Great! Writing test cases in Postman can be a powerful way to test your API endpoints thoroughly. Here's a detailed guide on various methods and techniques you can use when writing test cases in Postman:

1. **Response Validation with pm.test**:
   * **pm.test** is the primary method for writing test cases in Postman. It allows you to make assertions on various aspects of the response.
   * **Status Code Validation**:

javascriptCopy code

pm.test("Status code should be 200", function () { pm.response.to.have.status(200); });

* + **Response Body Validation**:

javascriptCopy code

pm.test("Response should contain expected data", function () { pm.response.to.have.jsonBody({ key1: "value1", key2: "value2" }); });

* + **Header Validation**:

javascriptCopy code

pm.test("Content-Type header should be application/json", function () { pm.response.to.have.header("Content-Type", "application/json; charset=UTF-8"); });

* + **Response Time Validation**:

javascriptCopy code

pm.test("Response time is less than 500ms", function () { pm.expect(pm.response.responseTime).to.be.below(500); });

1. **Chaining Assertions**:

You can chain multiple assertions in a single test case:

javascriptCopy code

pm.test("Multiple Assertions", function () { pm.response.to.have.status(200); pm.response.to.have.header("Content-Type", "application/json; charset=UTF-8"); pm.response.to.have.jsonBody({ key1: "value1", key2: "value2" }); });

1. **Environment and Global Variables**:

You can use environment and global variables to make your test cases more dynamic. These variables can hold values that change based on the environment or previous requests.

javascriptCopy code

var responseJson = pm.response.json(); // Using environment variable pm.environment.set("userId", responseJson.id); // Using global variable pm.globals.set("token", responseJson.token);

1. **Pre-request Scripts**:

You can write scripts that run before sending the request. This is useful for setting up data or dynamically generating request parameters.

1. **Post-request Scripts**:

Scripts that run after receiving the response. Useful for extracting data from the response or performing additional actions based on the response.

1. **Test Runner**:

Postman provides a test runner that allows you to run collections of requests and monitor the results of your test cases.

1. **Test Data**:

You can use different sets of test data by creating data files (e.g., CSV, JSON) and iterating through them using the Collection Runner.

1. **Dynamic Values**:

Use dynamic variables and functions provided by Postman to create dynamic test cases. For example, **pm.iterationData.get()** can be used to access data during iterations.

1. **Logging**:

You can log information to the Postman console to help with debugging:

javascriptCopy code

console.log("Logging a message");

1. **External Libraries**:

Postman allows you to use external libraries and npm modules to extend your testing capabilities.

1. **Custom Functions**:

You can define your own custom functions and use them in your test scripts to avoid code duplication.

By mastering these techniques and methods, you can create comprehensive and effective test cases in Postman for your Spring Framework-based application. Remember to refer to the Postman documentation for more details and examples on each of these methods and concepts.

meaning and when to use some of these key elements:

1. **pm**:
   * **pm** is a global object in Postman that provides access to various functionalities and data throughout your request lifecycle. It stands for "Postman."
   * You use **pm** to interact with environment variables, global variables, and other Postman-specific features.
   * Example: **pm.environment.set("variableName", "variableValue");**
2. **response**:
   * **response** is an object that represents the response received from the API request.
   * It allows you to access various properties and methods related to the response, such as the response body, status code, headers, and more.
   * Example: **var responseBody = pm.response.text();**
3. **to**:
   * **to** is a property of the **response** object that is often used to start building assertions and validations.
   * It helps you specify what you want to validate about the response.
   * Example: **pm.response.to.have.status(200);**
4. **have**:
   * **have** is a method that follows **to** and is used to specify the expected value or condition for a particular aspect of the response.
   * It is typically used for making assertions about the response, such as checking the status code, headers, or body content.
   * Example: **pm.response.to.have.header("Content-Type", "application/json");**
5. **body**:
   * **body** is a property that allows you to access the body of the response.
   * You can use it to perform validations on the response body, which is often in JSON format.
   * Example: **pm.response.to.have.jsonBody({ key1: "value1" });**

These elements are used together to create structured assertions and validations for your API responses. For example, you might use them as follows:

javascriptCopy code

pm.test("Validate Response", function () {

// Use pm.response to access the response

var response = pm.response;

// Start building assertions using to, have, and other methods

response.to.have.status(200);

response.to.have.header("Content-Type", "application/json; charset=UTF-8"); response.to.have.jsonBody({ key1: "value1", key2: "value2" });

// You can also use response.body to access the response body directly

var responseBody = response.body; // Additional validations and assertions can be performed here });

In this example, we use **pm.response** to access the response, **to** to start building assertions, **have** to specify the expected conditions, and **body** to access the response body for further validations.

These elements allow you to thoroughly test and validate API responses in Postman, ensuring that your API behaves as expected.

The error you're encountering is because you are trying to assert that the entire response body is equal to the string "Roshan," but the response body is actually a JSON array containing an object.

To check if the response body contains the string "Roshan," you should perform a partial string match or search within the response body, which is a JSON array in this case. You can use JavaScript to loop through the array and look for the desired string. Here's how you can modify your test:

javascriptCopy code

// Check if the response body contains the string "Roshan"

pm.test("Response body check for Roshan", function ()

{

// Parse the JSON response body

var responseBody = pm.response.json(); // Boolean flag to check if "Roshan" is found

var foundRoshan = false; // Loop through the array and check if any empName contains "Roshan"

for (var i = 0; i < responseBody.length; i++)

{

if (responseBody[i].empName === "Roshan")

{ foundRoshan = true; break; // Exit the loop if "Roshan" is found

}

} // Assertion based on whether "Roshan" was found pm.expect(foundRoshan).to.be.true; });

In this modified test case:

1. We first parse the JSON response body using **pm.response.json()** to work with it as a JavaScript array.
2. We then use a **for** loop to iterate through the array and check if any **empName** property equals "Roshan."
3. If "Roshan" is found in any **empName**, we set the **foundRoshan** flag to **true**.
4. Finally, we use **pm.expect** to assert that **foundRoshan** is **true**, indicating that "Roshan" was found in the response body.

This approach allows you to check if the string "Roshan" is present within the response body without requiring an exact match of the entire body.

Top of Form