# **Exercise: Postman & Newman**

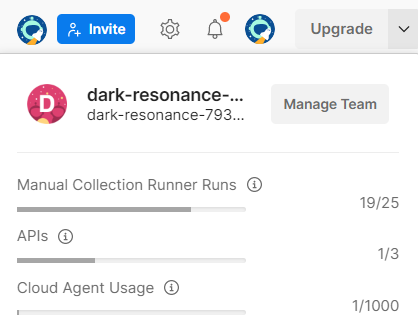
This document defines the exercise assignments for the   
**"QA Back-End Automation" Course @ SoftUni**

## Running the Collection Manually

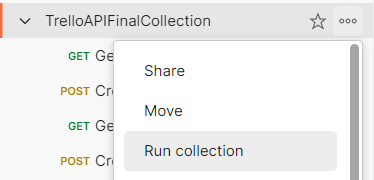
Before we get into automating a collection, we must first cover a necessary step. Ensure that the collection works when you run it by hand. This means checking once more that, as you go through each request one by one from the start to the finish, everything operates smoothly, the requests are doing what they should, and the tests are successful. If you find any tests that don't pass, take a moment to fix them.

## Postman Runner

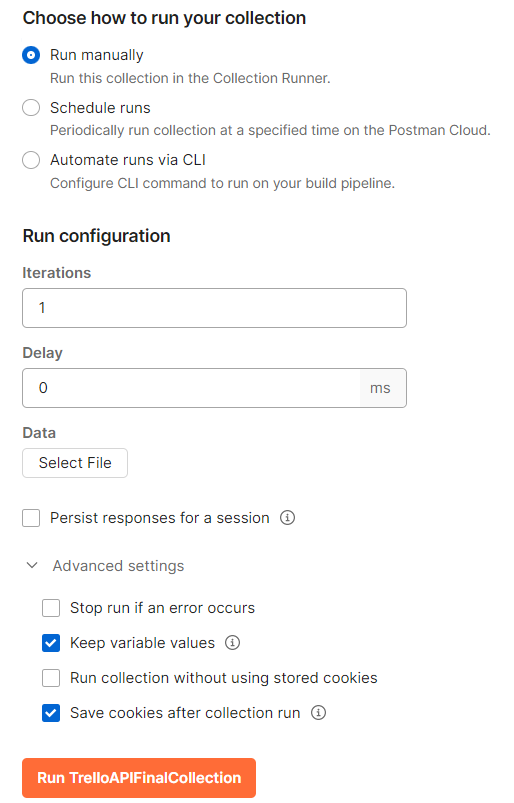
**NB!** For a long time, the Collection Runner was a free tool with no usage limits. However, starting in March 2023, Postman introduced significant restrictions on **how many times you can use the Collection Runner each month**. On the **free plan, you're limited to 25 runs per month**. A run counts each time you press the run button, regardless of how many requests your collection has. As you approach this limit, you'll see a warning, and eventually, you'll reach a soft limit. If you keep using the tool beyond this point, you'll hit a hard limit. To avoid reaching the hard limit and making the Collection Runner unusable, try to minimize how often you execute your collections.

**You can check how many runs do you have left from the Upgrade button.**

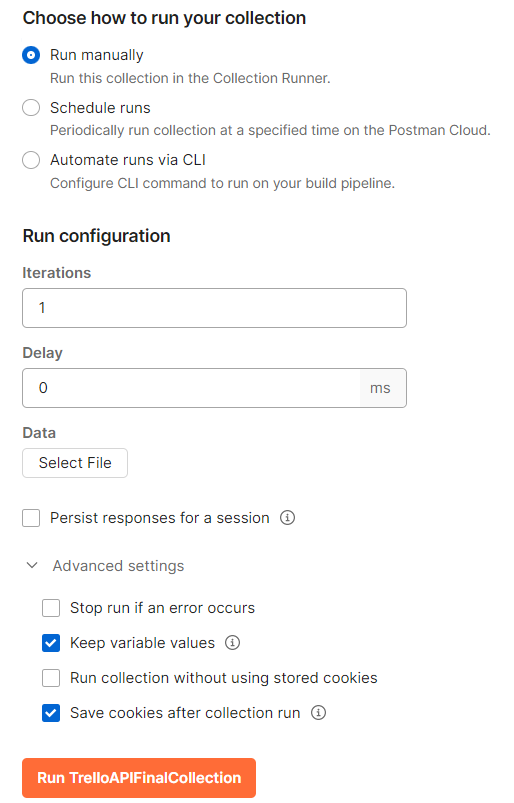
The **Collection Runner** is a tool that is **built into Postman** and allows us to **execute the entire collection with just a single click** instead of manually going over each individual request and clicking the send button. You can execute your API test collections within Postman by **selecting the** **Run collection option from the collection menu**.



The Collection Runner opens in a separate tab. Notice the run order first, which shows how requests will be executed in the same sequence as they appear in the collection. If there's a request you prefer not to run, simply disable it. If you're not satisfied with the execution order, you can easily change it by dragging the request to a new spot. In our case, the current order is perfectly fine.  
**We have few options here:** **Run Manually, Schedule runs, Automate runs via CLI**



### **Running the Collection Manually**



**Iterations:** Determines how many times the collection runs. If you input "2", the collection will run twice. Usually, one iteration sufficient.

**Delay:** Manages the timing between requests, like allowing time for a board to become available after creation, add a delay. This can prevent failures in subsequent requests.

**Advanced Settings:**

**Persist responses for a session**: This saves the responses from your API requests until you close the session, helping you to check them later for any issues.

**Keep variable values**: This keeps the new values of any variables you change during the run for use later in the same session.

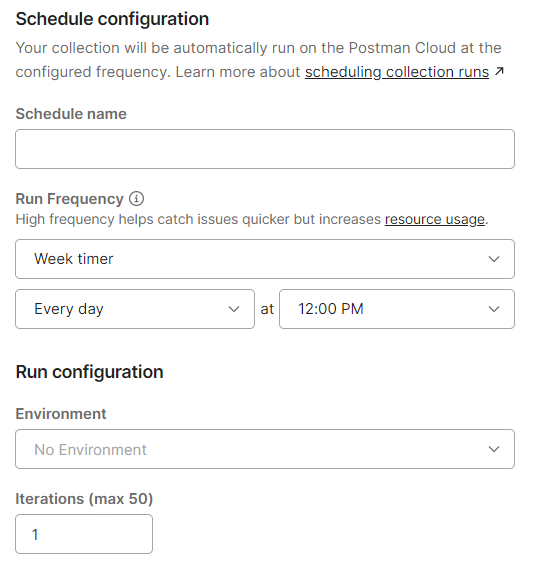
**Run collection without using stored cookies:** This runs your requests without any cookies saved before, making sure your tests are clean.

**Save cookies after collection run**: This saves new cookies from your run to use in future sessions or tests.  
  
**When you running the collection:**

* **Utilize Postman Console for Troubleshooting**: Before running the collection, clear any existing logs in the Postman console to streamline troubleshooting. Console logs are invaluable for debugging, showing executed requests and logged statements.
* **Inspect Request Details**: After running the collection, you can examine each request for additional insights, such as the request URL and response body. If certain details are missing, like the request body, refer to the Postman console for a complete overview.
* **Review and Debug**: The Collection Runner provides a preliminary step towards automation by offering a semi-automatic way to run collections. It generates a report detailing which requests succeeded or failed, aiding in troubleshooting.
* **Ensure Collection Runs Smoothly Before Automating Further**: Verify that your collection runs correctly with the Collection Runner before moving on to more advanced automation tools. If there are issues at this stage, it's easier to debug within Postman.

### **Schedule runs**

To set up a scheduled run for a collection in Postman, which will run automatically without you needing to   
click anything:



In Postman, the 'Schedule runs' feature is designed for automating collections to run at specified intervals:

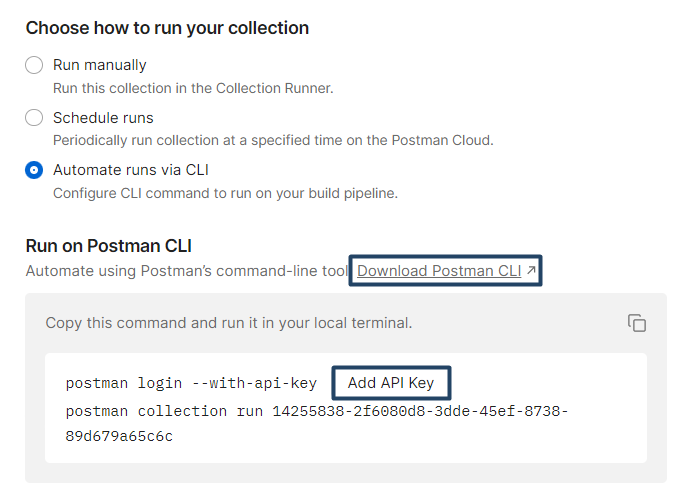
* You can set up automation by choosing 'Schedule runs' in the collection options.
* This allows you to name your schedule, like 'Daily API Check', and pick the frequency, such as daily at noon.
* The automation executes on Postman's cloud, so your local machine's status won't affect it.
* To view or manage these runs, you'd navigate to the 'Runs' tab in the collection's settings.
* It's important that the scheduled runs are configured with the correct initial variable values since they depend on these for proper execution.

However, keep in mind that on the free plan, Postman limits you to 25 collection runs per month, so while setting this up, you should be cautious not to exhaust your allowed runs.

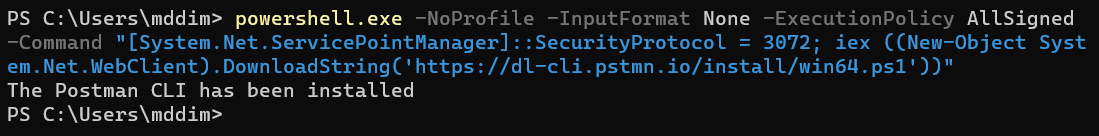
## Postman CLI

The Postman CLI is an integrated tool within Postman that enables the automation of collection runs through   
the command line.

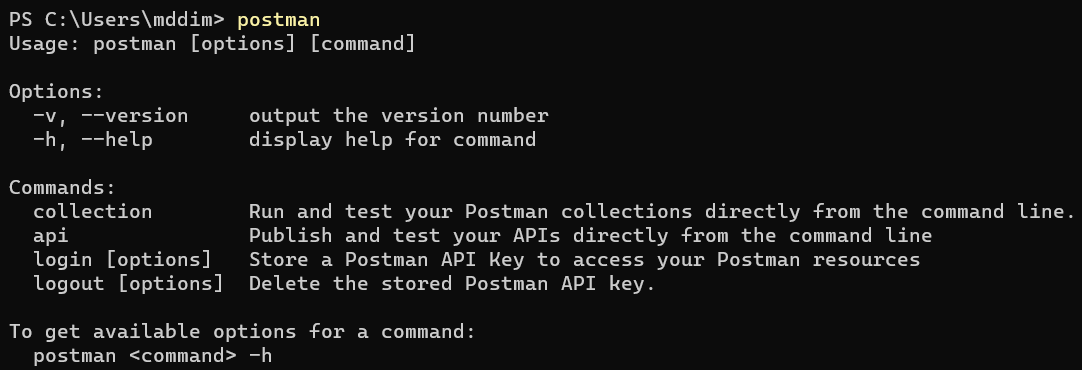
### **Installation and Basic Usage**



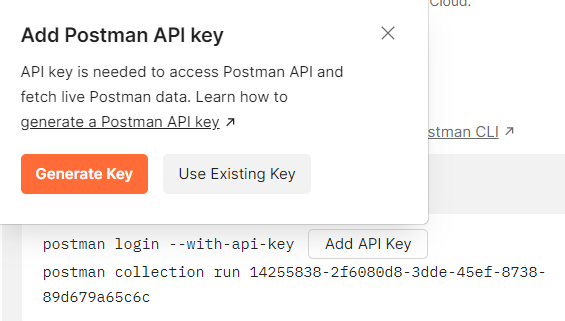
* It will open a new tab
* Choose your operation system
* Copy the command for your operating system in your terminal and this will install Postman CLI (this is Windows OS)



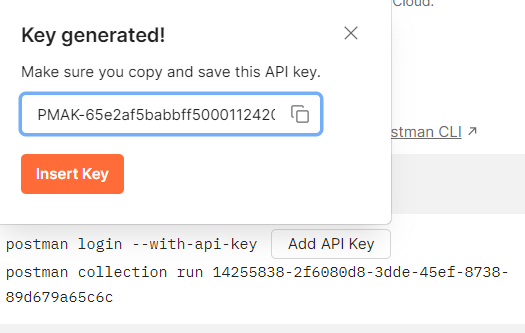
* After the installation, you can simply type **postman** in your terminal and it will give you some ideas of how we can use Postman



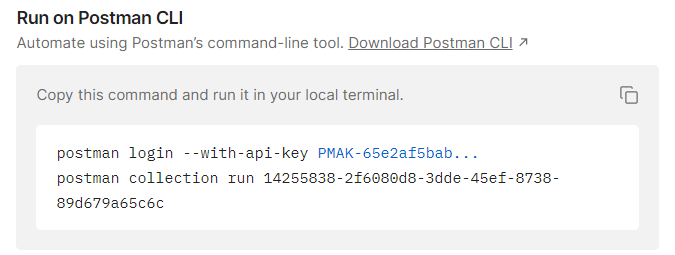
* After installing Postman CLI, you'll need an API key to access your Postman collections.
* So, go ahead and generate one



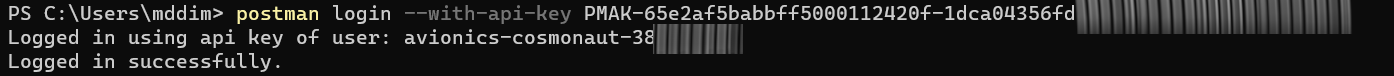
* Name your API key
* Copy and paste it somewhere safe. It is shown only once.



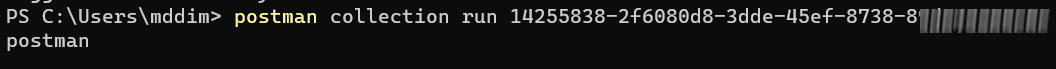
* When you click the insert button, it will add the API key to the command, so you need to copy the command and run it in your local terminal



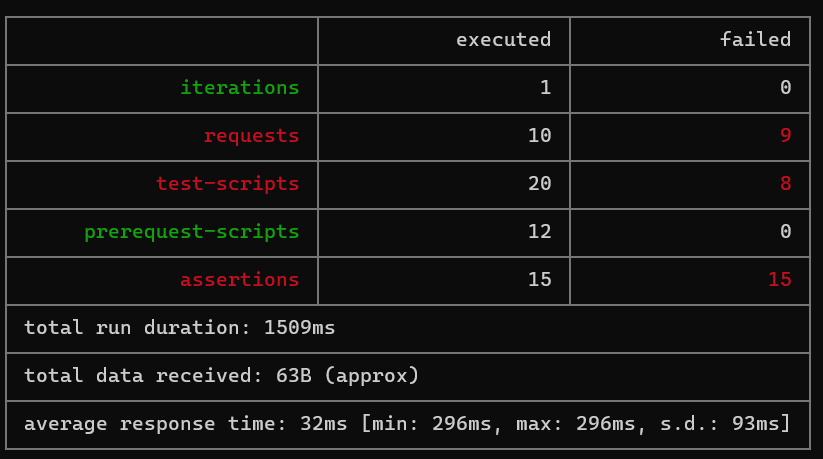
* The first thing that this command does, actually these are two commands, so the first one logs you in with the API key.



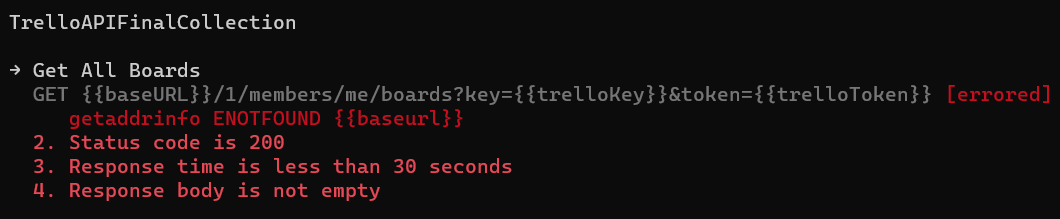
* And the second one runs the collection.



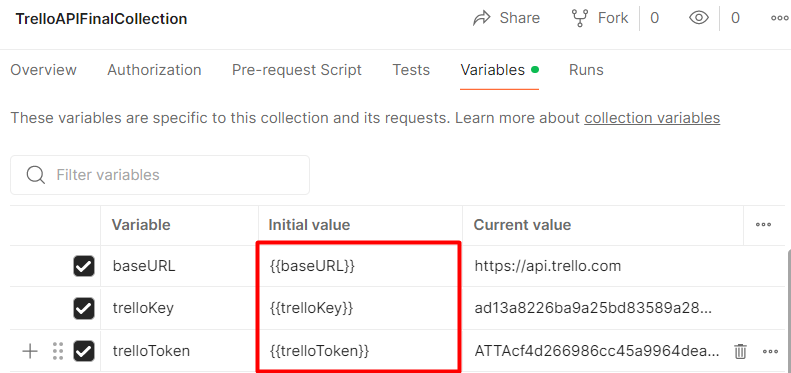
* But all of our tests are failing. Why is that?



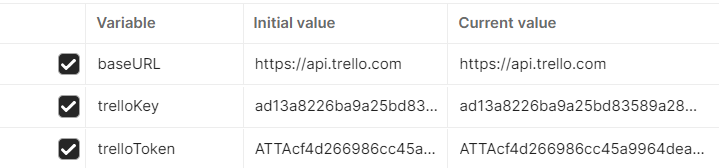
* We can see here on our first request that Postman doesn’t recognize what {{baseURL}} and doesn't have access to our trelloKey and trelloToken.



* In general, Postman can't resolve our variables, because their initial value is hidden.



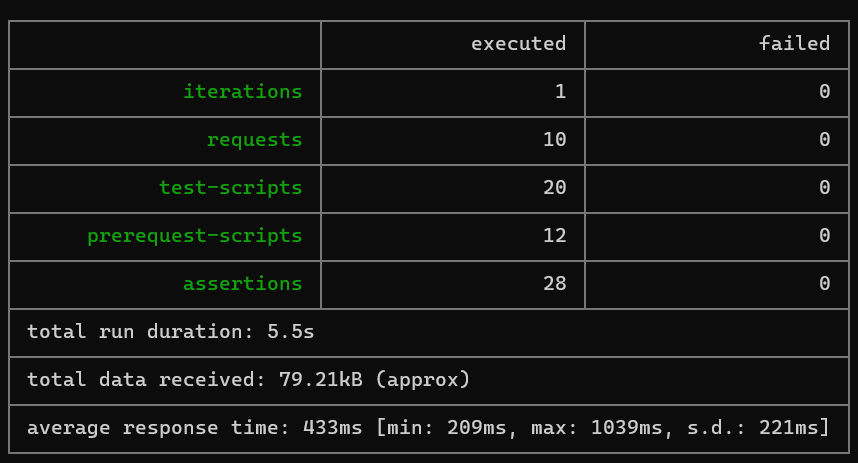
* So go ahead to your collection 🡪 Edit 🡪 Variables and "unhide" the initial value (copy and paste the current value to in the initial value)



* Don't forget to save the changes
* Go to the terminal and since you're already logged in, just run the second command

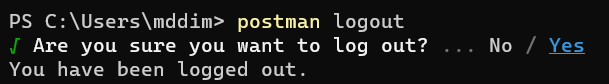


* And this time all our tests are passing. 😊

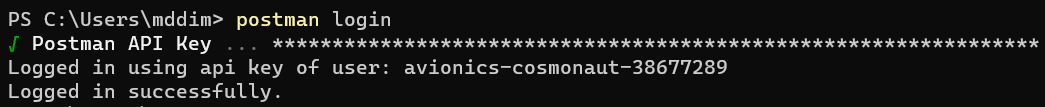


### **Useful Commands**

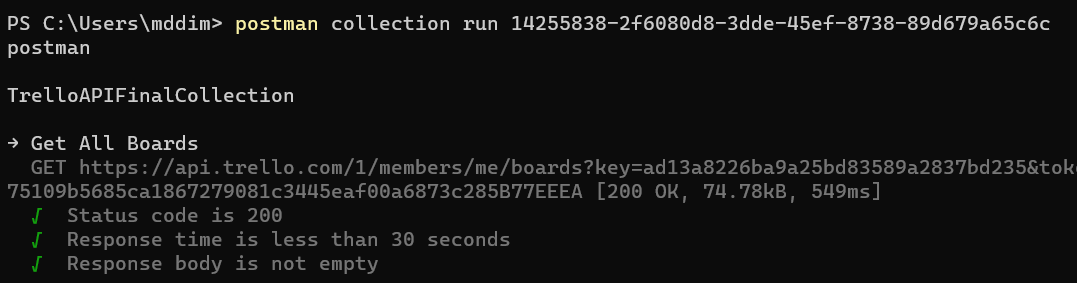
* **postman logout**



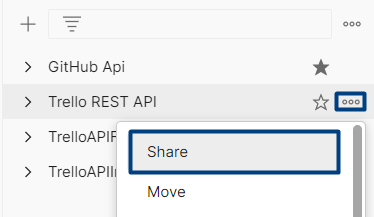
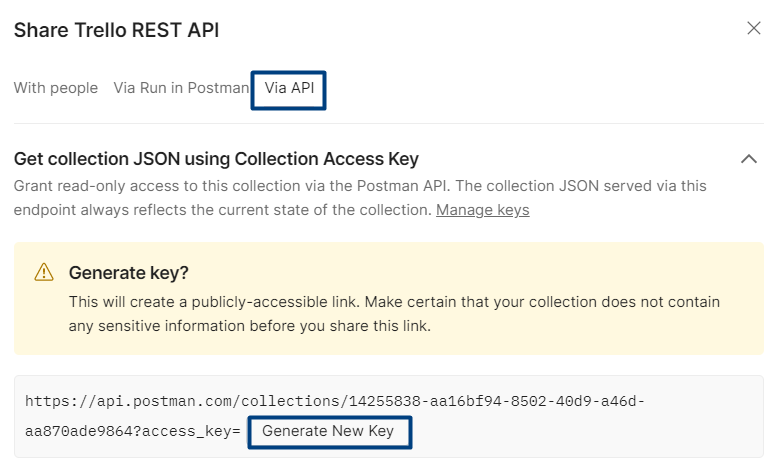
* **postman login**



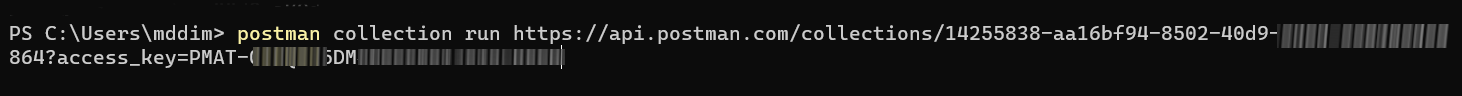
* **postman collection run {collection id}**



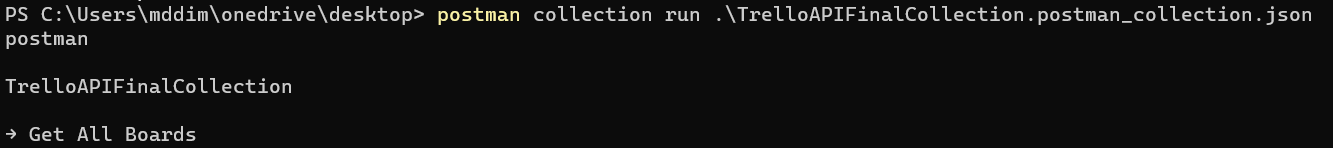
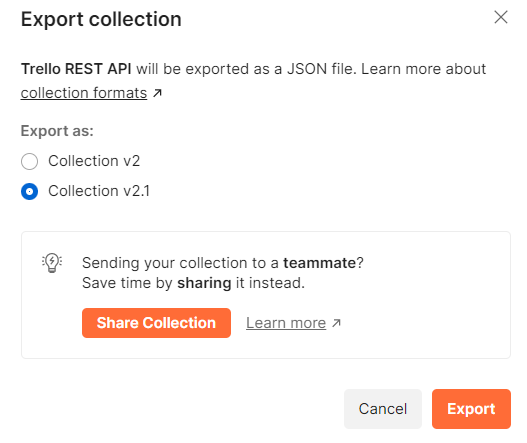
* **postman collection run {collection URL}**Here is how to get the URL of the collection?





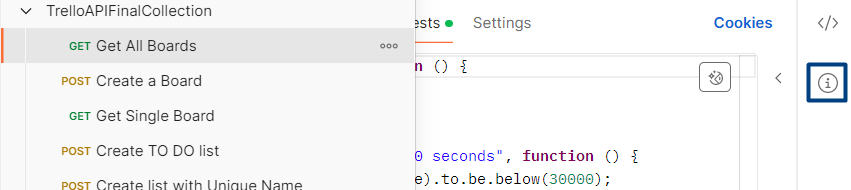
* **postman collection run {JSON file}**Export your collection and reference it locally. Navigate through the terminal to the directory where you saved the collection or specify the exact path to the collection.

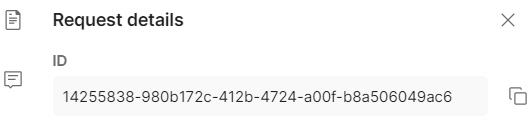


### **Additional Options**

Besides the main command 'postman collection run' that starts the execution of a collection, we can also add extra options to customize the run. These additional options are more advanced and you may not need them very often, but we need to point out that they exist and where to look into [**documentation**](https://learning.postman.com/docs/postman-cli/postman-cli-options/) if you need them.

* **--bail option -** Adding **--bail** option to the end of your postman collection run command, **will stop the collection run immediately if any test fails**. It works the same whether you're running the collection from a JSON file, a collection URL, or using a collection ID.
* **--delay-request [number] -** Specifies a **delay (in milliseconds) between requests**.
* **-i [requestUID] or [folderUID]** - **Runs only the specified folder UID or request UID** from the collection.

You can find the **UID of every collection, folder, request** by clicking on it and then on the **small "info" button** on the right side of Postman.  




## Newman

Postman CLI, lacks custom reporters and integrations with third party tools and services. This is where his older brother **Newman** comes to help. It is also created by Postman, but under an open-source model, which makes it easier to extend and create more advanced integrations.

### **Installation**

Newman runs inside the Node.js runtime environment. So, this is why Node.js needs to be installed locally on your computer. To check if you have Node.js installed run the following command on your terminal:

* **node -v** It will show you the current version installed on your computer.

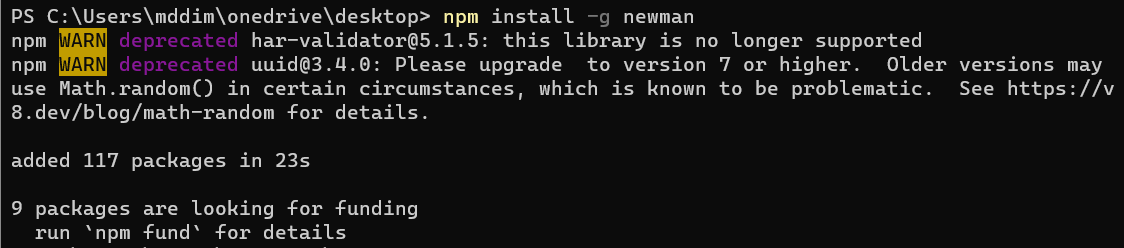
If you don't have Node.js installed, go to [**https://nodejs.org/en**](https://nodejs.org/en) and install the LTS version.

This is where you can find Newman documentation. Again, learn to check the documentation:

[**https://github.com/postmanlabs/newman#getting-started**](https://github.com/postmanlabs/newman#getting-started)

To install it globally, run:

* **npm install -g newman**



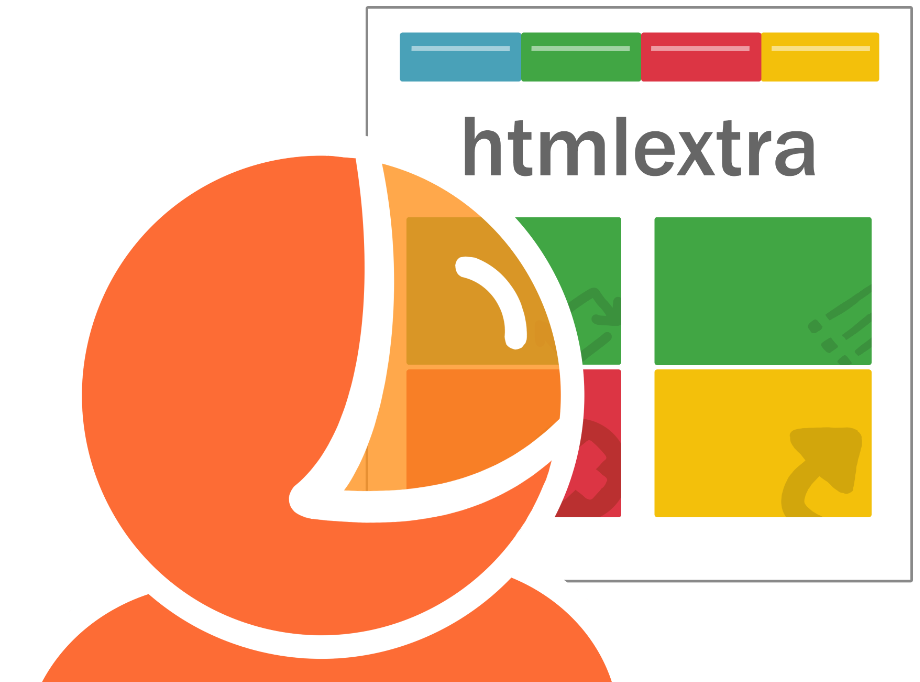
### **Running a Collection**

The command we need to write to run a Postman collection in Newman is:

* **newman run -** followed by a location where Newman can fetch that collection. This can be a local file or a remote URL.

You already have the JSON file with your collection, stored in your computer and you also have the URL of the collection, both from the exercises with Postman CLI. Go ahead and run the tests from the terminal.  
Pretty much it is the same as with Postman CLI, so what is so special about Newman? Newman has multiple reporters. Let's see what a reporter is.

### **Generating HTML reports**



#### **Install HTMLEXTRA**

The **htmlextra** Reporter is a plugin for Newman that generates comprehensive **HTML reports** for Postman collection runs. This tool is designed to provide an enhanced visualization of API test runs for better analysis and debugging​. You can find it here:

[**https://www.npmjs.com/package/newman-reporter-htmlextra**](https://www.npmjs.com/package/newman-reporter-htmlextra)

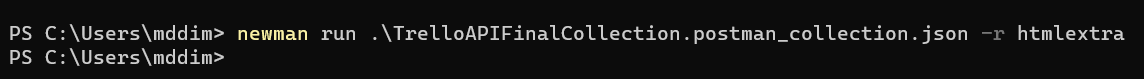
In order to install it you should run the following command:

* **npm install -g newman-reporter-htmlextra**

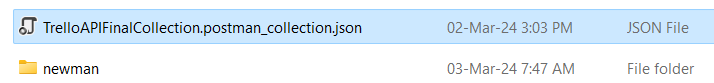
#### **Using Newman with HTMLEXTRA**

In order to use this reporter, specify htmlextra in Newman's -r or --reporters option. The following command will create a new report in the ./newman directory, if the directory does not exist, it will be created as part of the Newman run.

* **newman run {collection.json} -r htmlextra**



For a few seconds, nothing happens and you might think that this doesn't work, but if you navigate to where your collection is stored in your computer you will find a folder called newman was created.



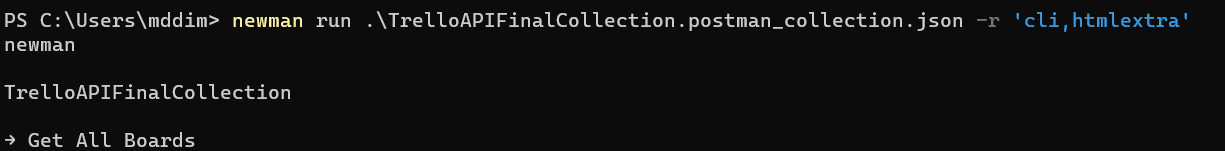
And in that folder, you'll have the HTML report. Go ahead and open it. It will open in your browser and it has pretty awesome features. Go ahead and take a look. Explore it. Click on Summary, Total Requests, Failed Tests, Skipped Tests. It contains a lot:

* A summary dashboard with the overview of the run
* Tabs for detailed request and response data
* Visual indicators for passed and failed tests
* Buttons or links for navigation within the report
* Options to view logs, headers, body data, and variables used during the run



But what happened with the CLI report? Whenever we are writing **-r** in Newman, we are specifying a reporter and only that reporter will be used. You can specify as many reporters as you like. So, in order to see the CLI report on the console and create an HTML report via htmlextra, we can specify that, like this:

* **newman run {collection.json} -r 'cli,htmlextra'**



**htmlextra** is highly customizable, through command-line options, so we suggest to explore its documentation.

[**https://www.npmjs.com/package/newman-reporter-htmlextra**](https://www.npmjs.com/package/newman-reporter-htmlextra)

### **Generating JUnit Reports**

JUnit is a standard report format that is widely supported by CI/CD servers.

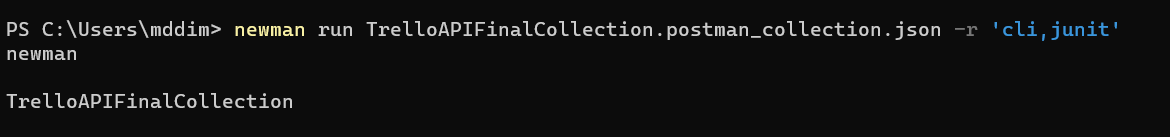
The JUnit report is stored in an XML format and is intended for programmatically reporting the test results.

So, it is essentially intended for the consumption of a machine, not for us humans.

The JUnit Reporter is already built in with Newman, so we don't need to install it separately and all we have to do

is specify that we want to generate the JUnit report.

* Instead of HTML extra, we are going to specify JUnit.



We're gonna see an XML file that was generated inside the Newman directory.

And we can try opening this file, but this is simply XML, it's not intended for us, but it's super useful for CI/CD.

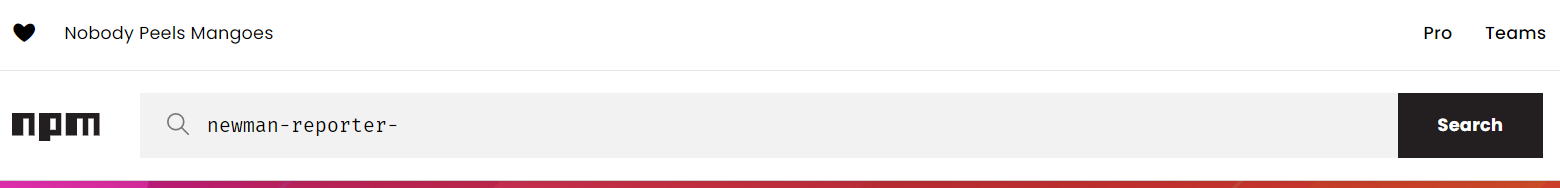
* If you want to specify the path, where you want to save the reporter, you have to use this additional option:  
  **--reporter-junit-export <path>**



* The same way we can generate **json report**. Try that by yourself. Instead of JUnit use json reporter.

### **Other Newman Reporters**

So, these are some of the reporters that Newman has, but we can find many more on [**https://www.npmjs.com/**](https://www.npmjs.com/)



It is important to start your search with **newman-reporter-** and it willshow youa list of results for Newman reporters. It is advisable to sort the results by popularity. Go ahead and try one. 😊

### **Advanced Configuration Options**

These additional options are a bit advanced and you will not need them very often. The intention here is to point out that they exist and where to look in the documentation if you need them.

* Specify the extent of delay between requests (milliseconds).  
  --delay-request
* Specify whether or not to stop a collection run on encountering the first test script error.  
  --bail [optional modifiers]

Can optionally accept modifiers, currently include folder and failure. folder allows you to skip the entire collection run in case an invalid folder was specified using the --folder option or an error was encountered in general. On the failure of a test, failure would gracefully stop a collection run after completing the current test script.

* Show detailed information of collection run and each request sent.  
  --verbose
* Run requests within a particular folder/folders or specific requests in a collection. Multiple folders or requests can be specified by using --folder multiple times, like so: --folder f1 --folder f2 --folder r1 --folder r2.  
  --folder <name>

\* And for a good night, try to run the collection with 'emojitrain' reporter:

