

## Fetch Single Product By invalid Id:

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for 'File', 'Edit', 'View', and 'Help'. Below the navigation bar, there are buttons for 'New', 'Import', 'Runner', and a dropdown menu. The title bar displays 'Project 3' and three tabs: 'GET Fetch Single Product by Id', 'GET Login with valid credentials', and 'GET Fetch Single Product by invalid...'. The current tab is 'GET Fetch Single Product by invalid...'. The status bar indicates 'No Environment'.

In the main workspace, there is a 'Tests' tab selected. A code snippet is shown:

```
1 pm.test("Fetch Single Product by invalid Id", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.success).to.eql(false);  
4     pm.expect(jsonData.products).to.eql(null);  
5});
```

Below the code, the 'Body' tab is selected, showing the response body in JSON format:

```
1 {  
2     "success": false,  
3     "products": null  
4 }
```

The status bar at the bottom right shows 'Status: 200 OK', 'Time: 50 ms', 'Size: 207 B', and a 'Save Response' button.

## Fetch Single Product By Id

The screenshot shows the Postman application interface. The layout is identical to the previous screenshot, with the 'File', 'Edit', 'View', and 'Help' tabs at the top. The title bar displays 'Project 3' and three tabs: 'GET Fetch Single Product by Id', 'GET Login with valid credentials', and 'GET Fetch Single Product by invalid...'. The current tab is 'GET Fetch Single Product by Id'.

In the main workspace, the 'Tests' tab is selected. A code snippet is shown:

```
1 pm.test("Fetch Single Product by Id", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.success).to.eql(true) && pm.expect(jsonData.products).to.not.eql(null);  
4});
```

The status bar at the bottom right shows 'Status: 200 OK', 'Time: 49 ms', 'Size: 36.85 KB', and a 'Save Response' button.

## Login with Valid Credentials:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'History', 'Collections' (which is selected), and 'APIs'. Under 'Collections', there are two items: 'E-Commerce' (36 requests) and 'Project3Collection' (44 requests). In the main panel, a collection named 'Login with valid credentials' is expanded. It contains a single GET request with the URL `http://localhost:55808/api/users?email=admin@admin.com&password=admin`. The 'Tests' tab is selected, displaying the following JavaScript code:

```
1 pm.test("LoggedIn Successfully", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.userId).to.eql(16);  
4     pm.expect(jsonData.activated).to.eql(true);  
5 });
```

Below the code, the 'Test Results' section shows one test case labeled 'Loggedin Successfully' with a status of 'PASS'. At the bottom right of the main panel, it says 'Status: 200 OK Time: 24 ms Size: 549 B Save Response'. The bottom of the screen shows the Windows taskbar with various pinned icons.

## Login with invalid credentials:

This screenshot shows the same Postman interface as the previous one, but with a different collection. The 'Project3Collection' is selected in the sidebar. A GET request titled 'Login with invalid credentials/ User not found' is shown with the URL `http://localhost:55808/api/users?email=sandeep@gmail.com&password=Test12`. The 'Tests' tab is selected again, with the following test script:

```
1 pm.test("Invalid Details", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.title).to.eql("Not Found");  
4     pm.expect(jsonData.status).to.eql(404);  
5 });
```

The 'Test Results' section shows one test case labeled 'Invalid Details' with a status of 'PASS'. The status bar at the bottom indicates 'Status: 404 Not Found Time: 47 ms Size: 323 B Save Response'. The Windows taskbar is visible at the bottom.

## Invalid login request method:

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Upgrade'. A 'Project 3' dropdown is open. The main workspace displays a POST request titled 'Invalid Request Method POST' to 'http://localhost:55808/api/users?email=sandeep@gmail.com&password=Test123'. The 'Tests' tab is selected, containing the following JavaScript code:

```
1 pm.test("Invalid Request Method", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.title).to.eql("Unsupported Media Type");
4     pm.expect(jsonData.status).to.eql(415);
5});
```

The 'Body' tab shows a JSON response with status 415 and type 'Unsupported Media Type'. The bottom status bar indicates 'Status: 415 Unsupported Media Type'.

## Register with new valid body:

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Upgrade'. A 'Project 3' dropdown is open. The main workspace displays a POST request titled 'Register with new valid body' to 'http://localhost:55808/api/users'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2     "Email": "ashaadode@gmail.com",
3     "Name": "Ashaad Aisha",
4     "DateOfBirth": "1993-10-07",
5     "Phone": "2265057573",
6     "Activated": true,
7     "Password": "Test123"
8 }
```

The 'Body' tab shows a JSON response with status 200 OK and message 'Register New User with valid body'. The bottom status bar indicates 'Status: 200 OK'.

## Fail to register existing user:

The screenshot shows the Postman application interface. A test collection named "Project 3" is open, containing a test case titled "Fail to register existing user". The test script is as follows:

```
1 pm.test("Fail to Register existing User", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.id).to.eq(0);  
4});
```

The "Tests" tab is selected. Below the script, the "Test Results (1/1)" section shows a single test result:

Result	Message
PASS	Fail to Register existing User

Details below the table indicate the test status: Status: 200 OK, Time: 33 ms, Size: 181 B. A "Save Response" button is also present.

## Missing data to register user:

The screenshot shows the Postman application interface. A test collection named "Project 3" is open, containing a test case titled "Missing data to register user". The test script is as follows:

```
1 pm.test("Missing data to register user", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.id).to.eq(0);  
4});
```

The "Tests" tab is selected. Below the script, the "Test Results (1/1)" section shows a single test result:

Result	Message
PASS	Missing data to register user

Details below the table indicate the test status: Status: 200 OK, Time: 28 ms, Size: 181 B. A "Save Response" button is also present.

## Update user data:

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Upgrade'. The main header displays 'Project 3' and 'Update Password/ ActivateDeactivate User'. The left sidebar lists various API endpoints with their status (e.g., 'Cannot find the address to delete', 'Delete the address by addressID'). The central workspace shows a 'PATCH' request to 'http://localhost:55808/api/users'. The 'Tests' tab is selected, containing the following JavaScript code:

```
1 pm.test("Update User Data", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.passwordChanged).to.eql(true);
4     pm.expect(jsonData.userActivatedDeactivated).to.eql(false);
5 });
```

The 'Test Results' section shows one test case labeled 'PASS' with the name 'Update User Data'. The bottom status bar indicates 'Status: 200 OK', 'Time: 95 ms', and 'Size: 549 B'. The taskbar at the bottom shows icons for various Windows applications like File Explorer, Task View, and Edge.

## Fetch all products:

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Upgrade'. The main header displays 'Project 3' and 'Fetch All Products'. The left sidebar lists various API endpoints with their status (e.g., 'Cannot find the address to delete', 'Delete the address by addressID'). The central workspace shows a 'GET' request to 'http://localhost:55808/api/products'. The 'Tests' tab is selected, containing the following JavaScript code:

```
1 pm.test("Fetch All Products", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.success).to.eql(true);
4     pm.expect(jsonData.products.count()>0);
5 });
```

The 'Test Results' section shows one test case labeled 'PASS' with the name 'Fetch All Products'. The bottom status bar indicates 'Status: 200 OK', 'Time: 214 ms', and 'Size: 2.49 MB'. The taskbar at the bottom shows icons for various Windows applications like File Explorer, Task View, and Edge.

## Invalid method to fetch products:

The screenshot shows the Postman application interface. The left sidebar lists various API endpoints. The main workspace displays a POST request to 'http://localhost:55808/api/products'. The 'Tests' tab is selected, containing the following JavaScript code:

```
1 pm.test("Invalid Method to Fetch All Products", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.status).to.eql(415);
4});
```

The 'Test Results' section shows one test case labeled 'Invalid Method to Fetch All Products' with a status of 'PASS'. The bottom status bar indicates a 'Status: 415 Unsupported Media Type'.

## Add Products by admin:

The screenshot shows the Postman application interface. The left sidebar lists various API endpoints. The main workspace displays a POST request to 'http://localhost:55808/api/products'. The 'Tests' tab is selected, containing the following JavaScript code:

```
1 pm.test("Add Product By Admin", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.success).to.eql(true);
4     pm.expect(jsonData.product).to.not.eql(null);
5});
```

The 'Test Results' section shows one test case labeled 'Add Product By Admin' with a status of 'PASS'. The bottom status bar indicates a 'Status: 200 OK'.

## Non-Admin Unable to add products:

The screenshot shows the Postman application interface. A test collection named "Non-Admin: Unable to add products" is selected. A POST request is defined to "http://localhost:55808/api/products". The "Tests" tab contains the following JavaScript code:

```
1 pm.test("Non-Admin: Unable to add product", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.success).to.eql(false);  
4     pm.expect(jsonData.product).to.eql(null);  
5 });
```

The "Test Results" section shows one test case labeled "Non-Admin: Unable to add product" with a status of "PASS". The status bar at the bottom indicates "Status: 200 OK" and "Time: 103 ms".

## Admin delete products:

The screenshot shows the Postman application interface. A test collection named "Admin: Delete Products Successfully" is selected. A DELETE request is defined to "http://localhost:55808/api/products". The "Tests" tab contains the following JavaScript code:

```
1 pm.test("Admin: Delete Product Successfully", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.isProductDeleted).to.eql(true);  
4     pm.expect(jsonData.isAdmin).to.eql(true);  
5     pm.expect(jsonData.isValidUserRequestData).to.eql(true);  
6 });
```

The "Test Results" section shows one test case labeled "Admin: Delete Product Successfully" with a status of "PASS". The status bar at the bottom indicates "Status: 200 OK" and "Time: 124 ms".

## Non-Admin Delete Products Unsuccessful:

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Upgrade'. A 'Project 3' dropdown is open. The main workspace displays a collection titled 'Non-Admin: Delete Product Unsuccessful'. A single test case is selected, showing a 'DELETE' request to 'http://localhost:55808/api/products'. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("Non-Admin: Delete Product Unsuccessful", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.isProductDeleted).to.eql(false);  
4     pm.expect(jsonData.isAdmin).to.eql(false);  
5     pm.expect(jsonData.isValidUserRequestData).to.eql(true);  
6 });
```

The 'Test Results' section shows one test case labeled 'Non-Admin: Delete Product Unsuccessful' with a status of 'PASS'. The bottom status bar indicates a 'Status: 200 OK', 'Time: 12 ms', and 'Size: 246 B'. The taskbar at the bottom shows various icons and the system clock.

## Admin- Product not found to delete:

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Upgrade'. A 'Project 3' dropdown is open. The main workspace displays a collection titled 'Admin:Product Not Found'. A single test case is selected, showing a 'DELETE' request to 'http://localhost:55808/api/products'. The 'Tests' tab contains the following JavaScript code:

```
1 pm.test("Admin: Product Not Found", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.isProductDeleted).to.eql(false);  
4     pm.expect(jsonData.isAdmin).to.eql(true);  
5     pm.expect(jsonData.isValidUserRequestData).to.eql(true);  
6 });
```

The 'Test Results' section shows one test case labeled 'Admin: Product Not Found' with a status of 'PASS'. The bottom status bar indicates a 'Status: 200 OK', 'Time: 34 ms', and 'Size: 253 B'. The taskbar at the bottom shows various icons and the system clock.

## Unknown/Invalid product deletion request:

The screenshot shows the Postman application interface. A collection named "Project 3" is selected. A test case titled "Unknown/ Invalid Product Deletion Request" is open, showing a DELETE request to `http://localhost:55808/api/products`. The "Tests" tab contains the following JavaScript code:

```
1 pm.test("Unknown/ Invalid Product Deletion Request", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.isProductDeleted).to.eql(false);  
4     pm.expect(jsonData.isAdmin).to.eql(false);  
5     pm.expect(jsonData.isValidUserRequestData).to.eql(false);  
6});
```

The "Test Results" section shows one test case labeled "Unknown/ Invalid Product Deletion Request" with a status of "PASS". The bottom status bar indicates a 200 OK response, 43 ms time, and 255 B size.

## Fetch Single Product By Id:

The screenshot shows the Postman application interface. A collection named "Project 3" is selected. A test case titled "Fetch Single Product by Id" is open, showing a GET request to `http://localhost:55808/api/products/2`. The "Tests" tab contains the following JavaScript code:

```
1 pm.test("Fetch Single Product by Id", function () {  
2     var jsonData = pm.response.json();  
3     pm.expect(jsonData.success).to.eql(true) && pm.expect(jsonData.products).to.not.eql(null);  
4});
```

The "Test Results" section shows one test case labeled "Fetch Single Product by Id" with a status of "PASS". The bottom status bar indicates a 200 OK response, 46 ms time, and 36.84 KB size.

## Fail to Fetch Single Product By Invalid Id:

The screenshot shows the Postman application interface. A collection named "Project 3" is selected. A test script is written to check if a product with ID 10 exists. The response status is 200 OK, indicating a failure.

```
pm.test("Fetch Single Product by invalid Id", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.success).to.eql(false);
    pm.expect(jsonData.products).to.eql(null);
});
```

Test Results (1/1):

All	Passed	Skipped	Failed
	PASS		

Details: Status: 200 OK Time: 70 ms Size: 215 B Save Response

## Adding products to cart with valid inputs

The screenshot shows the Postman application interface. A test script is written to verify that a POST request to add products to the cart with valid inputs results in a status code of 200. The response status is 200 OK, indicating success.

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

Test Results (1/1):

All	Passed	Skipped	Failed
	PASS		

Details: Status code is 200 Status: 200 OK Time: 8.10 s Size: 973 B Save Response

## Adding products to cart with valid inputs to check status

The Postman interface shows a collection named "Project 3" with a single test step. The test script checks if the status code is "OK". The results show one test passed: "Status code name has string". The status bar indicates a 200 OK response.

```
pm.test("Status code name has string", function () {
    pm.response.to.have.status("OK");
});
```

Status: 200 OK Time: 102 ms Size: 966 B Save Response

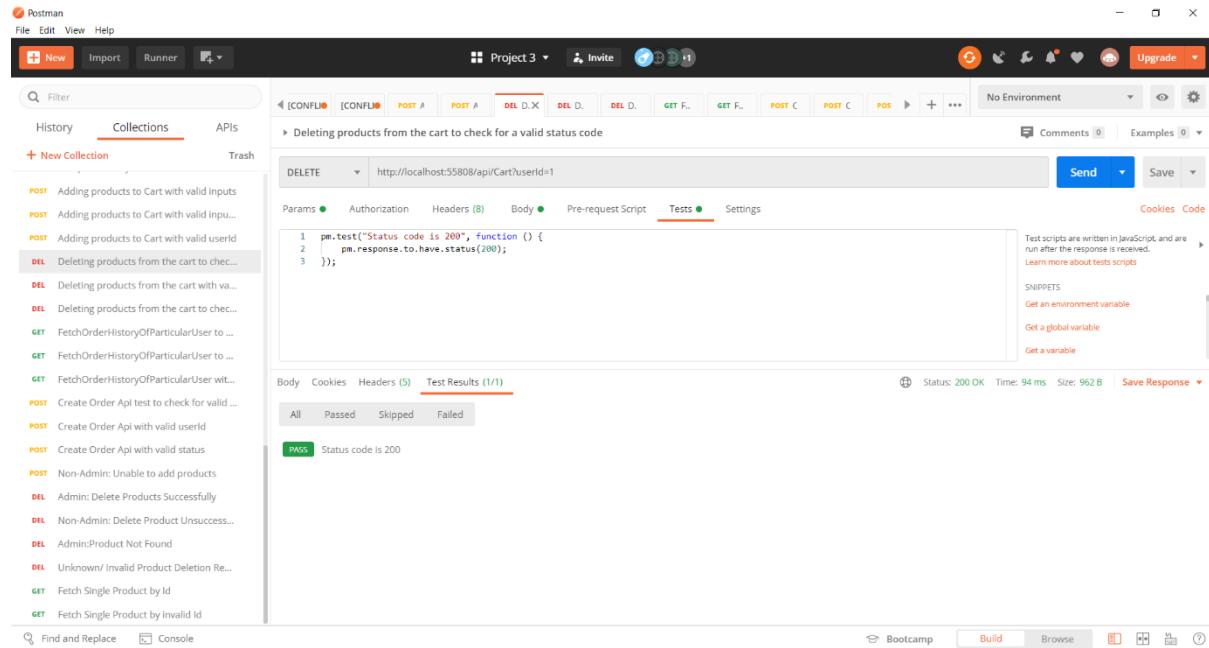
## Adding products to cart with valid user id

The Postman interface shows a collection named "Project 3" with a single test step. The test script checks if the body matches a string containing "userId=1". The results show one test passed: "Body matches string". The status bar indicates a 200 OK response.

```
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include('userId=1');
});
```

Status: 200 OK Time: 344 ms Size: 966 B Save Response

## Deleting products from cart to check for a valid status code



Postman

File Edit View Help

Project 3 Invite

DELETE http://localhost:55808/api/Cart?userId=1

Params Authorization Headers (8) Body Pre-request Script Tests Settings

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
```

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

Body Cookies Headers (5) Test Results (1/1)

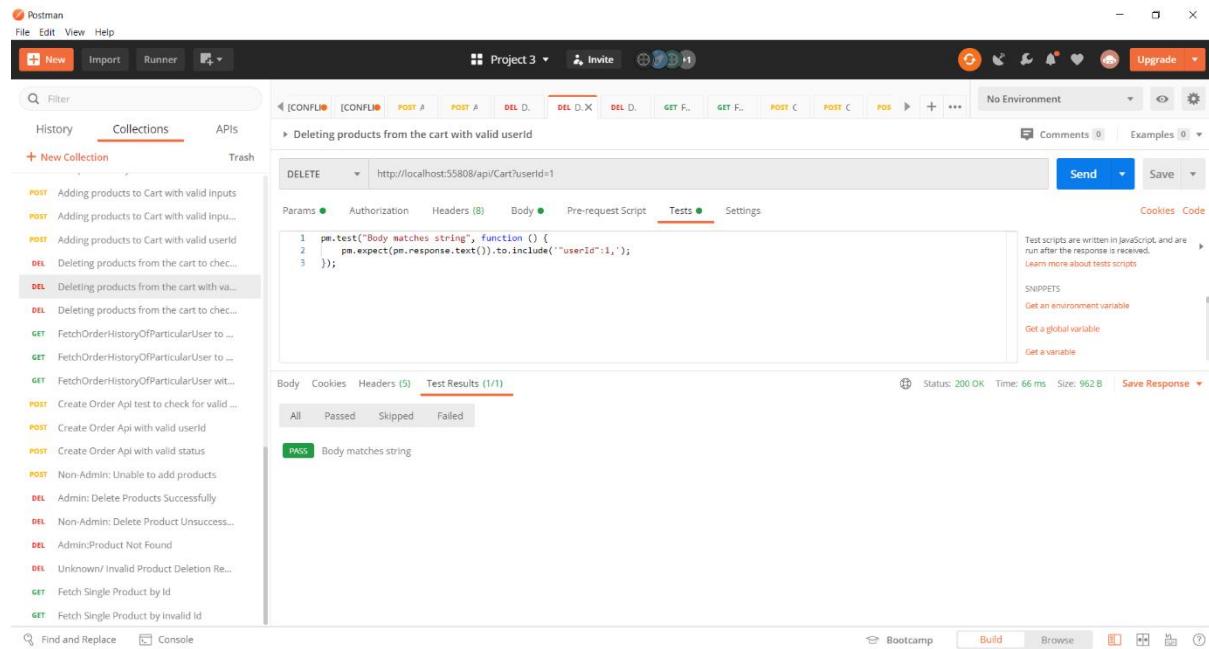
All Passed Skipped Failed

PASS Status code is 200

Status: 200 OK Time: 94 ms Size: 962 B Save Response

Find and Replace Console

## Deleting products from cart with valid user id



Postman

File Edit View Help

Project 3 Invite

DELETE http://localhost:55808/api/Cart?userId=1

Params Authorization Headers (8) Body Pre-request Script Tests Settings

```
1 pm.test("Body matches string", function () {
2   pm.expect(pm.response.text()).to.include('userId:1');
3 });
```

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

Body Cookies Headers (5) Test Results (1/1)

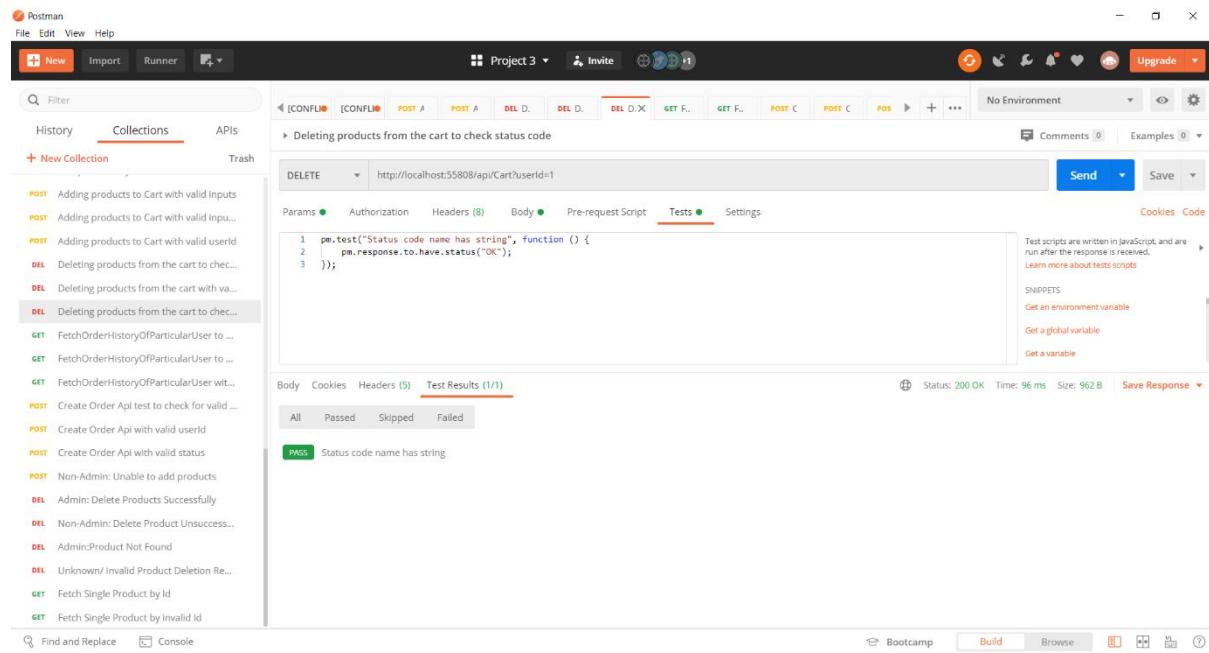
All Passed Skipped Failed

PASS Body matches string

Status: 200 OK Time: 66 ms Size: 962 B Save Response

Find and Replace Console

## Deleting products from product to check for a valid status code

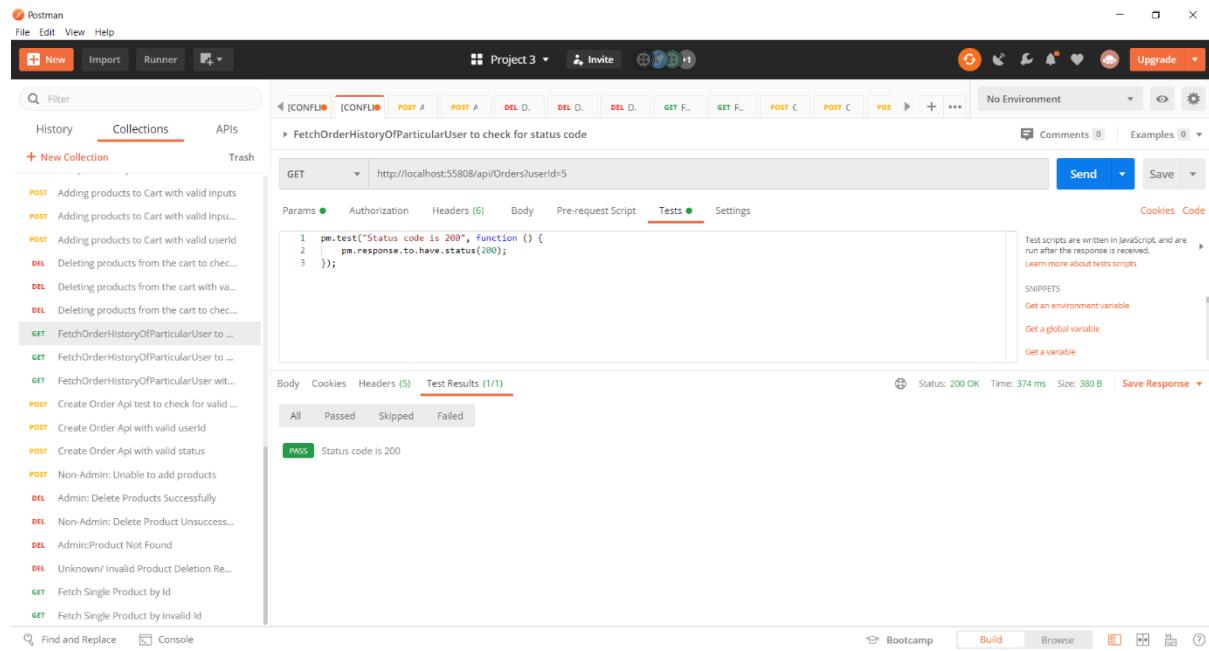


The Postman interface shows a collection named "Project 3" with a single test step titled "Deleting products from the cart to check status code". The request method is DELETE, and the URL is `http://localhost:55808/api/Cart?userId=1`. The "Tests" tab contains the following JavaScript code:

```
1 pm.test("Status code name has string", function () {
2     pm.response.to.have.status("OK");
3 });
```

The "Test Results" section shows one test result: "Status code name has string" with a status of PASS. The overall status of the request is 200 OK, time taken is 96 ms, and size is 962 B.

## Fetching Users Order History to check for a valid status code



The Postman interface shows a collection named "Project 3" with a single test step titled "FetchOrderHistoryOfParticularUser to check for status code". The request method is GET, and the URL is `http://localhost:55808/api/Orders?userId=5`. The "Tests" tab contains the following JavaScript code:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
```

The "Test Results" section shows one test result: "Status code is 200" with a status of PASS. The overall status of the request is 200 OK, time taken is 374 ms, and size is 380 B.

## Fetching Users Order History to check for status

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'Project 3' (with an invite link), and 'Upgrade'. Below the navigation is a toolbar with buttons for 'New', 'Import', 'Runner', and others. The main workspace shows a collection named 'FetchOrderHistoryOfParticularUser to check for status'. A GET request is selected with the URL `http://localhost:55808/api/Orders?userId=2`. The 'Tests' tab is active, containing the following JavaScript code:

```
1 pm.test("Status code name has string", function () {
2   pm.response.to.have.status("OK");
3 });
```

The 'Test Results' section shows one test result: 'Status code name has string' (PASS). The status bar at the bottom indicates 'Status: 200 OK', 'Time: 17 ms', and 'Size: 5.06 KB'. Other tabs like 'Body', 'Cookies', and 'Headers' are visible.

## Fetching Users Order History with valid user id

This screenshot is identical to the previous one, showing the same Postman interface and collection. The difference is in the test script, which now checks for a specific user ID in the response body:

```
1 pm.test("Body matches string", function () {
2   pm.expect(pm.response.text()).to.include({"userId":2,""});
3 });
```

The 'Test Results' section shows one test result: 'Body matches string' (PASS). The status bar at the bottom indicates 'Status: 200 OK', 'Time: 30 ms', and 'Size: 5.06 KB'.

## Create Order and push order details and update product quantity api to check for valid status code

The screenshot shows the Postman application interface. A test script in the 'Tests' tab of a POST request to 'http://localhost:55808/api/Orders' is run successfully, returning a status code of 200. The results table shows one pass: 'Status code is 200'.

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
```

Body	Cookies	Headers (5)	Test Results (1/1)
All	Passed	Skipped	Failed
PASS Status code is 200			

Status: 200 OK Time: 2.21 s Size: 3.11 MB Save Response

## Create Order api with valid user id

The screenshot shows the Postman application interface. A test script in the 'Tests' tab of a POST request to 'http://localhost:55808/api/Orders' is run successfully, returning a status code of 200. The results table shows one pass: 'Body matches string'.

```
1 pm.test("Body matches string", function () {
2   pm.expect(pm.response.text()).to.include('"userId":1');
3 });
```

Body	Cookies	Headers (5)	Test Results (1/1)
All	Passed	Skipped	Failed
PASS Body matches string			

Status: 200 OK Time: 1975 ms Size: 3.11 MB Save Response

## Create order api to check for a valid status

POST Adding products to Cart with valid inputs

POST Adding products to Cart with valid input...

POST Adding products to Cart with valid user...

DEL Deleting products from the cart to chec...

DEL Deleting products from the cart with va...

DEL Deleting products from the cart to chec...

GET FetchOrderHistoryOfParticularUser to ...

GET FetchOrderHistoryOfParticularUser to ...

GET FetchOrderHistoryOfParticularUser wit...

POST Create Order Api test to check for valid ...

POST Create Order Api with valid user...

POST Create Order Api with valid status

POST Non-Admin: Unable to add products

DEL Admin: Delete Products Successfully

DEL Non-Admin: Delete Product Unsuccess...

DEL Admin:Product Not Found

DEL Unknown/ Invalid Product Deletion Re...

GET Fetch Single Product by Id

GET Fetch Single Product by invalid id

## Get certain user's address

GET Get certain user's address

GET Address data didn't exist at DB

POST Add Address to DB

PUT Edit certain address detail from certain ad...

PUT Cannot find the address id from DB to edit...

DEL Cannot find the address to delete

DEL Delete the address by addressId

GET Found Comments for product

GET No Comments Found for Product

POST Add the comment on the product

PUT Modify the comment

PUT Fail to modify the comment

DEL Delete the comment

DEL Invalided Delete the comment by different ...

GET Login with valid credentials

## Address data didn't exist at DB

The screenshot shows the Postman interface with a collection named "Project3Collection". A GET request is being tested with the URL `http://localhost:55808/api/addresses?User_Id=100`. The "Tests" tab contains the following JavaScript code:

```
pm.test("Address didn't exist", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.hasAddress).to.eq(false);
});
```

The response body is shown as:

```
1 []
2 [
3     "addresses": [],
4     "hasAddress": false
5 ]
```

The status bar indicates a 200 OK response with a time of 74 ms and a size of 217 B.

## Add Address to DB

The screenshot shows the Postman interface with a collection named "Project3Collection". A POST request is being tested with the URL `http://localhost:55808/api/addresses`. The "Tests" tab contains the following JavaScript code:

```
pm.test("Add Address Successfully", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.id).to.eq(2);
    pm.expect(jsonData.action).to.eq(true);
});
```

The response body is shown as:

```
1 [
2     {
3         "id": 2,
4         "action": true
5     }
6 ]
```

The status bar indicates a 200 OK response with a time of 92 ms and a size of 196 B.

## Edit certain address detail from certain address ID number

The screenshot shows the Postman application interface. The left sidebar displays collections and requests, including an 'E-Commerce' collection with various test cases like 'Get certain user's address' and 'Edit certain address detail from certain ad...'. The main workspace shows a 'PUT' request to 'http://localhost:55808/api/addresses?Address\_Id=1'. The 'Tests' tab contains a failing JavaScript test script:

```
1 pm.test("Edit Address Successfully", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.id).to.eq(1);
4});
```

The status bar at the bottom indicates a 'Status: 200 OK' response.

## Cannot find the address Id from DB to edit address

The screenshot shows the Postman application interface. The left sidebar displays a collection named "Project3Collection" with various API requests. A specific PUT request is selected in the center panel, targeting the URL `http://localhost:55808/api/addresses?Address_Id=100`. The "Tests" tab is active, containing the following JavaScript code:

```
pm.test("Invalid address detail modify", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.id).to.eql(0);
});
```

The "Body" tab shows the response body as a JSON object with the following content:

```
{ "id": 0, "action": false }
```

The status bar at the bottom indicates a 200 OK response with a time of 69 ms.

## Cannot find the address to delete

The screenshot shows the Postman application interface. The left sidebar displays a collection named "Project3Collection" with various API requests. A specific DELETE request is selected in the center panel, targeting the URL `http://localhost:55808/api/addresses/18`. The "Tests" tab is active, containing the following JavaScript code:

```
pm.test("Cannot delete Address", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.id).to.eql(18);
    pm.expect(jsonData.action).to.eql(false);
});
```

The "Body" tab shows the response body as a JSON object with the following content:

```
{ "id": 18, "action": false }
```

The status bar at the bottom indicates a 200 OK response with a time of 59 ms.

## Delete the address by addressID

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, New, Import, Runner, and Upgrade. The main toolbar has buttons for POST, GET, PUT, and DELETE methods. A collection named "Project 3" is selected, containing 36 requests. The current request is a DELETE operation to "http://localhost:55808/api/addresses/13". The "Tests" tab is active, displaying a JavaScript test script:

```
1 pm.test("Delete Address successfully", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.id).to.eq(13);
4     pm.expect(jsonData.action).to.eq(true);
5 });
```

The "Body" tab shows a JSON response with the following content:

```
1 {
2     "id": 13,
3     "action": true
4 }
```

The status bar at the bottom indicates Status: 200 OK, Time: 92 ms, Size: 205 B, and Save Response.

## Found Comments for product

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, New, Import, Runner, and Upgrade. The main toolbar has buttons for POST, GET, PUT, and DELETE methods. A collection named "Project 3" is selected, containing 36 requests. The current request is a GET operation to "http://localhost:55808/api/comment?productId=3". The "Tests" tab is active, displaying a JavaScript test script:

```
1 pm.test("Found Comments", function () {
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.hasComments).to.eq(true);
4 });
```

The "Body" tab shows a JSON response with the following content:

```
1 {
2     "comments": [
3         {
4             "commentId": 2,
5             "productId": 3,
6             "userId": 1,
7             "content": "Modify the comment",
8             "createdAt": "2020-06-01T19:38:56",
9             "updatedAt": "2020-06-01T19:38:56"
10        }
11    ]
12}
```

The status bar at the bottom indicates Status: 200 OK, Time: 40 ms, Size: 379.29 KB, and Save Response.

## No Comments Found for Product

The screenshot shows the Postman interface with a collection named "Project3Collection". A GET request is made to `http://localhost:55808/api/comment?Product_Id=100`. The response status is 200 OK, and the body contains the JSON object `{"comments": [], "hasComments": false}`.

```
pm.test("No Comments Found", function () {
  var jsonData = pm.response.json();
  pm.expect(jsonData.hasComments).to.eql(false);
});
```

Body (JSON)

```
[{"id": 20, "actionSuccessfully": true}]
```

## Add the comment on the product

The screenshot shows the Postman interface with a collection named "Project3Collection". A POST request is made to `http://localhost:55808/api/comment?Product_Id=1&User_Id=1`. The response status is 200 OK, and the body contains the JSON object `{"id": 20, "actionSuccessfully": true}`.

```
pm.test("Add the comment successfully", function () {
  var jsonData = pm.response.json();
  pm.expect(jsonData.actionSuccessfully).to.eql(true);
});
```

Body (JSON)

```
[{"id": 20, "actionSuccessfully": true}]
```

## Modify the comment

The Postman interface shows a successful API call to modify a comment. The request URL is `http://localhost:55808/api/comment?Comment_Id=2`. The response status is 200 OK, time 158 ms, size 216 B. The response body is:

```
1 {
2   "id": 2,
3   "actionSuccessfully": true
4 }
```

## Fail to modify the comment

The Postman interface shows a failed API call to modify a comment. The request URL is `http://localhost:55808/api/comment?Comment_Id=100`. The response status is 200 OK, time 94 ms, size 217 B. The response body is:

```
1 {
2   "id": 0,
3   "actionSuccessfully": false
4 }
```

## Delete the comment

The Postman interface shows a successful DELETE request to the endpoint `http://localhost:55808/api/comment?Comment_Id=12&User_Id=3`. The response status is 200 OK, and the JSON body contains the following data:

```
1 {  
2   "id": 12,  
3   "actionSuccessfully": true  
4 }
```

## Invalid Delete the comment by different user

The Postman interface shows an invalid DELETE request to the endpoint `http://localhost:55808/api/comment?Comment_Id=2&User_Id=1`. The response status is 200 OK, and the JSON body contains the following data:

```
1 {  
2   "id": 2,  
3   "actionSuccessfully": false  
4 }
```