Portfolio: Postman the complete guide

## Section 1: Introduction and first steps in Postman

I had head of postman before, but never got to actually use the application, so I had to get used to the interface at first, which is a bit different to the one shown on the Udemy course (I suppose it’s an older version). So I followed along the course and tried out some simple GET requests with query parameters.  
  
Graphical user interface, text, application, email, Teams

Description automatically generated

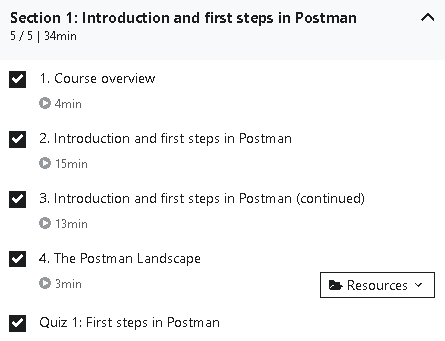
And a simple POST request:  
Graphical user interface, text, application, email

Description automatically generated  
  
Then to get a global variable:  
Graphical user interface, text, application, email

Description automatically generated

And to create a simple test:  
Graphical user interface, text, application, email

Description automatically generated  
Finally to run all these requests that we created together since we saved them as “httpbin” collection, but we had to make sure that the get uuid variable runs before the post request in order for the id to be correct. This is the result:  
Graphical user interface, application

Description automatically generated with medium confidence  
  
Finished the first section on 03/10:  


## Section 2: Creating with API requests

In this section, we started by creating a request bin that could handle our requests that we created on Postman. This was the result:  
Graphical user interface, text, application, email

Description automatically generated

Then we explored path variables, which I figured out quite quickly by the clear examples given on Udemy. We were also shown how to change something in a request url:  
Table

Description automatically generated

Next up, we explored the different tabs and we created some examples on the request bin that we created.  
Headers:  
Graphical user interface, text, application, email, Teams

Description automatically generated  
Text

Description automatically generated  
Body:  
Graphical user interface, text, application, email

Description automatically generated  
A picture containing table

Description automatically generated  
  
  
We also learnt the difference between form-data and x-www-form-urlencoded, which will url encode the values that we submit. We can also make use of json, but we need to specify it in the raw tab and make sure it’s selected in json format, which will change or add an header called Content-Type with the value “application/json”, so the server already knows that it’s written in json format.  
  
Lastly for binary, we use this to send information to send to your api or server, but that we can’t directly enter through postman (for example an image).  
  
Then we also saw that it’s possible to copy a request and import it on Postman and to paste it as a raw text.  
Text

Description automatically generated  
It will fill in all the information (headers, body, etc.) on Postman

For the cookies part, I didn’t have a cookie header on my request bin, so I couldn’t really show pictures, but I followed along and understood everything that was shown on the course.  
Table

Description automatically generated  
Graphical user interface, application, Teams

Description automatically generated  
But I could add a domain name and add a cookie myself, which I did:  
Graphical user interface, application

Description automatically generated

Next time we use a request on example.com, this cookie will also be shown.  
  
How to save a collection and how to save a request to a collection was already shown in section one, but I followed along just in case of any additional information. It was sometimes difficult to follow along since the interface has changed a lot compared to the one shown on the Udemy course.

## Section 3: Writing tests and scripts

We started off this section by learning some more about tests in Postman, like that a test is commonly just a simple assertion , that one request can have multiple tests and that tests only run after the request is completed.  
  
We quickly tried an example out in Postman as shown below, we used a snipped provided in Postman: “ Status code: Code is 200.”  
  
We ran this and got this response:  
Graphical user interface, text, application, email

Description automatically generated  
  
And in the tab “test results” we see the result of out test, which in this case was passed as shown below:  
Graphical user interface, text, application, chat or text message

Description automatically generated  
  
Next up we analyzed the snippet used.  
  
-pm.test: function for writing test specification, anything put in the assertions that causes an error, won’t affect other pieces of code.  
-First parameter is the test name, which is a string. In our example it was “Status code is 200”  
-Second parameter is a so-called callback function will be called when the underlying execution (in this case the assertions) has finished.  
- response assertion API, this can make assertions on the response object (so on the status code, headers, body, etc) in the example it was: pm.response

Next up we wanted to test an API, in we used the Trello API, which we had to register on for a free account. We got a short introduction and made a simple board to test the API.  
  
First task: Create new board  
  
We explored the Trello reference on their developer website , which showed the code to create a board: Graphical user interface, text, application

Description automatically generated  
To create a request we first need a developer API key, which is used to authorize your application. I got mine and created a second parameter in my post request to create a board: Graphical user interface, text, application, email

Description automatically generated  
Next up we need to a third parameter: Token, which I generated and it gives access to my Trello account:  
Graphical user interface, text, application, email

Description automatically generated  
After clicking on send, our board is created, which we can also see on the Trello main website:  
Graphical user interface, application

Description automatically generated

We then wrote some tests as shown in the picture below where it’s shown that it’s possible to write multiple test with multiple assertions and test multiple properties.  
Graphical user interface, text, application, email

Description automatically generated  
This is also possible within nested objects by using a dot between properties, depending on their level:   
Graphical user interface, text, application

Description automatically generated with medium confidence

## Section 4: Writing tests and scripts using variables

We can make use of variables using the double curly brackets as shown in this example:

Graphical user interface, text, application, email

Description automatically generated

We can access variables in the scripts by: pm.globals.get(“variable\_key”);  
and to set a value: pm.globals.set(‘variable\_key”, variable\_value);  
delete: pm.globals.unset(“variable\_key”)  
where variable\_key and variable\_value are your chosen key and value.  
  
So in our example, we can create a board and set the variable “boardId” by using this line of code:  
pm.globals.**set**("boardID", "response.id"); it uses

const response = pm.response.json();

line that we wrote on line 4 earlier.  
  
Environment variables are the same, but more narrow, you can set them by clicking in the top right click corner on “environment quick look” (the eye icon) and then on “add”. It looked quite a lot different than what was shown in the video.Table

Description automatically generated with medium confidence

You can still decide whether you want to make use of an environment or not (if you don’t want to, you just select “no environment”) and moreover it can overwrite global variables.

To access variables in scripts:

pm.environment.set(‘variable\_key”, variable\_value);

pm.environment.get(‘myVariable);

pm.environment.unset(‘myVariable);

We can also use pre-request scripts, these make the requests more dynamic and without assertions.  
These scripts can be written in the pre-request scripts tab as shown below.

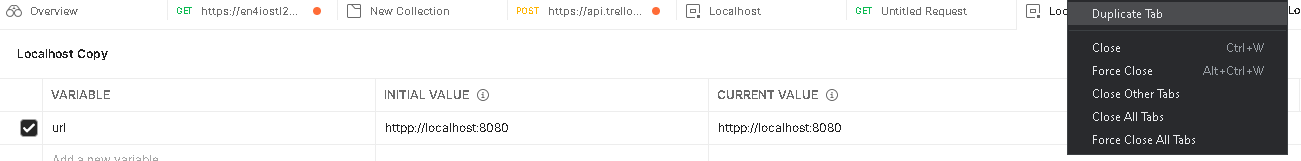
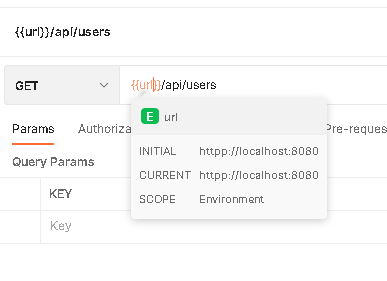
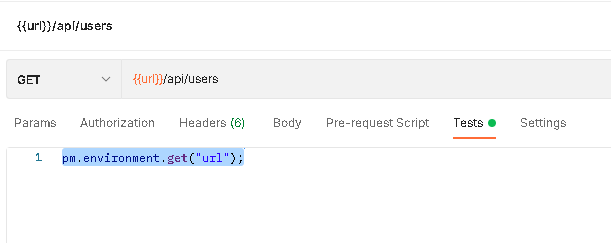
Graphical user interface, text, application, email

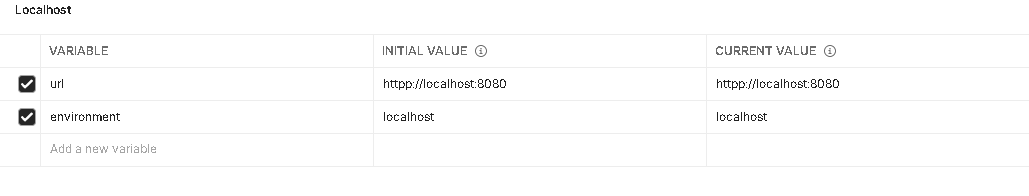
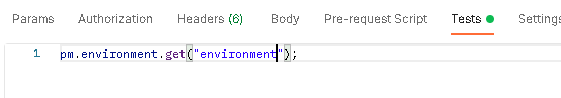
Description automatically generated

We can also have the number be random by using the Math.random function in javascript:  
Graphical user interface, application

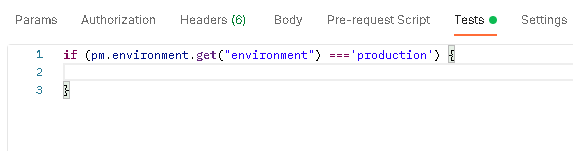
Description automatically generated

**Variable Scopes:**  
Global>Collection>Environment>Data>Local  
(for example if a variable is set in global and environment, environment variable has priority)  
  
Which variable type to use?  
  
**Global Variables**: ideal for quick results and prototyping, most commonly used  
pm.globals.get(“url”)  
pm.variables.get(“url”)  
  
**Environment variables**: Ideal for working with different servers, keeps variables away from global namespace, results in less crowded global namespace, or to pass data between different requests or environments.  
pm.variables.get()  
  
**Collection Variables**: Can’t be updated using scripts, but ideal for storing some constants that do not change during execution. For URLs or auth creds if only one environment exists.  
pm.variables.get(), preferably not in combination with environment variables

To setup different URLs using environments:  
  
Duplicate tab and change each environment value, but keep the variable name.  
  
To then use it in a request:  
  
  
For scripts:   


And so when you change the environment, the URL changes accordingly.  
  
To know in which environment you’re working:  
  
and just change url to environment:  


To run or do tests in specific environment:

  
To debug tests in script:

Console.log(response);  
Console.log(pm.environment.get(“environmentVariable”));

## Section 5: [Deprecated - To be removed on 12/31/2021]

## Section 6: Advanced assertions

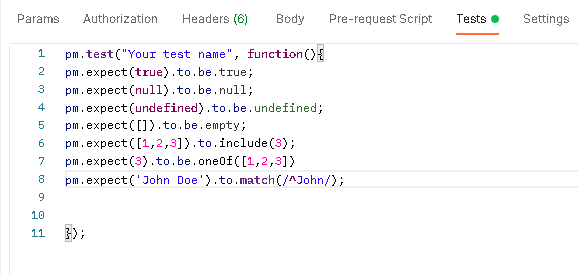
Parse response body:  
JSON 🡪 pm.response.json()

XML🡪 xml2Json(responseBody)

HTML 🡪 cheerio(pm.response.text())

Plain-text🡪 pm.response.text()

CSV🡪 csv-parse/lib/sync  
  
Graphical user interface, text, application, email

Description automatically generated  
equal instead of eql: tests everything, including a= b, so it would be false in this case  
  
Some examples of chai assertion library:  


Imagine if we would have a response body looking like this:  
  
“CompanyID”: 1555,

“RegionId”: 1234,

“filters”: [

{

“id”: 1,  
“name”: VENDOR,  
“isAllowed”:false

},

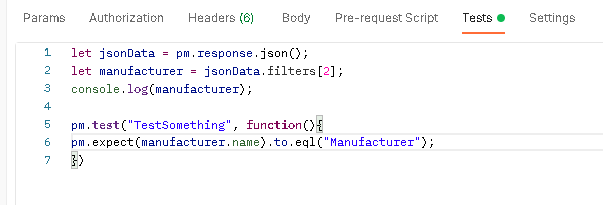
{

“id”: 2,  
“name”: COUNTRY,  
“isAllowed”:true

},

{

“id”: 3,  
“name”: MANUFACTURER,  
“isAllowed”:true

}  
]  
  
Then to test something on the level of filter, so in an array where the 2 resembles the place in array, since array starts counting from 0 and we wanted to test the third object, which so at index 2 and to go on the lower level, we separate by using a dot before writing name: manufacturer.name  
  
for properties like randomly generated string, you’ll have to use manufacturer[‘randomGeneratedString].etc instead of manufacturer.randomGeneratedString.etc

## 