200 OK
Date: Thu, 03 Jul 2008 11:12:06 GMT
Server: Apache
X-Powered-By: PHP/5.2.5
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding, Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip, Cookie;string-contains=enwikiToken;string-contains=enwikiLoggedOut;string-contains=enwiki_session;string-contains=centralauth_Token;string-contains=centralauth_Session;string-contains=centralauth_LoggedOut
Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
Content-Length: 54218
Content-Type: text/html; charset=utf-8
X-Cache: HIT from sq39.wikimedia.org
X-Cache-Lookup: HIT from sq39.wikimedia.org:3128
Age: 3
X-Cache: HIT from sq38.wikimedia.org
X-Cache-Lookup: HIT from sq38.wikimedia.org:80
Via: 1.0 sq39.wikimedia.org:3128 (squid/2.6.STABLE18), 1.0 sq38.wikimedia.org:80 (squid/2.6.STABLE18)
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="keywords" content="Main Page,1778,1844,1863,1938,1980 Summer Olympics,2008,2008 Guizhou riot,2008 Jerusal
    ...
    ... This content has been removed to save space
    ...
    <li id="privacy"><a href="http://wikimediafoundation.org/wiki/Privacy_policy" title="Wikimedia:Privacy policy">Privac
    y policy</a></li>
    <li id="about"><a href="/wiki/Wikipedia:About" title="Wikipedia:About">About Wikipedia</a></li>
    <li id="disclaimer"><a href="/wiki/Wikipedia:General_disclaimer" title="Wikipedia:General disclaimer">Disclaimers</a>
  </li>
  </ul>
</div>
<script type="text/javascript">if (window.runOnLoadHook) runOnLoadHook();</script>
<!-- Served by srv93 in 0.050 secs. --></body></html>
Connection closed by foreign host.
josh@blackbox:~$
```

CRUD by HTTP methods

In computer programming create, read, update, and delete (as an acronym CRUD) are the four basic functions of persistent storage. The Data Distribution Service for real-time systems (DDS) is an Object Management Group (OMG) machine-to-machine (sometimes called middleware) standard that aims to enable scalable, real-time, dependable, high-performance and interoperable data exchanges using a publish–subscribe pattern.

Operation	HTTP	DDS
Create	POST	write
Read (Retrieve)	GET	read / take
Update (Modify)	PUT / PATCH	write
Delete (Destroy)	DELETE	dispose

Example demo API project

What: Online Currency Converter API

Functionality for API clients:

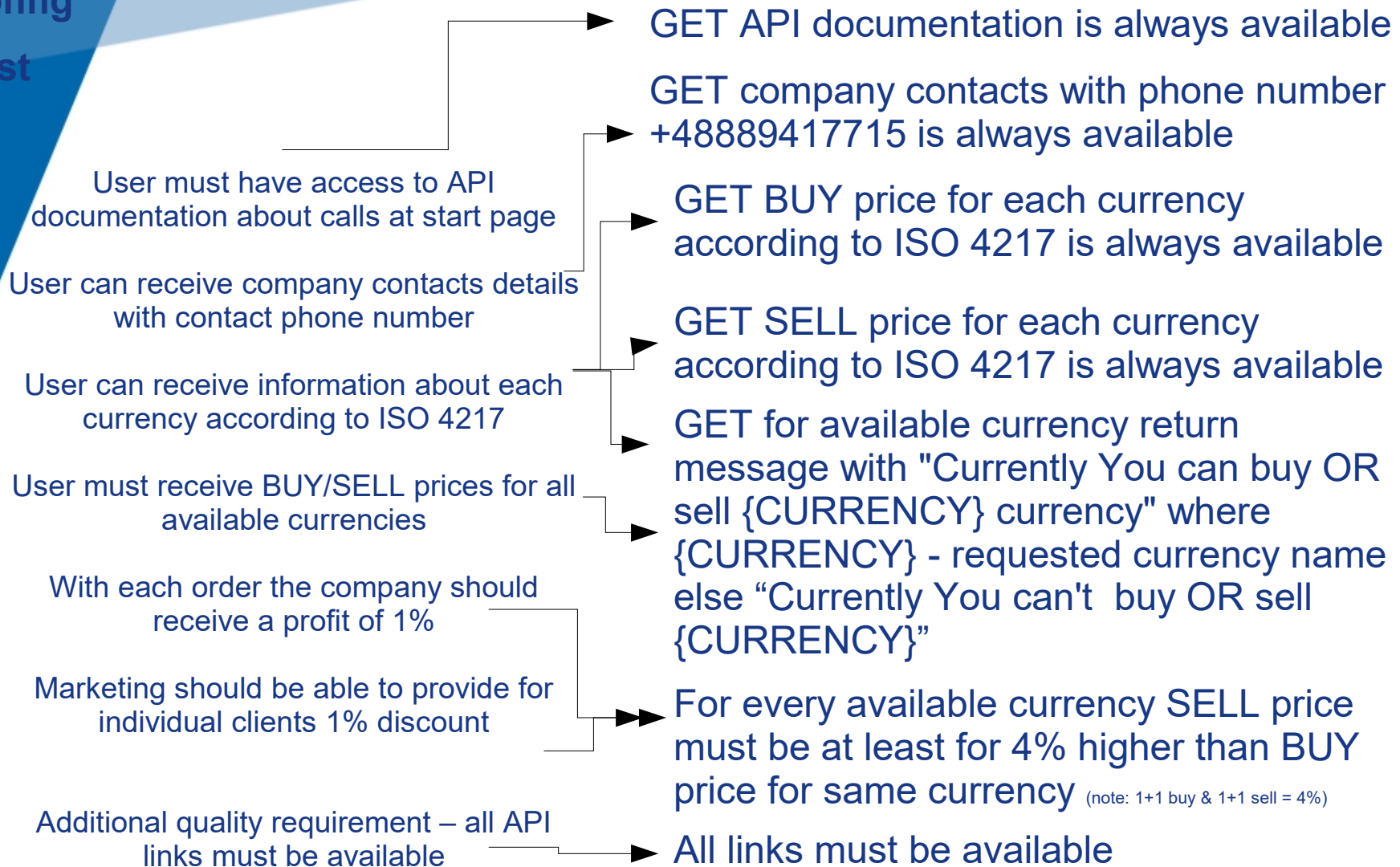
- Receive information about calls available in API
- Receive information about currency BUY price
- Receive information about currency SELL price



Business requirements

- User must have access to API documentation about calls at start page
- User can receive information about each currency according to ISO 4217
- User must receive BUY/SELL prices for all available currencies
- With each order the company should receive a profit of 1%
- Marketing should be able to provide for individual clients 1% discount
- User can receive company contacts details with contact phone number

Develop system requirements



- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Install tools

Software	Resource
Postman native apps	https://www.getpostman.com/
or Postman Chrome app	https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=en https://chrome.google.com/webstore/detail/postman-interceptor/aicmkgpgakddgnaphhhplifpcfhicfo?hl=en
Node.js with NPM	https://nodejs.org/en/download/
Newman	https://www.npmjs.com/package/newman \$ npm install newman --global;
Notepad++	https://notepad-plus-plus.org/download/
Git bash	https://git-scm.com/downloads

Deploy “Demo API v.1.0”

1. Run Git bash
2. Navigate to any folder where create work directory, as ex.:
cd c:
3. Clone repo with demo API:
`git clone https://github.com/AndriiStepura/letslearnapitesting.git`
4. Open directory with demo API:
`cd c:\letslearnapitesting`
5. Install json-server for emulate API with command:
`npm install -g json-server`
6. run first version of demo API with command:
`json-server api_1.0.json`
7. Open in browser this address 127.0.0.1:3000 (or `http://localhost:3000`)

Tests prioritization

Let's choose which system requirements will be covered with tests:

Smoke tests:

GET API
documentation always
available

GET company contacts
with phone number
always available

GET BUY list always
available

GET SELL list always
available

Whole set of tests:

GET for any available currency return
message with info "Currently You can
buy/sell {CURRENCY} currency".

GET for any unavailable currency return
message "Currently You can't buy/sell
{CURRENCY} currency"

GET BUY price for each currency
according to ISO 4217 always available

GET SELL price for each currency
according to ISO 4217 always available

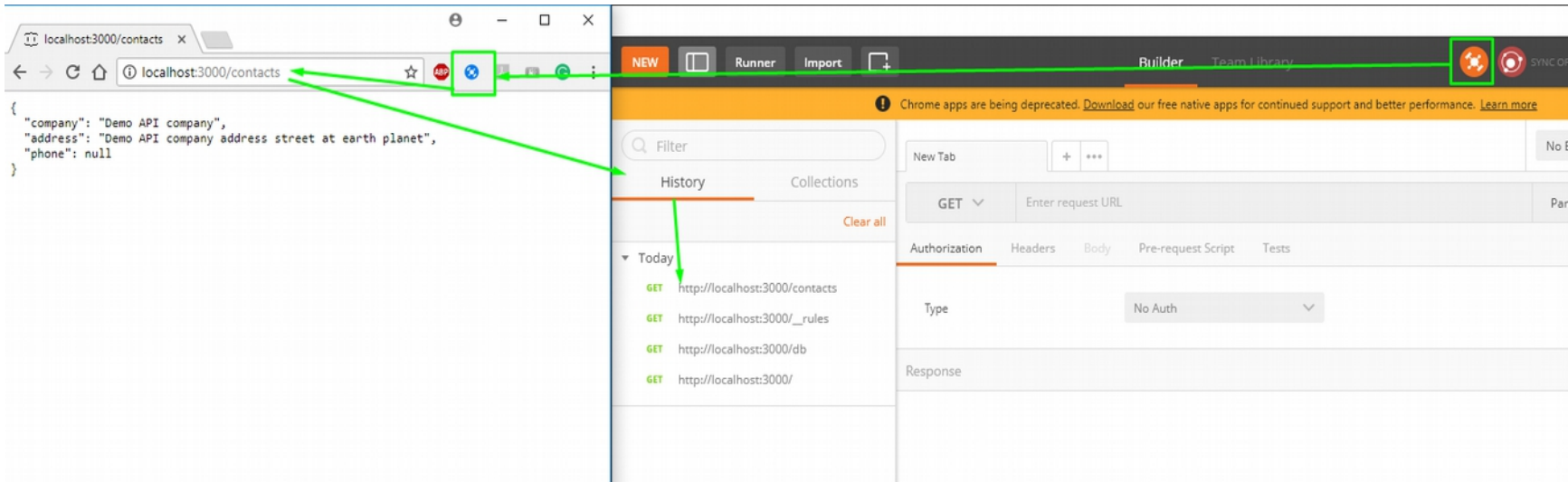
For every available currency SELL price
must be at least for 4% highest than BUY
price for same currency

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Exploratory testing with Postman (intercept requests)

For fast start capture from your browser you may use Postman Chrome:

1. Run Postman Chrome App
2. Set Postman Interceptor - ON mode
3. Open API home page 127.0.0.1:3000 and switch to any resource

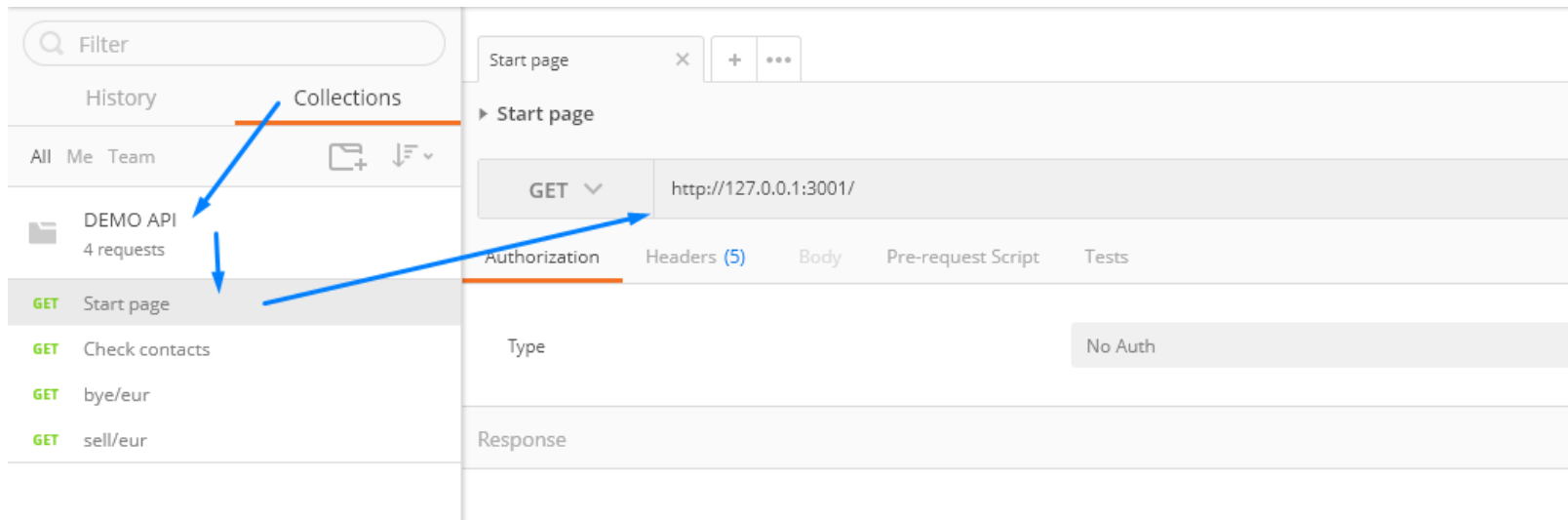
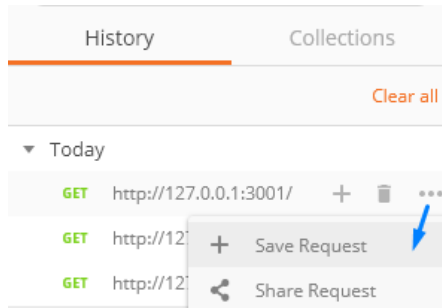


Your browser REST methods from Chrome are available in Postman History

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Exploratory testing with Postman

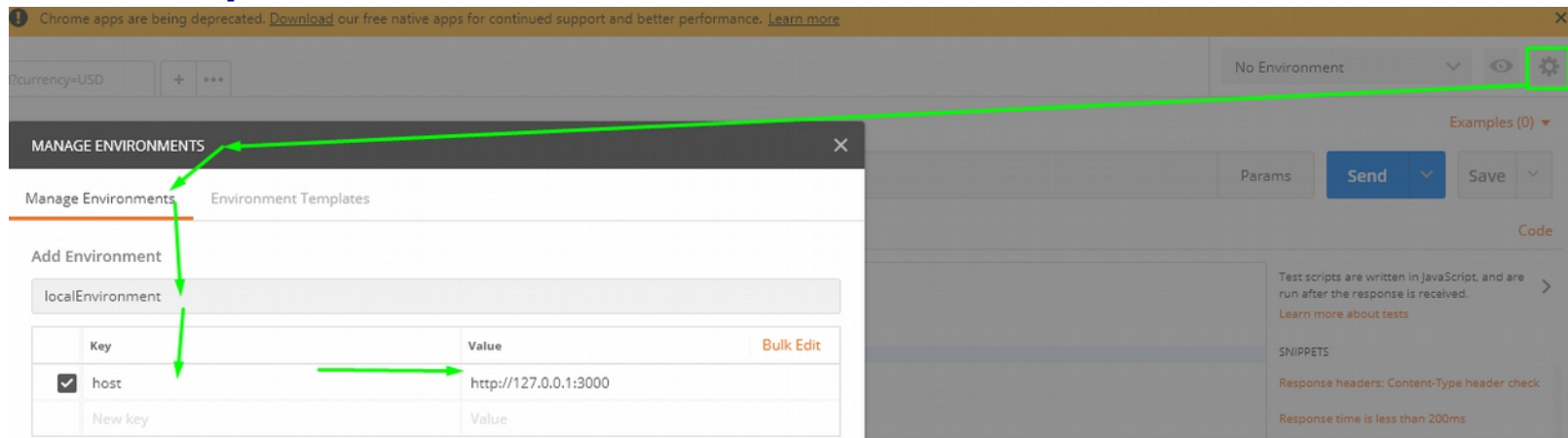
Save requests as collection for future use:



- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Postman environment variables

At top right corner click settings and add new environment, as example:
localEnvironment
with variable key/value (*variableName* - *value*)
host - **http://127.0.0.1:3000**



now we may use this variable in endpoints as `{{variableName}}` (ex. `{{host}}`)



Exploratory testing with Postman (intercept requests)

Postman may capture your request from browser, apps, mobile devices...

1. For Postman native apps navigate to postman folder/app-* as ex.:
Run in cmd console "postman --proxy-server=localhost:5555" or manually
2. Create proxy connection
3. Open API home page 127.0.0.1:3000 and switch to any resource.

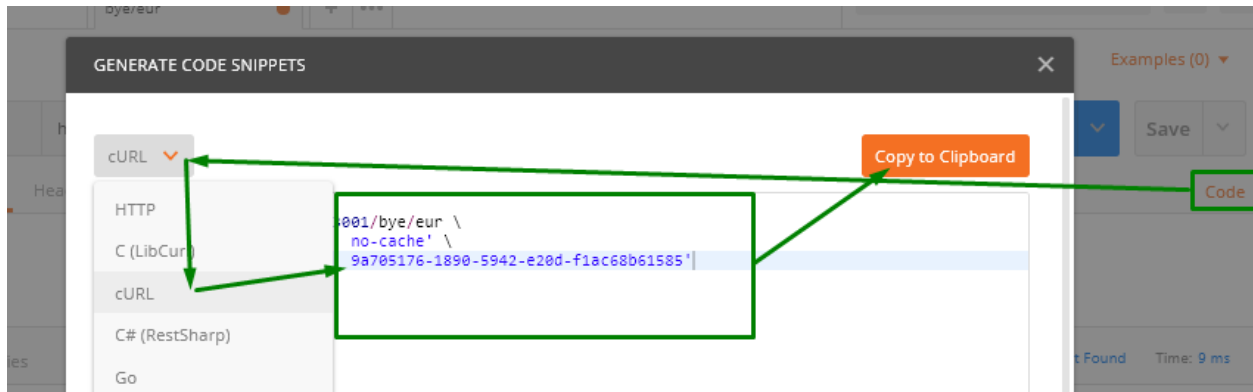
/*IT'S TASK FOR
HOME WORK*/

Your requests are available in Postman History now

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Report bugs with Postman code (cURL)

Faster way to report about bugs with steps to reproduce is Postman code:



Postman code is available for many program languages, as ex. - cURL:

```
$ curl -X GET http://127.0.0.1:3001/bye/eur
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload    Total   Spent    Left   Speed
100  146  100  146    0     0   146      0  0:00:01 --:--:--  0:00:01 142k<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot GET /bye/eur</pre>
</body>
</html>
```

Re-testing by Postman Runner with smoke tests set example

Re-testing with smoke test implementation example:

1. According to requirements:
let's create collection

*GET API documentation always available
GET BUY list always available
GET SELL list always available
GET company contacts with phone +48889417715 number always available*

2. For each add in "Test" tab verification rule.

Note – very useful as base use snippets at right column.

3. Click at collection folder arrow and run all tests set

/*LET'S TRY TO CREATE THESE
TESTS NOW OR GET ALL READY
EXAMPLES AT NEXT SLIDE*/

Tips:

//how to parse JSON response in Java Script?

```
var jsonData = JSON.parse(responseBody);
```

```
tests["Parameter is equal 3"] = jsonData.ifExistArrayParameter.parameterName == 3;
```

Re-testing by Postman Runner with smoke tests set example

Test case: Test script:

GET API documentation
always available

```
GET {{host}}/  
tests["Start page is available"] = responseCode.code === 200;
```

GET BUY list always
available

```
GET {{host}}/buy  
tests["Buy page is available"] = responseCode.code === 200;
```

GET SELL list always
available

```
GET {{host}}/sell  
tests["Sell page is available"] = responseCode.code === 200;
```

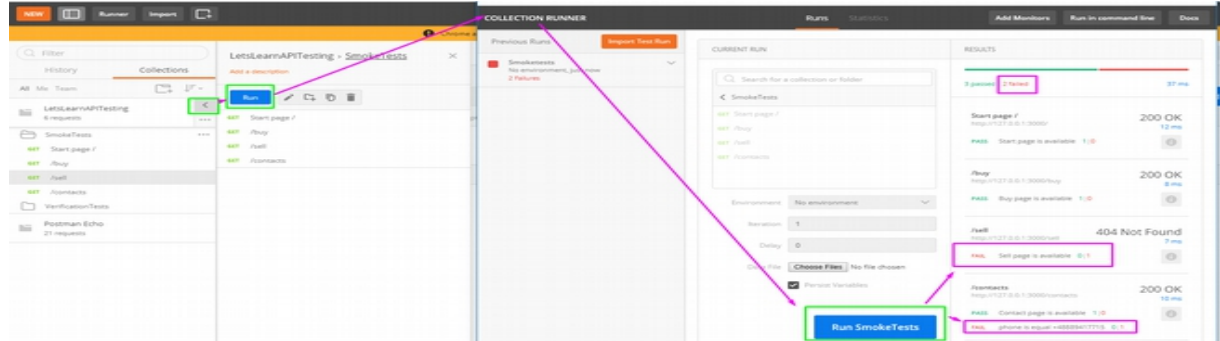
GET company contacts
with phone number
always available

```
GET {{host}}/contacts  
tests["Status code is 200"] = responseCode.code === 200;  
var jsonData = JSON.parse(responseBody);  
tests["phone is equal +48889417715"] = jsonData.data.parameterName  
== "+48889417715";
```

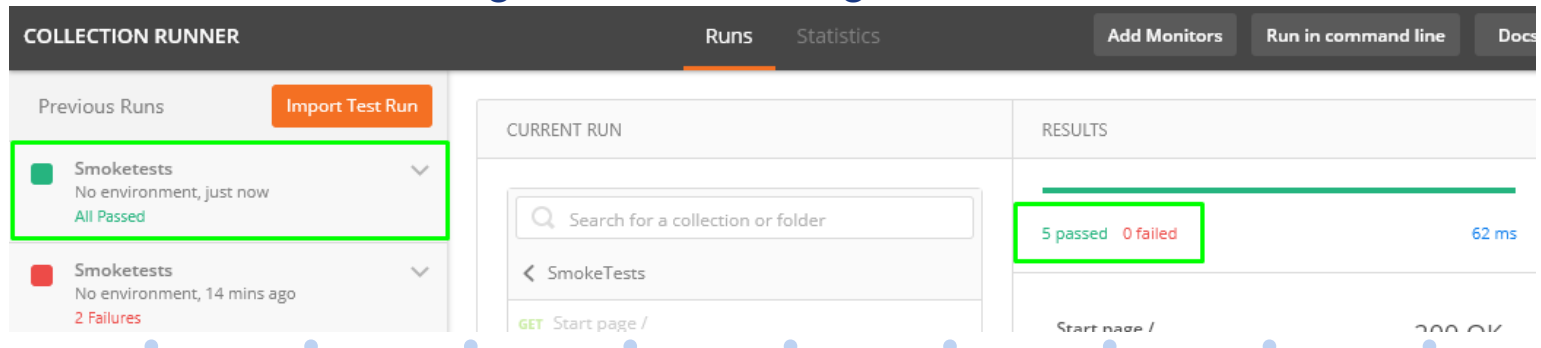

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Re-testing by Postman Runner at smoke tests set example

After smoke tests run we receive 2 failed tests



so, our demo API version is not appropriate candidate to release. After report found issues and they were fixed we received new version api_1.1, let's deploy it with commands Ctr+C (for stop previous 1.0) and for run 1.1: **json-server api_1.1.json**
After verifying that all smoke tests passed with the new candidate, we can continue increasing the test coverage with new tests.



Verification tests with Postman Runner at DDT example

Let's create verification tests for fully test set coverage of all requirements:
According to requirements:

GET BUY price for each currency according to ISO 4217 always available
GET SELL price for each currency according to ISO 4217 always available
GET for any available currency return message with info at /buy and /sell:
"Currently You can buy {CURRENCY} currency"
"Currently You can sell {CURRENCY} currency"
GET for any not available currency return message with info at /buy and /sell:
"Currently You can't buy {CURRENCY} currency"
"Currently You can't sell {CURRENCY} currency"
For every available currency SELL price must be at least for 4% highest than BUY price for same currency

1. Let's create tests as for just one currency, as example USD.
2. Change checked currency with tests for variable currencyName.
3. Run tests feed with data from csv file with currencies list.

*/*LET'S TRY CREATE THESE TESTS NOW OR
GET ALL READY EXAMPLES AT NEXT SLIDE*/*

Tips:

//How to transfer data between requests?

//You may save response of it as an environment variable

postman.setEnvironmentVariable("savedVariableName", jsonData.parameterName1);

//use it anywhere, as example in tests:

tests["Compare from response with saved"] = jsonData.parameterName2 > environment.savedVariableName;

//Don't forget to remove an unused variable

postman.clearEnvironmentVariable("savedVariableName");

Verification tests with Postman Runner at DDT example

Test case: Test script:

GET for available currency return messages with info at /buy /sell:

"Currently You can buy {CURRENCY} currency"
"Currently You can sell {CURRENCY} currency"

GET {{host}}/buy?currency=USD

```
tests["BUY USD page is available"]=responseCode.code===200;varjsonData=JSON.parse(responseBody);tests["USD BUY exchangeRate exist"]=jsonData[0].exchangeRate>=0;if(jsonData[0].exchangeRate>0){
tests["Message for BUY USD is: \"Currently You can buy USD currency\""]=jsonData[0].message=="Currently You can buy ZWL currency";postman.setEnvironmentVariable("buyPrice", jsonData[0].exchangeRate);
}else{tests["Message for BUY USD is: \"Currently You can buy USD currency\""]=jsonData[0].message=="Currently You can't buy ZWL currency";}
```

GET for not available currency return messages with info:

"Currently You can't buy {CURRENCY} currency"
"Currently You can't sell {CURRENCY} currency"

For every available currency
SELL price must be at least for
4% highest than BUY price for
same currency

GET {{host}}/sell?currency=USD

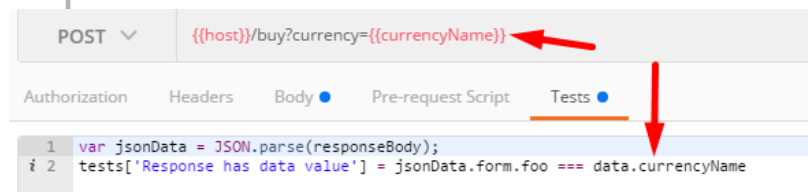
```
tests["SELL USD page is available"]=responseCode.code===200;varjsonData=JSON.parse(responseBody);
tests["USD SELL exchangeRate exist"]=jsonData[0].exchangeRate>=0;if(jsonData[0].exchangeRate>0){
tests["Message for SELL USD is: \"Currently You can sell USD currency\""]=jsonData[0].message=="Currently You can sell ZWL currency";tests[data.currencyName+" SELL price "+jsonData[0].exchangeRate+" at least for 4% highest than BUY price "+environment.buyPrice]=jsonData[0].exchangeRate>=environment.buyPrice/1.04;
}else{tests["Message for SELL USD is: \"Currently You can't sell USD currency\""]=jsonData[0].message=="Currently You can't sell ZWL currency";if(environment.buyPrice){
tests[data.currencyName+" SELL price is not available when BUY price exists"]=true;}}
```

GET BUY price for each currency
according to ISO 4217 always
available

GET SELL price for each
currency according to ISO 4217
always available

2. Change checked currency USD at all requests endpoints and test for variable currencyName.

Tips:



Verification tests with Postman Runner at DDT example

Test case: Test script:

GET BUY price for each currency
according to ISO 4217 always
available
GET SELL price for each
currency according to ISO 4217
always available

GET {{host}}/buy?currency={{currencyName}}

```
tests["BUY "+data.currencyName+" page is
available"]=responseCode.code===200;varjsonData=JSON.parse(responseBody);tests[data.currencyName+"
BUY exchangeRate exist"]=jsonData[0].exchangeRate>=0;if(jsonData[0].exchangeRate>0){
tests["Message for BUY "+data.currencyName+" is: \"Currently You can buy "+data.currencyName+"
currency\""]=jsonData[0].message=="Currently You can buy "+data.currencyName+"
currency";postman.setEnvironmentVariable("buyPrice", jsonData[0].exchangeRate);
}else{tests["Message for BUY "+data.currencyName+" is: \"Currently You can't buy "+data.currencyName+"
currency\""]=jsonData[0].message=="Currently You can't buy "+data.currencyName+" currency";
}
```

GET {{host}}/sell?currency={{currencyName}}

```
tests["SELL "+data.currencyName+" page is
available"]=responseCode.code===200;varjsonData=JSON.parse(responseBody);tests[data.currencyName+"
SELL exchangeRate exist"]=jsonData[0].exchangeRate>=0;if(jsonData[0].exchangeRate>0){
tests["Message for SELL "+data.currencyName+" is: \"Currently You can sell "+data.currencyName+"
currency\""]=jsonData[0].message=="Currently You can sell "+data.currencyName+"
currency";tests[data.currencyName+" SELL price "+jsonData[0].exchangeRate+" at least for 4% highest than
BUY price "+environment.buyPrice]=jsonData[0].exchangeRate>=environment.buyPrice/1.04;
}else{tests["Message for SELL "+data.currencyName+" is: \"Currently You can't sell "+data.currencyName+"
currency\""]=jsonData[0].message=="Currently You can't sell "+data.currencyName+"
currency";if(environment.buyPrice){tests[data.currencyName+" SELL price is not available when BUY price
exists"]=true;
}}
```

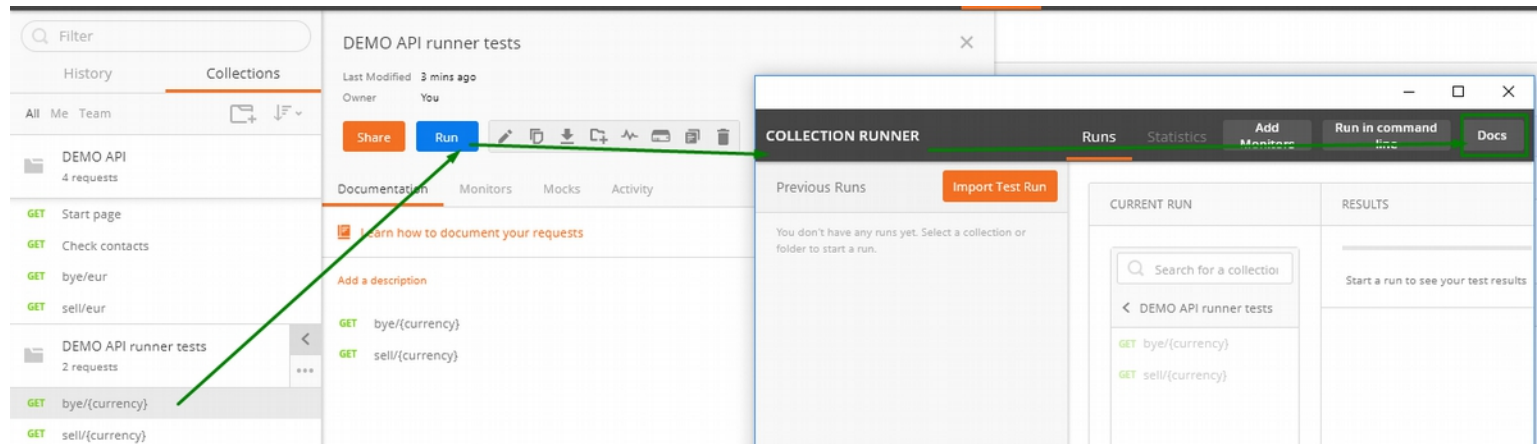
Next step – we need to

“Run tests and feed them with data from .csv file with currencies list”...

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Verification tests with Postman Runner at DDT example

- 3.1. Change currency to variable ex.: `http://127.0.0.1:3000/buy/{{currencyName}}`
- 3.2. Open Postman Runner



- 3.4. Open docs in top right corner and download example files

Data

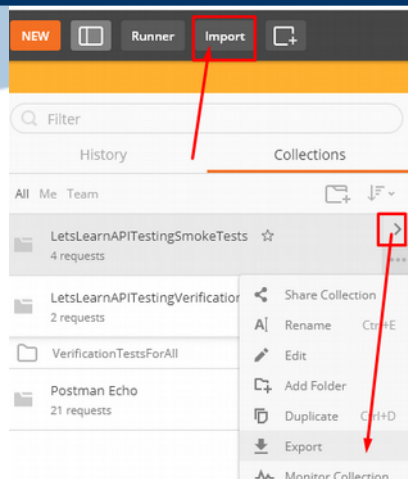
This is used to supply a data file to be used for the collection run. Read more about [data files](#).

- 3.5. Change file for data feed. Your test, as example with few volume of test data, as example three currencies:

USD
EUR
GBP

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

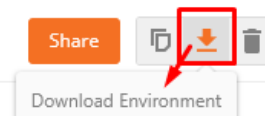
Collections Export/Import and Postman native vs Postman Chrome debug mode



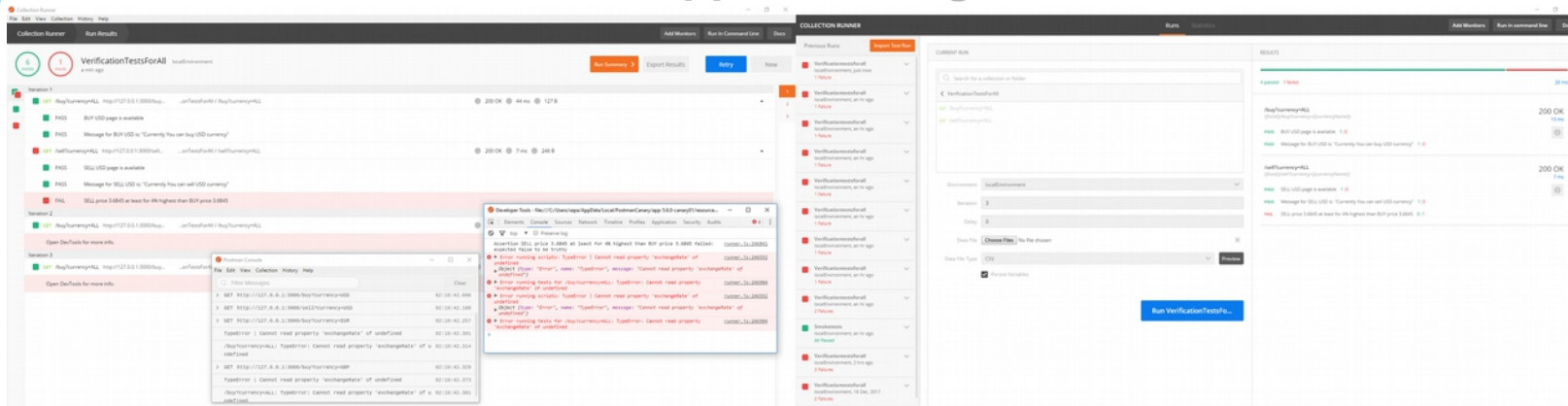
//How to Export/Import collections:

//How export Environments (top right settings button) for Manage Environments:

localEnvironment



//Postman native vs Chrome apps in debug at Runner mode:



Postman native pros:

- Request details
- DevTools, Console

Re-testing by Postman Runner with smoke tests set example

After verification tests run we discovered new bugs, so receive new Demo API version for tests, let's deploy it:

json-server api_1.2.json

and again run our tests runner feed with short currency list.

Now when all our tests passed we are sure that our tests scripts are correct and we can continue and increase test coverage by increment test data for fully covered requirements.

Requirement - for each currency according to ISO 4217

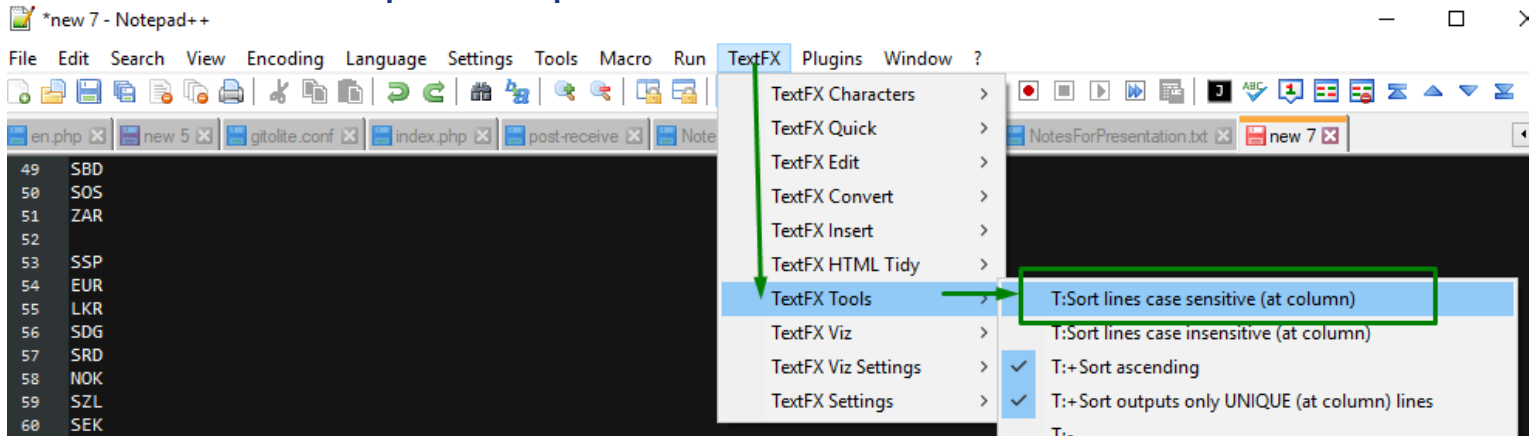
Let's prepare test data...

- 1 Theory
- 2 Analyse
- 3 Config
- 4 Test
- 5 CI

Verification tests with Postman Runner at DDT example

Data preparation, create unique currency list from ISO 4217:

1. Get list from iso.org
2. Sort list for left just unique values



3. Feed this data for the runner and receive newest Demo API version:

json-server api_1.3.json and wait ...

json-server api_1.4.json and wait ...

json-server api_1.5.json and wait ... or maybe let's run faster?

Regression testing with Postman CLI tool Newman

Let's run Postman tests faster with Newman, just four steps are enough:

1. Install Newman - "npm install newman --global"
2. Export collection from Postman and save in the folder as ex.: with name "fulltestscollection.json"
3. Export environment variables as ex.: "envar.json" (optional, if they are not created all variables at Pre-request Scripts and Tests)
4. Run collection and save reports in all available formats as ex.:

```
newman run collection.json --environment envar.json --iteration-data ddt.csv --iteration-count 178 --reporters html,cli,json,junit
```

or with short keys:

```
newman run collection.json -e envar.json -d ddt.csv -n 178 -r html
```

Full tests run failed, so our current demo API version is not appropriate candidate to release. After report found issues and they were fixed we received a new version api_1.2, let's deploy it with commands Ctr+C (for stop previous 1.1) and to run 1.2: json-server api_1.2.json
After verifying that all our created **1258 tests** (12 smoke + 1246 verification) are passed we can approve current candidate to release.

Introduction to API testing with Postman tools or how create 1258 tests in 27 minutes

Gratitude:

Thanks for great tools!



POSTMAN

<https://www.getpostman.com/>

Thanks for great team review:





<http://w2business.pl/>
W2BUSINESS
QA academy

Aleksander Zaleski – QA review
<https://www.linkedin.com/in/aleksander-zaleski-4821b543/>

James Wafer – Translation and grammar review
<https://www.linkedin.com/in/james-wafer-44712165/>



Introduction to API testing with Postman tools or how create 1258 tests in 27 minutes



[All](#) [Images](#) [Videos](#) [News](#) [Maps](#) [More](#) [Settings](#) [Tools](#)

About 21,300,000 results (0.57 seconds)

Postman | API Development Environment

<https://www.getpostman.com/> ▼

Postman is the complete toolchain for API developers, used by more than 3 million developers and 30000 companies worldwide. Postman makes working with APIs faster and easier by supporting developers at every stage of their workflow, and is available for Mac OS X, Windows, Linux and Chrome users.

Get the App

You should too! Download the free app today. Native apps. Choose ...

Postman A complete API ...

Postman is the most complete API Development Environment ...

Navigating Postman

Navigating Postman. Postman provides a multi-window and ...

[More results from getpostman.com »](#)

Docs

Installation and updates - Sending the first request - Requests - Pro

Postman - Sign In


Enterprise user? Sign in here. Sign In. Create Account instead ...


Support


Choose topic. Postman App · Monitoring · Mock Service ...

The Postman

1997 · Drama/Action · 2h 57m

 :)





 [Play trailer on YouTube](#)

6/10
IMDb

9%
Rotten Tomatoes

87% liked this film
Google users



We hope now Postman is not a drama for you!