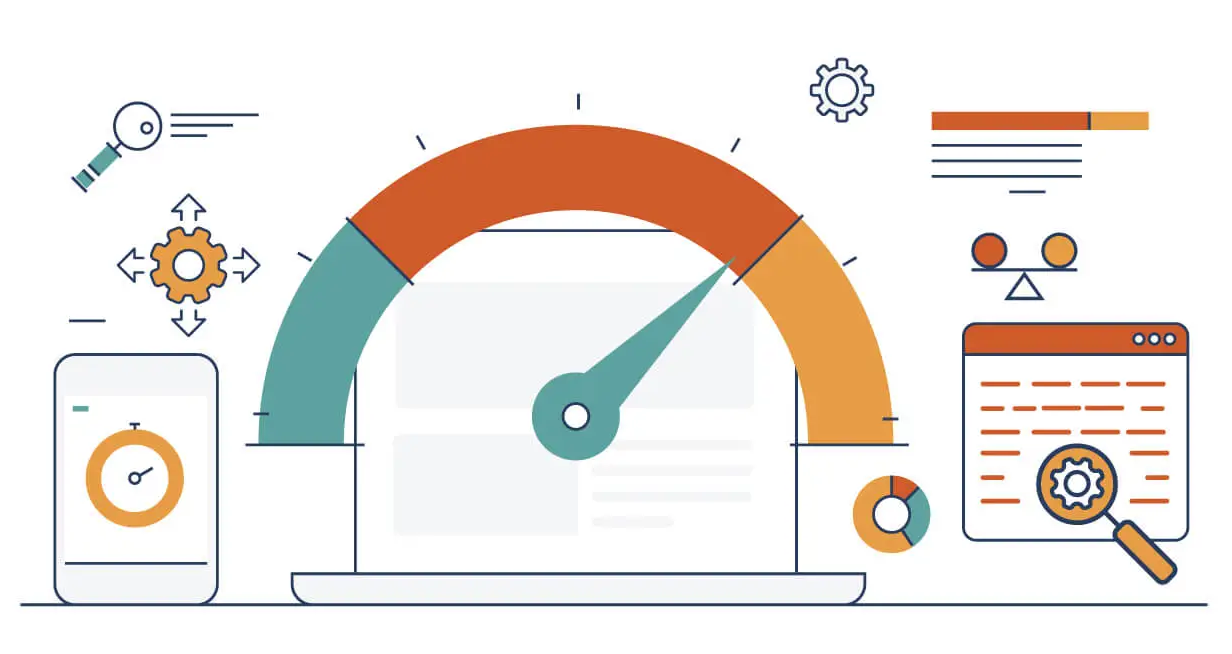
# Workshop: Performance Testing with JMeter and K6



In this workshop, we will explore performance testing methodologies using two powerful tools: **Apache JMeter** and **K6 Framework**.

### How to Run the Project

You should have installed **Docker**.

Follow these steps to get the application running in a Docker container.

1. **Download** the **EShop.zip** file, which contains all the necessary files.
2. **Unzip** the **EShop.zip** file into your preferred directory on your machine.
3. **Build** and **Run the Docker Containers**.

Ensure you have **Docker** and **Docker Compose** installed. Then, run the following command to build and start the containers:

**docker-compose up --build**

This command will load the Docker image into your local Docker environment.

1. **Access** the API

Once the containers are up and running, you can access the API at <http://localhost:5000/api>.

1. API **Documentation**

API documentation is available at <http://localhost:5000/api-docs>.

## Perform tests with JMetter

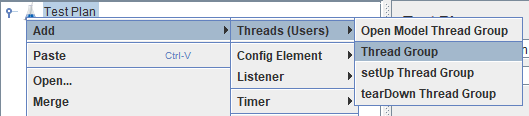
### Open Apache JMeter

* You should have already installed JMetter. If you don’t have it you can download the JMeter binary – "**apache-jmeter-{version}.zip**" from Binaries section of [**Apache JMeter Official Website**](https://jmeter.apache.org/download_jmeter.cgi).

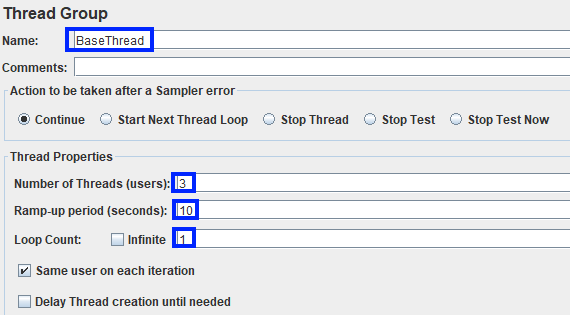
Start JMeter by running the **jmeter.bat** (Windows) or **jmeter.sh** (Linux/Mac) script from the **JMeter bin directory**.

### Add a Thread Group

Right-click on the **Test Plan** in the left-hand tree. Select **Add > Threads (Users) > Thread Group**.



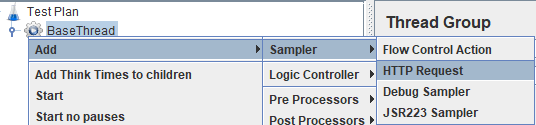
This will create a **Thread Group** where you can define the **number of users** (**threads**), **ramp-up period** and **loop count**. Do not forget to change **thread name**.



### Blog Category Get All Request

#### Add an HTTP Get Request

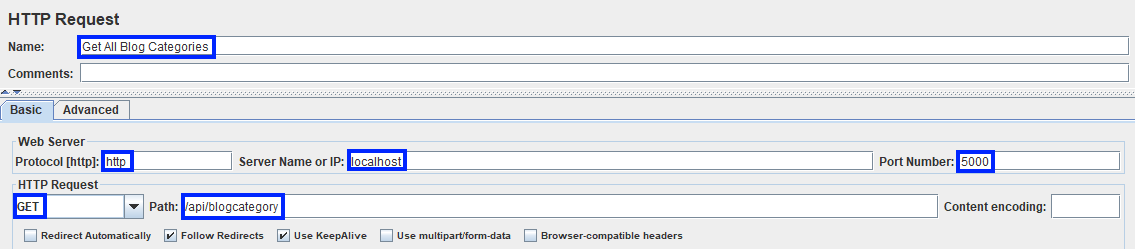
Right-click on the **Thread Group** you just created. Select **Add > Sampler > HTTP Request**.



This will add an HTTP Request Sampler to the Thread Group.

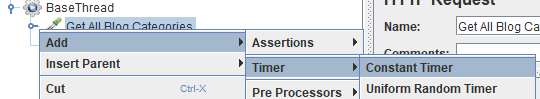
#### Configure the Get HTTP Request

* **Name**: Give a name to your HTTP Request
* **Server Name or IP**: Enter "**localhost**" (or the IP address of the server if it's different).
* **Port Number**: Enter **"5000** ".
* **HTTP Request Method**: Enter "**Get**".
* **Path**: Enter "**/api/blogcategory/**".
* **Protocol**: The server uses HTTP, so leave this as http

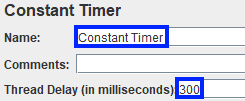


#### Add a Timer

Right-click on the **Request**. Select **Add > Timer > Constant timer** (or any other listener like Summary Report, etc.).

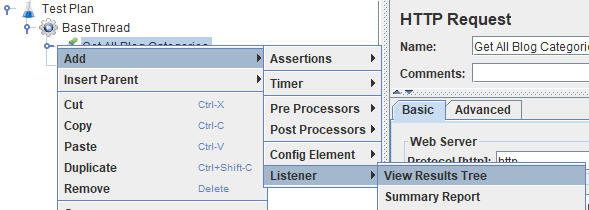


This will add **delay** to your request. Change the **name** and **thread delay**.



#### Add a Listener to View Results

Right-click on the **Thread Group**. Select **Add > Listener > View Results Tree** (or any other listener like Summary Report, etc.).



This will help you see the response of your request when you run the test.

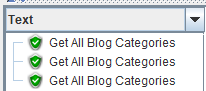
#### Run the Test

**Before running the tests do not forget to start EShop application.**

Click on the **Start** button (green play icon) in the JMeter toolbar.



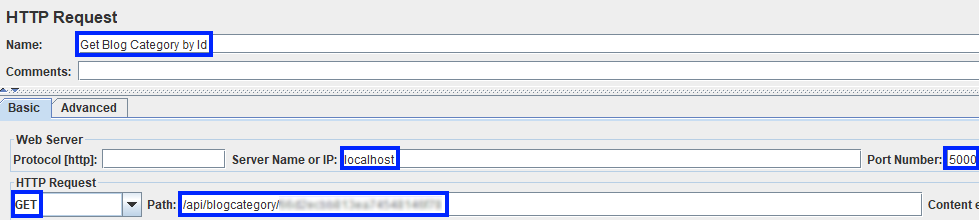
After the test completes, check the **View Results Tree** (or other listener) to see the details of the request and response.



### Blog Category Get One by Id Request

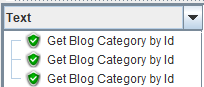
**Add and Configure an HTTP Get Request**

Right-click on the **Thread Group** you just created. Select **Add > Sampler > HTTP Request**. Change name add server name and port number. Then change request method to get and path to "**/api/blogcategory/{id}**". You can get id directly from swagger.



**Add a Timer and Listener to View Results**

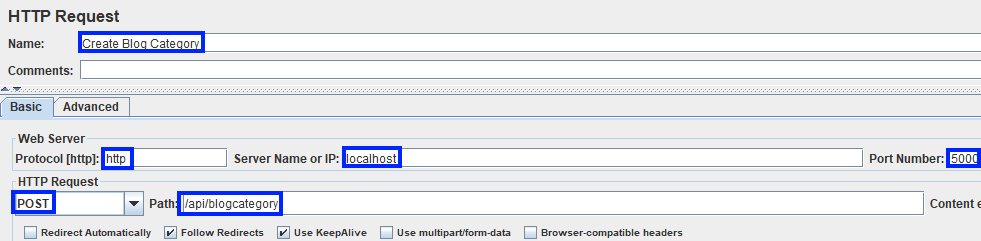
Click on the **Start** button (green play icon) in the JMeter toolbar and you should see this response:



### Blog Category Create Request

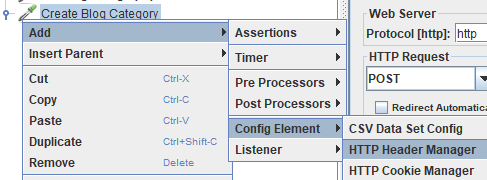
#### Add and Configure an HTTP Post Request

Right-click on the **Thread Group** you just created. Select **Add > Sampler > HTTP Request**. Change name add server name and port number. Then change request method to get and path to "**/api/blogcategory**".

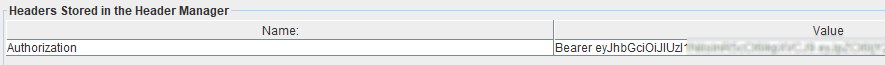


#### Add HTTP Header Manager

You need to add custom header (Authorization), so right-click on the **HTTP Request** you added. Select **Add > Config Element > HTTP Header Manager**.

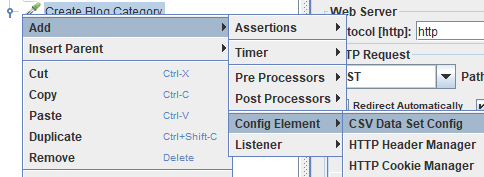


Add the required headers here. You can get token directly by logging from swagger:

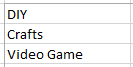


#### Add CSV Data Set Config

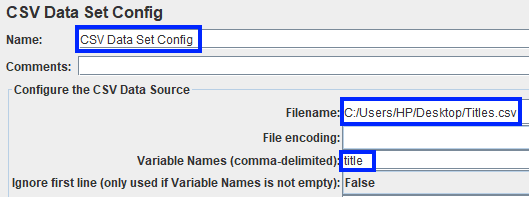
You need to add unique blog category titles, so right-click on the **HTTP Request** you added. Select **Add > Config Element > CSV Data Set Config**.



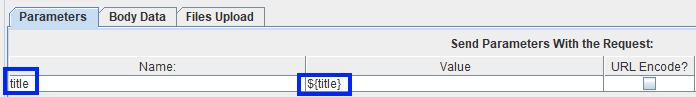
Then create CSV file with example data.



Then change the name, filename and add "**title**" as **variable name**.



After this edit **request parameters**:



#### Add a Timer and Listener to View Results

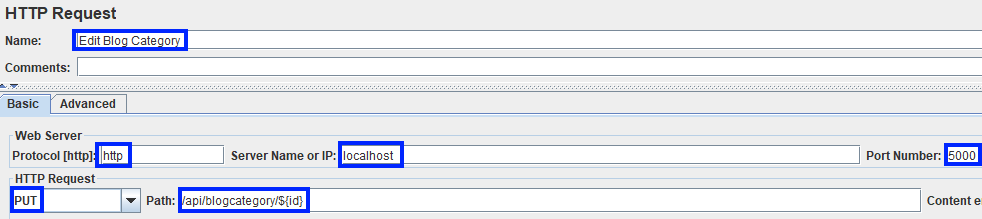
Click on the **Start** button (green play icon) in the JMeter toolbar and you should see this response:



### Blog Category Edit Request

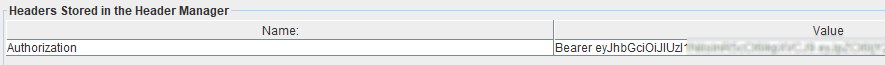
#### Add and Configure an HTTP Put Request

Right-click on the **Thread Group** you just created. Select **Add > Sampler > HTTP Request**. Change name add server name and port number. Then change request method to get and path to "**/api/blogcategory/{id}**".



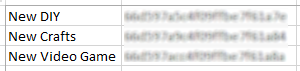
#### Add HTTP Header Manager

You need to add custom header (Authorization), so right-click on the **HTTP Request** you added. Select **Add > Config Element > HTTP Header Manager**. The required headers here. You can get token directly by logging from swagger:



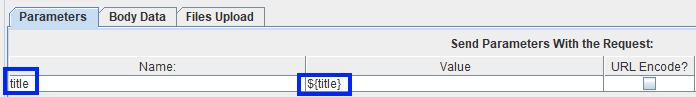
#### Add CSV Data Set Config

You need to add unique blog category titles, so right-click on the **HTTP Request** you added. Select **Add > Config Element > CSV Data Set Config**. Then create CSV file with example data. You can get ids directly from swagger.



Then change the name, filename and add "**title, id**" as **variable name**.

After this edit **request parameters**:



#### Add a Timer and Listener to View Results

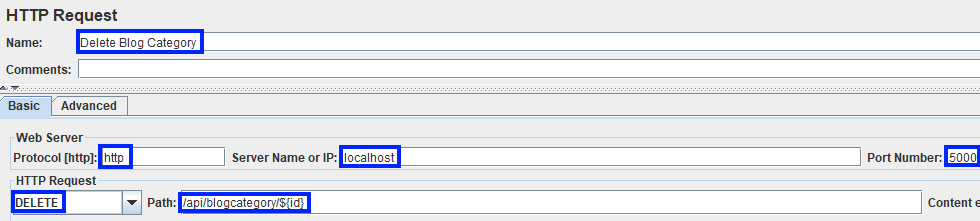
Click on the **Start** button (green play icon) in the JMeter toolbar and you should see this response:



### Blog Category Delete Request

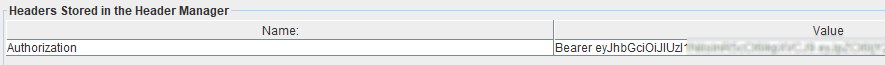
#### Add and Configure an HTTP Delete Request

Right-click on the **Thread Group** you just created. Select **Add > Sampler > HTTP Request**. Change name add server name and port number. Then change request method to get and path to "**/api/blogcategory/{id}**".



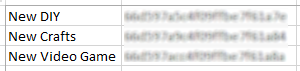
#### Add HTTP Header Manager

You need to add custom header (Authorization), so right-click on the **HTTP Request** you added. Select **Add > Config Element > HTTP Header Manager**. The required headers here. You can get token directly by logging from swagger:



#### Add CSV Data Set Config

You need to add unique blog category titles, so right-click on the **HTTP Request** you added. Select **Add > Config Element > CSV Data Set Config**. Then create CSV file with example data. You can get ids directly from swagger.



Then change the name, filename and add "**id"** as **variable name**.

#### Add a Timer and Listener to View Results

Click on the **Start** button (green play icon) in the JMeter toolbar and you should see this response:



## Use K6 Framework

First create **test.js** file where you will write your tests. In your terminal, **navigate to the directory containing the test.js file**. You can run all the test with the following command:

k6 run test.js

### Enquiry Get All Request

Let’s write a script that is designed to perform a load test on the GET endpoint http://localhost:5000/api/enquiry/.

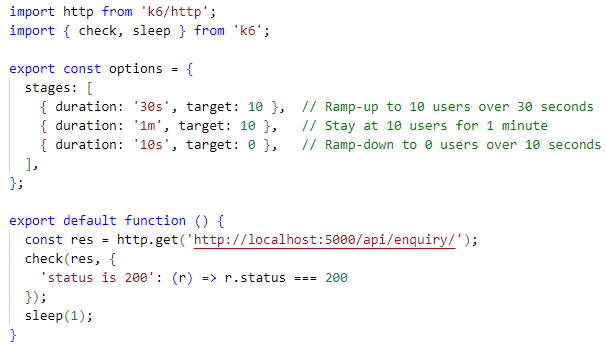
**Load Pattern**:

* Gradually increases the number of virtual users (VUs) to **10 over 30 seconds**.
* Maintains **10 VUs for 1 minute**.
* Gradually decreases VUs to **0 over** **10 seconds**.

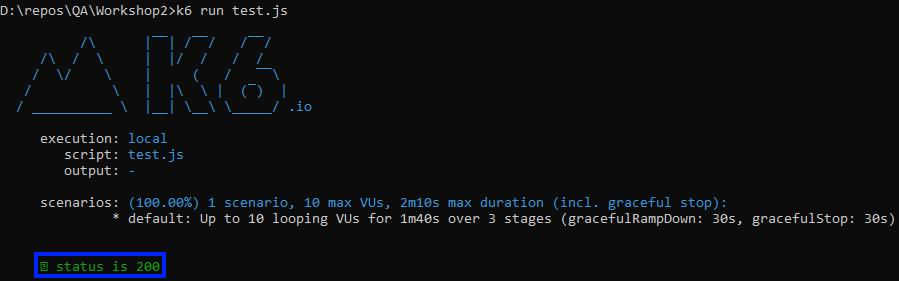
**Validation Checks**:

* Ensures the **HTTP response** status is **200** (**OK**).

The script pauses for **1 second between each request**.



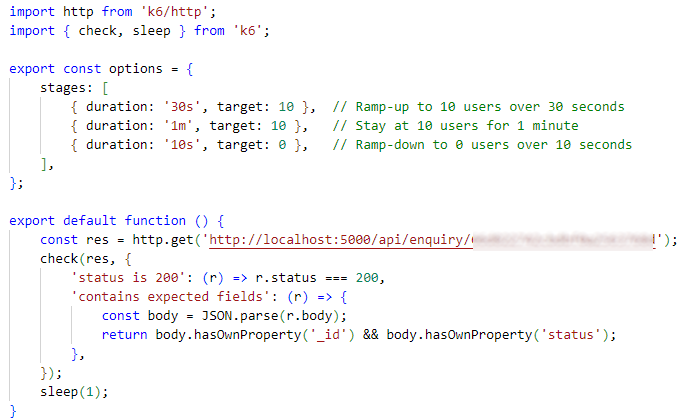
Run the test and you should see this output:



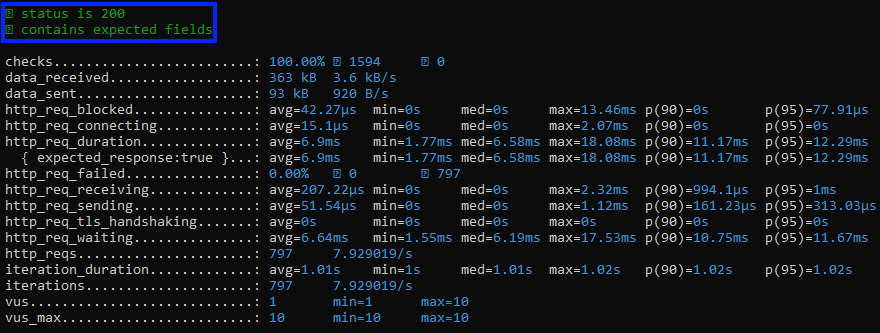
### Enquiry Get One by Id Request

Let’s write a script that is designed to perform a load test on the GET by Id endpoint http://localhost:5000/api/enquiry/{id}.

The script first ramps up to **10 virtual users over 30 seconds**, maintains this **load for 1 minute**, and then gradually reduces the number of users to zero over **10 seconds**. During each iteration, the virtual **users send a GET request** to the specified API endpoint and the response is checked to ensure a **200 status code** and the **presence of specific** **fields** (**id** and **status**). **A 1-second delay** is added **between each request** to mimic real-world user interactions. **You can get id directly from swagger**.

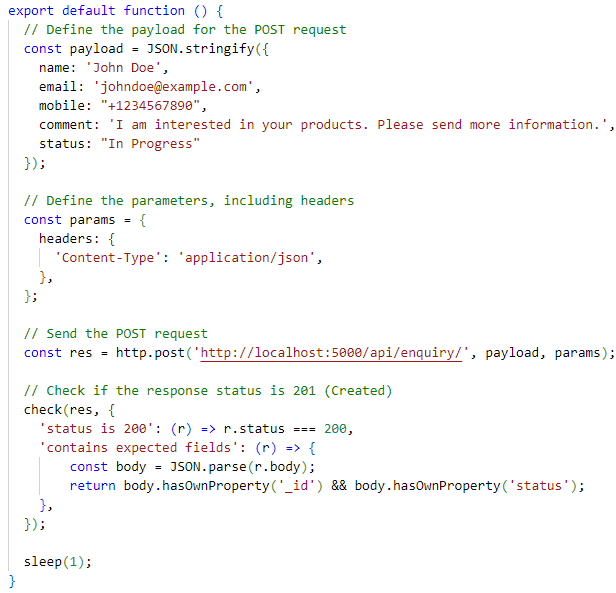


Run the test and you should see this output:

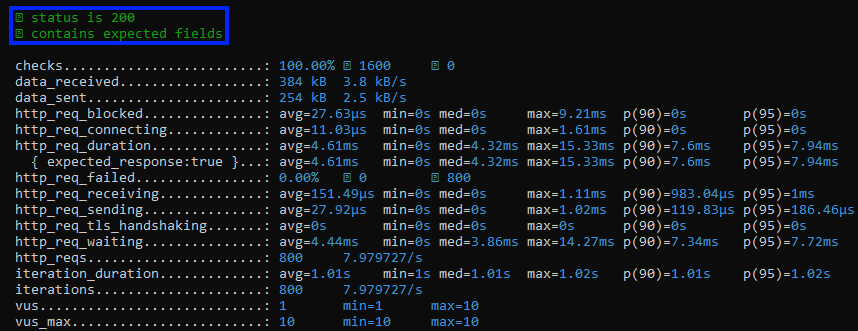


### Enquiry Create Request

The next script performs a load test on the **POST** endpoint http://localhost:5000/api/enquiry/, simulating up to **10 virtual users** over a **period of 1 minute**. It sends **POST requests** with a JSON payload **containing user details** and **checks** that the server responds with a **status** of **200** and includes expected fields like **id** and **status** in the response. The script helps assess the API's ability to handle concurrent requests while ensuring correct functionality under load.



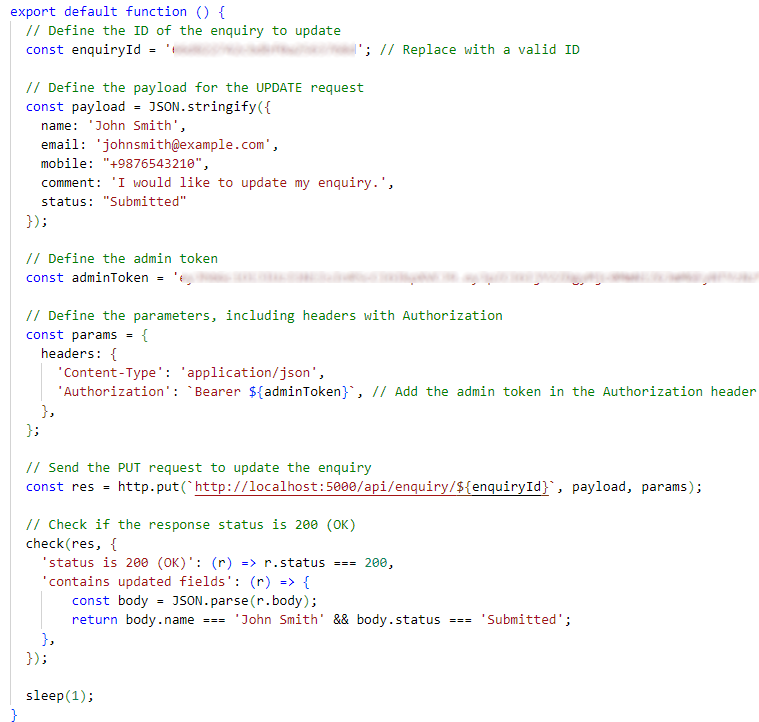
This is how your terminal should look like after running tests. You can also see **detailed information** as **checks**, **sent** **request**, **interactions** and many others:



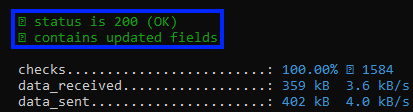
### Enquiry Edit Request

Let’s write a script that is designed to perform a load test on the Edit endpoint **http://localhost:5000/api/enquiry/{id}**.

This K6 load testing script simulates updating an enquiry via an **HTTP** **PUT** **request**, using an **admin authentication** **token** for **secure access**. The test gradually ramps up to **10 virtual users over 30 seconds**, maintains this load for **1 minute**, and then ramps down to **0 users over 10 seconds**. During the test, the script sends a request to **update an enquiry's details**, such as the **name**, **email**, **mobile** **number**, **comment**, and **status**. The **admin token** is included in the **request headers for authorization**. After sending the request, the script checks whether the response status is **200 OK** and **verifies that the updated fields** (**name** and **status**) **are accurate**. A **1-second delay** is added **between** **requests** to mimic real-world user behavior, allowing for performance evaluation under moderate load.



Run the test and you should see this output:



Feel free to write and **customize your own tests** to suit your **specific requirements**. You can **create various test** **scenarios** to evaluate performance, functionality, or different aspects of the system under load, ensuring it meets your desired outcomes.

Enjoy ☺