Q**Quick Introduction to Postman and API Testing for Beginners**

**Introduction**

**A quick introduction to APIs**

* API stands for Application Programming Interface.
* The last word, interface, is the most important one to understand.
* An API is an interface to some data on a server, typically stored in a database. This interface allows a program to communicate and thus exchange data with that server. Without this interface, the server would be inaccessible to the outside world.
* an API is an interface to some data stored somewhere remotely, on a different server.
* Postman can connect to a server through an API and exchange data.

**How to install Postman**

* Postman uses a freemium pricing model, and for many use-cases, it is free to use.
* There are two ways of running Postman:
  + in your browser by going to postman.com
  + as a standalone app that you need to install on your computer (Postman is available for Windows, macOS, or Linux)
* DO NOT USE the deprecated Google Chrome extension
* Go to postman.com and create an account.

**Your First Postman request**

* Through the course, we will be using a simple API of a tool rental store which allows us to view tools & place an order
* To know how to use the API, we need to study the API documentation
* Not possible to know how to use an API without some kind of documentation. It is like if you buy a complex machine but get no instruction on how to use it.
* **Tip:** Copy/pasting URLs/params/data from the API documentation ensures you make fewer mistakes.
* **Heads-up!** Make sure you don’t add any newlines or spaces when pasting text in Postman

### Resources

* API documentation:

[https://github.com/vdespa/quick-introduction-to-postman/blob/main/simple-tool-rental-api](https://github.com/vdespa/quick-introduction-to-postman/blob/main/simple-tool-rental-api.md)

[.md](https://github.com/vdespa/quick-introduction-to-postman/blob/main/simple-tool-rental-api.md)

**The HTTP Protocol Explained**

* Postman is the client and has sent a message to the server running the tool rental API
* We call this message a request, as it is requesting some data.
* The message that the server sends in response is called a response
* To make this communication possible, we have used HTTP
* HTTP is a protocol that enables the client, in this case, Postman, and the server which runs the API to communicate.
* A protocol is essentially a set of rules that both parties need to follow.
* **Heads-up!** It is essential to understand some basics of HTTP to be able to read any API documentation and use APIs.
* HTTPS is the secure & encrypted version of HTTP; use HTTPS whenever possible.

The HTTP request has:

* Request method (sometimes called HTTP verb)
* Address / URL
* Headers
* Body

The HTTP response has:

* Status code
* Headers
* Body

**Creating Requests in Postman**

**Postman Collections**

* To save a request, click on save and create or select a collection
* In Postman, we try to avoid having configurations in your requests, just in case something changes.
* Save the baseUrl in a Postman collection variable

- Works only if you have saved the request in a collection

* The variable name is between curly brackets {{baseUrl}}
* Current value
* This is used by Postman when submitting a request
* Private to you / your Postman account
* Not shared with others
* Initial value
  + Exposed to others when sharing the collection
  + Not used by Postman

**Query Parameters**

* **Heads-up!** Postman collection variables are unresolved if the request is not saved in the same collection where the variable is defined.
* Response body: this way of formatting data is called JSON
* Query parameters are a way to send data to the API
* For this API, query parameters in this case are a way to filter data, to get a subset of the data
* Query parameters can be optional or mandatory (as specified by the API documentation)
* Which query parameters are available can only be known by reading the API
* A mistake that every beginner does: Category vs category
* **Tip**: Always copy/paste names from the documentation to avoid making mistakes
* What are [ ] empty brackets: this is not an error, it is just an empty list

**Path Variables**

* Path parameters are required if the endpoint mentions them
* The notation for path variables is :variableName (don’t forget the colon!)
* Path variables in Postman are just a placeholder
* The name of the variable is NOT being sent with the request
* You can have both query parameters and path parameters
  + No question mark
* **Tip**: use the Postman console to inspect the requests and responses

**Query Parameters and Path Variables**

Path parameters

* Mandatory
* Sends data to the APIs
* Part of the endpoint/path

Query

* Mandatory or optional
* Sends data to the APIs
* Start after the question mark ?

**API Authentication**

* API can be public or private
* Private API require authentication
* Even public API may require authentication when creating new data or updating an existing one
* The purpose of the API client registration is to obtain an access token, which is like a password
* The term client does not refer to a customer (think about client-server)
* For us, the API client is Postman
* when working with APIs, we will not get a login form where we can enter a username and password
* We use tokens which are like a temporary password
* Tokens are usually added to headers or as query parameters (see the API documentation)

**Troubleshooting HTTP status codes**

* Typical errors
  + 404 - check the URL or the HTTP request method
  + 400 - check your request body, ensure that JSON is valid
  + 409 - client registered

**HTTP headers**

* HTTP Headers are found in the request and in the response
* Typical request headers
  + Content Type - it is telling the API that the request body is in JSON format
  + Authorization - contains authentication information
* Typical response headers
  + Content Type - it is telling the client (Postman) that the response body is in JSON format

**Using random data in requests(random variables)**

* + Postman offers a long list of random variables
  + Go to any Postman request and start typing {{$ and select an item from the list
  + For example: ${{randomFullName}}
  + In JSON, keep the double quotes if the returned value is a string
  + Show the Postman console to inspect which values have been sent

### Resources

* Dynamic variables in Postman

<https://postman-quick-reference-guide.readthedocs.io/en/latest/dynamic-variables.html>

**How not to use Postman**

* + Not for dealing with user interactions, like filling out forms, and clicking buttons.
  + Not a good tool for performance testing or any other kinds of tests where you send to send a lot of requests in a short time-frame

It can be used for security testing, but this is not the primary focus of the tool.

**Patch request Method**

* + The PATCH request method (if supported by the API), allows you to change existing data
  + With a PATCH request, you don’t need to provide the entire data object, only the properties that need to be changed.

**Delete Request Method**

* + Use DELETE to remove data (if the API supports this)
  + Typically no request body is required
  + Use a GET request to see if the delete was successful.

**Preparing for Automation**

**Automation Basics**

* + Manually testing an API is a lot of work
  + When someone makes a change to the API, we have to manually test all endpoints and parameters to see if the API is working as before.
  + We can let Postman test the API by writing API tests.
  + Automation means that we let Postman do the testing work, and we only step in if something goes wrong.

**Your first API test**

* + Postman has no idea if a request was successful or not
  + We typically write tests to assert if the response contains something that we expect
  + We already know the request, so it is less interesting to write tests for that
  + Postman uses JavaScript for writing scripts
  + The most basic test is to check for the status code

pm.test("Status code is 200", () => { pm.response.to.have.status(200);

});

* + Heads-up! Always make your test fail
  + To make assertions on the response body, you need to parse the JSON response

const response = pm.response.json();

* + This is a typical test structure:

pm.test("Basic test structure", () => { pm.expect(1).to.eql(1);

});

* + Use Postman console to show the value of the property
  + Use both response.status and response[‘status’]

pm.test("Status is UP", () => {

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

});

**Postman Variables**

* + copy/pasting data from one request to the other is annoying and time-consuming.
  + Postman allows you to create different variable types:
    - Collection variables
      * Available only for a collection
    - Environment variables
      * Available only for an environment
      * Useful when you wish to reuse the same collection against different servers running the API, like localhost, testing, and production.
    - Global variables
      * Available for the entire workspace

**Working with Postman variables with scripts**

* + You can define or set a variable value manually, through the Postman UI or from scripts.
  + Getting a collection variable:

pm.collectionVariables.get(“apiToken”)

* + If the Postman variable does not exist, the value of the expression above will be

undefined.

* + To set a collection variable, you can use an expression like the following:

pm.collectionVariables.set(“firstName”, “John”)

* + **Heads-up!** Don’t confuse set with get!
  + To get or set a global variable, just replace collectionVariables with globals

in the expressions above.

**Extracting data from Response**

* + Setting variables from scripts is most useful when we use data from the response, instead of hard-coding a value

const response = pm.response.json();

const tools = response.filter((tool)=> tool.available === true); pm.globals.set(“toolId”, tools[0].id);

### Resources

* JavaScript filter function

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Arr](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter) [ay/filter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)

**Assertions on objects**

* + - If no book is being found, it makes sense to write a test to ensure the request will fail

pm.globals.set("toolId", tool.id); pm.test("Tool found", () => {

pm.expect(tool).to.be.an('object'); pm.expect(tool.inStock).to.be.true; pm.expect(tool.inStock).to.eql(true);

});

# Automated collection runs

## Running the collection manually

* + We have all elements in place that ensure we can do proper test automation:
    - We have tests that ensure the API works as expected
    - We have variables that ensure we don’t need to copy/paste data
  + Go from request to request and run the collection
  + If some tests are still failing, I recommend you pause the video and take a minute to fix them.

Collection Runner

* + The collection runner is a tool built-in Postman that allows us to execute the entire collection with just one click, instead of going through each request.
  + You can drag and drop requests to change the execution order
  + You can disable requests from the execution
  + Iterations: how man times to run the collection (default 1)

Request Execution Order

* + The default execution order is the one given by the collection
  + We can skip the request “Register API client” by using:

postman.setNextRequest("Create order");

* + **Heads-up!** Use postman.setNextRequest and not pm.setNextRequest (the latter won’t work).
  + Alternative: move Register API client at the end

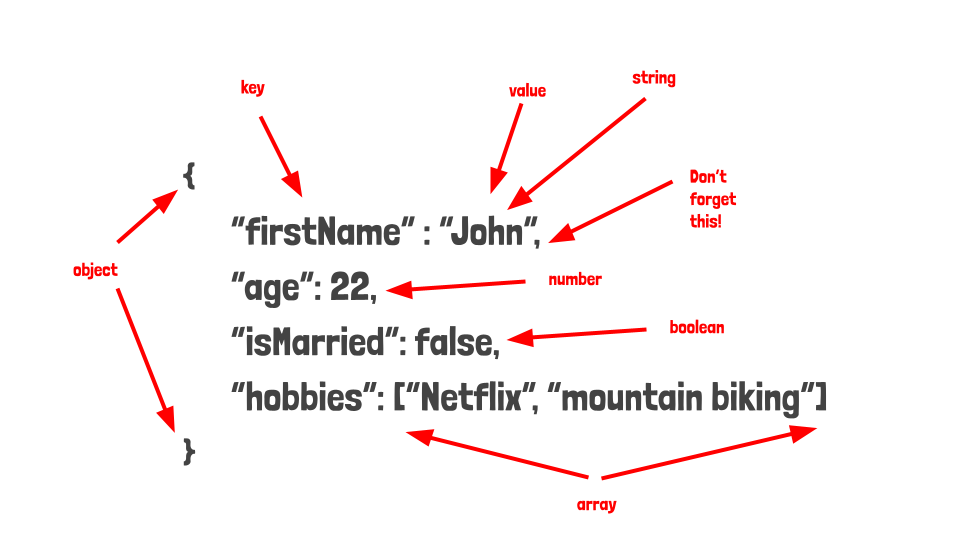
postman.setNextRequest(null);

* + DON’t create an endless loop!

postman.setNextRequest("Status");

**JSON format explained**

* In practical terms, we use JSON to transfer data from one machine to the other
* JSON has a simple key-value format
* The advantage of JSON is that it is both readable by humans as well as by computers.
* { } - curly brackets denote an object
* [ ] - square brackets denote a list (an array of elements)

Make sure you write valid JSON, otherwise the API won’t understand you.

* Typical JSON errors
  + No quotes for strings
  + Simple quotes
  + No comma between lines
  + Comma at the end

**GET VS POST**

GET

* + no data should be changed with GET
  + You can call GET multiple times with no effect
  + While technically possible, it does not carry a payload.

POST

* + Each time you call POST, new data will be created
  + Usually has a payload (request body)