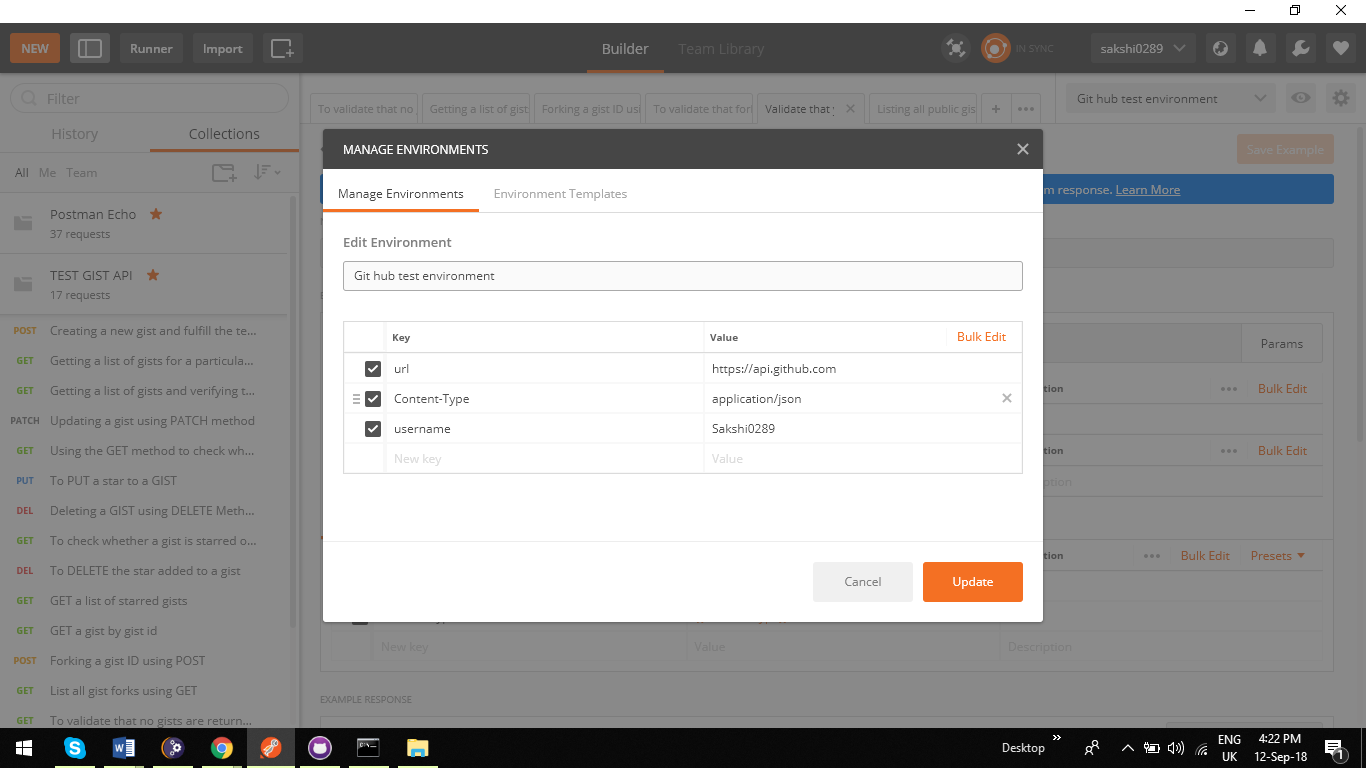
**GIST API testing doc:**

**Prerequisite**: POSTMAN GUI should be installed.

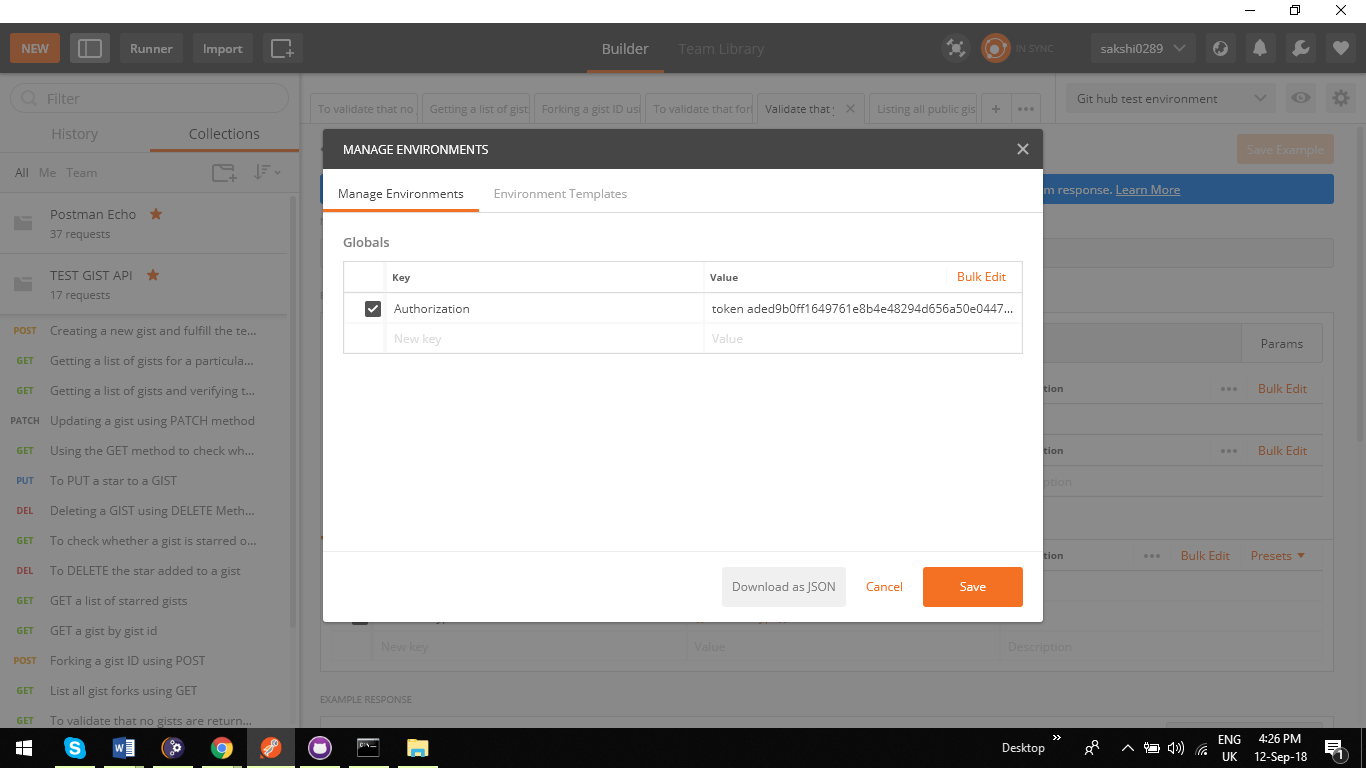
**Points to note:**

1. I have created all the test cases using **POSTMAN** **GUI**. Request you to run using POSTMAN GUI
2. The test cases are created in a **collection** called “TEST GIST API”.
3. I have created an **environment** called “Git hub test environment” for the testing purpose where I have initialized three variables which can be used for the entire collection if we use that environment. It is done to maintain the ease, so we do not have to write the values again and again for each test case. Please see the screenshot below:
4. url
5. Content-Type
6. username



1. I have initialized one **global** **variable** called **Authorization** which has the OAuth token for validation purpose. It could have been an environment variable as well but just to showcase that we can use global variables also I have initialized it globally.

**Note**: The test cases may not work with my OAuth token as I will push the code to git hub. Request you to use your own OAuth token which can be changed by editing the global variable section and updating your own value. That is the only change you must make. No need to make changes in the test cases. Importance of making it a global variable 😊



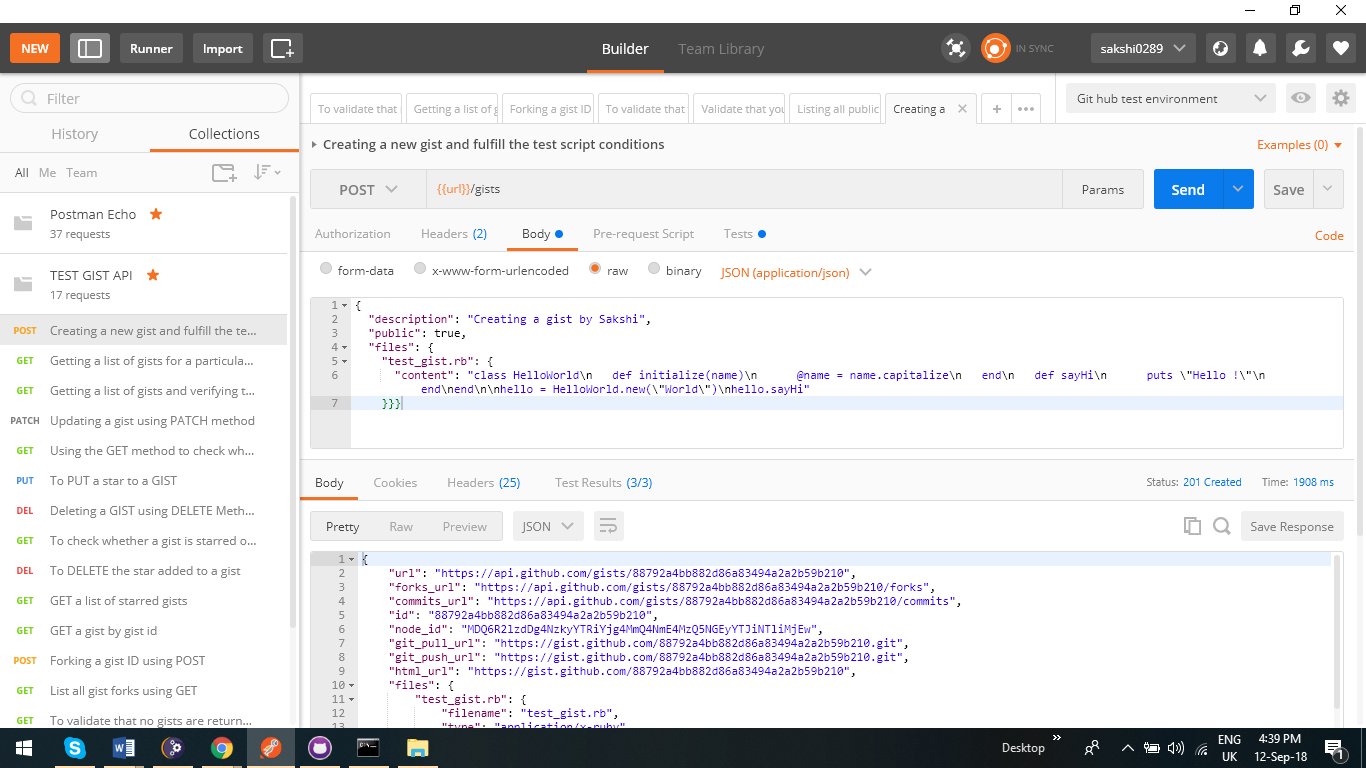
1. Every test case has some or other test scripts mentioned for validating the response. Please do have a look at that as well.

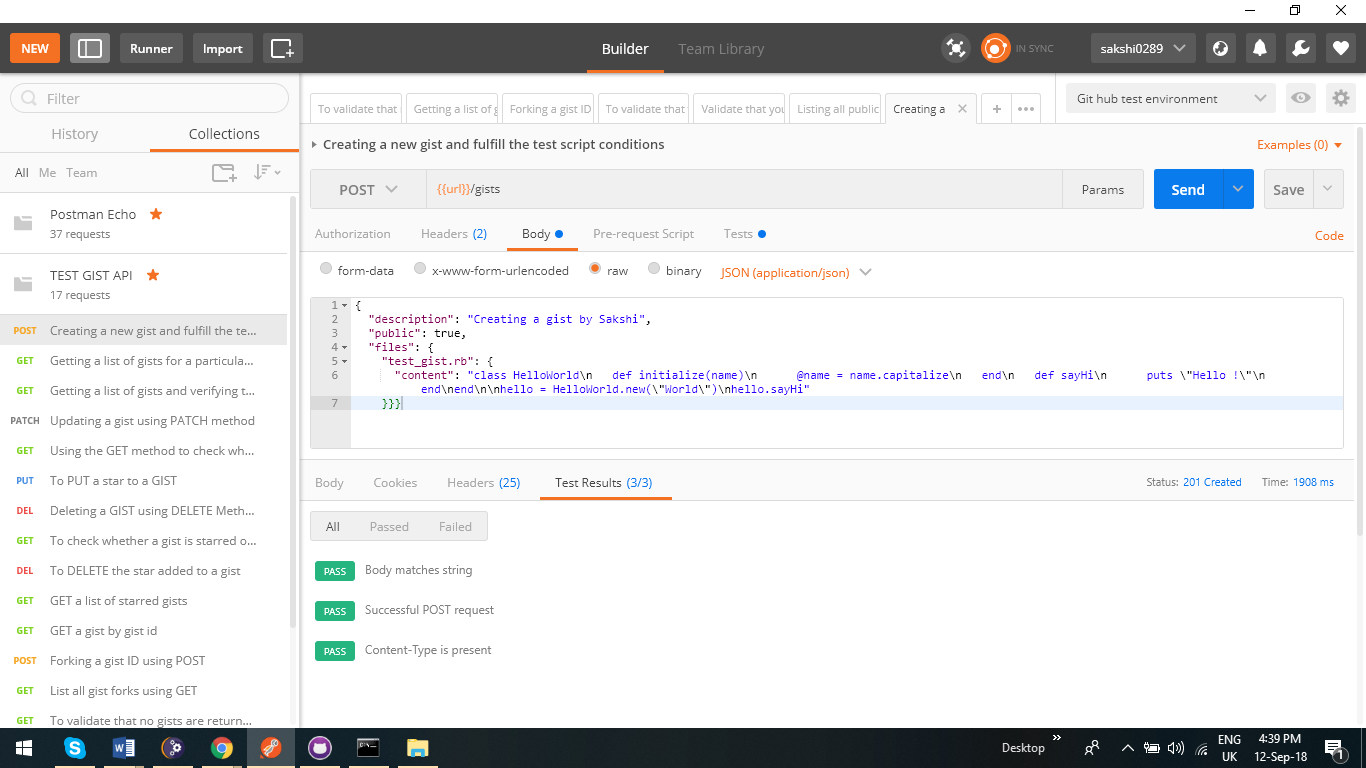
**Test Cases**

1. **Creating a new gist and fulfill the test script conditions**

In this we are creating a new gist by passing the body of the gist we want to create and passing the Authorization code for the header.

As we can see we can see the gist is created with response as 201. The 3 test conditions we have put are passing in the other screenshot.

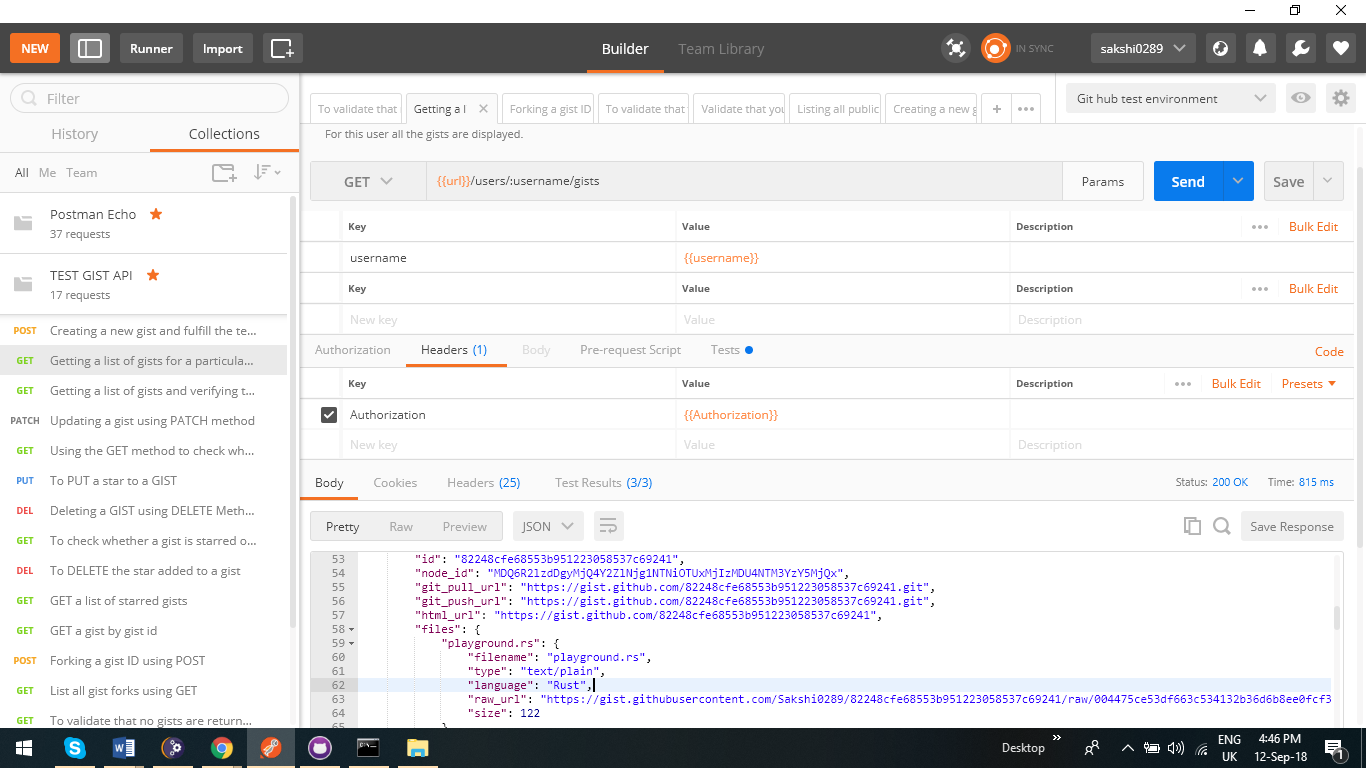


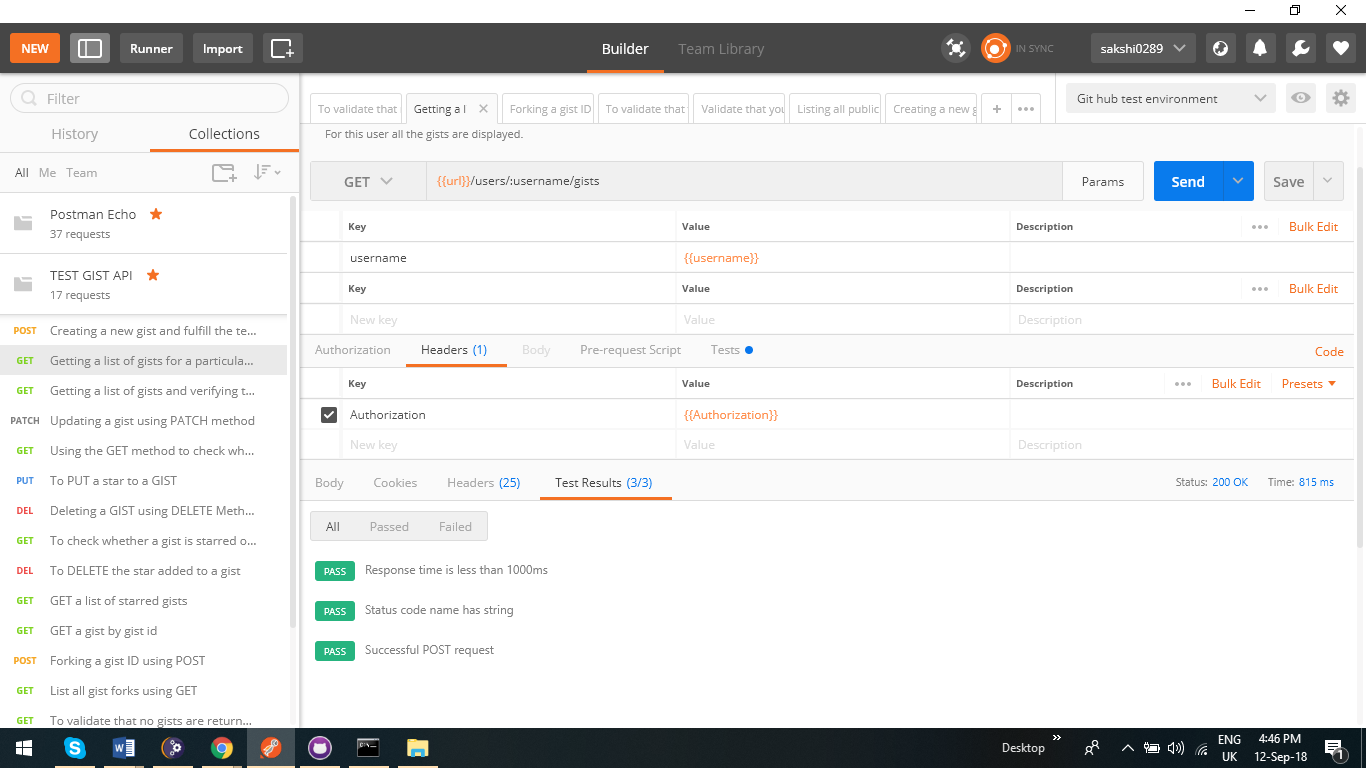


1. **Getting a list of gists for a particular user by GET method**

As we can see we are getting a list of username for that particular user in this case : Sakshi0289

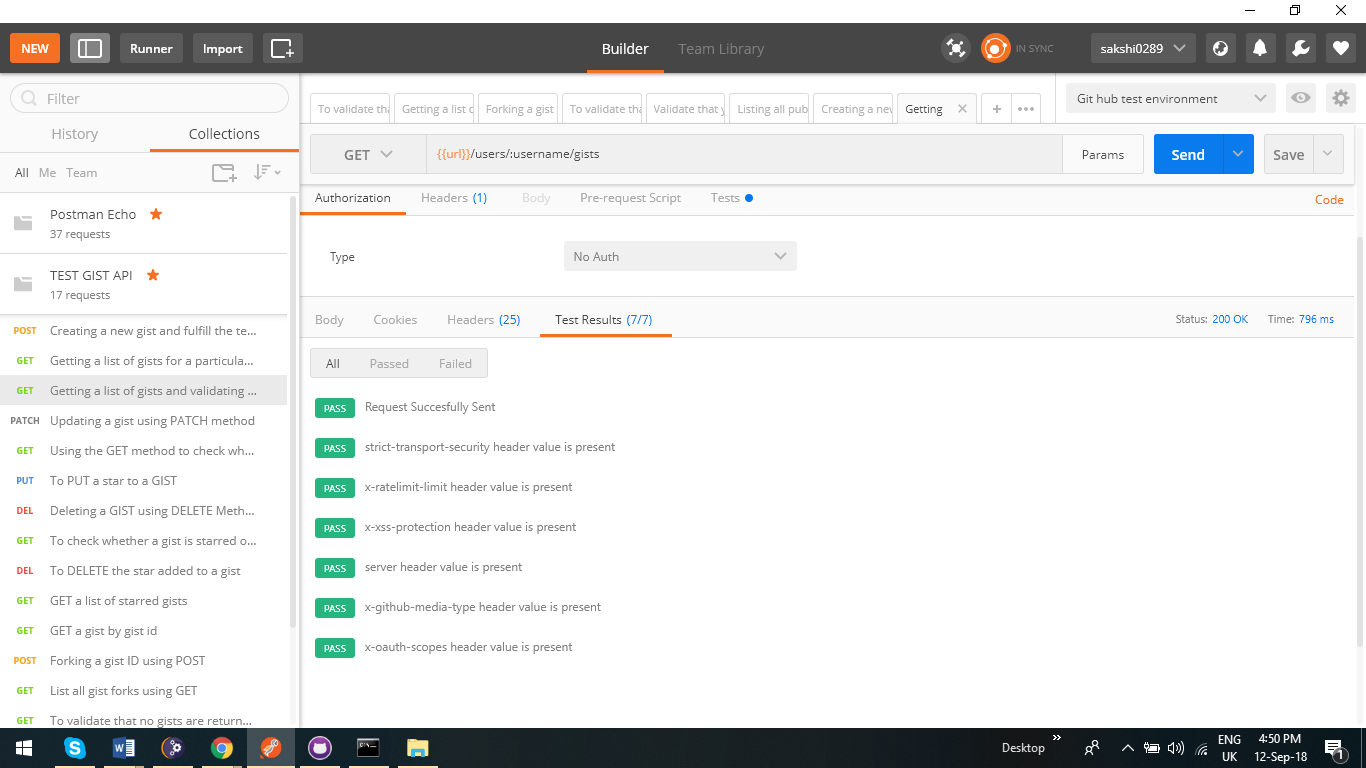
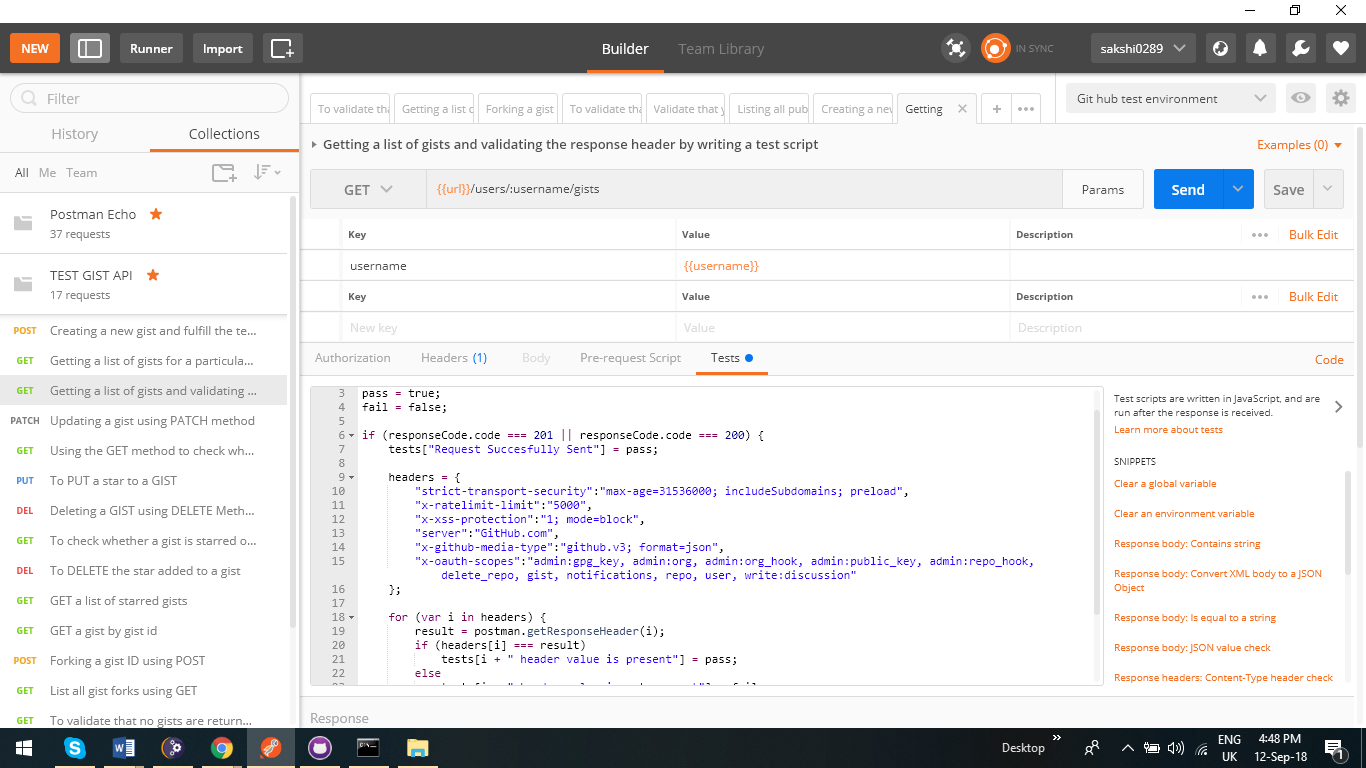
You can use your user by changing the name in environment variable.





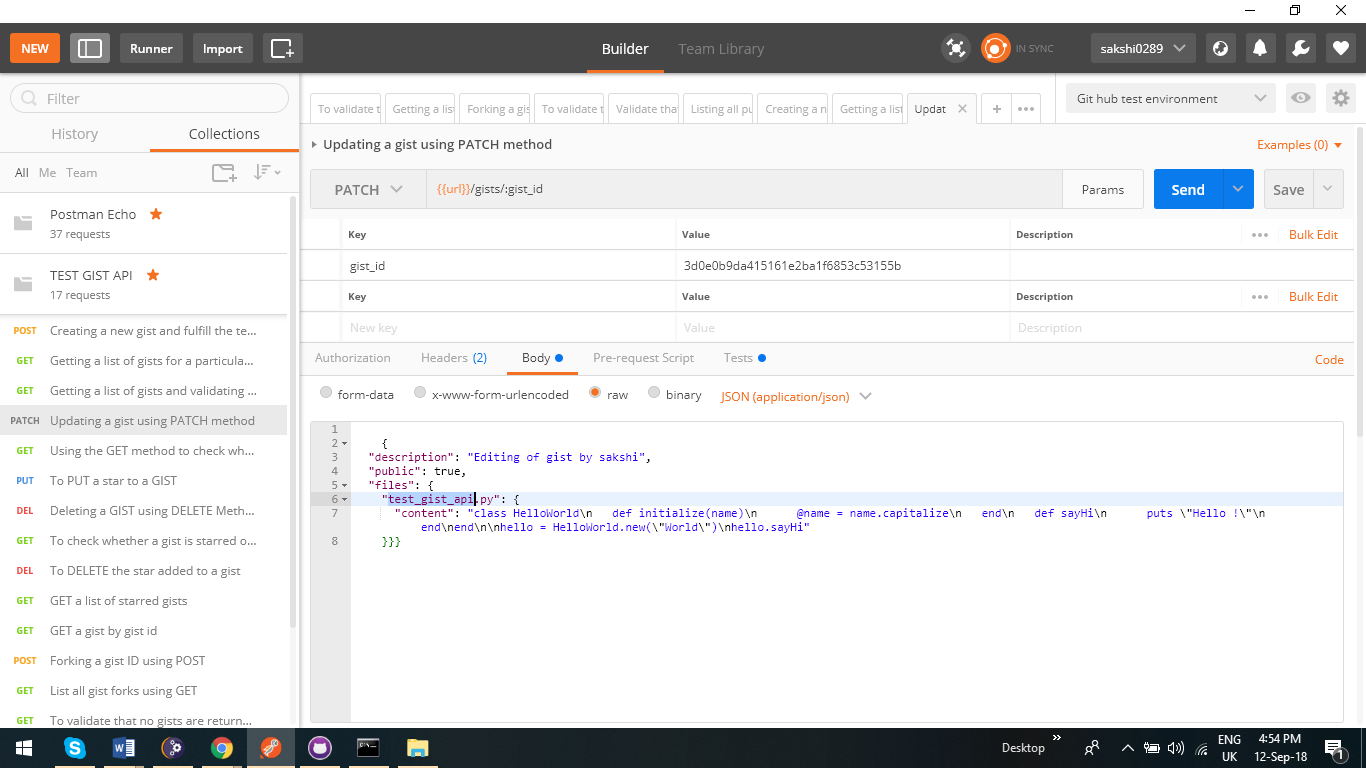
1. **Getting a list of gists and validating the response header by writing a test script**

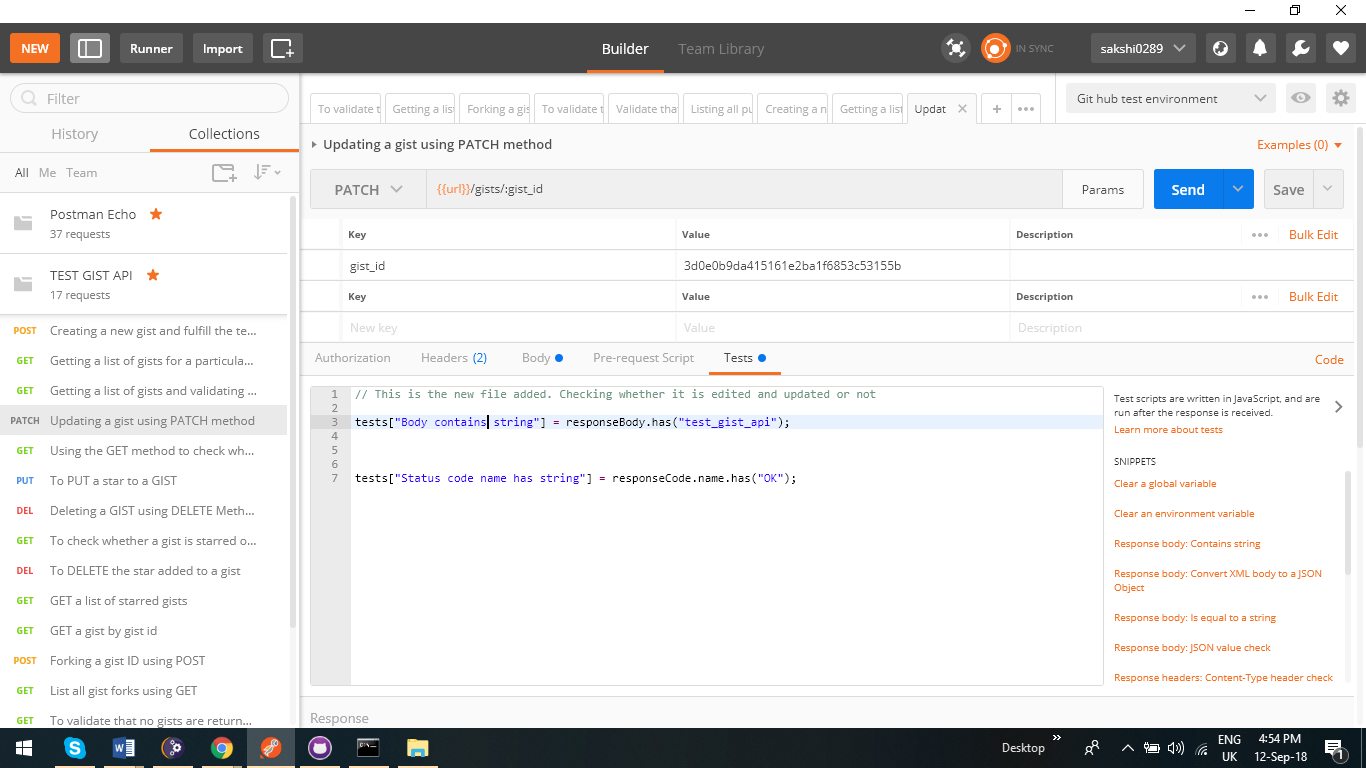
WE have written a javascript in test script to validate the response headers which we get. The second screenshot shows that all the 7 headers are present in the response body.

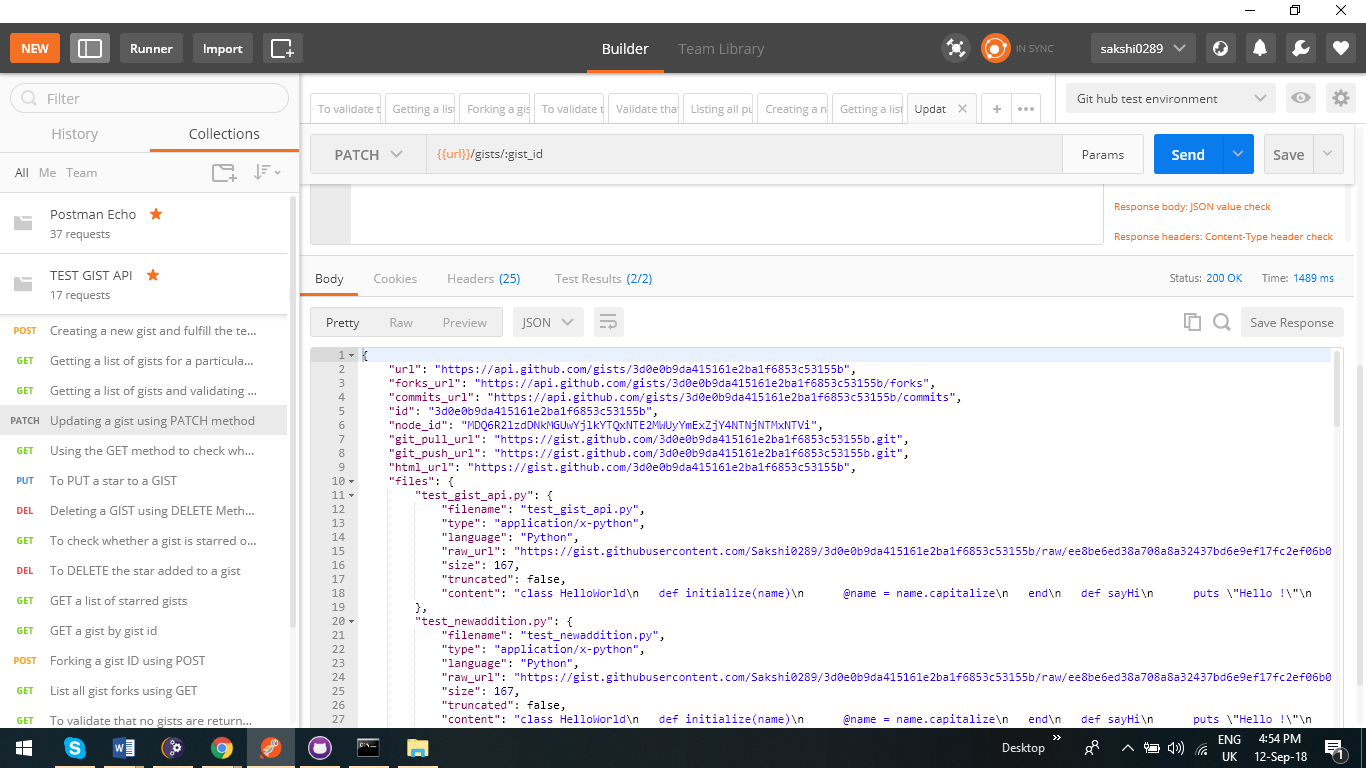
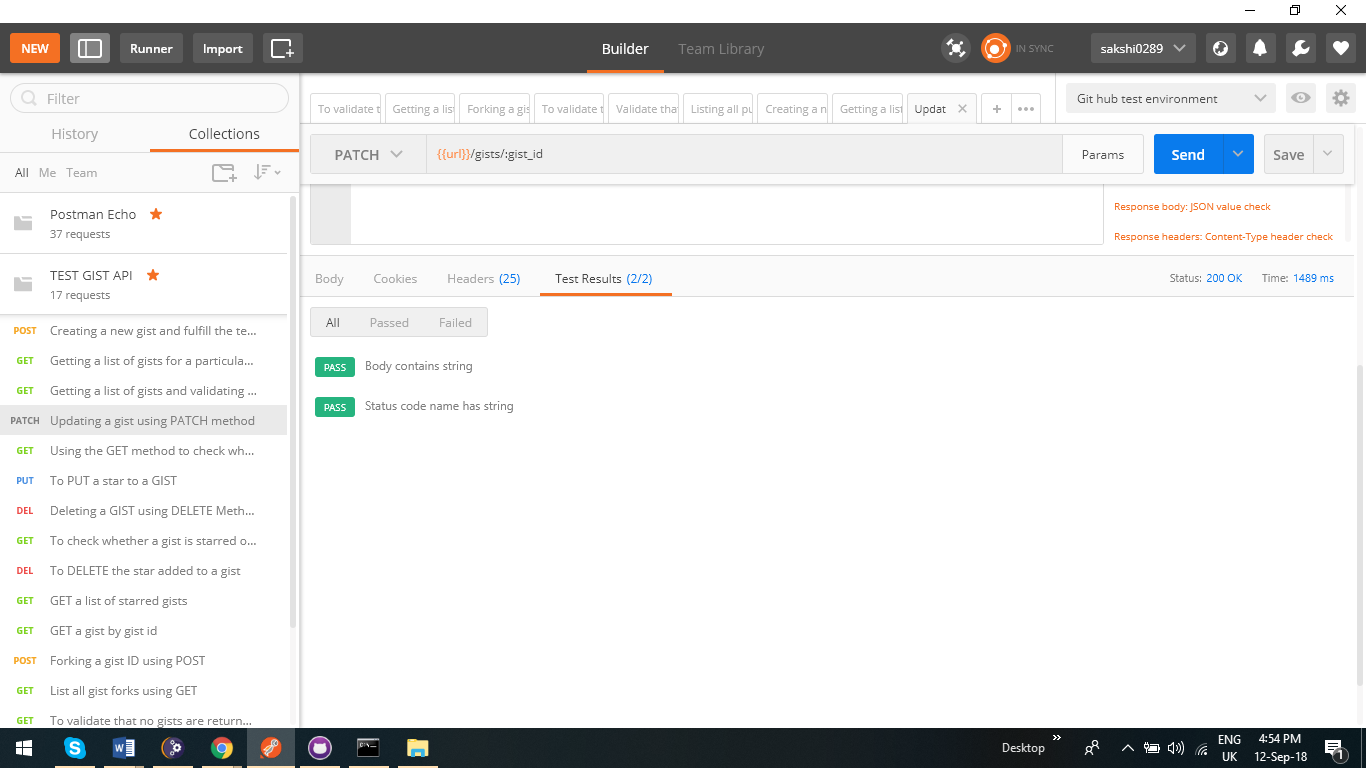


1. **Updating a gist using PATCH method**

We are updating a gist with another file by passing the gist id as parameter and validating by test script that the new file name is present in the response. The last screenshot in the test validate that the test has passed.

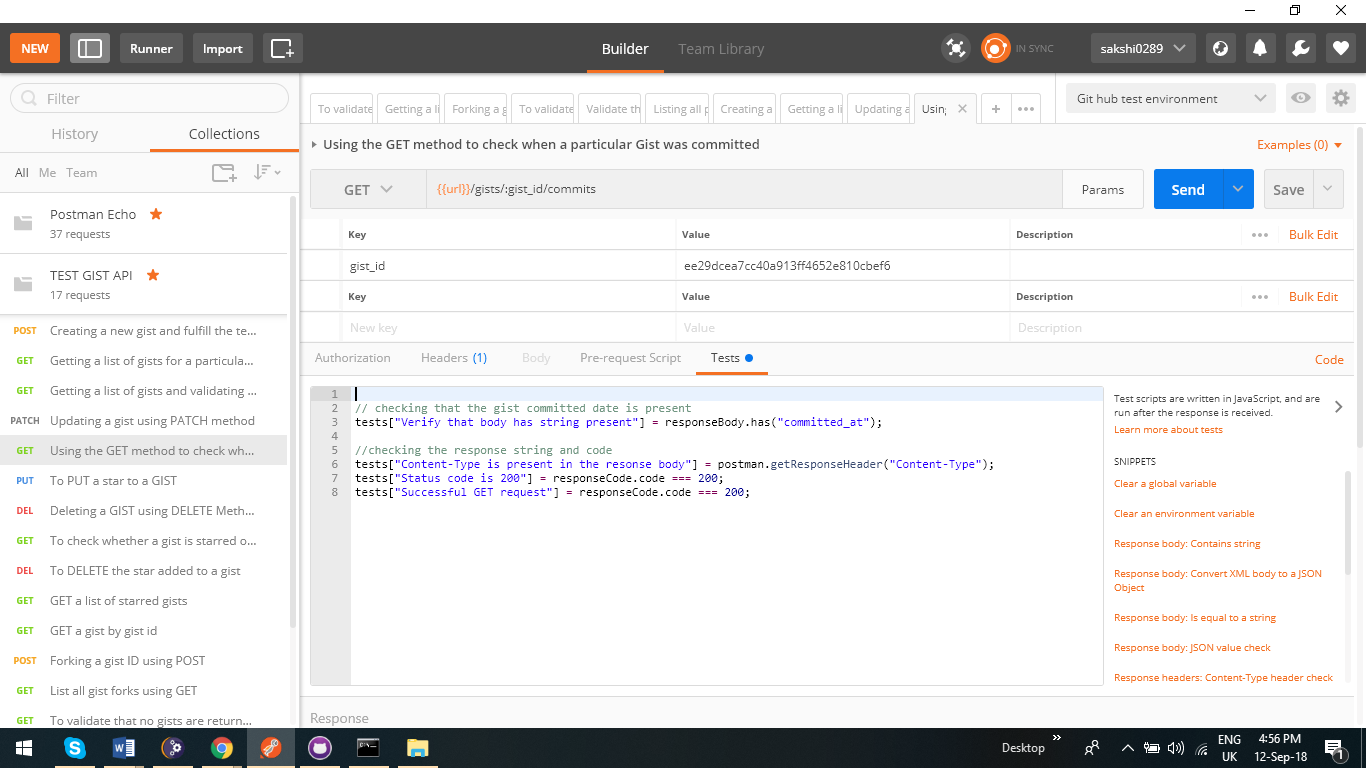


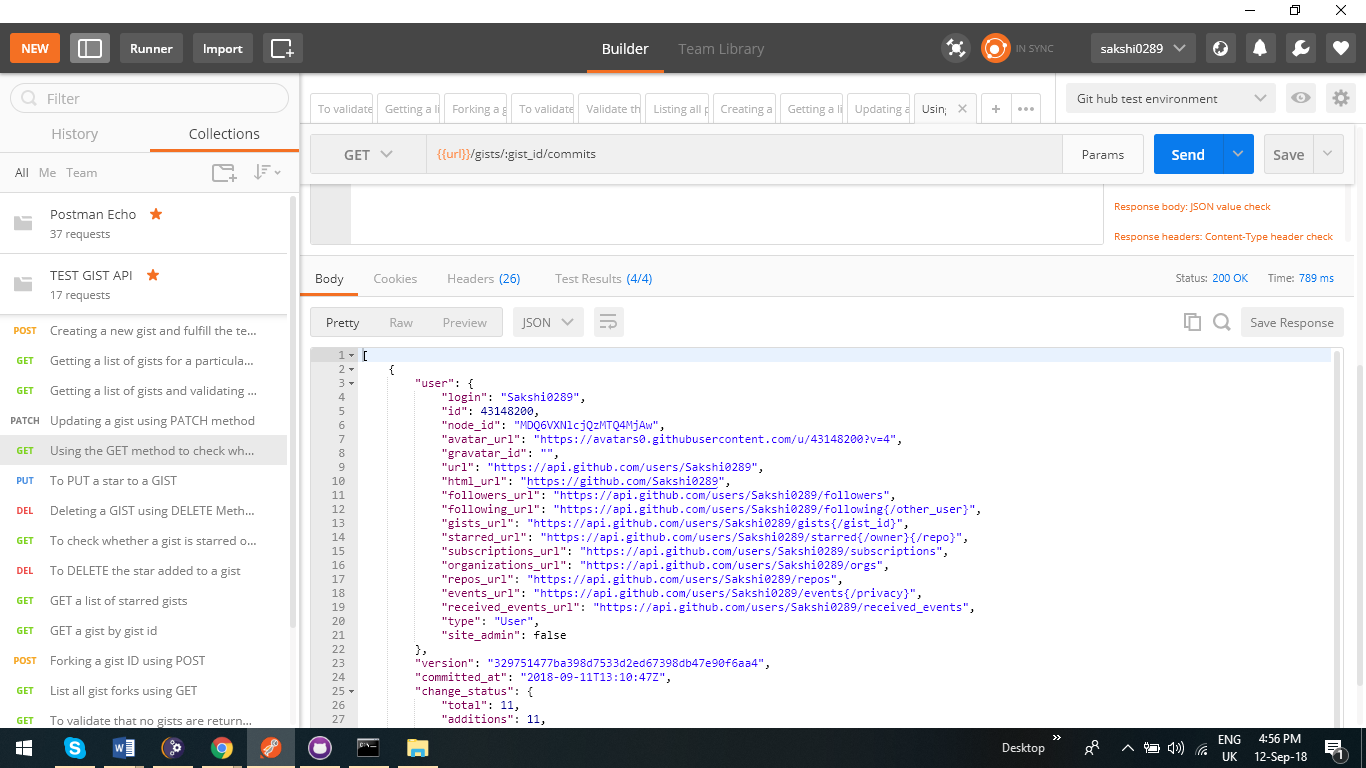


1. **Using the GET method to check when a particular Gist was committed**

It tells us the time when the particular gist was committed. As the response string contains the string which we have validated via test script, the test case passed.

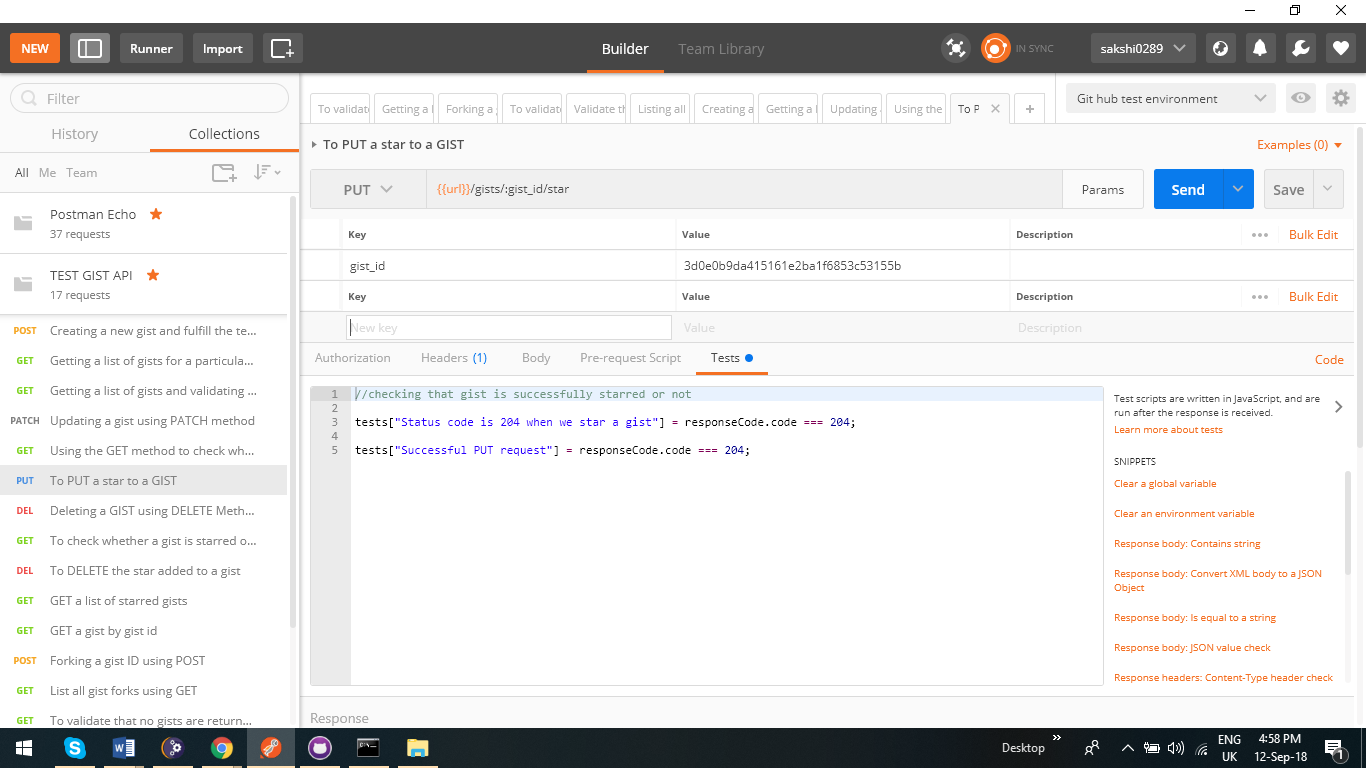


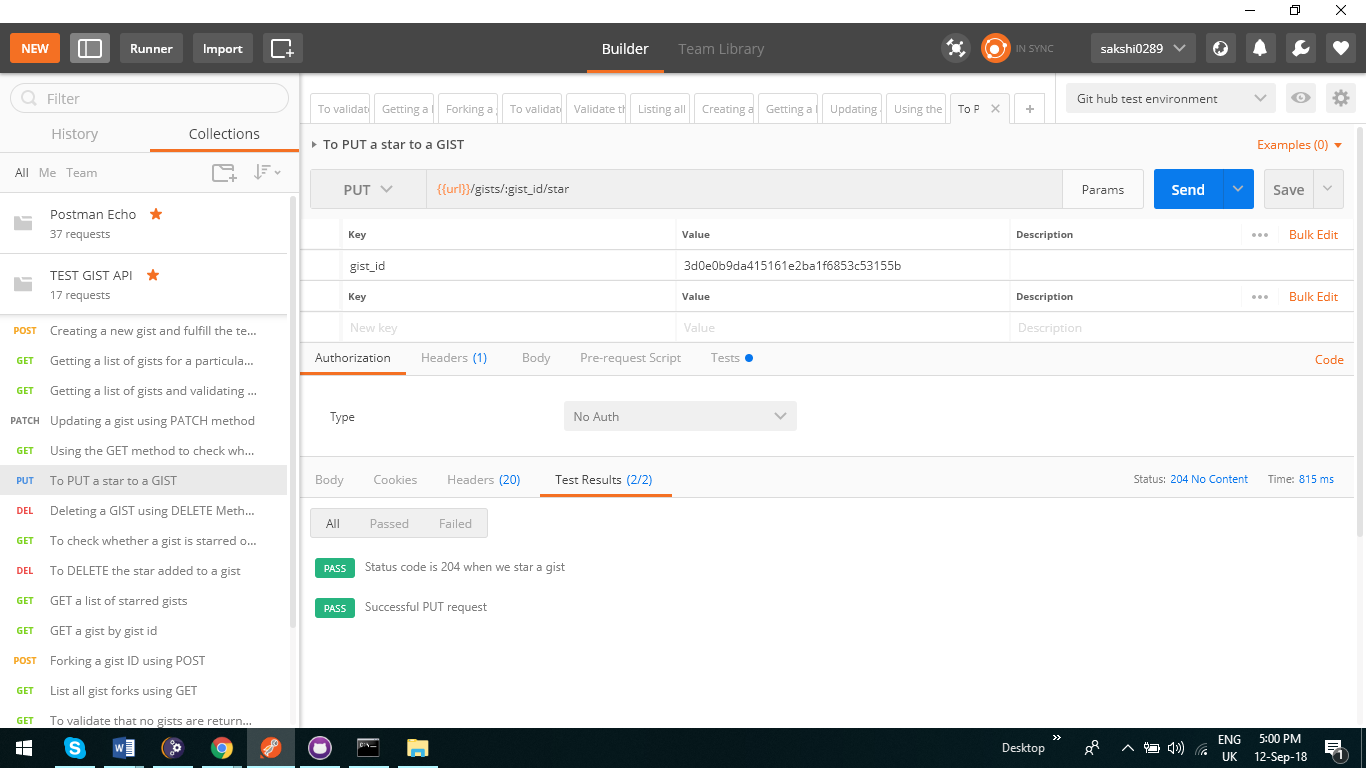


1. **To PUT a star to a GIST**

It is a simple test to star a gist via PUT method.

Status code is 204 when we star a gist which is a test script validation as seen below.

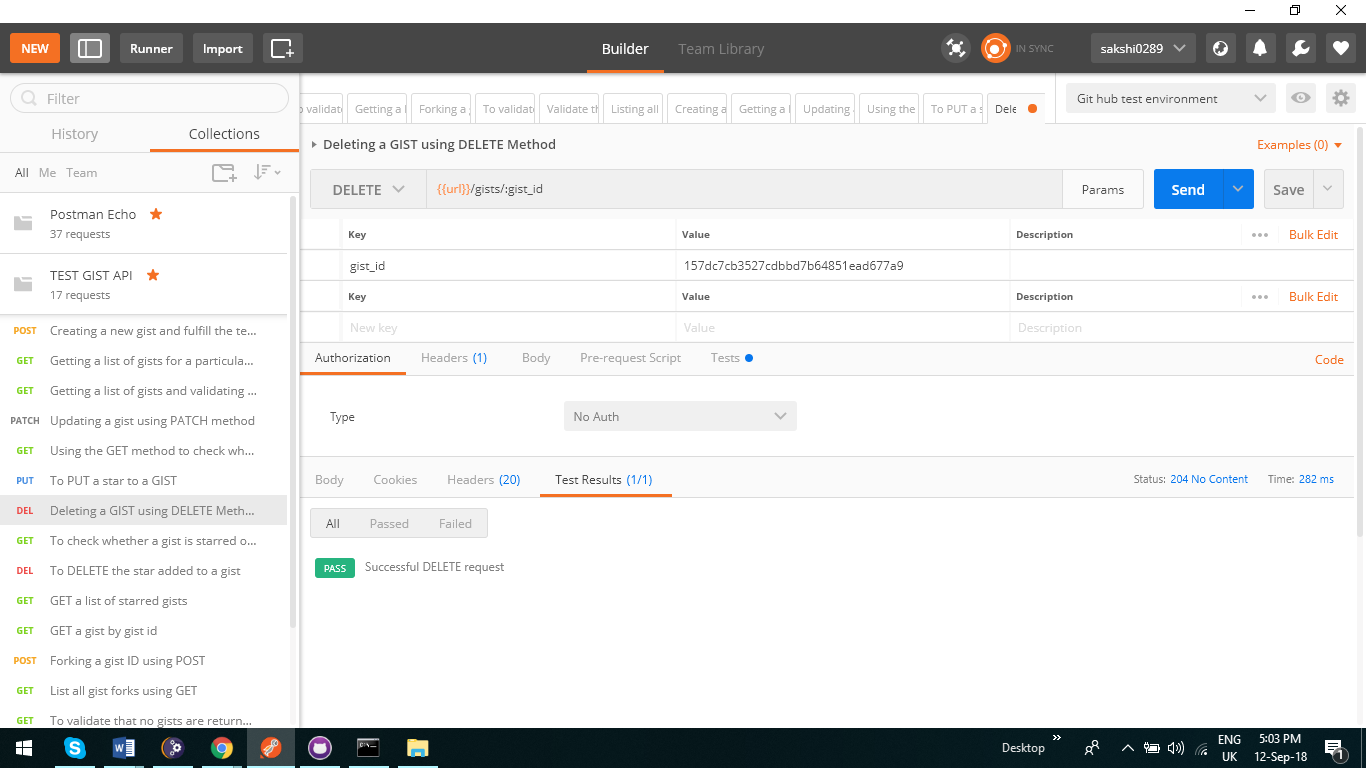




1. **Deleting a GIST using DELETE Method**

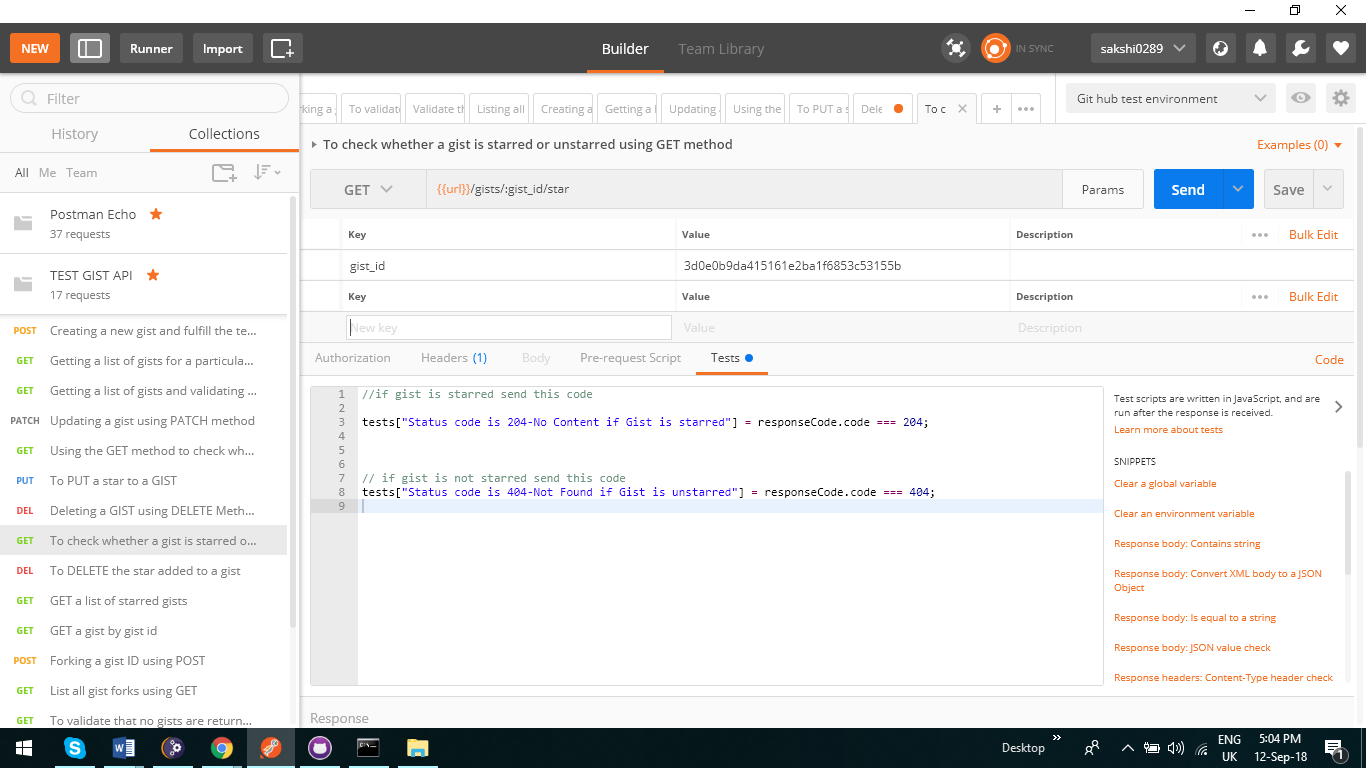
**We do that by passing the gist id of the gist we want to delete**



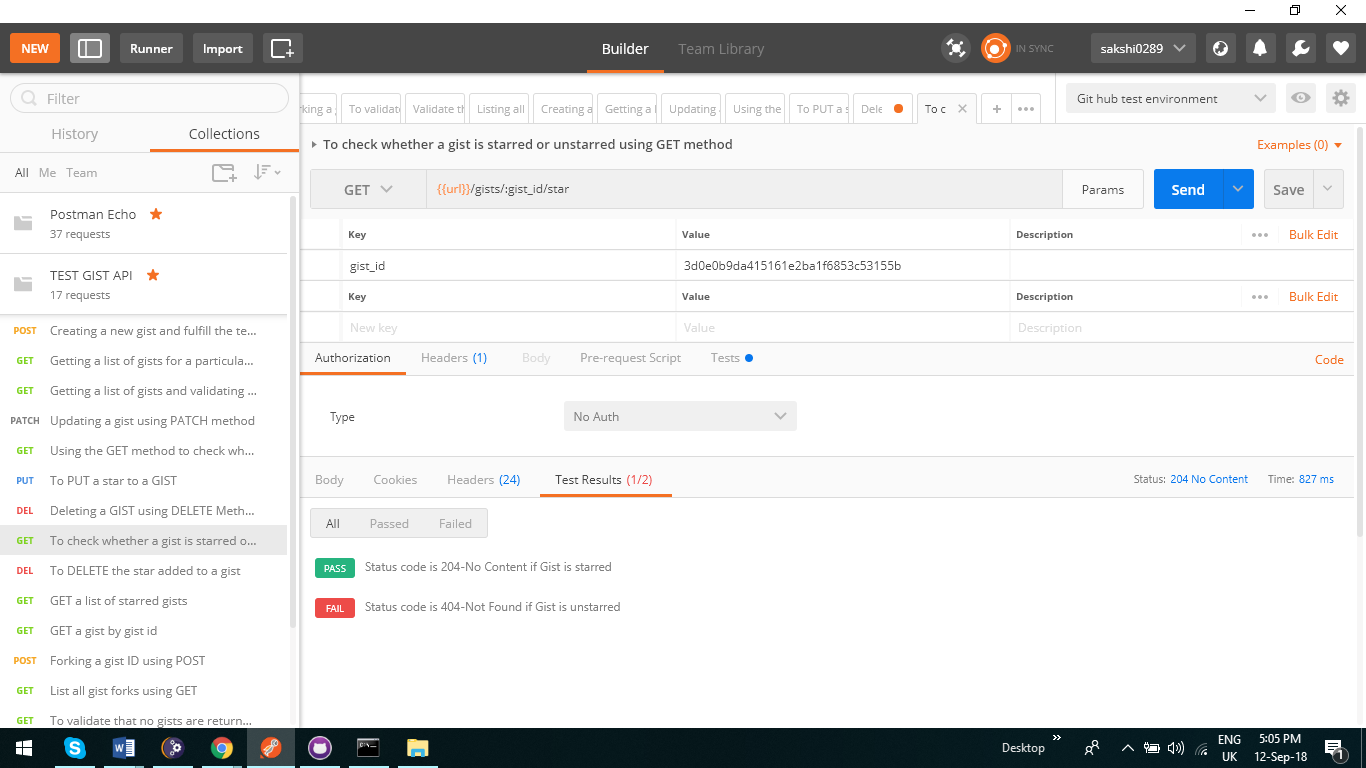


1. **To check whether a gist is starred or unstarred using GET method**

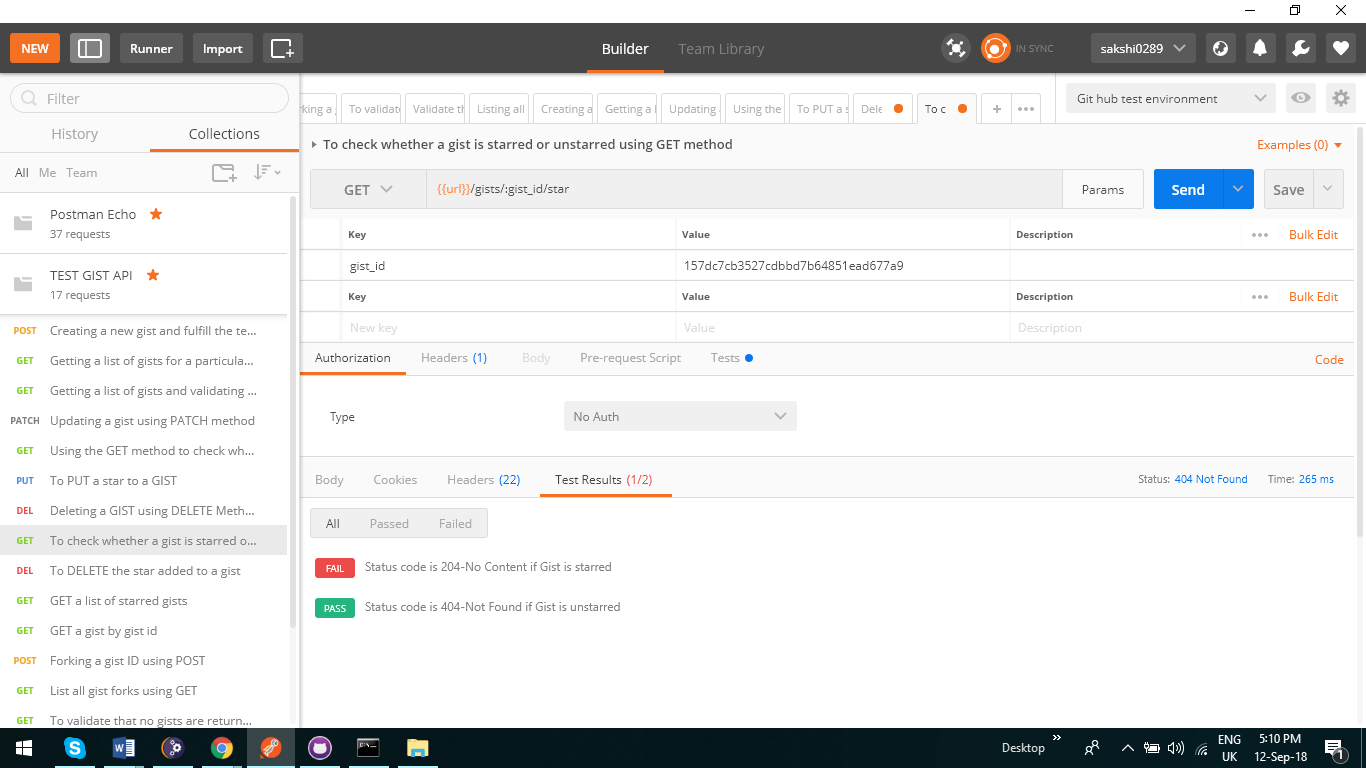
We have added a test script that If a gist is starred then we get 204, and if gist is unstarred we get 404. Adding screenshot of both



**The first one is a pass which states that the gist is starred as code is 204.**

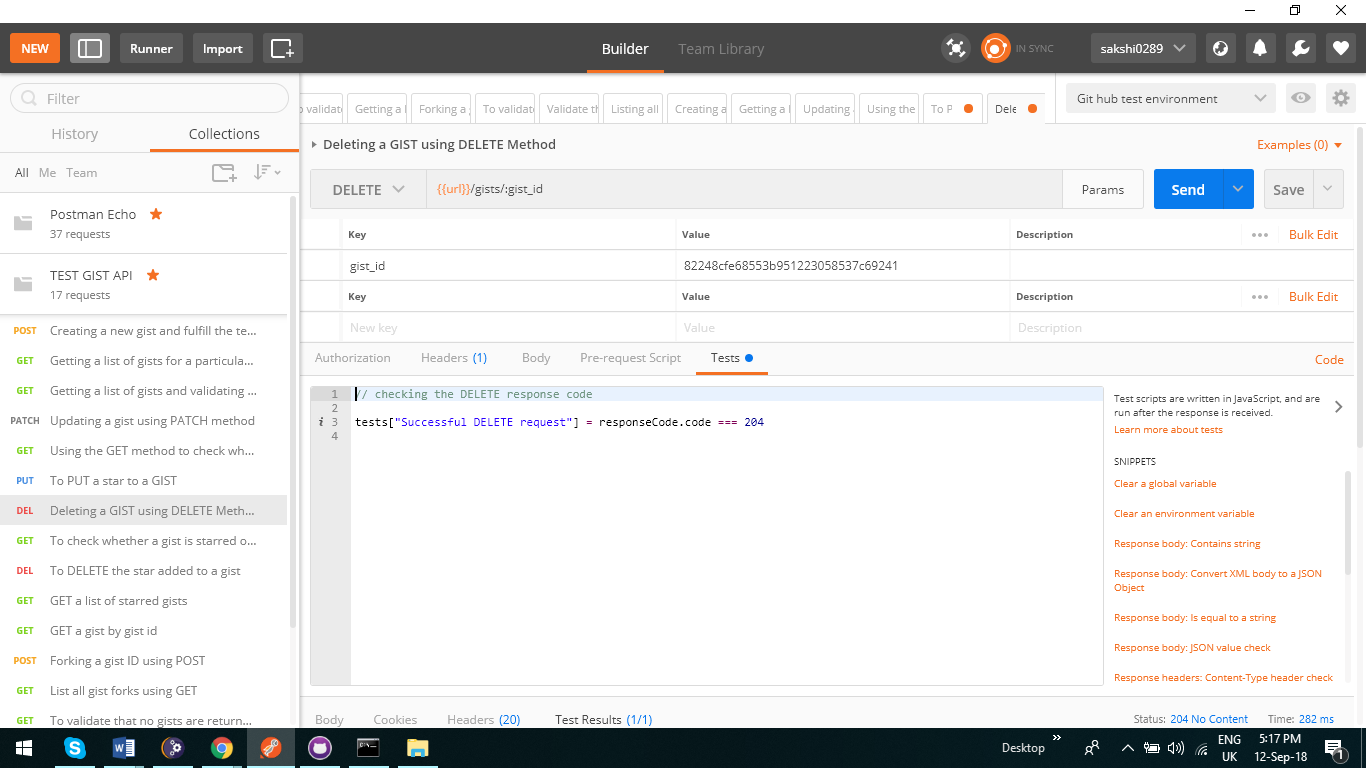


**The second states that it is an unstarred gist as status is 404.**

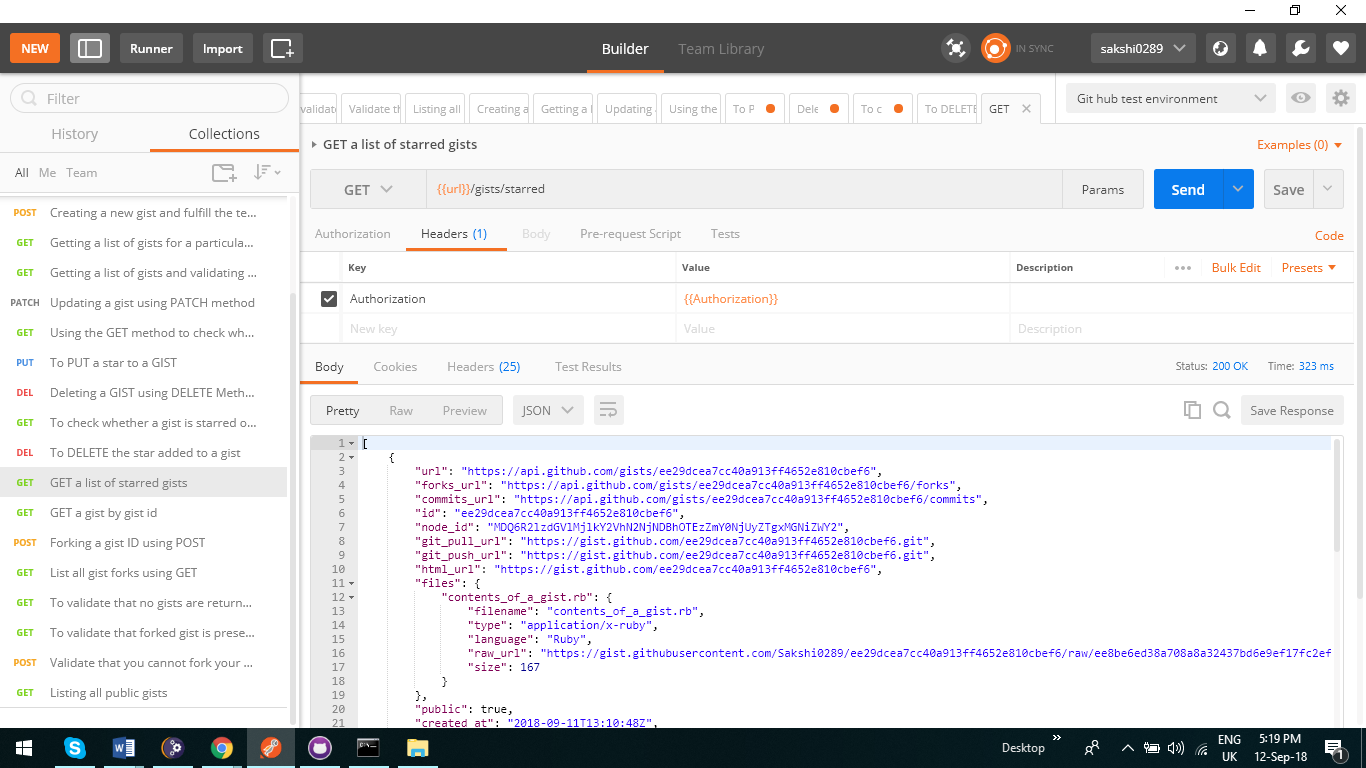


1. **To DELETE the star added to a gist**

As the name suggests it deletes the star added to the gist.

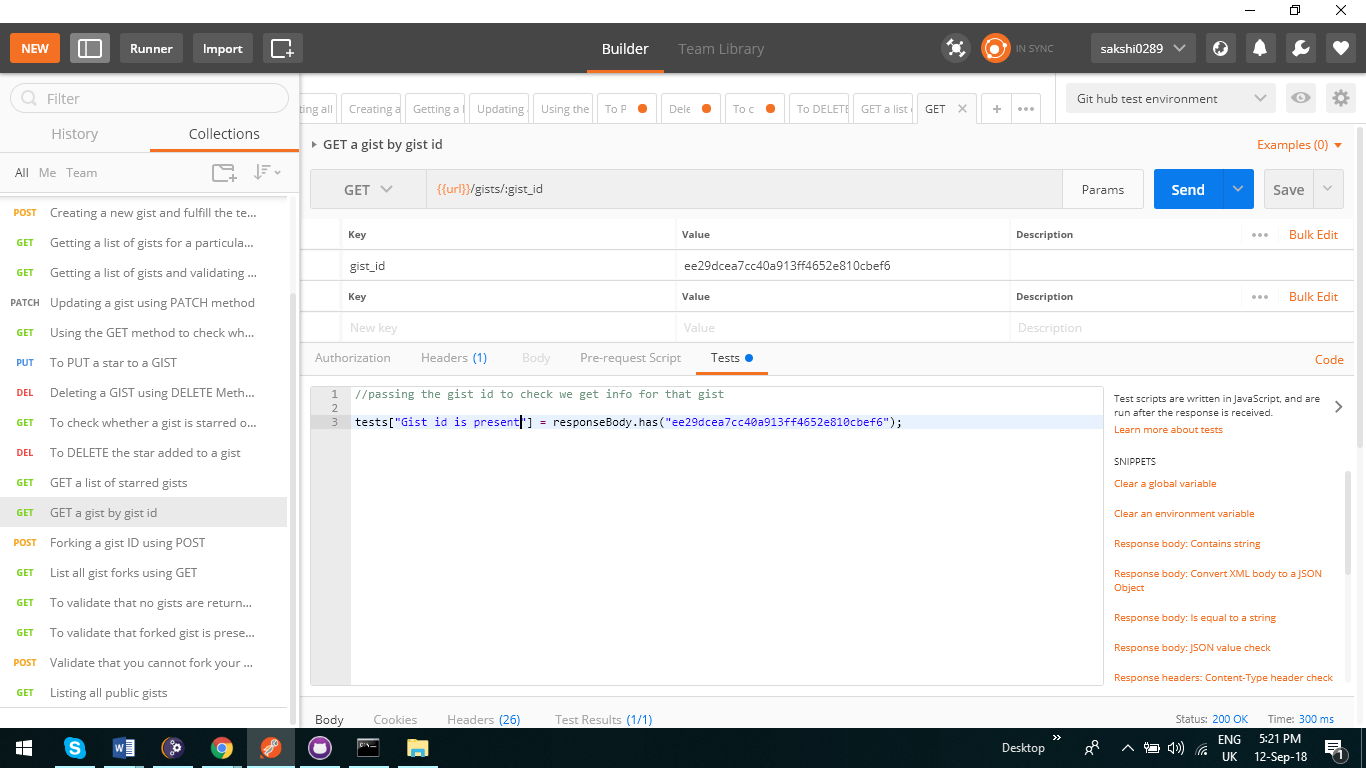


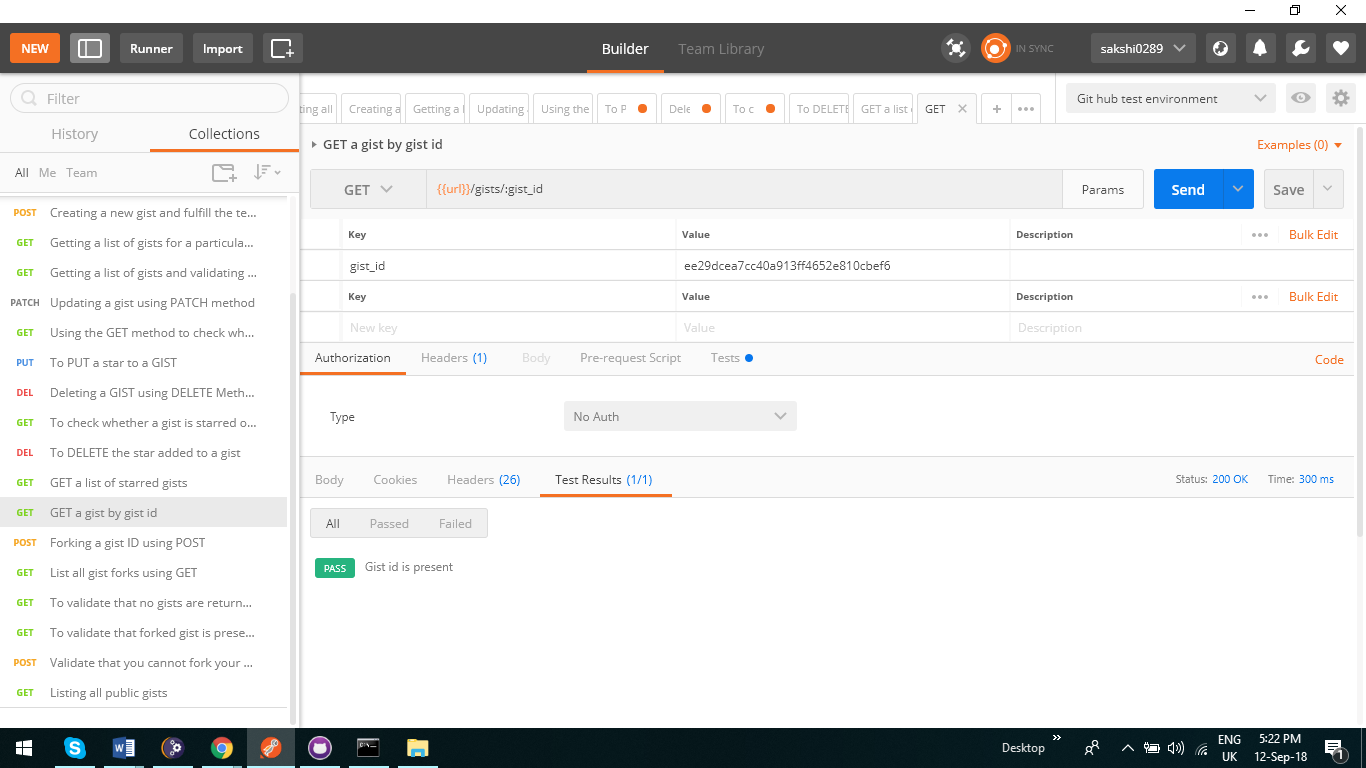
1. **GET a list of starred gists**



1. **GET a gist by gist id**

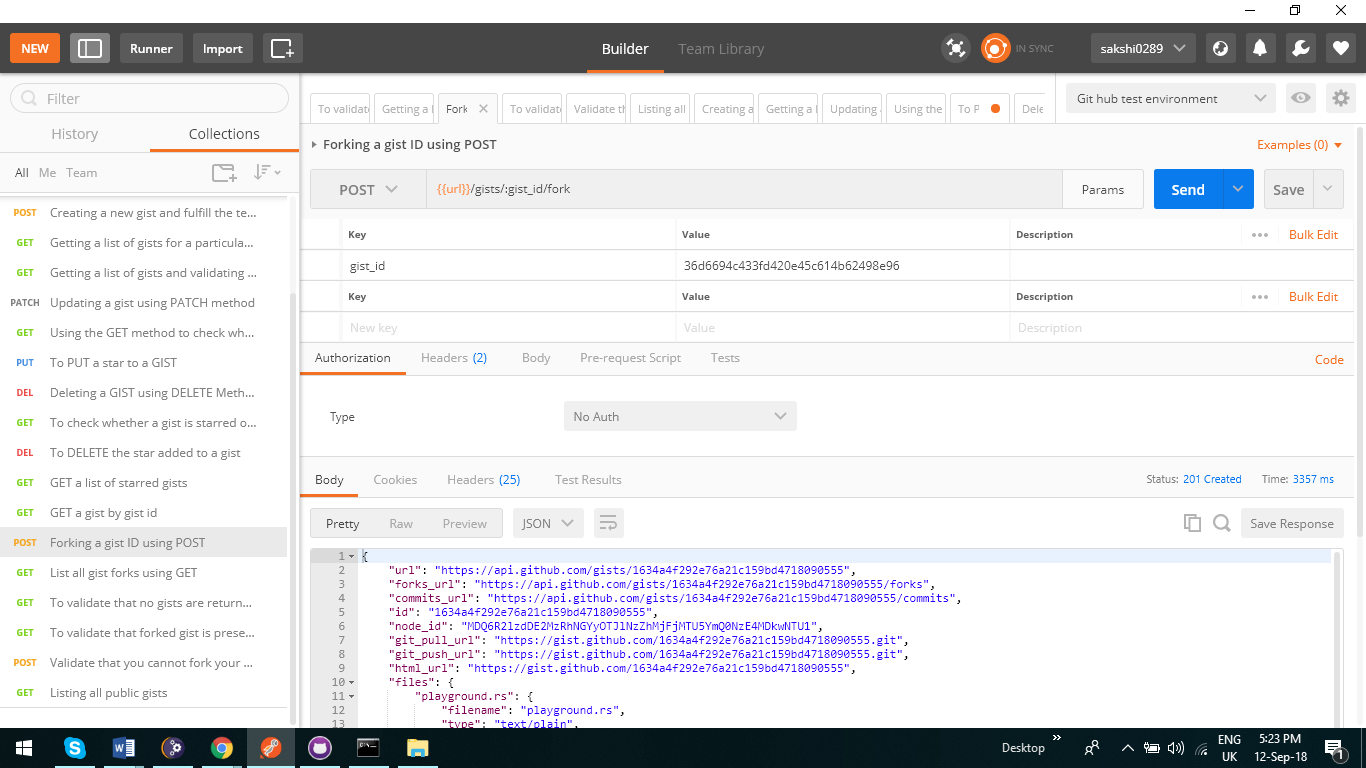
**To get a gist description by passing the gist id and validating that gist id is present in the response by test script.**





1. **Forking a GIST using POST**

Validating that a gist from another repository is forked.



1. **List all gist forks using GET**

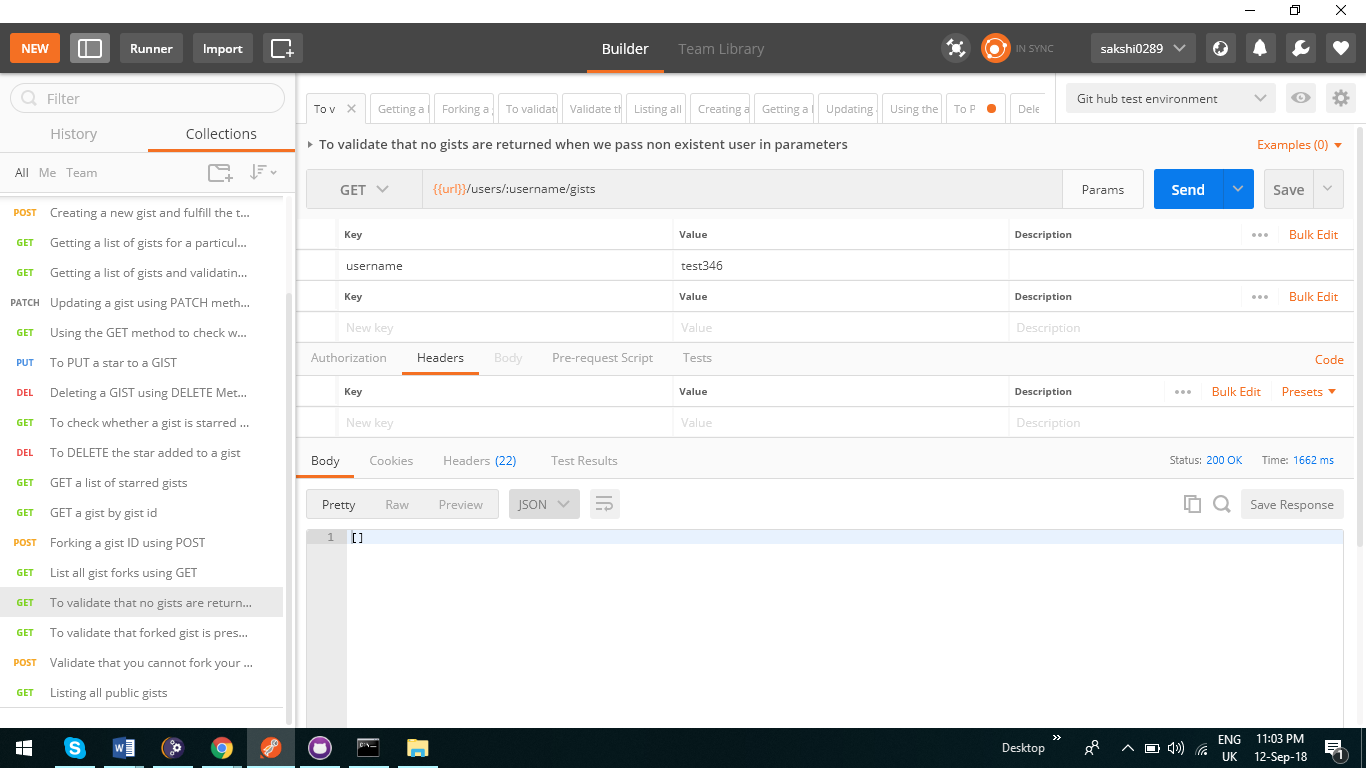
**Listing all gists that have been forked. We should be able to see the gist that has been forked from the above testcase.**

**I have created a test script where I have mentioned the string of the forked gist. It should to successfully test that the gist has been forked.**



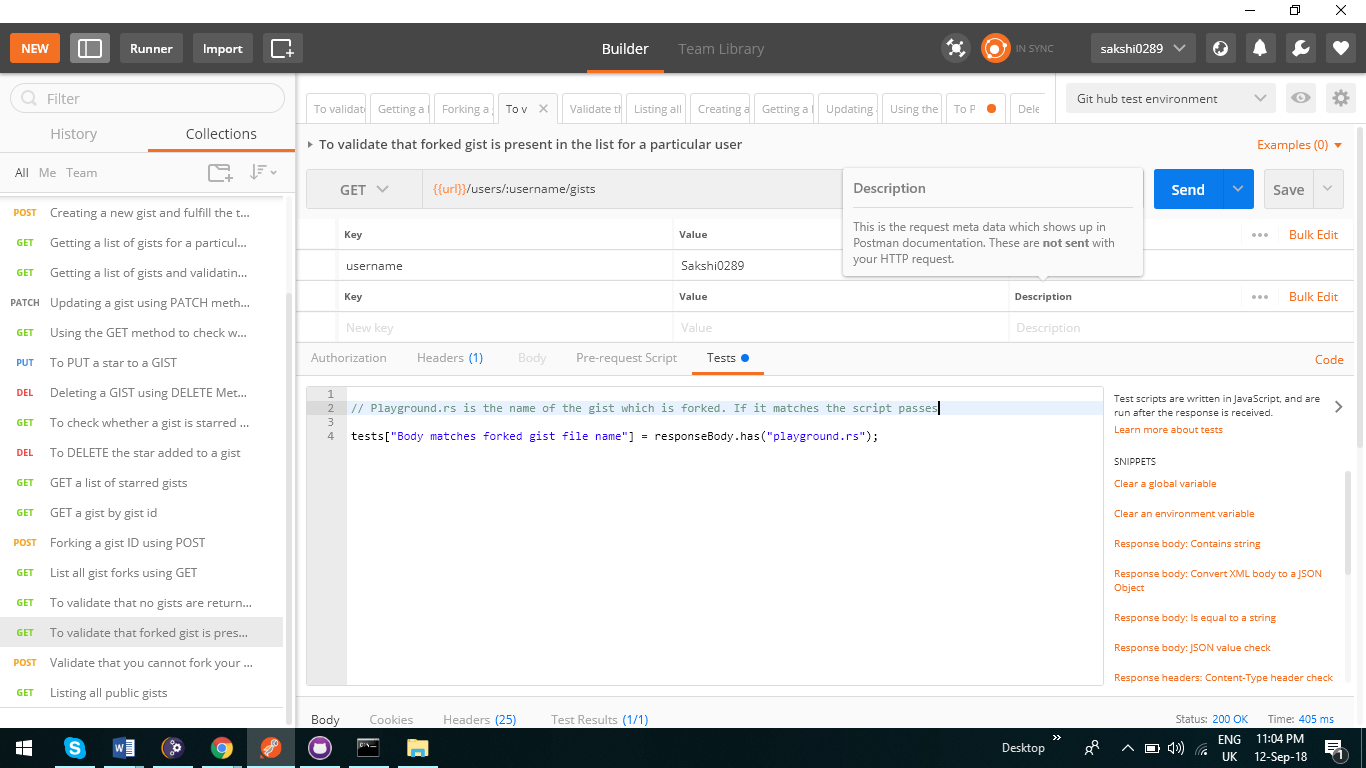
1. **To validate that no gists are returned when we pass nonexistent user in parameters**

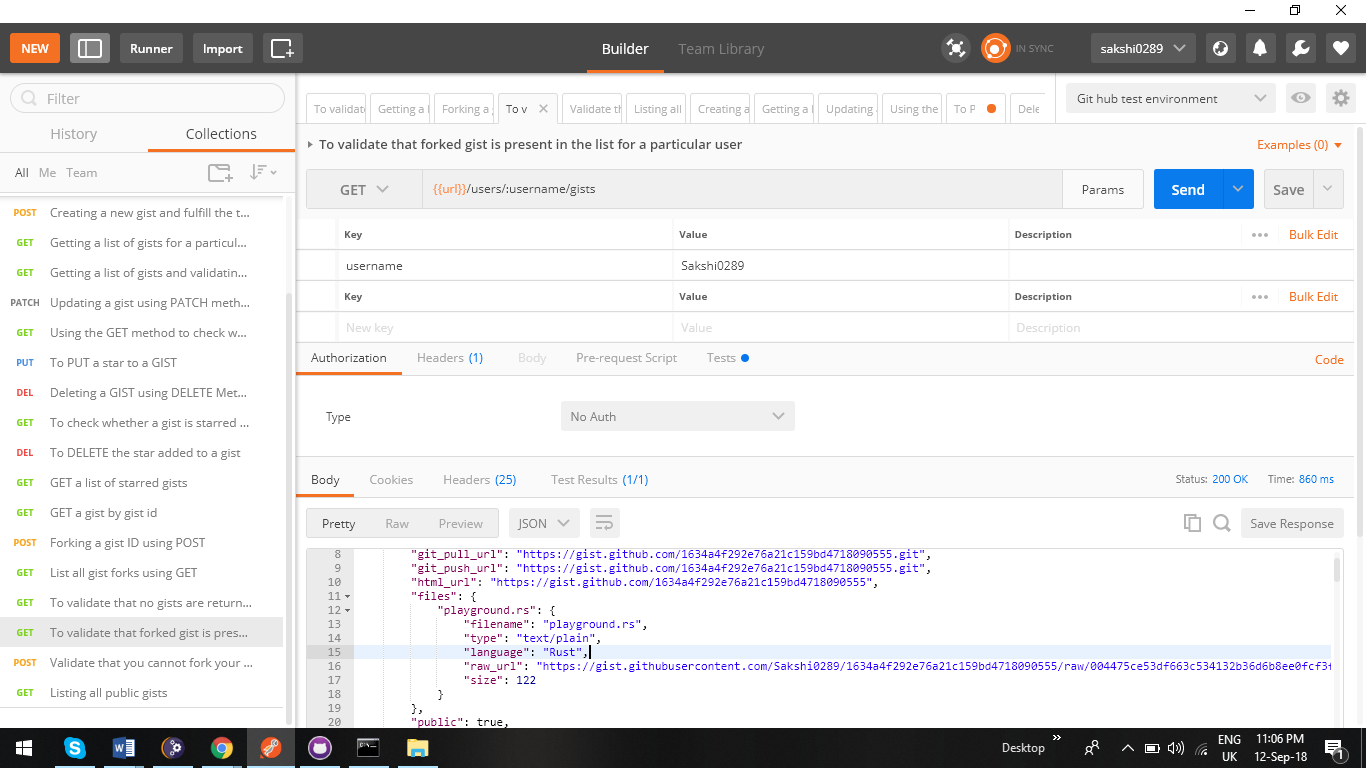
Here we pass a username as test 346 which is not registered in git hub and hence no gists are present

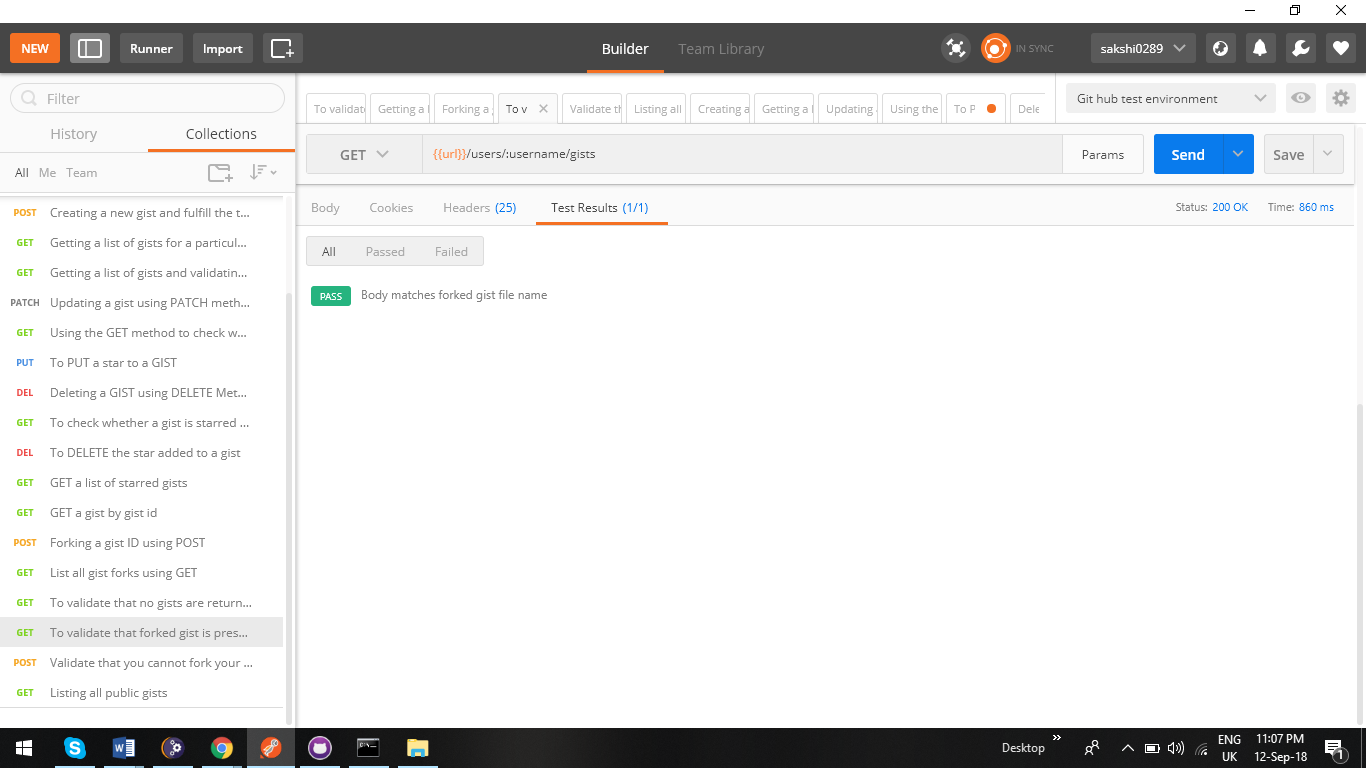


1. **To validate that forked gist is present in the list for a particular user**

**Here the gist we have forked above should be present. We have validated that point by putting the name of the fie in test script and validating it. If the response body contains the forked file name it means the gist have been forked successfully**

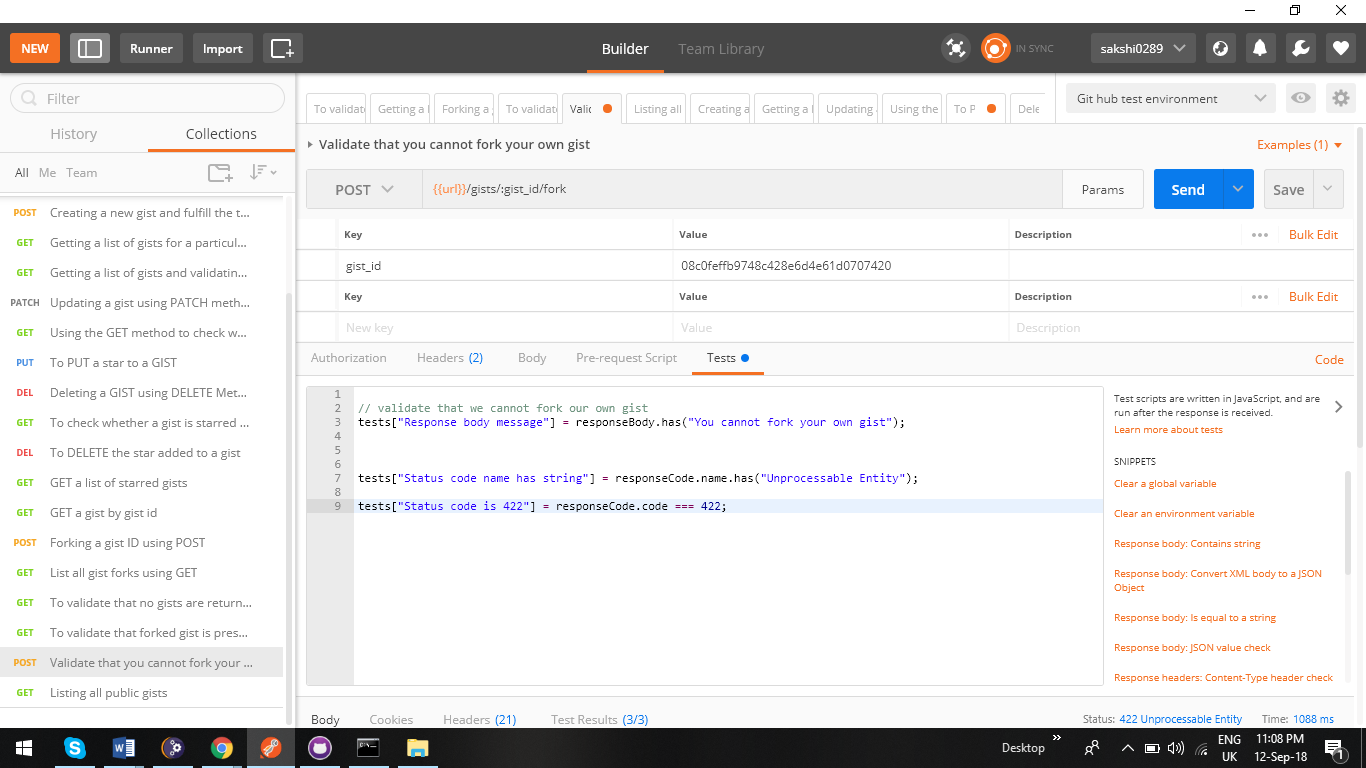


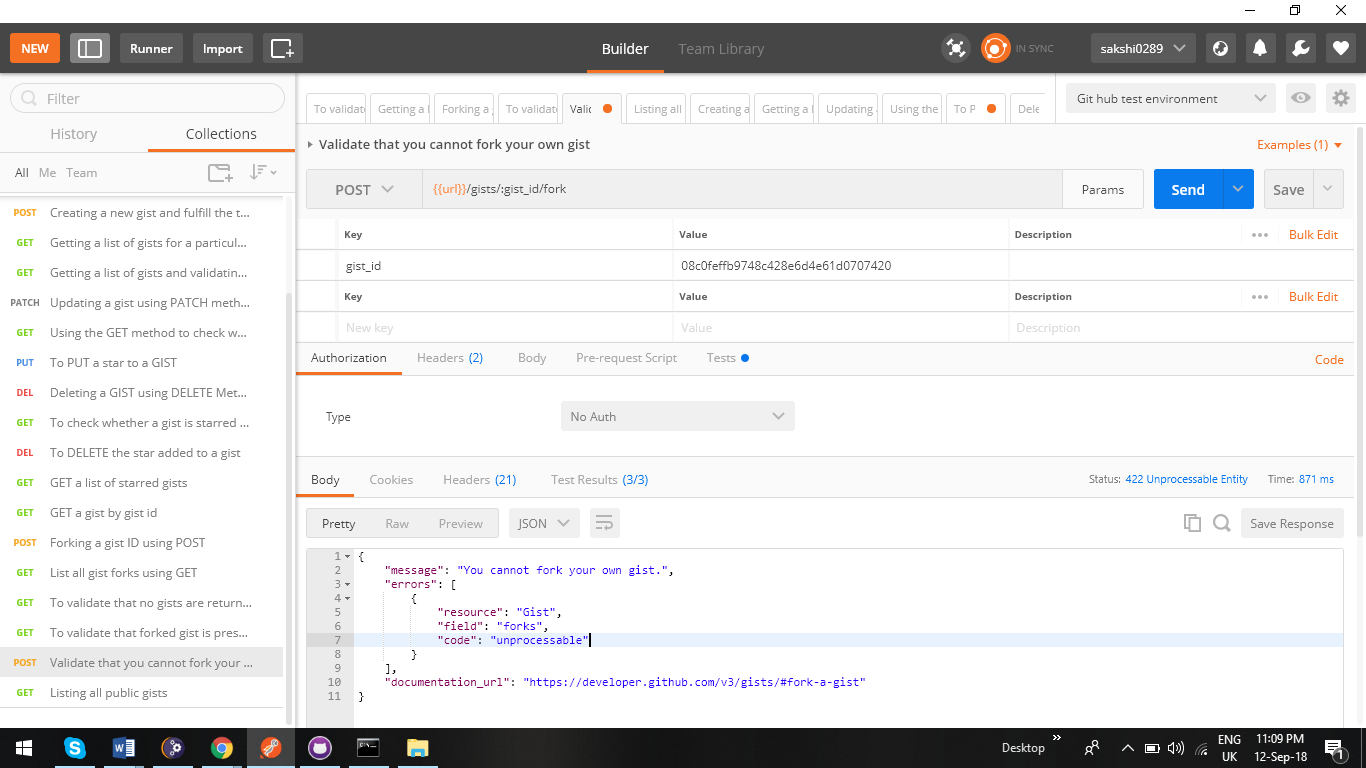


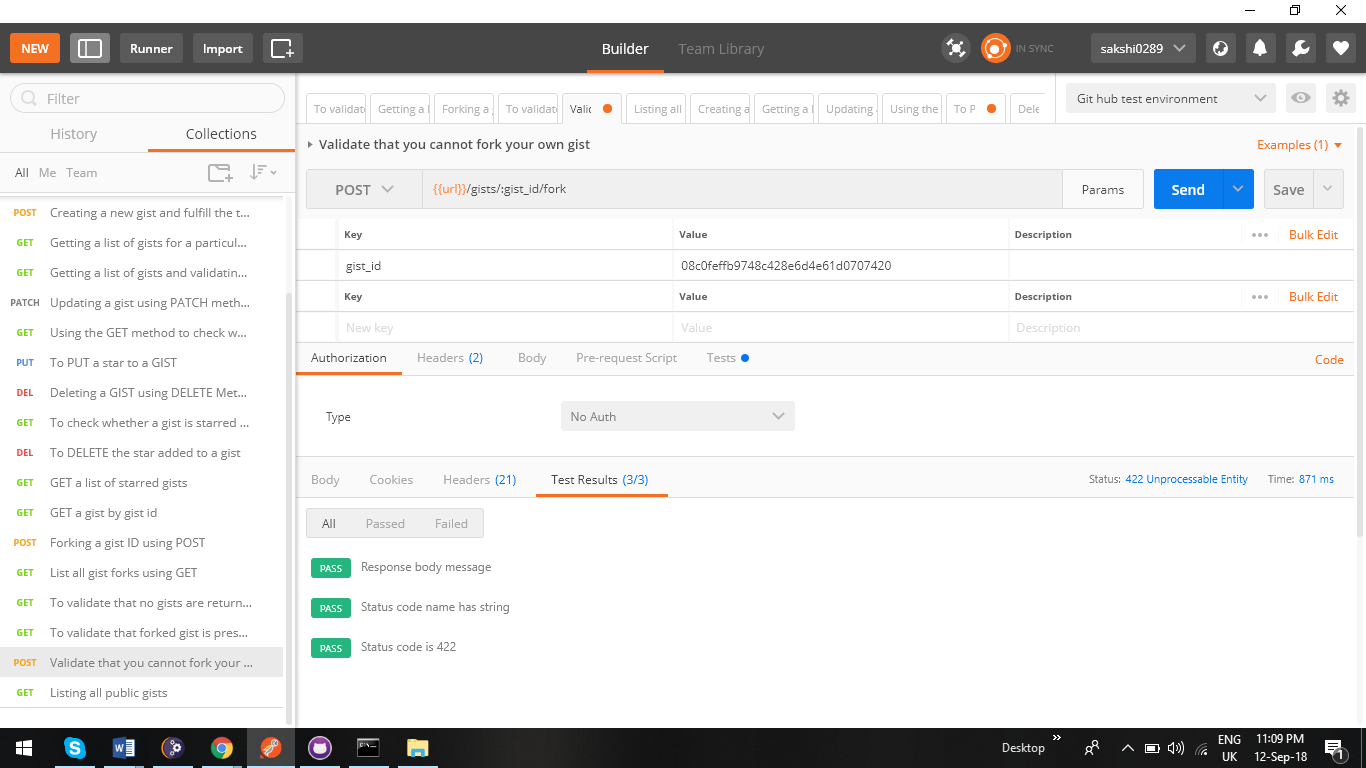


**16.Validate that you cannot fork your own gist**

We have written a test script which validates that we cannot fork our own gist. Response code should be 422 and message should contain “You cannot fork your own gist”

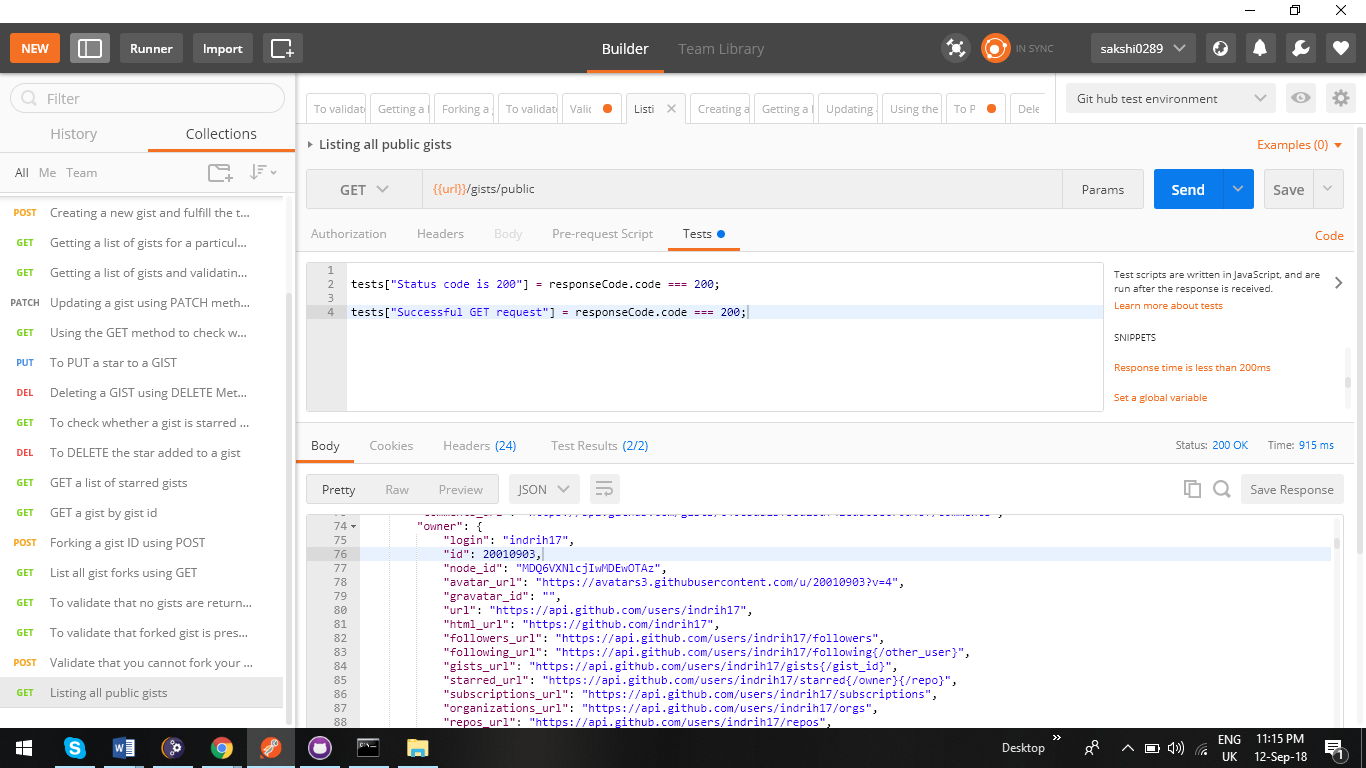






1. **Listing all public gists**

As the name suggests it means listing all public gists. It does not require authorization as the name suggests



**18 Editing a gist by deleting a file already present and adding a new file**

Editing a gist by deleting a file already present and adding a new file on behalf by passing the gist id and details in the body using PATCH method

Validating by adding a test script of the file newly added and ensuring response body contains the newly added string.

