ATLANTIC TECHNOLOGICAL UNIVERSITY

ASSIGNMENT COVER SHEET

To Be Completed by The Student

Lecturer’s Name: Ms. Maria Griffin

Assessment Title: Test Frameworks

Submission Date: 28-Nov-2022

Student’s Name: Tony Mathew Thomas Id. Number: L00171045

Course / Stage Master’s in DevOps

Subject/Module: DevOps Software Engineering (2022/23)

Word Limit: Actual Word Count: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I confirm that the work submitted has been produced solely through my own efforts.

Student’s signature: Tony Mathew Thomas Date: 28-Nov-2022

|  |
| --- |
| **Note**: **PENALTIES**   * The total marks available for an assessment will be reduced by 15% for work submitted up to one week late. The total marks available are reduced by 30% for work up to two weeks late. * Assessment work received more than two weeks late, without prior approval by the lecturer will receive a mark of zero. * Marks awarded will be reduced by 10 % if submitted work is greater than 10% above or below the assigned word limit. * A further hard or electronic copy of your submitted work may be requested, and therefore you must keep a copy on disc. * Incidents of alleged plagiarism and cheating are dealt with in accordance with the Institute’s Assessment Regulations   **Plagiarism:** Presenting the ideas, words of someone else without proper acknowledgement. Refer to the Institutes’ procedures and guidelines for the assessment of learners. |

# Introduction

The objective of this lab is to learn about the benefits of using a test framework by focusing on one of the popular and widely used tools to get an understanding of how using one will make testing easier by enabling automation. There is also a comparison with another tool to find out the strengths and limitations of each.

# Aims/Objectives

* Use Postman to find out the possibilities of automation testing features in it
* Try out the advanced features of it rather than the basic functionalities like API testing environments and variables.
* Restrict usage of tools or framework to free and open-source ones rather than enterprise or licensed ones.
* Try out the features on the API endpoints of a locally running project rather than one available publicly.
* Compare and contrast features available in another tool used for automation testing

# Methods

1. Use the desktop version of Postman which can be installed in a local machine.
2. Set up a simple Web API project using .Net Core with two endpoints for testing. First one to return a collection of objects and second one to return one item in the collection.
3. Create a collection in postman with two requests one for each of the endpoints created in the sample project.
4. Create examples for each of request by selecting the “Add Example” option given by postman upon right-clicking a request.
5. Create a Mock Server pointing to the newly created Collection. This will give us a mock server URL for testing with the output from the examples created in the collection.
6. Create a new collection for testing the requests with the base URLs of each replaced with the URL given by the Mock Server.
7. Use this same collection for creating a Monitor in Postman. Schedule the Monitor to run every hour. Set the notification email Id to receive failure notification.
8. Try changing the base URL of one request to a wrong one to see whether the notification emails are landing in the inbox.
9. Create a flow using the existing collection created for monitoring. Use blocks such as Send Request and Create Durables to create a chained flow to fetch a value from the output of first request and use it as input in the second request.

# Results & Analysis

* Creation of a Mock Server is very easy using an existing collection. Postman seems to have created a mock domain in their server for us to test based on the collection we pointed. Figure 1 to 4 shows a collection created with two requests and examples created for the requests. Figure 5 shows a Mock Server created with Copy URL option
* It’s possible to modify the output of the examples which is what the Mock Server will return on requests to the Mock Server.
* Set up a collection with the Mock Server URLs and this is used to create a Monitor. Changing the URL to a wrong one was deliberately done to check the behaviour during a failure. The failure email notifications are received instantaneously.
* The lowest frequency of Monitoring is 1 hour in the free plan. While there are plans as frequent by a minute under paid plans of Postman.
* Monitoring of the URLs in the collection happens in the background while postman is running without any user intervention. The result of monitoring is presented in a graphical format. Figure 8 shows the result for the next 3 hours including two initial tests that was done manually.
* There basically two types of monitors in Postman:

1. Uptime Monitors: This is meant to monitor a website in general rather than a collection of endpoint URLs. The frequency available here are the same as that of collection monitors. And has the capability of notifying team members.
2. Collection Monitors: This is meant to monitor a collection of URLs and needs a collection to be setup before. All the URLs in the collection are probed during each interval of monitoring.

* There are options to stop and resume a monitor as needed. Also, the options to set timeout time limit and the delay time to wait between requests in a single cycle (Postman Learning Center, n.d.).
* There is an option to share the result with others (Postman Learning Center, n.d.).
* There is option to integrate with slack so that notifications can be sent slack channels which will be helpful for support teams who need to act up on any failure or incidents.
* Monitors also supports execution of scripts as this is available as a feature along with collections. Collections have the ability to execute Pre-request script that allows us to run a JavaScript code either before the beginning of the collection or before sending a request in the collection. This allows us to share data between request as well as share a global data at the collection’s scope.
* Creating a flow is simple in postman. It helps to automate monotonous tasks that involves several API calls. Flow has the components to make API calls, create variables, log data, and execute scripts
* A sample flow was created on top of the mock collection for the demonstration:
  1. The output received from the first POST request is received as a list. The API endpoint returns a list of hotels and its properties including a unique Id.
  2. The Id property of the first hotel in the list is used in the remaining of the flow. A durable variable is created for this purpose. A durable variable persists through out the life of a flow
  3. The durable variable is used in the next component that sends a GET request with the value in the durable variable as its query string parameter.
  4. The result of the GET request is the details of the hotel against which the Id is passed.
  5. The result of the GET request is printed onto the console using the Log / Terminal component which can be viewed in the Postman’s console window upon execution of the flow. Figure 9 to 12 shows the screen shots captured while setting up and running of the sample flow created.

# Conclusion

All the tools used were under free or community ones which did not incur any payment. This assignment proves that Postman is not a simple tool used merely to test API endpoints. Its capability for automating is immense. The Mock Server feature gives us the benefit of serving realistic data which gives the advantage of having mock API endpoints that could be created for the testers or UI developers to develop their scripts without having to wait for the backend developers to finish them. This will enable a team to parallelly work on same tasks that are dependent. The Monitors setup monitors a collection of endpoints rather than a single URL for a site. The ability to schedule a Monitor takes out the need for manual intervention for monitoring a collection of endpoints. The result provided are at the level of each endpoint. The graphical view provided by the Monitor’s result do support a level of slicing and dicing which again comes in handy for support teams.

The flow feature is really meant to be used for automation by testers. A flow can be a series of steps with a lot of API requests, variables and scripts which once created could be repeated any number of times. This is very helpful when we require to develop a reusable script for integration testing. A flow could be run after each deployment to test the sanity. The components called blocks available for creating a flow though does not look comprehensive, is sufficient for API testing and is easy to understand and simple and straightforward to configure. A flow of medium complexity with approximately ten API requests could be created in less than half an hour.

The Flow feature of postman closely resembles Selenium which is a tool specifically written for automation of testing. Although, the scope for Selenium is much wider since it supports a variety of applications to test. Both provides a canvas in which we could drag and drop blocks/components to configure a workflow. The collection of components available in Selenium are enormous compared to the limited number of blocks in Flow. The most striking difference between Postman Flow and Selenium is that Postman Flow’s capability is limited to testing API endpoints, whereas Selenium can be developed to test the UI of an application including desktop application and browser web pages. The screen grab feature of Selenium is very powerful in closely mimicking user interactions through mouse and keyboard. Another difference is the different languages supported in Selenium. Besides JavaScript, C#, Python, Ruby, and Java are some of the languages made available opposed to the single language JavaScript available in Postman for scripting. The scope of coding too is wide; different stages in a workflow can have scripts to change the behaviour of the component making Selenium a more powerful tool with lot of flexibility compared to just the Pre-request scripting in Postman. Another feature missing in Postman and available in Selenium is the record and playback feature which allows us to record a series of user interactions that includes mouse clicks and key strokes and save it for repeating it in future as an automated testing script.

Postman was initially created and still is meant to be a tool primarily for developers silently building on its automation testing features. Meanwhile, Selenium was created for testers and gained popularity as a powerful framework for integration testing, although it has to ability to do performance testing and load testing. The scripting capability along with toolset and screen record and playback feature makes it standout from many other tools available in the market. All these existing features of Selenium comes with a high cost of licensing from an enterprise point of view. Whereas Postman is free unless we want to have a lesser interval than 1 hour in Monitors.

From my personal experience, both tools offer excellent support for testers to write automation scripts for repeated tests after each deployment of an application. Selenium is a fully fledged automation testing framework. It can mimic a series of human activities performed on an application’s interface. I still like the simplicity of Postman it should be the tool of choice when we have a list of endpoints to test and not bothered about the user interfaces which is typically the scenario for backend developers who are not bothered about the look and feel of the application.

# References

Postman Learning Center, n.d. *Monitoring API uptime.* [Online]   
Available at: https://learning.postman.com/docs/monitoring-your-api/uptime-monitors/  
[Accessed 26 11 2022].

Postman Learning Center, n.d. *Setting up mock servers.* [Online]   
Available at: https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/  
[Accessed 16 11 2022].

# Appendix

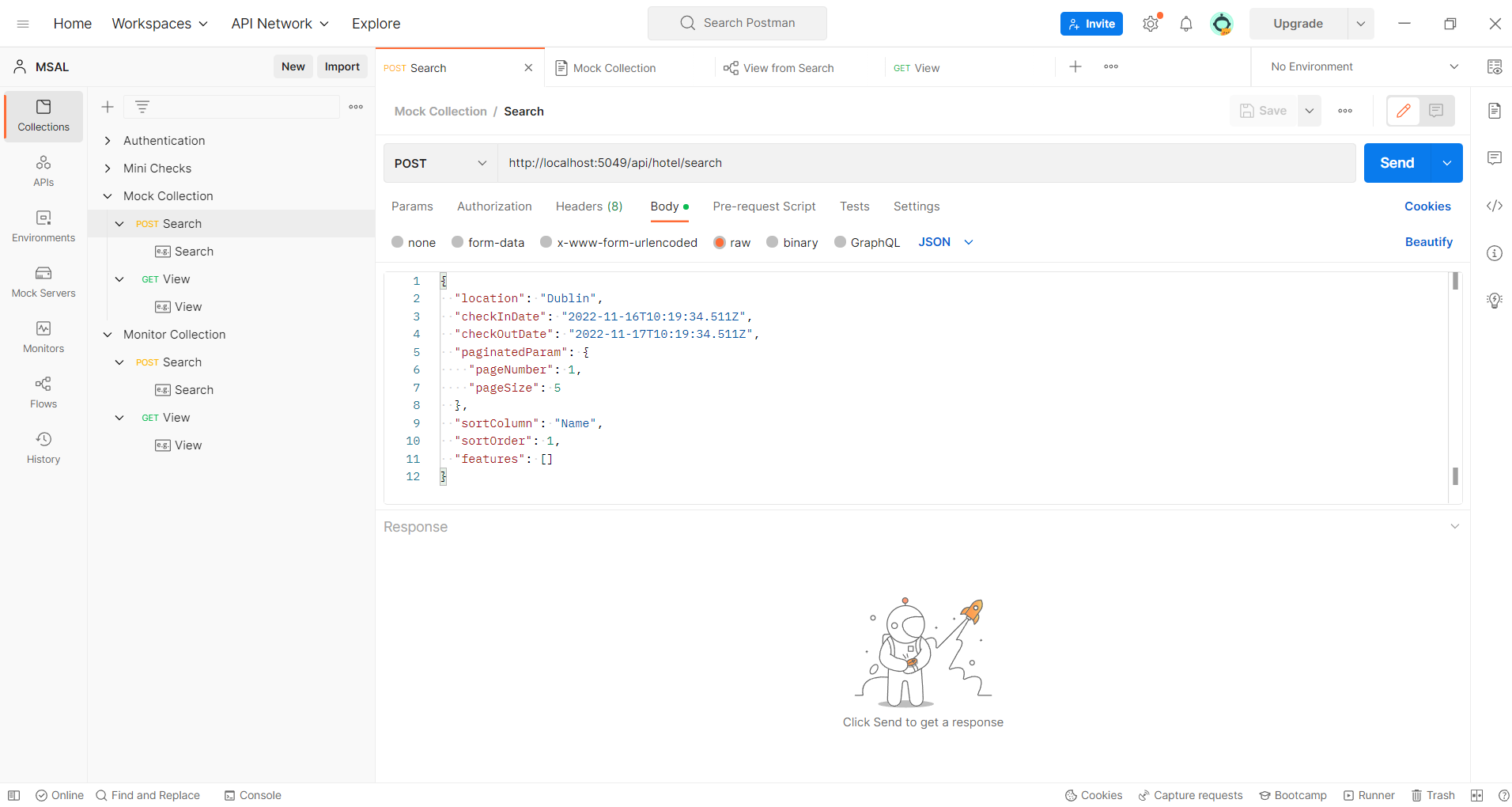


Figure 1. A POST request in Mock Collection

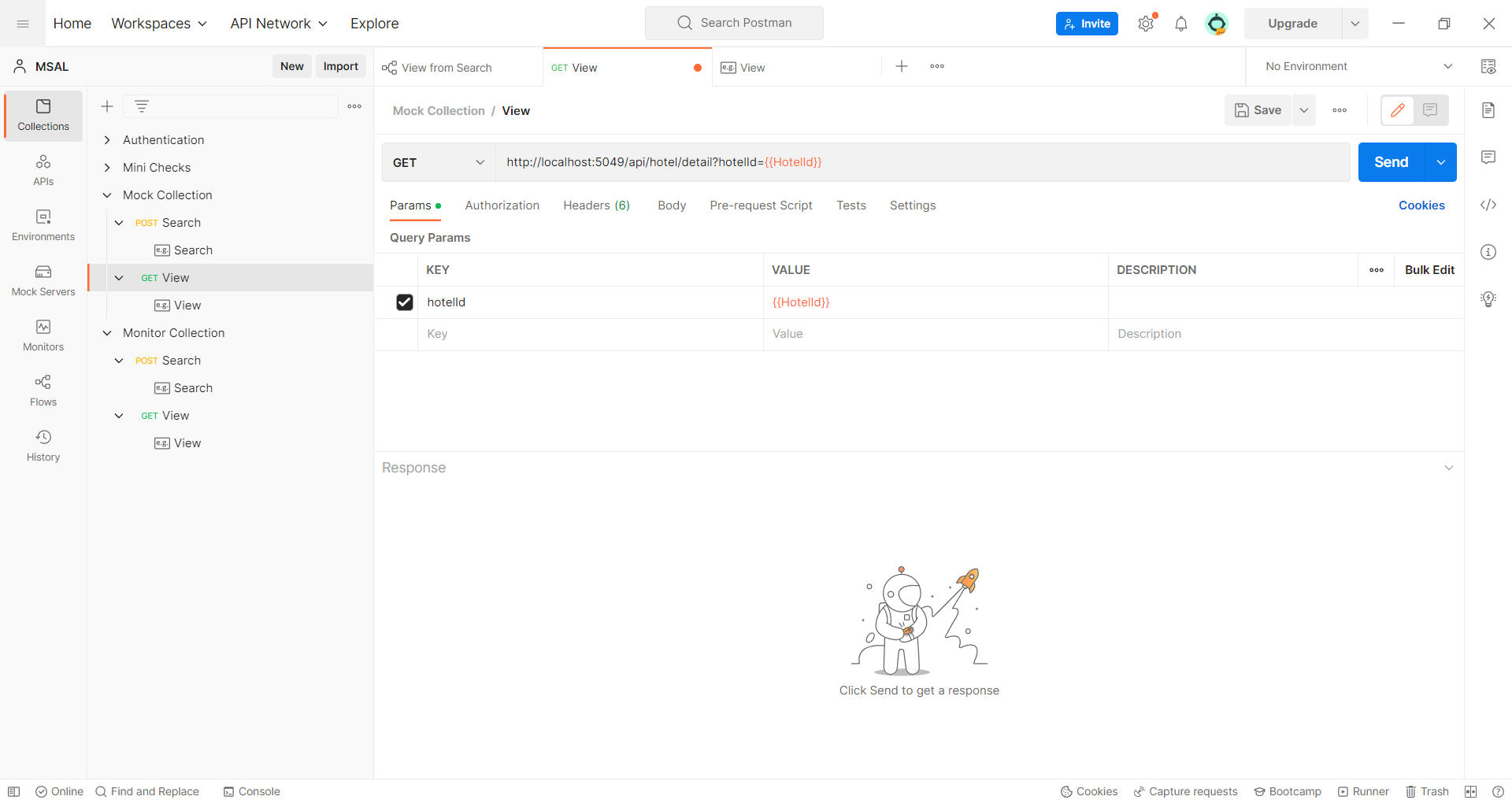


Figure 2. View GET request in Mock Collection

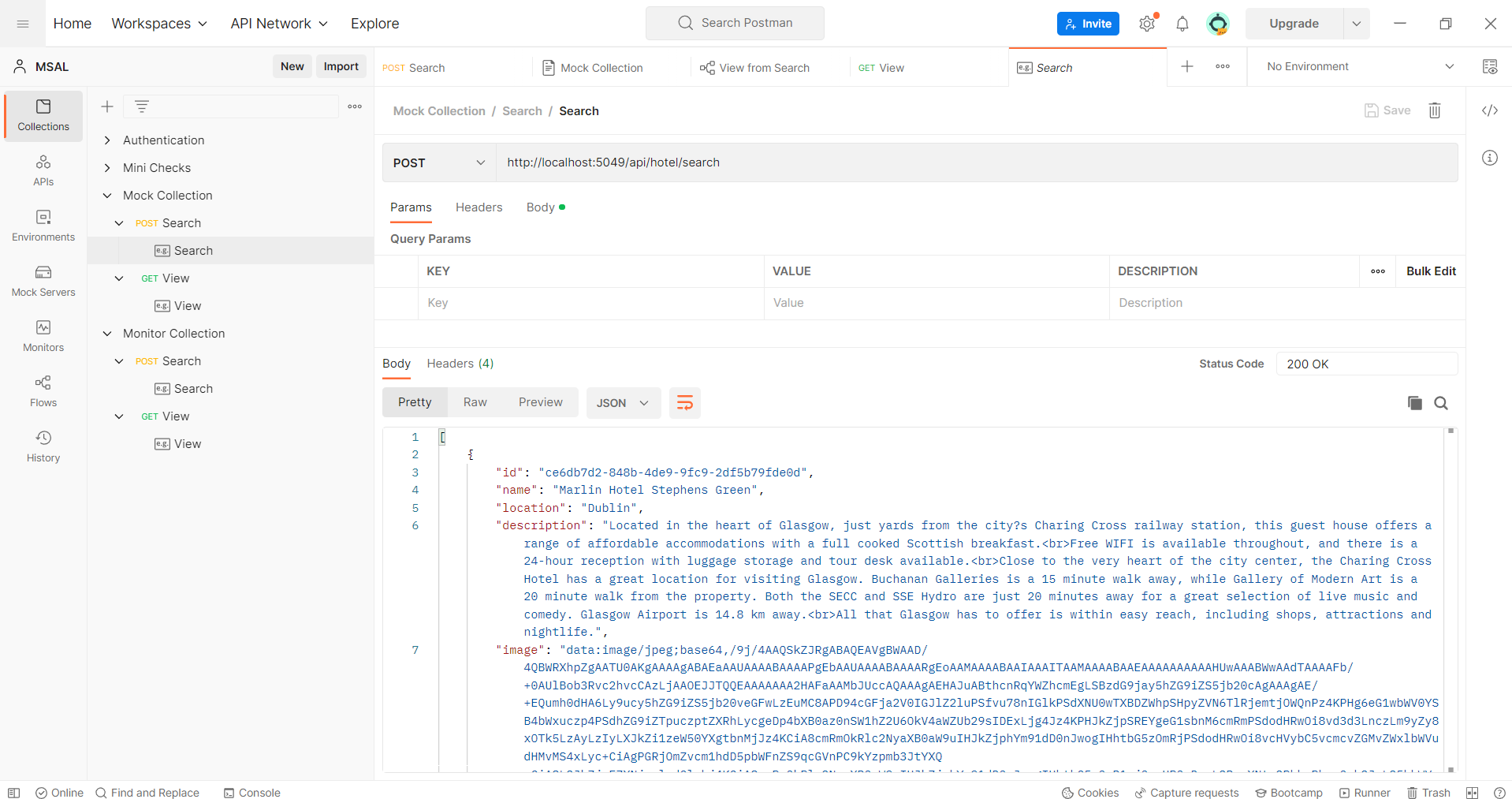


Figure 3. Example created for the Search POST request

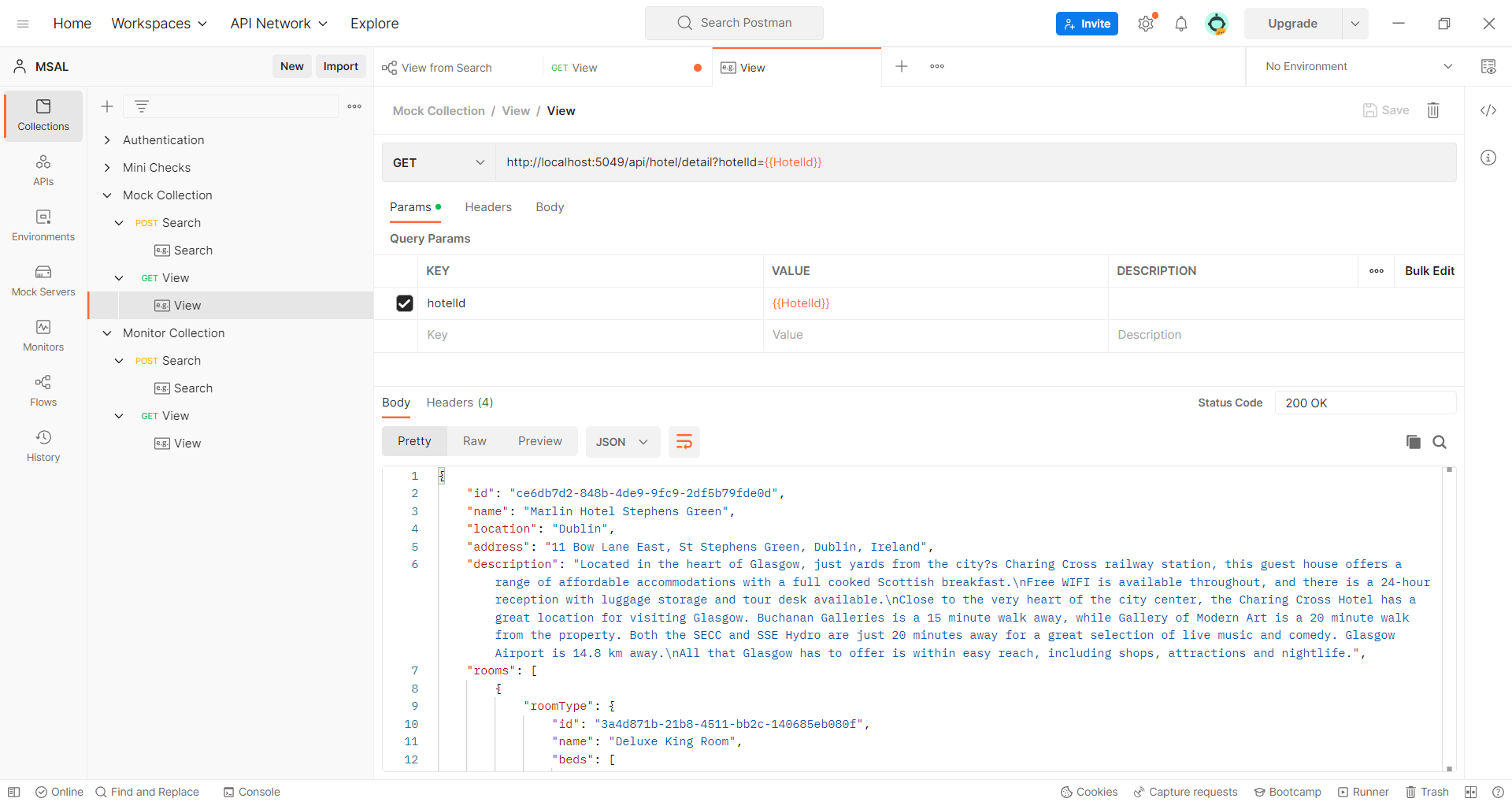


Figure 4. Example created for the View GET request

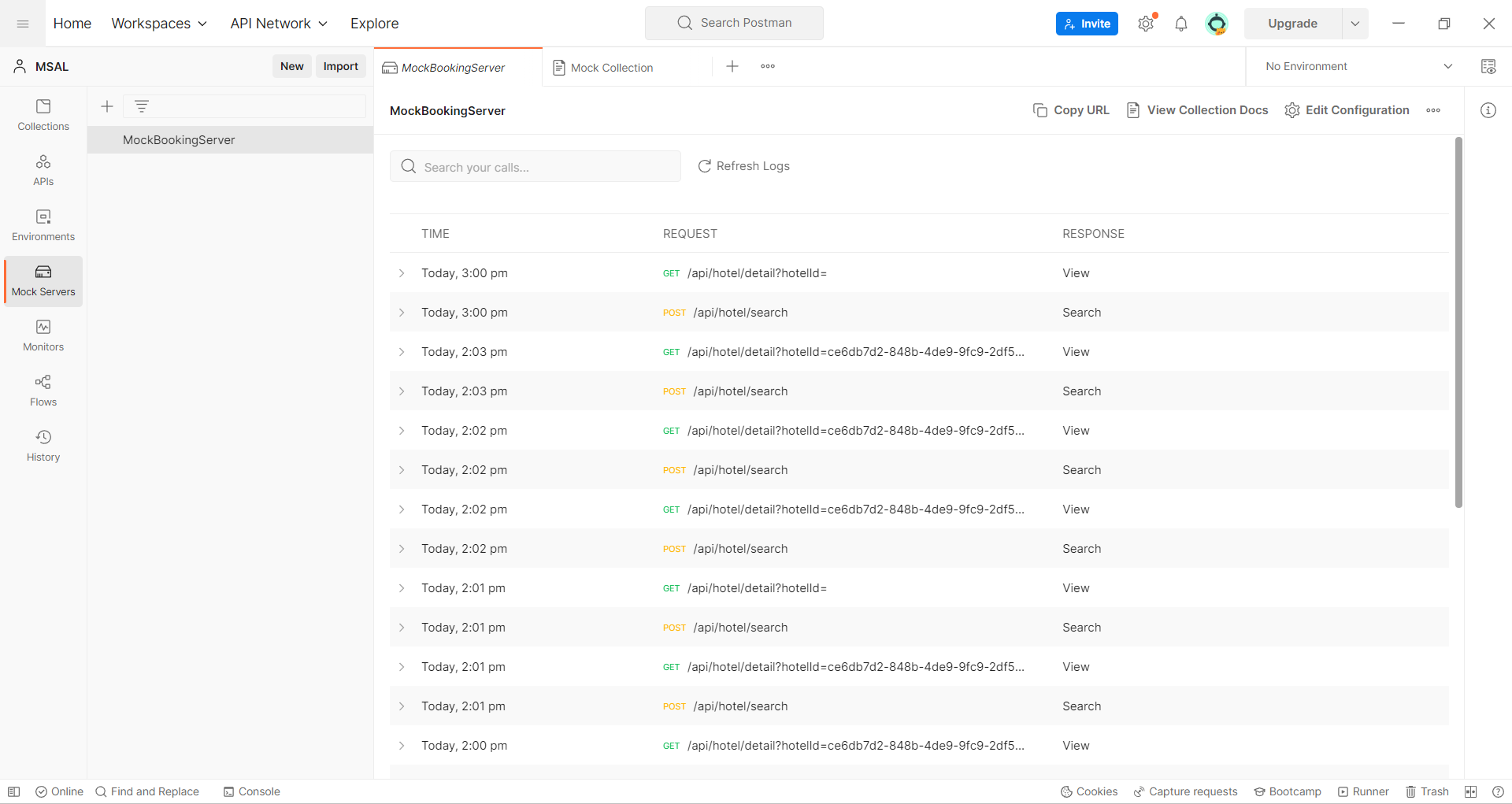


Figure 5. A Mock server with the Copy URL option

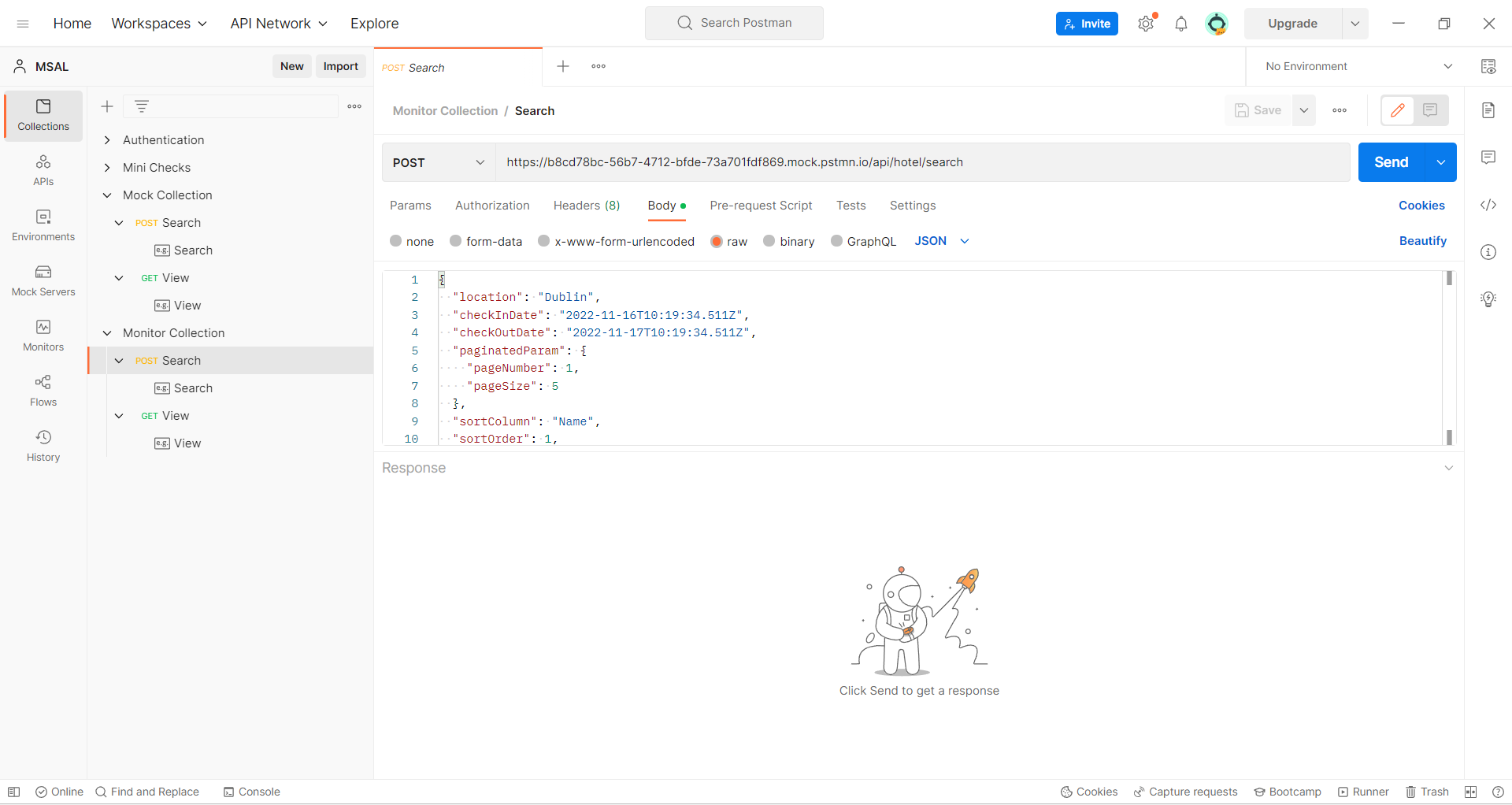


Figure 6. Monitor Collection with Search POST request with Mock Server as its base URL

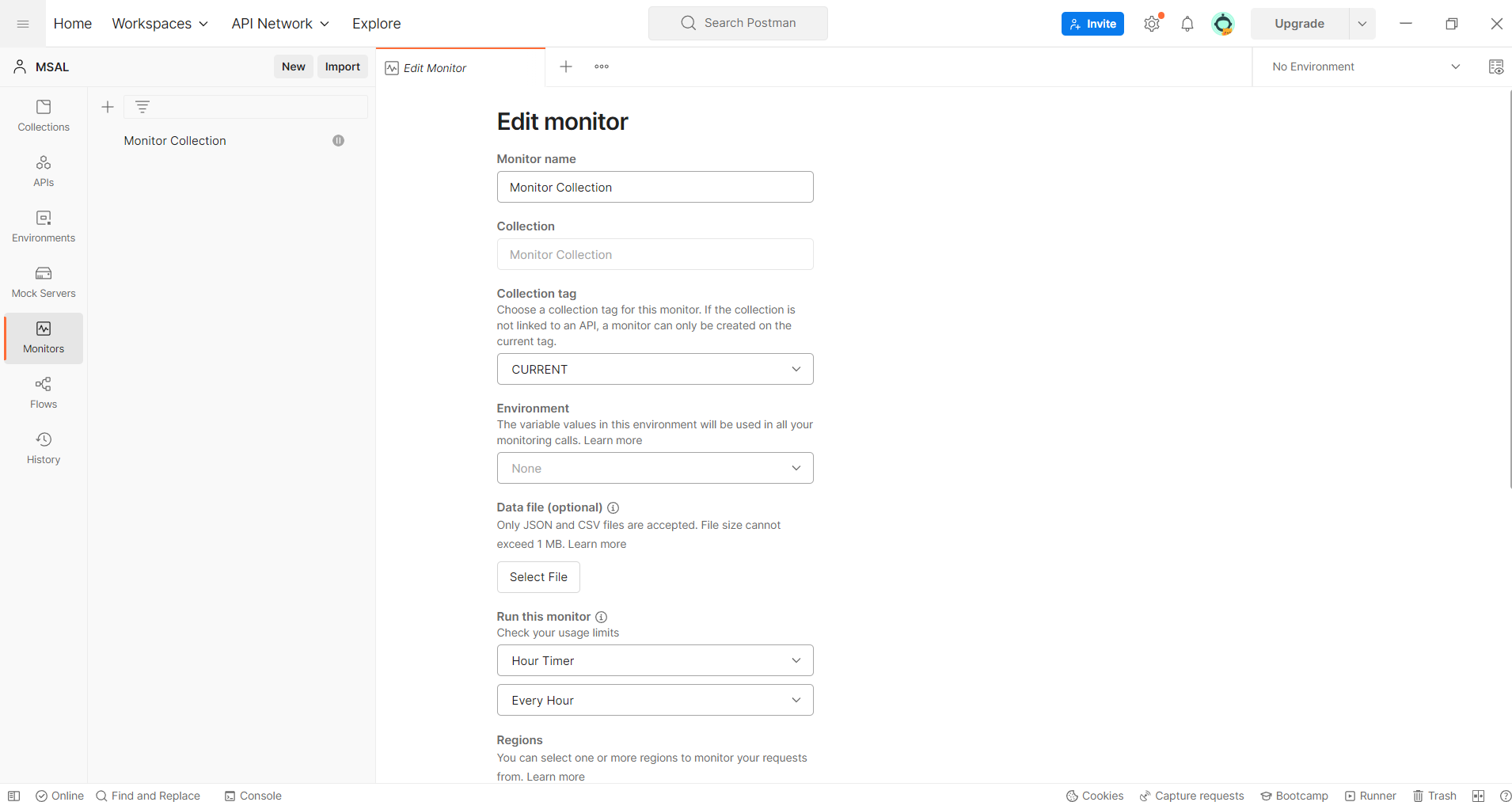


Figure 7a. Configure/Edit Monitor

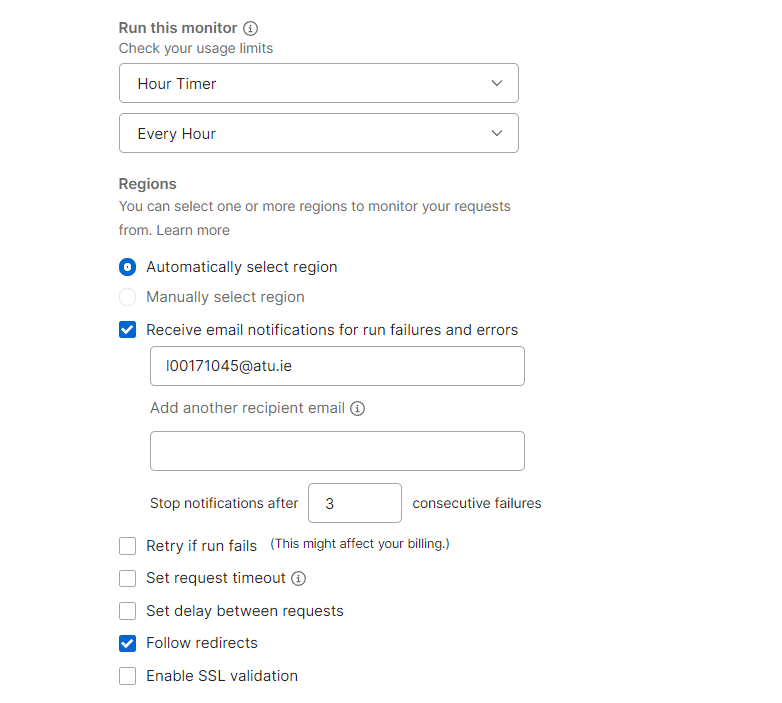


Figure 7b. Configure monitor window with email id to be notified and interval schedule.

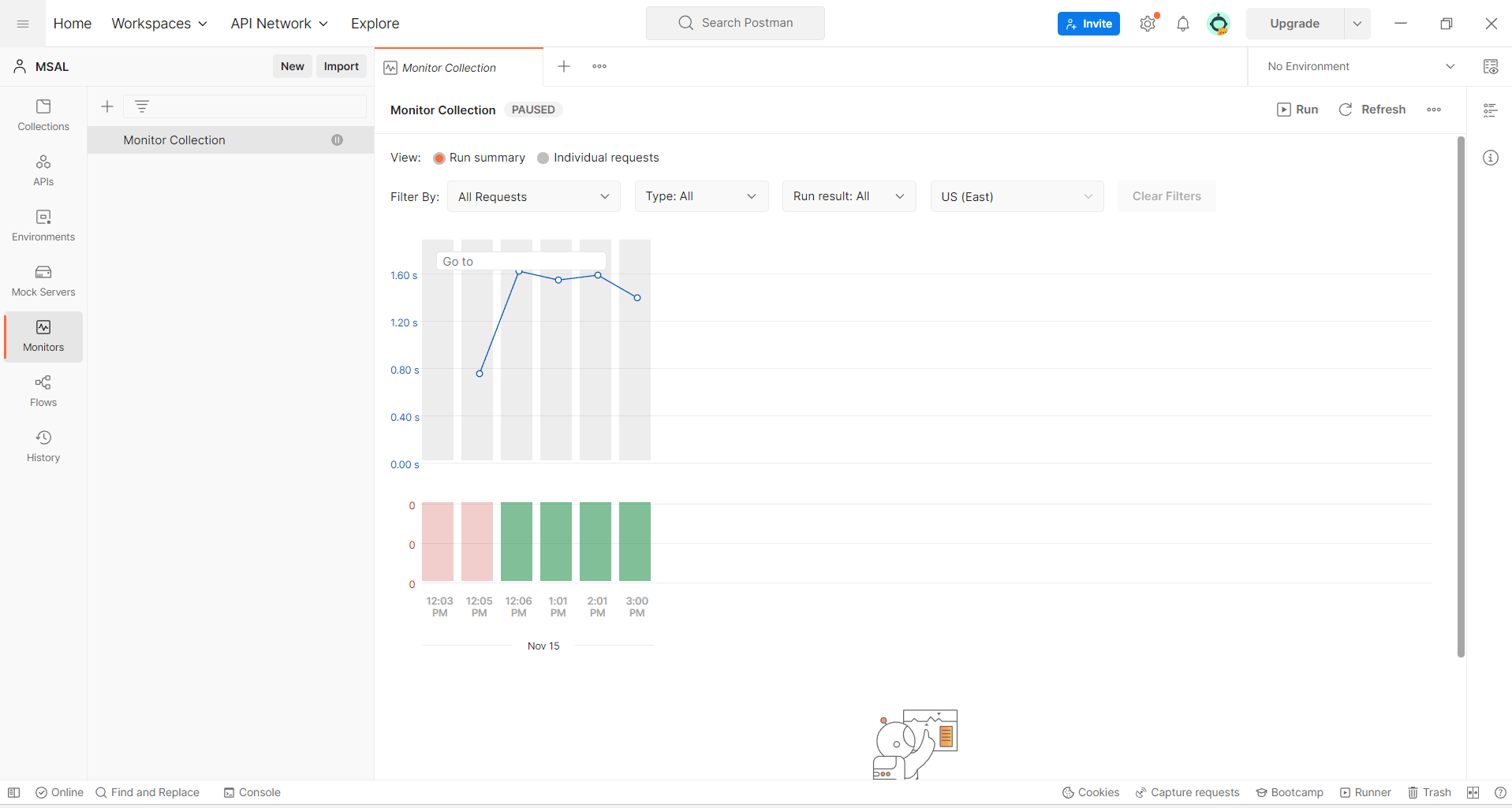


Figure 8. Hourly monitoring result

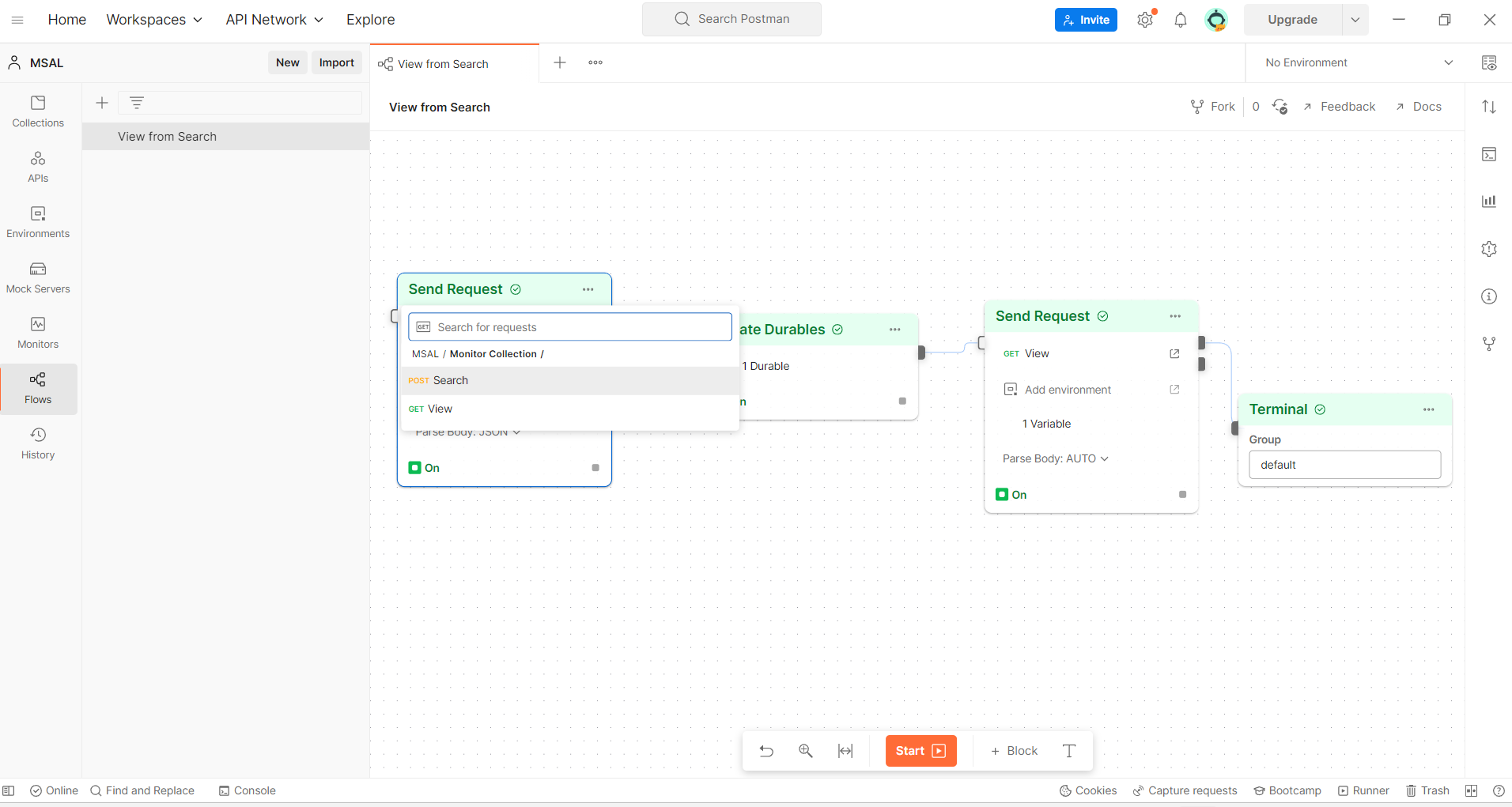


Figure 9. Send request block with Search POST URL selected in Flow

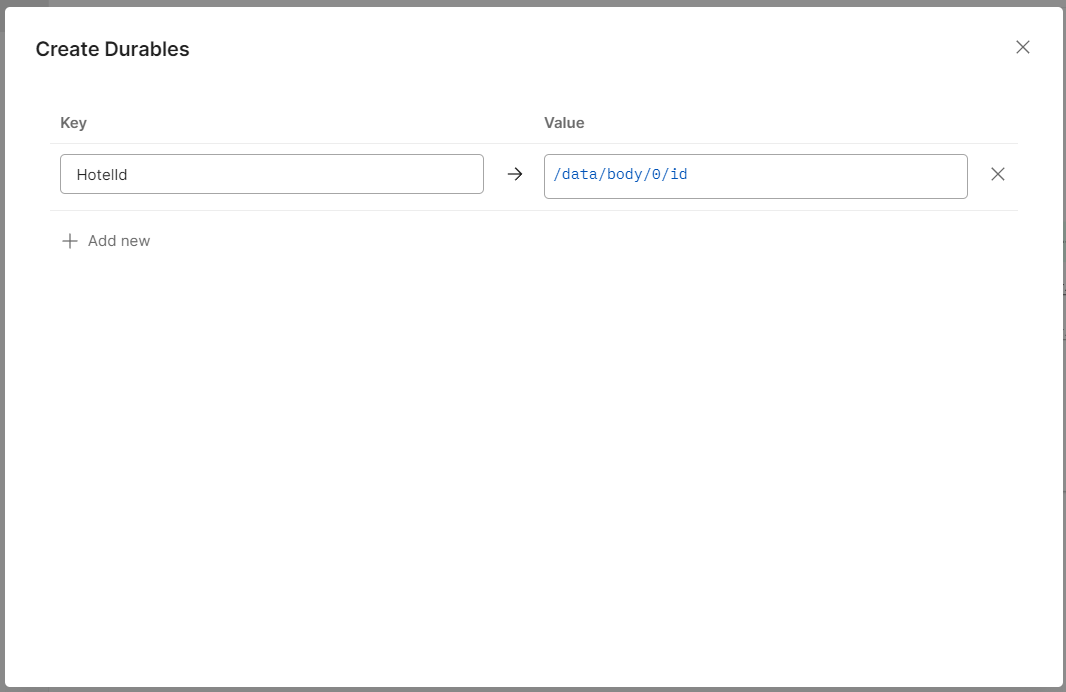


Figure 10. Create Durables block with a value set for the View GET URL query string parameter

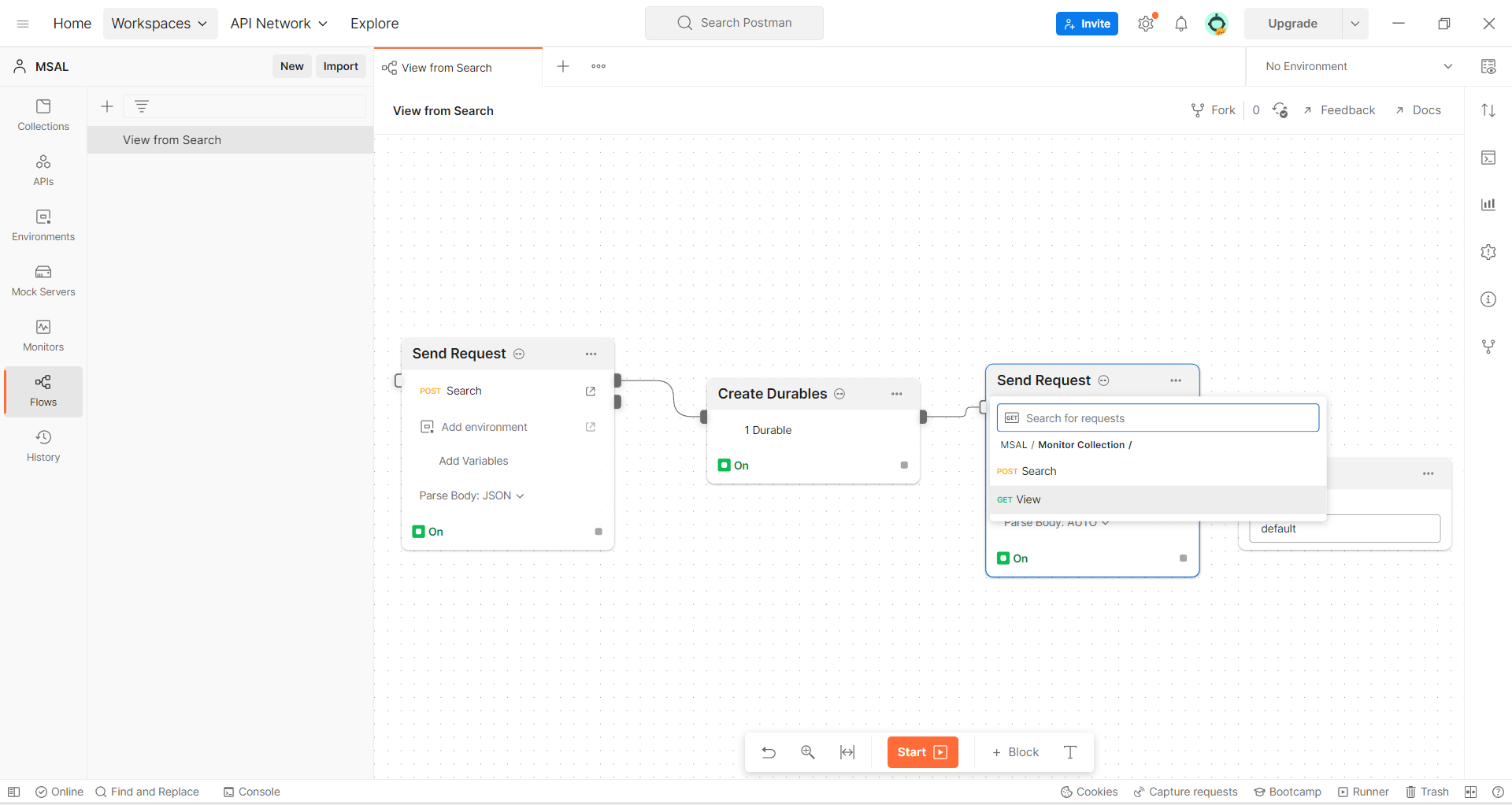


Figure 11. Second Send Request block for View GET URL

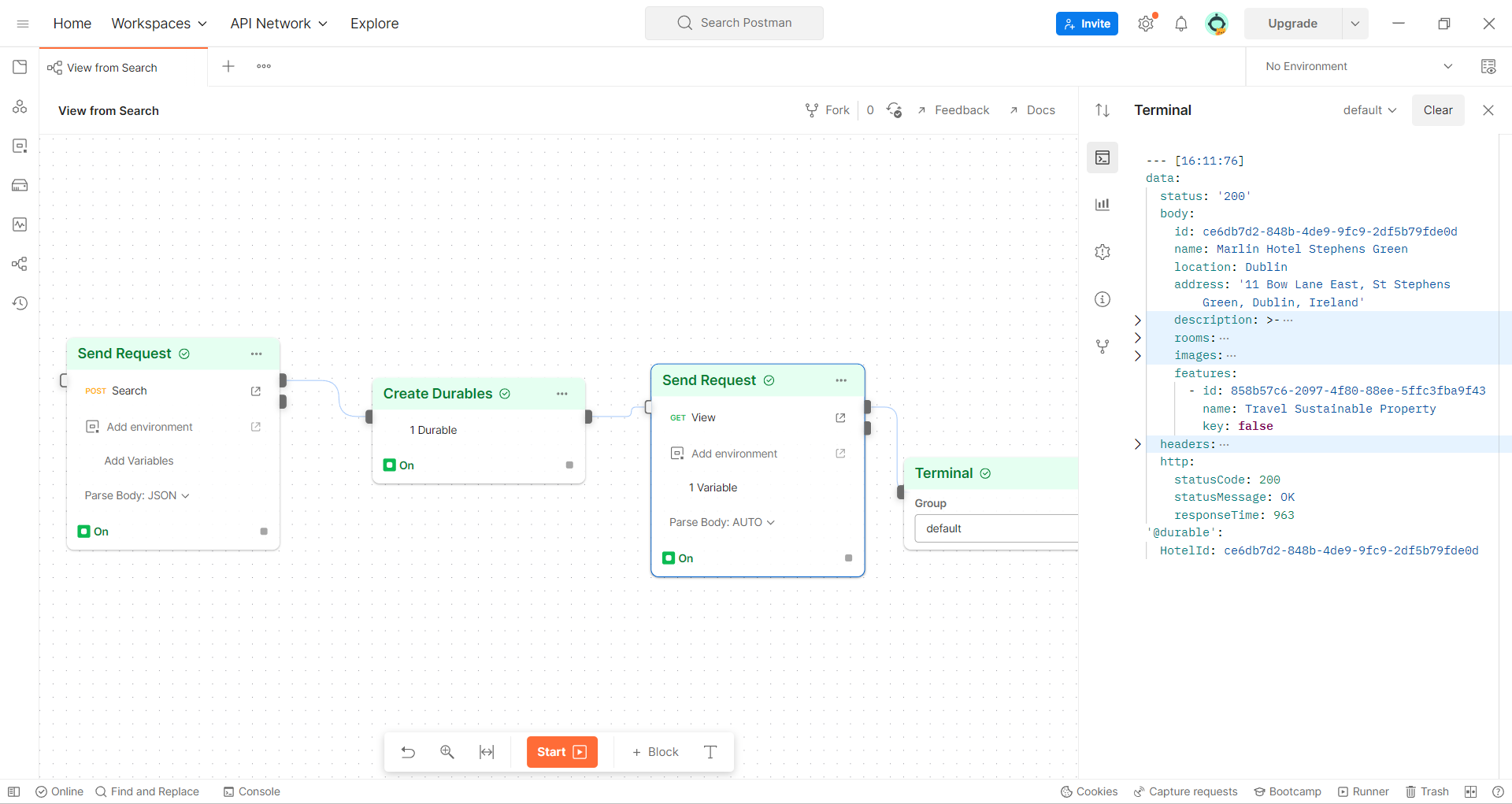


Figure 12. A successful run with Terminal output showing the result of View GET URL and Durable variable