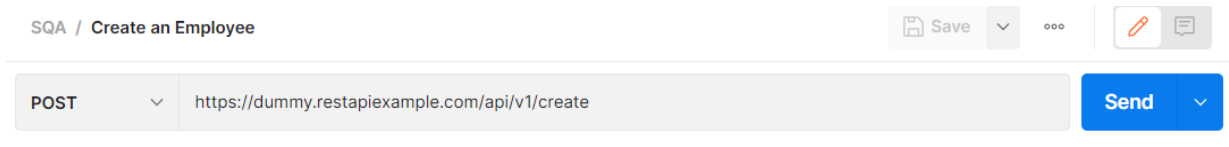


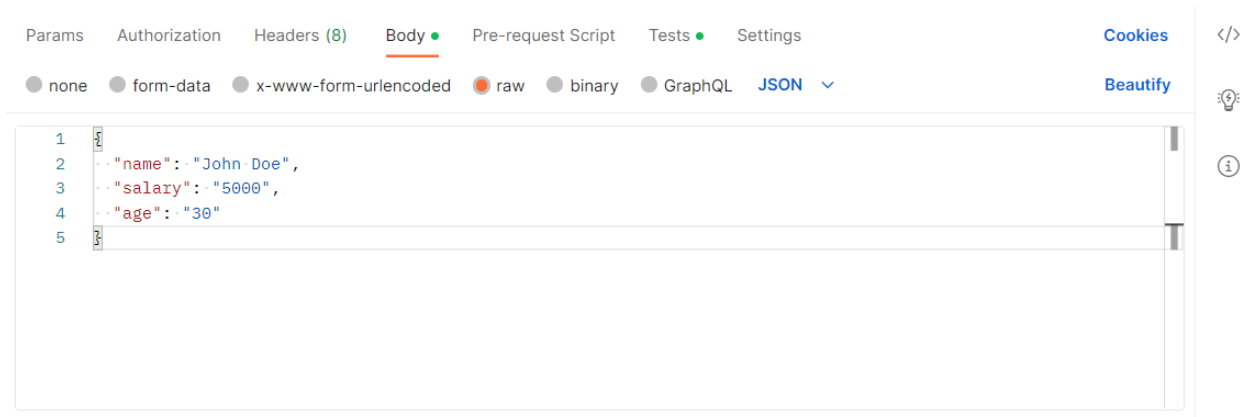
Test Case one

Goal/Purpose: Test if an employee can be successfully created with valid input data.

Request Type & URL:



Request Body:



Assertions(“tests”):

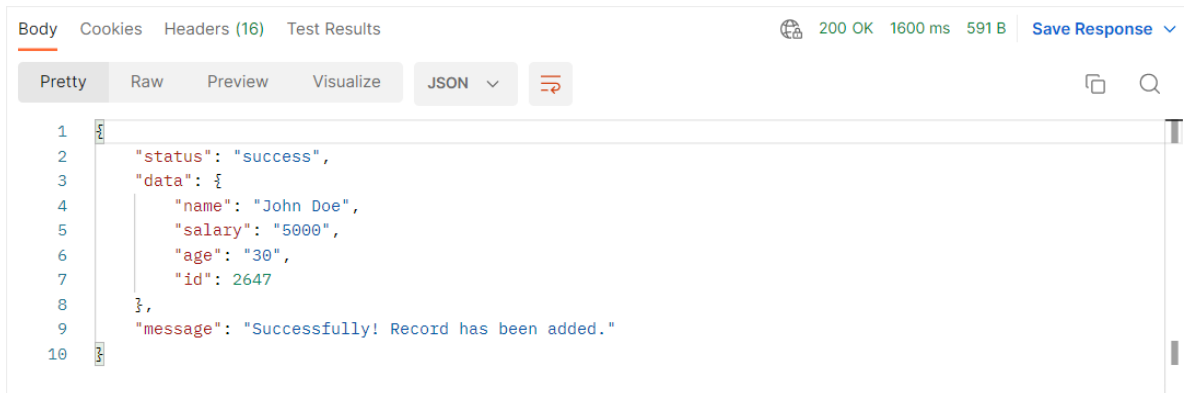
- Verify that the status code is 200.
- Verify that the response body is in JSON format.
- Verify that the response body contains the created employee data with the correct name, salary, and age values.



Test Explanations:

1. This assertion checks that the API call returns a status code of 200, indicating a successful response.
2. This assertion checks that the response body from the API is in JSON format, which is what we expect from this API.
3. These assertions check that the response body contains the data for the newly created employee with the correct name.
4. These assertions check that the response body contains the data for the newly created employee with the correct salary.
5. These assertions check that the response body contains the data for the newly created employee with the correct age values.

Response Body:

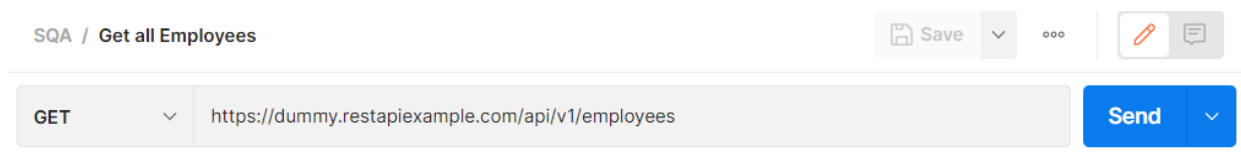


```
1 {
2   "status": "success",
3   "data": {
4     "name": "John Doe",
5     "salary": "5000",
6     "age": "30",
7     "id": 2647
8   },
9   "message": "Successfully! Record has been added."
10 }
```

Test Case Two

Goal/Purpose: Test if all employees can be successfully retrieved.

Request Type & URL:



Request Body:

We don't need to include a request body as we are sending a GET request to retrieve all employees.

Assertions(“tests”):

- Verify that the status code is 200.
- Verify that the response body is in JSON format.
- Verify that the response body contains an array of all employees.

The screenshot shows the 'Tests' tab in Postman. The left pane contains three lines of JavaScript code for assertions:

```
1 pm.response.to.have.status(200);
2 pm.response.to.have.jsonBody();
3 pm.expect(pm.response.json().data).to.be.an('array');
```

The right pane provides information about test scripts, stating they are written in JavaScript and run after the response is received. It also includes a 'Snippets' section with links to 'Get an environment variable', 'Get a global variable', 'Get a variable', 'Get a collection variable', and 'Set an environment variable'.

Test Explanations:

1. The first assertion checks that the status code of the response is 200, indicating that the request was successful.
2. This assertion checks that the response body from the API is in JSON format, which is what we expect from this API.
3. The third assertion checks that the "data" field in the response contains an array of employee objects, indicating that all employees were successfully retrieved.

Response Body:

The screenshot shows the 'Body' tab in Postman. The response is a JSON object with the following structure:

```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "employee_name": "Tiger Nixon",
      "employee_salary": 320800,
      "employee_age": 61,
      "profile_image": ""
    }
  ]
}
```

The top bar of the interface shows the status '200 OK', response time '1083 ms', and size '1.08 KB'. There is a 'Save Response' button on the right.

.....

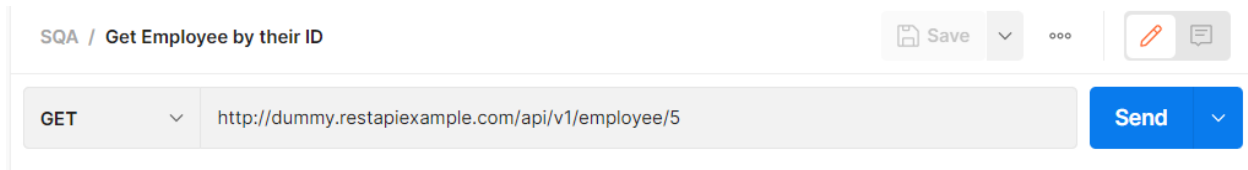
.....



Test Case Three

Goal/Purpose: Test if an employee can be successfully retrieved by their ID.

Request Type & URL:

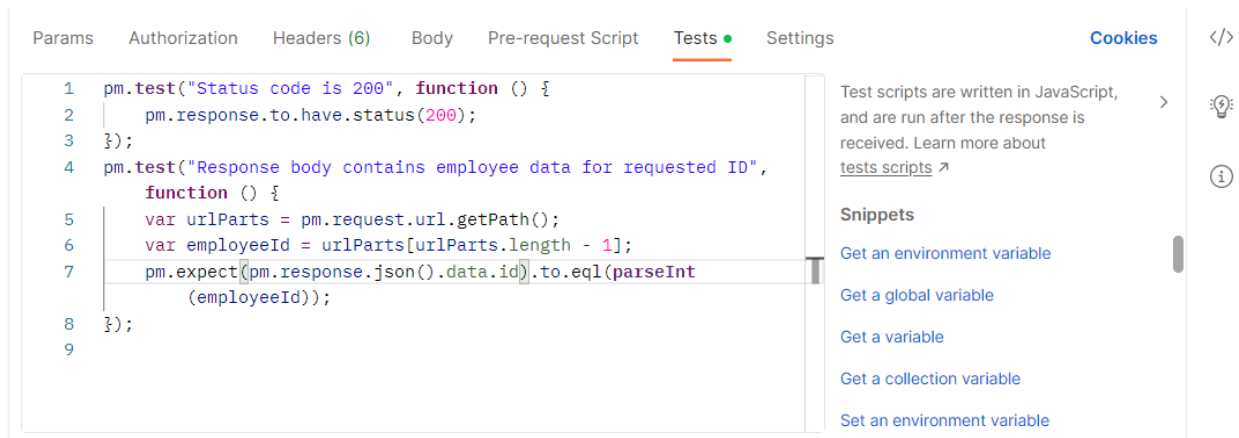


Request Body:

We don't need to include a request body as we are sending a GET request to retrieve all employees.

Assertions(“tests”):

- Verify that the status code is 200.
- Verify that the response body contains the employee data for the requested ID.



Test Explanations:

1. This assertion checks if the HTTP status code returned by the API response is 200. A response with a status code of 200 indicates that the request was successful.
2. This assertion is checking whether the response body contains the correct employee data for the requested ID. First, it uses `pm.request.url.getPath()` to extract the ID of the employee from the URL. The `getPath()` function returns an array of path segments in the URL, so `urlParts[urlParts.length - 1]` is used to get the last segment of the URL which is the employee ID. Then, it checks that the `id` property of the employee object in the response body matches the extracted employee ID using `pm.expect(pm.response.json().data.id).to.eql(parseInt(employeeId))`. If the employee ID in the URL matches the `id` property of the employee object in the response body, then the test case will pass.

Response Body:

The screenshot shows a REST client interface with the following details:

- Body** tab is selected.
- Test Results (2/2)** is shown in the top right.
- 200 OK** status, **884 ms** time, and **623 B** size are displayed.
- Save Response** button is available.
- Pretty** view is selected for the response body.
- The response body is a JSON object:

```
1 {
2   "status": "success",
3   "data": {
4     "id": 5,
5     "employee_name": "Airi Satou",
6     "employee_salary": 162700,
7     "employee_age": 33,
8     "profile_image": ""
9   },
10  "message": "Successfully! Record has been fetched."
11 }
```

Test Result:

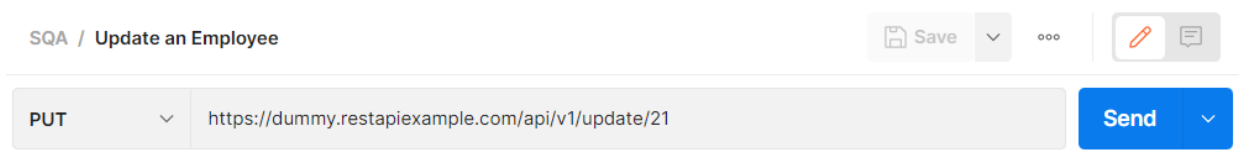
The screenshot shows the **Test Results (2/2)** section with the following details:

- Test Results (2/2)** is selected in the top bar.
- 200 OK** status, **884 ms** time, and **623 B** size are displayed.
- Save Response** button is available.
- All** filter is selected.
- Two test results are shown, both with a **PASS** status:
 - PASS** Status code is 200
 - PASS** Response body contains employee data for requested ID

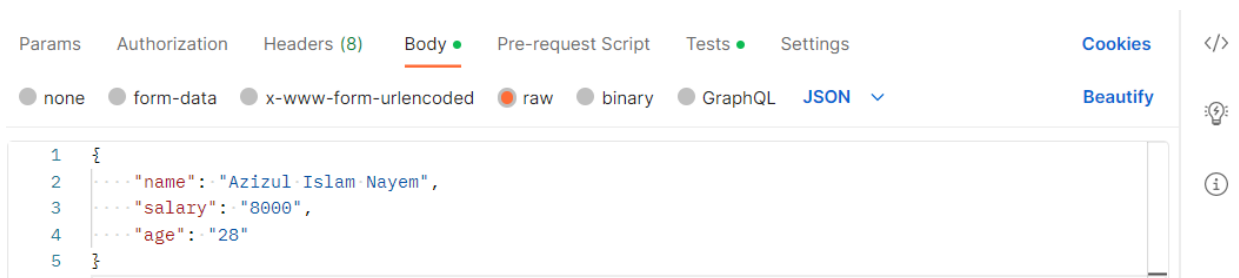
Test Case Four

Goal/Purpose: Test if an employee can be successfully updated with valid input data.

Request Type & URL:

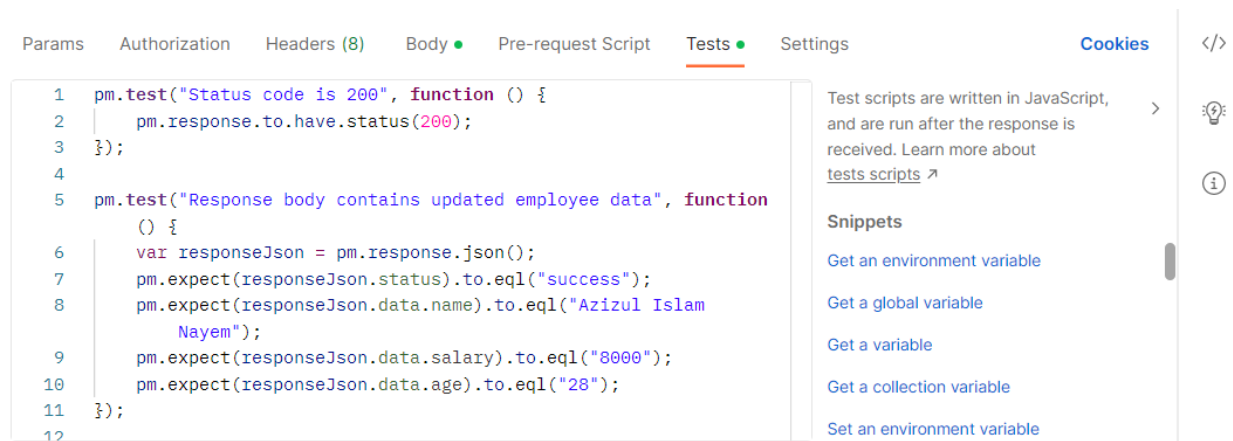


Request Body:



Assertions(“tests”):

- Verify that the status code is 200.
- Verify that the response body contains the updated employee data.

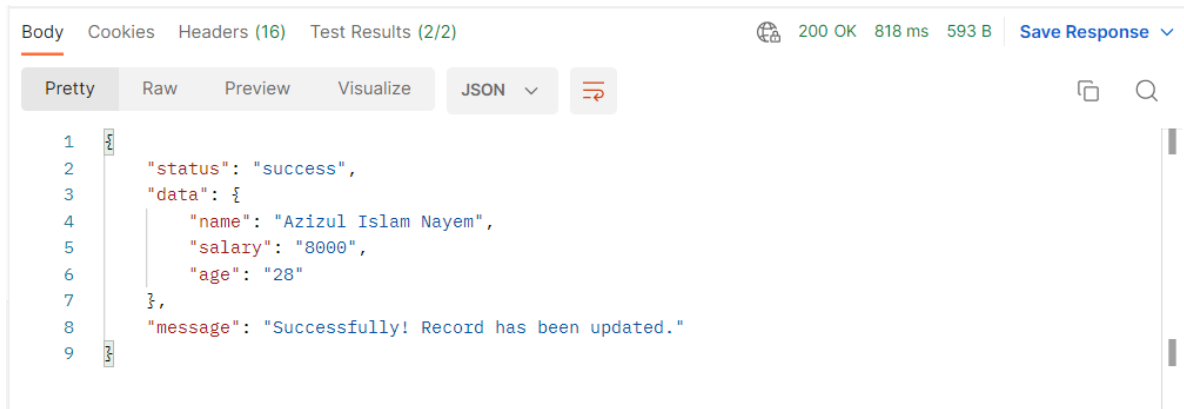


Test Explanations:

1. This assertion checks if the HTTP status code returned by the API response is 200. A response with a status code of 200 indicates that the request was successful.

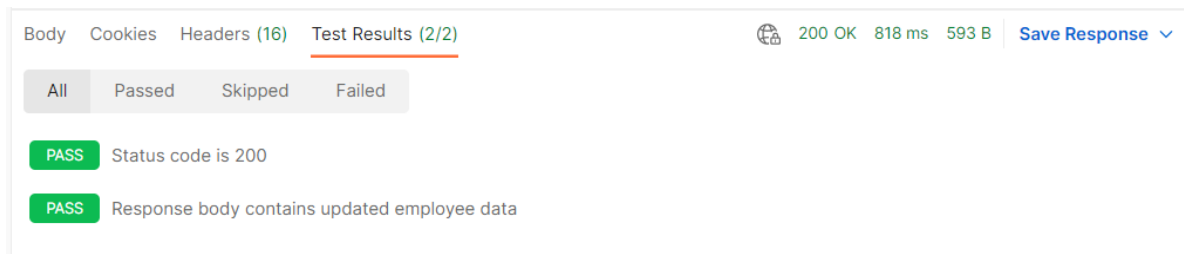
-
2. This assertion is testing if the response body contains the updated employee data. It does this by accessing the response body as a JSON object using `pm.response.json()`, and then verifying that the status field of the data object is equal to "success". It then checks that the name, salary, and age fields of the data object match the expected values of "Azizul Islam Nayem", "8000", and "28" respectively using the `to.eql` method. If all the assertions pass, it means that the employee update was successful and the response body contains the expected updated employee data.

Response Body:



```
1  {
2    "status": "success",
3    "data": {
4      "name": "Azizul Islam Nayem",
5      "salary": "8000",
6      "age": "28"
7    },
8    "message": "Successfully! Record has been updated."
9  }
```

Test Result:

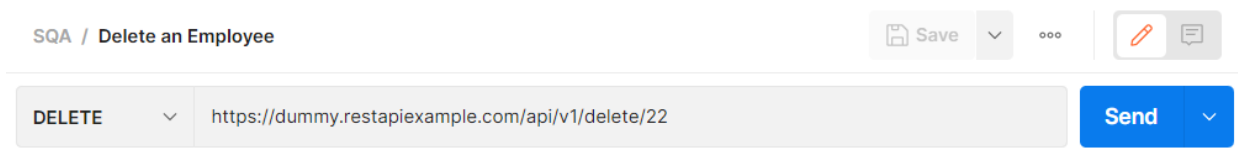


All	Passed	Skipped	Failed
PASS	Status code is 200		
PASS	Response body contains updated employee data		

Test Case Five

Goal/Purpose: Test if an employee can be successfully deleted by their ID.

Request Type & URL:



SQA / Delete an Employee

DELETE ▼ <https://dummy.restapiexample.com/api/v1/delete/22> Send ▼

Request Body:

In the case of the DELETE request type, the employee ID to be deleted is usually passed as a parameter in the URL and does not require a request body. Therefore, the request body is not needed for this test case.

Assertions(“tests”):

- Verify that the status code is 200.
- Verify that the response body contains a success message.

The screenshot shows the 'Tests' tab in a REST client interface. The left pane contains two JavaScript test scripts:

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });  
4 pm.test("Response body contains a success message", function () {  
5   var responseJson = pm.response.json();  
6   pm.expect(responseJson.status).to.eql("success");  
7   pm.expect(responseJson.message).to.eql("Successfully! Record  
   has been deleted");  
8 });  
9
```

The right pane provides information about test scripts, including a note that they are written in JavaScript and run after the response is received. It also includes a 'Snippets' section with links to get or set environment, global, and collection variables.

Test Explanations:

1. This assertion checks if the HTTP status code returned by the API response is 200. A response with a status code of 200 indicates that the request was successful.
2. This assertion is checking if the response body contains a success message after deleting an employee. The `pm.response.json()` function parses the response body into a JSON object, which is then used to access the status and message properties of the response. The first assertion `pm.expect(responseJson.status).to.eql("success");` verifies that the value of the status property is "success". The second assertion `pm.expect(responseJson.message).to.eql("Successfully! Record has been deleted");` verifies that the value of the message property is "Successfully! Record has been deleted". Together, these assertions check that the response body contains a success message after deleting an employee.

Response Body:

The screenshot shows the 'Body' tab in a REST client interface. The response is displayed in JSON format:

```
1 {  
2   "status": "success",  
3   "data": "22",  
4   "message": "Successfully! Record has been deleted"  
5 }
```

The interface also shows the status '200 OK', the time '2.90 s', and the size '551 B'. There are tabs for 'Body', 'Cookies', 'Headers (16)', and 'Test Results (2/2)'. The 'Body' tab is currently selected, and the response is shown in a 'Pretty' format.

Test Result:

The screenshot shows the 'Test Results' tab of a REST client. At the top, it displays '200 OK', '2.90 s', '551 B', and a 'Save Response' button. Below this, there are four filter buttons: 'All', 'Passed', 'Skipped', and 'Failed'. The 'All' button is selected. Two test results are listed, both marked as 'PASS' in green boxes:

- PASS Status code is 200
- PASS Response body contains a success message

Test Case Six

Goal/Purpose: Test the response time for retrieving all employees from the API.

Request Type & URL:

The screenshot shows the REST client interface for a GET request. The top bar displays 'SQA / Measure response time for getting all employees' and a 'Save' button. The request configuration bar shows the method 'GET' and the URL 'https://dummy.restapiexample.com/api/v1/employees'. A 'Send' button is visible on the right.

Request Body:

We don't need to include a request body as we are sending a GET request to retrieve all employees.

Assertions("tests"):

- Verify that the response time is less than 1000ms.
- Verify that the status code is 200.
- Verify that the response body contains an array of all employees.

The screenshot shows the 'Tests' tab of the REST client. The left pane contains the following JavaScript test scripts:

```
1 pm.test("Response time is less than 1000ms", function () {
2   pm.expect(pm.response.responseTime).to.be.below(1000);
3 });
4
5 pm.test("Status code is 200", function () {
6   pm.response.to.have.status(200);
7 });
8
9 pm.test("Response body contains an array of all employees",
10   function () {
11     var responseJson = pm.response.json();
12     pm.expect(responseJson.status).to.eql("success");
13     pm.expect(responseJson.data).to.be.an("array").that.is.not.
14       empty;
15   });
```

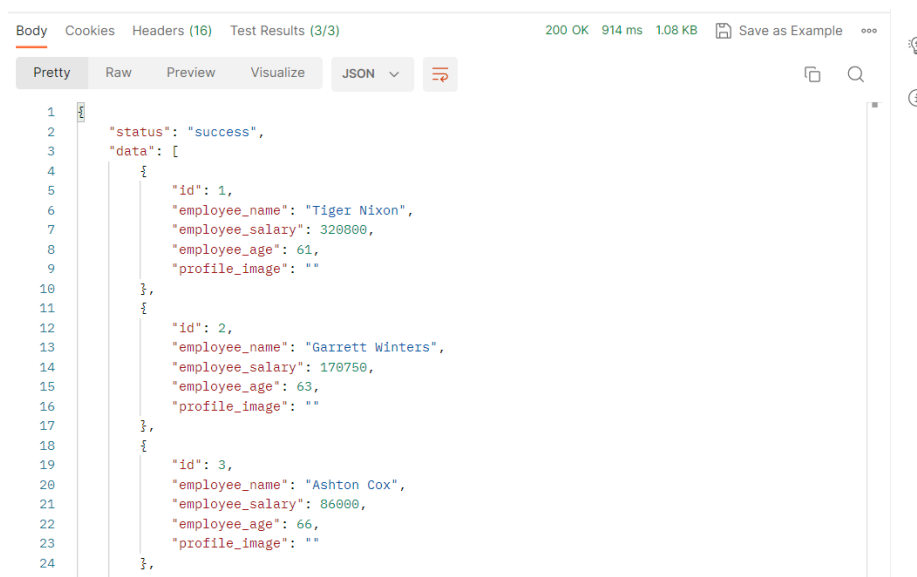
The right pane provides information about test scripts and a list of snippets:

- Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)
- Snippets
 - Get an environment variable
 - Get a global variable
 - Get a variable
 - Get a collection variable
 - Set an environment variable
 - Set a global variable
 - Set a collection variable

Test Explanations:

1. The first assertion is checking if the response time of the API request is less than 1000ms, which is one second. This is to ensure that the API is responding in a timely manner.
2. The second assertion is checking if the status code of the API response is 200, which indicates that the request was successful.
3. The third assertion is checking if the response body of the API contains an array of all employees. It first checks if the "status" field in the response body is equal to "success", and then checks if the "data" field is an array that is not empty. This is to ensure that the API is returning all employees as expected.

Response Body:



```
Body Cookies Headers (16) Test Results (3/3) 200 OK 914 ms 1.08 KB Save as Example
Pretty Raw Preview Visualize JSON
1
2  "status": "success",
3  "data": [
4    {
5      "id": 1,
6      "employee_name": "Tiger Nixon",
7      "employee_salary": 320000,
8      "employee_age": 61,
9      "profile_image": ""
10   },
11   {
12     "id": 2,
13     "employee_name": "Garrett Winters",
14     "employee_salary": 170750,
15     "employee_age": 63,
16     "profile_image": ""
17   },
18   {
19     "id": 3,
20     "employee_name": "Ashton Cox",
21     "employee_salary": 86000,
22     "employee_age": 66,
23     "profile_image": ""
24   },
25 ]
```

```
152   "id": 22,
153   "employee_name": "Yuri Berry",
154   "employee_salary": 675000,
155   "employee_age": 40,
156   "profile_image": ""
157 },
158 {
159   "id": 23,
160   "employee_name": "Caesar Vance",
161   "employee_salary": 106450,
162   "employee_age": 21,
163   "profile_image": ""
164 },
165 {
166   "id": 24,
167   "employee_name": "Doris Wilder",
168   "employee_salary": 85600,
169   "employee_age": 23,
170   "profile_image": ""
171 }
172 ],
173 "message": "Successfully! All records has been fetched."
174 }
```

Test Result:

BodyCookiesHeaders (16)Test Results (3/3)

200 OK914 ms1.08 KB

Save as Example

AllPassedSkippedFailed

PASS

Response time is less than 1000ms

PASS

Status code is 200

PASS

Response body contains an array of all employees

Cookies

Capture requests

Runner

Trash

-----THE END-----