**Postman The Complete Guide - REST API Testing**

**Section 1: Introduction and the first steps in Postman**

A picture containing graphical user interface

Description automatically generated

Snipets > Set a global variable

GET

*let* response = pm.response.json();

pm.globals.set("uuid", response.uuid);

POST

{

    "name": "John",

    "email": "johm@axample.com",

    "id": "*{{uuid}}*"

}

Snipets > Status code: Code is 200

pm.test("Status code is 200", *function* () {

    pm.response.to.have.status(200);

});

**Section 2: Creating API requests with Postman**

Если переменная с адресом сохранена как переменная коллекции (Не глобальная) то запрос будет фэйлится пока не будет сохранен в этой коллекции

{{baseUrl}}/tools

**Test – Set Collection Variables**

*let* response = pm.response.json();

pm.collectionVariables.set("apiToken", response.accessToken);

There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server.

[Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#headers)

[HTTP headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consist of one single line, which can be quite long.

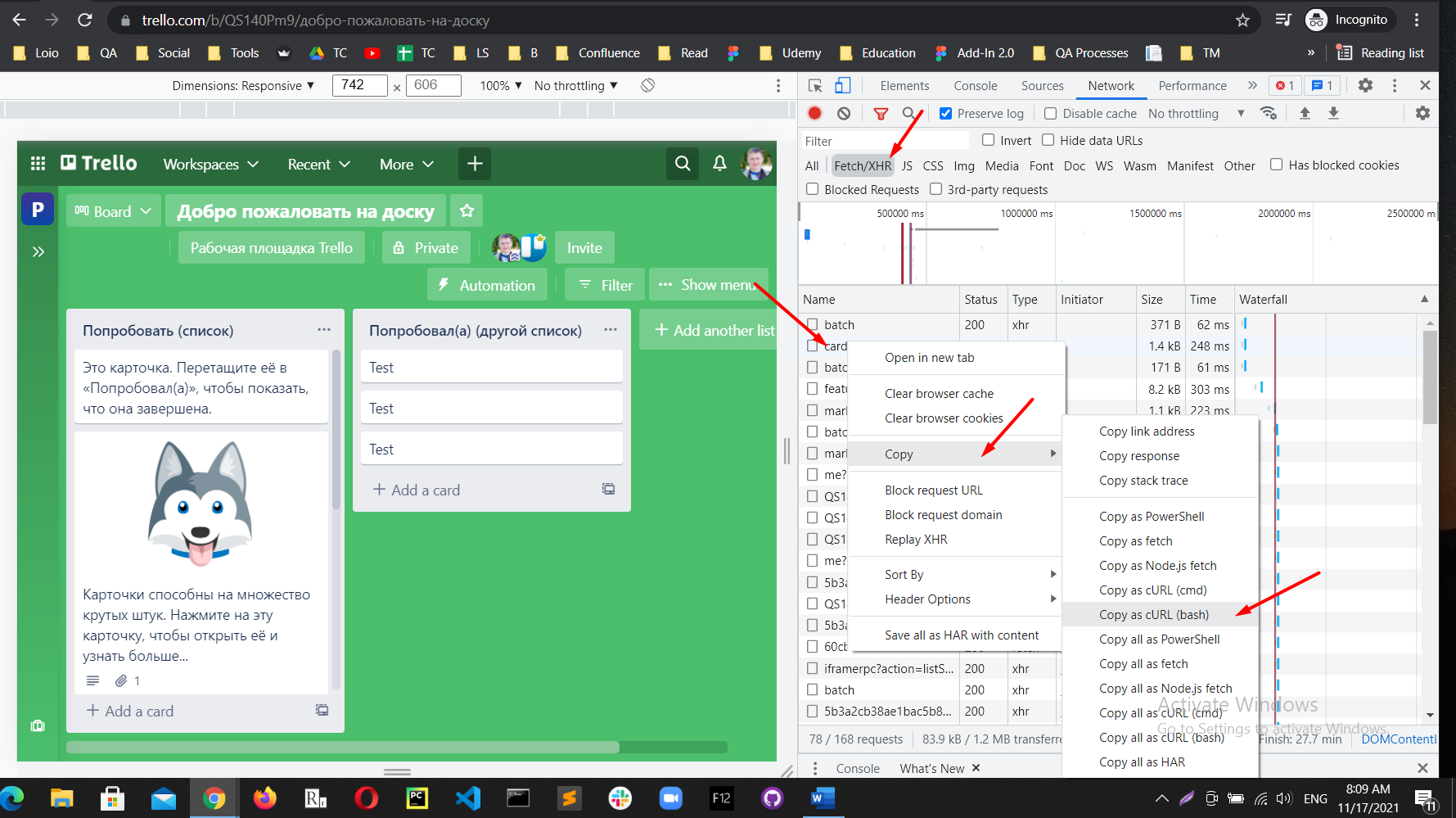
Many different headers can appear in requests. They can be divided in several groups:

* [General headers](https://developer.mozilla.org/en-US/docs/Glossary/General_header), like [Via](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Via), apply to the message as a whole.
* [Request headers](https://developer.mozilla.org/en-US/docs/Glossary/Request_header), like [User-Agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent) or [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept), modify the request by specifying it further (like [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language)), by giving context (like [Referer](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer)), or by conditionally restricting it (like [If-None](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-None)).
* [Representation headers](https://developer.mozilla.org/en-US/docs/Glossary/Representation_header) like [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) that describe the original format of the message data and any encoding applied (only present if the message has a body).

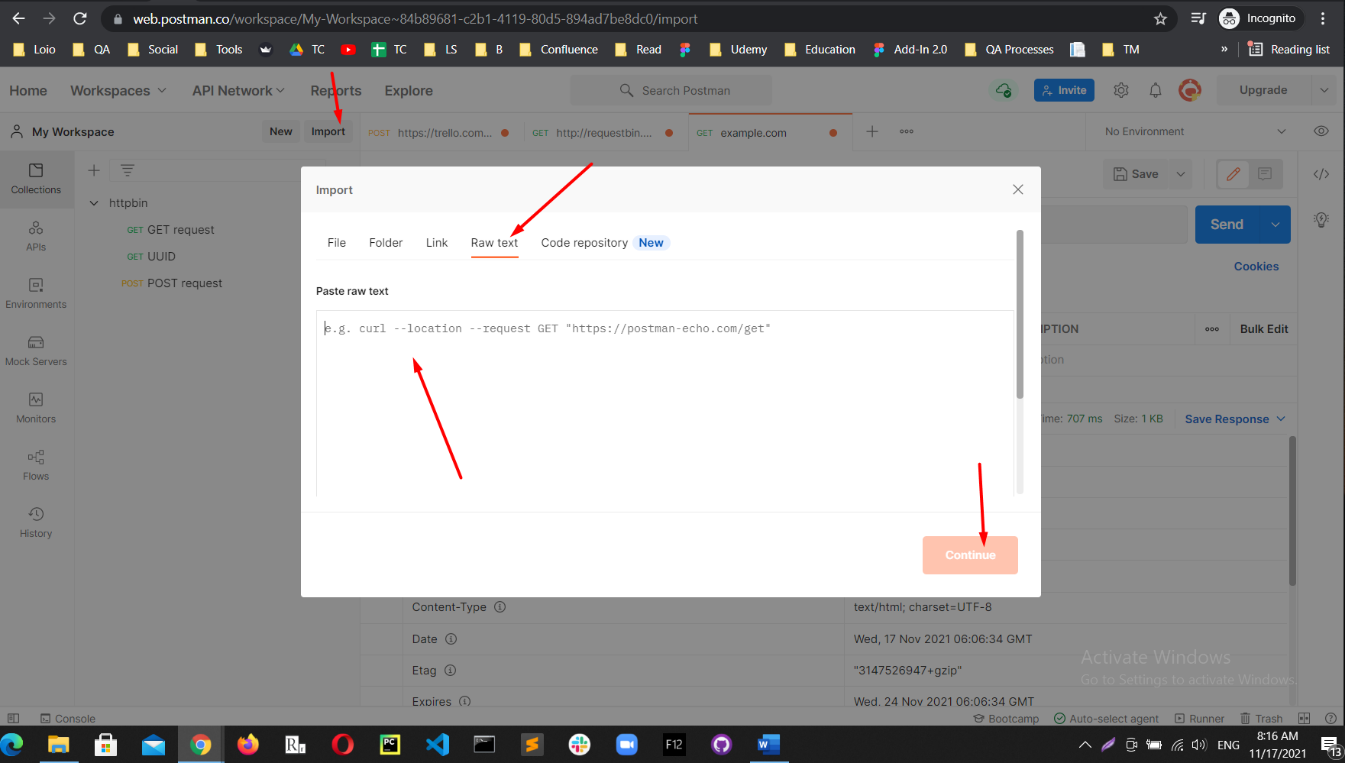
Text

Description automatically generated

Copy API request from browser and import it on Postman



In postman



eGDvNuNzjWORru49z2tuyzL8Uvjd20lB

**Section 3: Writing tests and scripts**

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

#### **Установка и получение переменных**

// глобальные переменные

pm.globals.set(“key”, “value”);

pm.globals.get(“key”);

// переменные окружения

pm.environment.set(“key”, “value”);

pm.environment.get(“key”);

// локальные переменные

pm.variables.set(“key”, “value”);

pm.variables.get(“key”); // если нет локальной, будет искать на уровне выше

#### **Тестирование или asserts**

// с использованием анонимной функции и специальных assert конструкций

pm.test(“Название теста”, **function** () {

pm.response.to.be.success;

pm.expect(“value”).to.be.true;

pm.expect(“other”).to.equal(“other”);

});

// с использованием простого условия и массива tests

tests[“Название теста”] = (“a” != “b”);

tests[“Название теста 2”] = true;

#### **Создание запросов**

// пример get запроса

pm.sendRequest(“https://postman-echo.com/get”, function (err, res) {

console.log(err);

console.log(res);

});

// пример post запроса

**let** data = {

url: “https://postman-echo.com/post”,

method: “POST”,

body: { mode: “raw”, raw: JSON.stringify({ key: “value” })}

};

pm.sendRequest(data, **function** (err, res) {

console.log(err);

console.log(res);

});

#### **Получение ответа для основного запроса**

pm.response.json(); // в виде json

pm.response.text(); // в виде строки

responseBody; // в виде строки

#### **Работа со встроенными библиотеками**

Документация регламентирует наличие некоторого количества встроенных библиотек, среди которых — tv4 для валидации json, xml2js конвертер xml в json, crypto-js для работы с шифрованием, atob, btoa и др.

// подключение xml2js

**var** xml2js = require(“xml2js”);

// преобразование простого xml в json объект

xml2js.parseString("<root>Hello xml2js!</root>", **function**(err, res) {

console.log(res);

});

Некоторые из библиотек, например, как tv4 не требуют прямого подключения через require и доступны по имени сразу.

#### **Получение информации о текущем скрипте**

pm.info.eventName; // вернет test или prerequest в зависимости от контекста

pm.info.iteration; // текущая итерация в Runner

pm.info.iterationCount; // общее количество итераций

pm.info.requestName; // название текущего запроса

pm.info.requestId; // внутренний идентификатор запроса

#### **Управление последовательностью запросов из скрипта**

Стоит отметить, что данный метод работает только в режиме запуска всех скриптов.

// установить следующий запрос

postman.setNextRequest(“Название запроса”); // по названию

postman.setNextRequest(ID); // по идентификатору

// остановить выполнение запросов

postman.setNextRequest(null);

После перехода на следующий запрос Postman возвращается к линейному последовательному выполнению запросов.

#### **Создание глобального хелпера**

В некоторых случаях вам захочется создать функции, которые должны быть доступны во всех запросах. Для этого в первом запросе в секции “Pre-request Script” напишите следующий код:

// создаем и сохраняем хелпер для глобального использования

pm.environment.set("pmHelper", **function** **pmHelper**() {

**let** helpers = {};

helpers.usefulMethod = **function**() {

console.log(“It is helper, bro!”);

};

**return** helpers;

} + '; pmHelper();');

А в последующих скриптах пользуемся им так:

// получаем объект

**var** pmHelper = eval(pm.environment.get("pmHelper"));

// вызываем наш метод

pmHelper.usefulMethod();

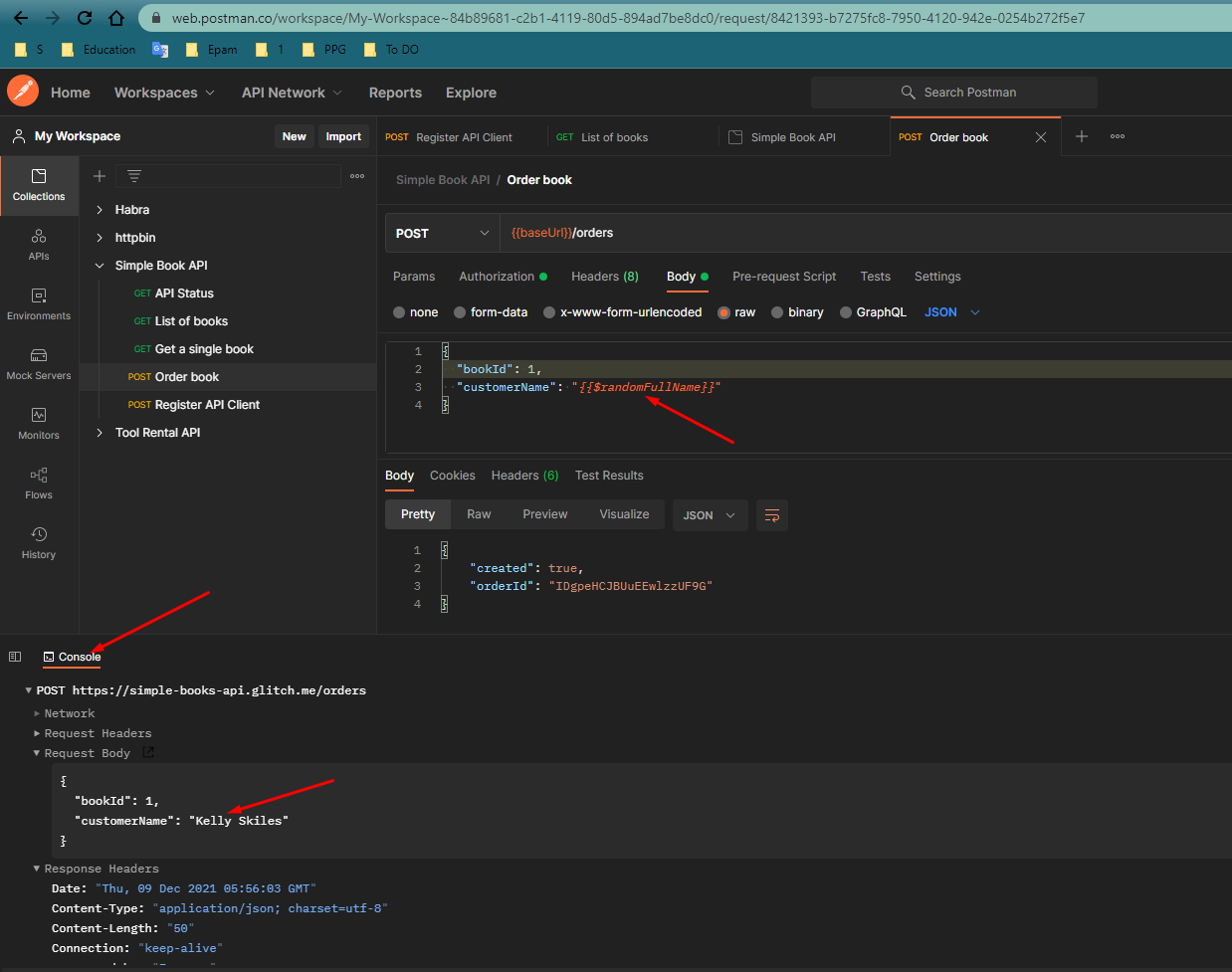
Random Test Data

{

  "bookId": 1,

  "customerName": "*{{$randomFullName}}*"

}



Обновление части информации

PATCH

{{baseUrl}}/orders/zJVpqu33DVrUiJ4NeO5NU

{

  "customerName": "John *{{$randomLastName}}*"

}

Парсим ответ

*const* response = pm.response.json();

*console*.log(response.status);

*console*.log(response['status']);

pm.test("Status should be OK", () *=>* {

   pm.expect(response.status).to.eql("OK");

Затем проверяем что ответ равняется нужному нам значению

Extracting data from the response

*const* response = pm.response.json();

pm.globals.set("orderId", response.orderId);

Request execution order

postman.setNextRequest("List of books");

postman.setNextRequest(null); // STOP !!!

Response body: JSON value check

pm.test("Board should be private", *function* () {

*var* jsonData = pm.response.json();

    pm.expect(jsonData.prefs.permissionLevel).to.eql("private");

});

**Section 4: Writing tests and scripts using variables**

Accessing variables in the scripts

pm.globals.get(“variable\_key”);

pm.globals.set(“variable\_key”, variable\_value); // environment

pm.globals.unset(“variable\_key”); // Delete variable

pm.globals.clear(); // Delete all global variables

Pre-request script in Postman

pm.environment.set(“boardName”, getRandomBoardName());

function getRandomBoardName() {

const boardName = “My new board” + parseInt(Math.rendom() \* 10000);

console.log(boardName);

return boardName;

}

var currentBoardNumber **=** postman.getEnvironmentVariable("currentBoardNumber");

**if** (currentBoardNumber) {

    currentBoardNumber**++**;

} **else** {

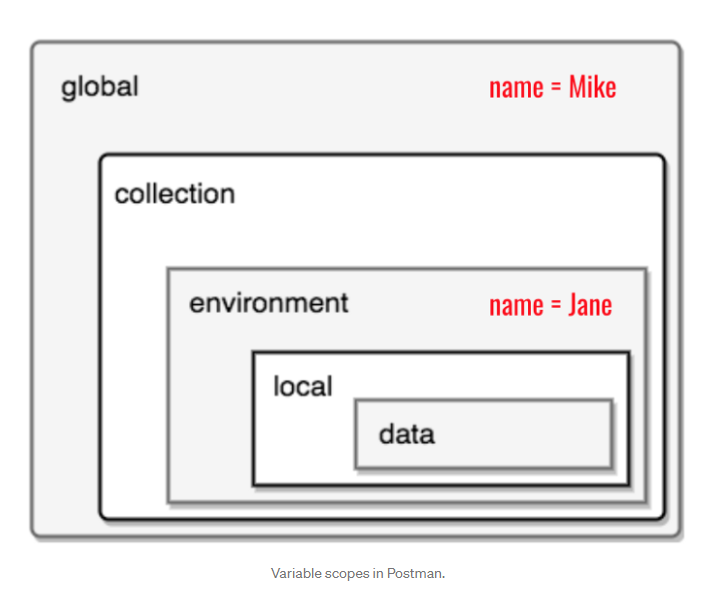
    currentBoardNumber **=** 1;

}

var boardName **=** "My board " **+** currentBoardNumber;

postman.setEnvironmentVariable("boardName", boardName);

postman.setEnvironmentVariable("currentBoardNumber", currentBoardNumber);



## **What are local variables anyway?**

**Really hard to explain.** You define them, use them once and they disappear in the dark — or something like that. **Local variables can only be created from scripts.** There is no GUI that allows you to define such variables or to inspect them.

## **Best practice**

* **Try to avoid them** as they can be hard to understand and can lead to unwanted side effects when used.
* **Try to avoid them as they**behave differently in the Postman App compared to the Collection runner and Newman — so be aware of that.

# **Data variables**

If you need to run the same request or collection against multiple data-sets, the most common solution is to define the data-sets as external files and to run the collection using the Collection Runner or Newman. There are not a lof alternatives to data variables if you go this way.

Data variables can only be set from a CSV or a JSON file and they will exist only during the execution of an iteration.

If we don’t know wat environment (test, prod ) is set, we can use this script

if (pm.environment.get(“environment”) === ‘production’ {

}

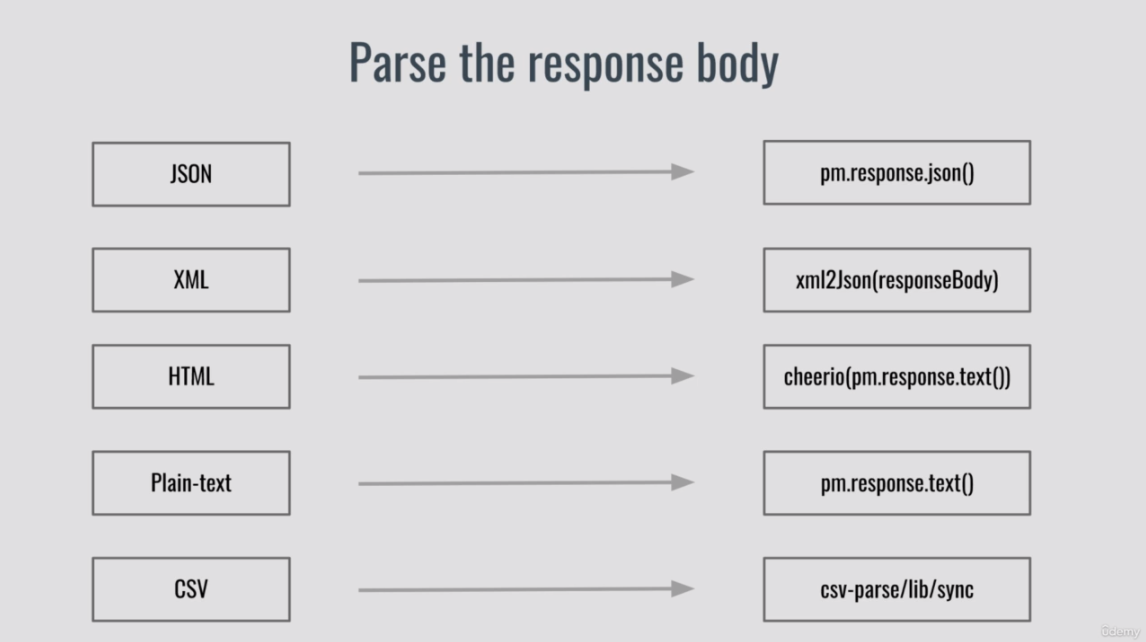
Needed to add in environment variables t

environment: “production”;

**Section 5: [Deprecated – To be removed on 12/31/2021 API workflow using Github & Twilio**

ghp\_rOMzuaO7c12l60Dg6PKgT78A9ZcsfP22CVnu

**Section 6: Advanced assertions**



[Chai Assertion Library](https://www.chaijs.com/)

<https://www.chaijs.com/api/bdd/>



Assertions on arrays

let jsonData **=** pm.response.json();

let manufacturer **=** jsonData.filters[2];

console.**log**(manufacturer);

pm.**test**("Manufacturer should not be allowed", **function**(){

    pm.expect(manufacturer.name).to.eql("MANUFACTURER");

    pm.expect(manufacturer.isAllowed).to.be.false;

})

Assertions on arrays cycle

let jsonData **=** pm.response.json();

let manufacturer;

**for** (let filter **of** jsonData.filters){

**if** (filter.name **==** "MANUFACTURER"){

    manufacturer **=** filter;

    }

}

console.**log**(manufacturer);

pm.**test**("Manufacturer should not be allowed", **function**(){

    pm.expect(manufacturer.name).to.eql("MANUFACTURER");

    pm.expect(manufacturer.isAllowed).to.be.false;

})

Assertions on nested objects

let jsonData **=** pm.response.json();

let commentsStatus **=** jsonData.prefs.comments.status;

console.**log**(commentsStatus);

pm.**test**("Comments status should be disabled", **function**(){

    pm.expect(commentsStatus).to.eql("disabled");

})

let totalPerMemberStatus **=** jsonData.limits['54bba24af6196bd5f824e563'].boards.totalPerMember.status;

console.**log**(totalPerMemberStatus);

pm.**test**("TotalPerMember status should be ok", **function**(){

    pm.expect(totalPerMemberStatus).to.eql("ok");

})

Assertions on nested objects for in

let jsonData **=** pm.response.json();

let commentsStatus **=** jsonData.prefs.comments.status;

console.**log**(commentsStatus);

pm.**test**("Comments status should be disabled", **function**(){

    pm.expect(commentsStatus).to.eql("disabled");

})

let totalPerMemberStatus;

**for** (let key **in** jsonData.limits){

    console.**log**(key, jsonData.limits[key]);

**if**(jsonData.limits[key].hasOwnProperty('boards')){

        totalPerMemberStatus **=** jsonData.limits[key].boards.totalPerMember.status

    };

}

pm.**test**("TotalPerMember status should be ok", **function**(){

    pm.expect(totalPerMemberStatus).to.eql("ok");

})

Testing headers and cookies

So far we have only looked into the status code and the response body for writing tests. **But you can also make assertions on the headers or cookies.**

#### **Headers**

This is how you retrieve a **header** from the response:

pm.response.headers.get('X-Cache')

and in a test:

Header exists: pm.response.to.have.header(X-Cache');

Header has value: pm.expect(pm.response.headers.get('X-Cache')).to.eql('HIT');

#### **Cookies**

In a similar fashion you can test cookies as well.

Cookie exists:

pm.expect(pm.cookies.has('sessionId')).to.be.true;

Cookie has value:

pm.expect(pm.cookies.get('sessionId')).to.eql(’ad3se3ss8sg7sg3');

**Section 7: Automatically running tests**

NodeJs is a JavaScript runtime environment that can execute JavaScript without opening a browser. Newman runs inside the NodeJs runtime environment, so this is why NodeJs needs to be installed (locally on your computer or on the server running Jenkins or any other CI server).

**NPM is the Node Package Manager**. This is like a repository of projects and has knowledge of what requirements each project has. For example, you want to install Newman, but it depends on other libraries which themselves depend on other libraries. To figure out all the dependencies each project needs, you just need to have NPM installed and NPM will do this work for you.

#### **Installation**

**Windows** - basic installation steps can be found here: http://nodesource.com/blog/installing-nodejs-tutorial-windows/

General tip: always restart Windows after installing new software.

**macOS**

- with Homebrew (my preferred way) - https://www.dyclassroom.com/howto-mac/how-to-install-nodejs-and-npm-on-mac-using-homebrew

- without - https://coolestguidesontheplanet.com/installing-node-js-on-macos/

**Linux**

- with Ubuntu

- for other distros, please just search for "install node and npm on ..."

Here's the abbreviated guide, highlighting the major steps:

1. Open the official page for [Node.js downloads](https://nodejs.org/en/download/) and download Node.js for Windows by clicking the "Windows Installer" option
2. Run the downloaded Node.js .msi Installer - including accepting the license, selecting the destination, and authenticating for the install.
   * This requires Administrator privileges, and you may need to authenticate
3. To ensure Node.js has been installed, run node -v in your terminal - you should get something like v6.9.5
4. Update your version of npm with npm install npm --global
   * This requires Administrator privileges, and you may need to authenticate
5. Congratulations - you've now got Node.js installed, and are ready to start building!

In terminal

npm install – newman

to clear windows CMD use Typing 'cls' + Enter (Return) is 4 keystrokes

Run collection from link

newman run <https://www.getpostman.com/collections/80c6fbed419878f064e4>

Run collection from file

newman run JSON/Postman\_collection.postman\_collection.json

**Section 8: Running Newman with other CI servers / tools**

**Section 9: Workflows and Scenarios**

postman.setNextRequest("Request 3");

**if** (pm.globals.**get**("runnedOnce")) {

    postman.setNextRequest(**null**);

}

pm.globals.**set**("runnedOnce", 1);

// Parse the response body

var jsonData **=** pm.response.json();

// Get the first object (element) from jsonData array

var firstBoard **=** jsonData.**shift**();

**if** (firstBoard) {

    // Print the value

    console.**log**(firstBoard.id);

    // Set the variable boardId with the first board id from the list retrieved

    pm.environment.**set**("boardId", firstBoard.id);

    // Set the next request to the delete board

    postman.setNextRequest("DeleteBoard");

} **else** {

    postman.setNextRequest(**null**);

}

**Section 10: Data driven tests: Running a requests multiple times with different data sets**