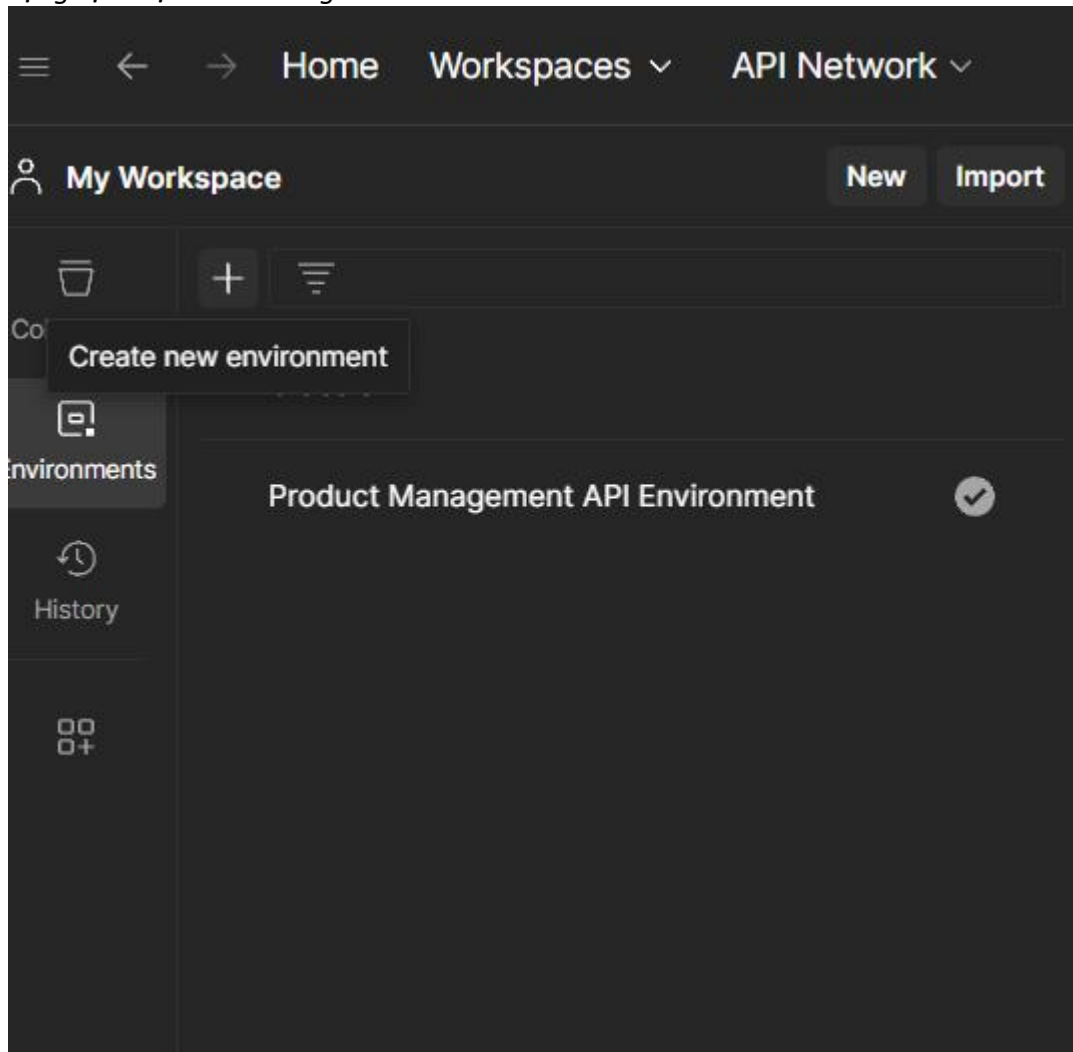


I. Yêu cầu liên quan đến dữ liệu

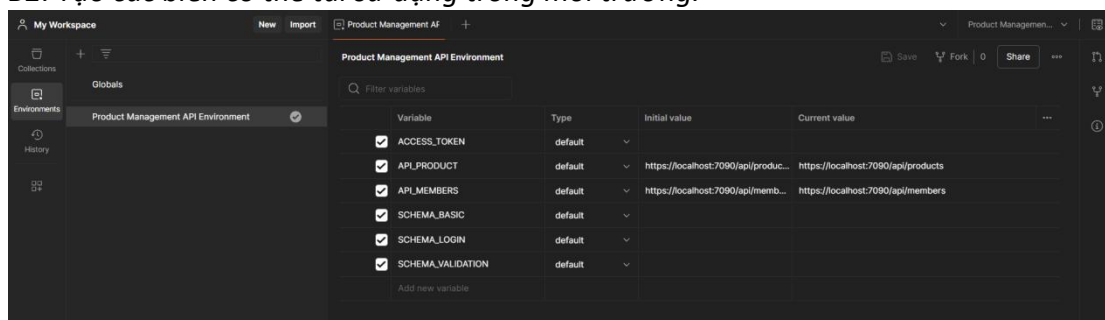
- Tối thiểu 222 dòng dữ liệu trong bảng Product
- Default-page-size : 50
- Max-page-size: 200

II. Cách tạo và sử dụng các biến môi trường trong các Request

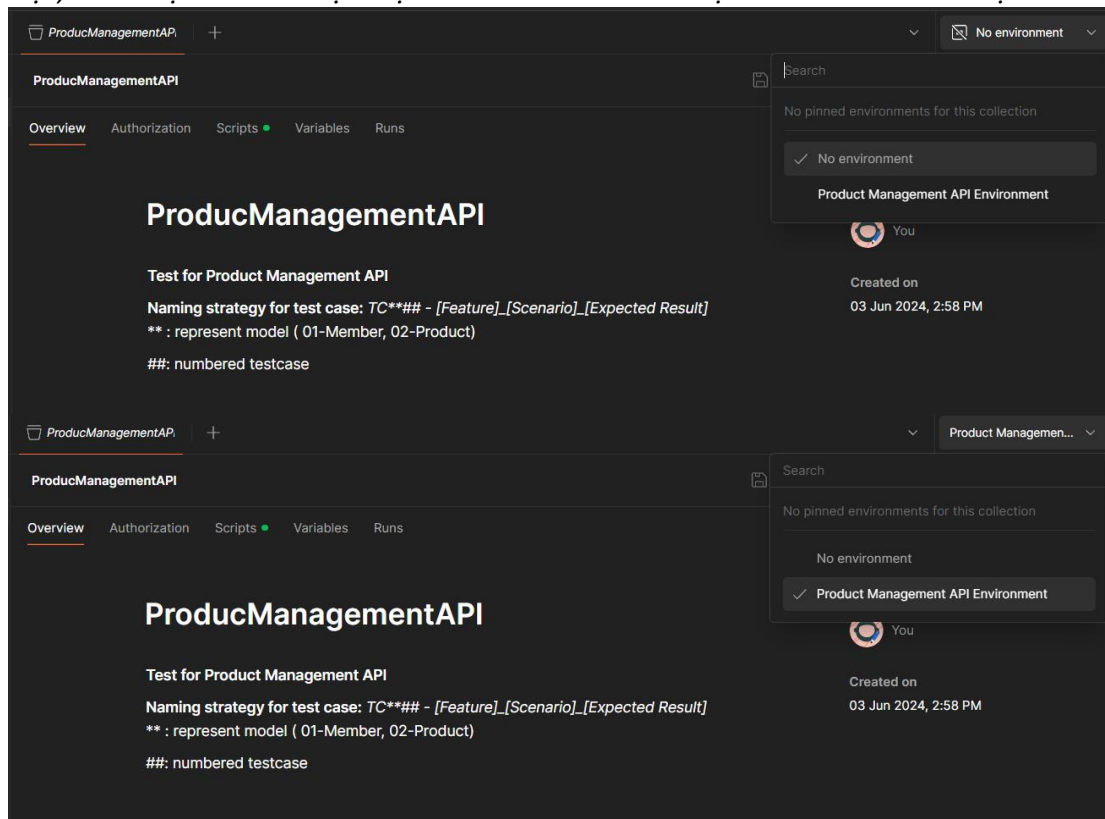
B1: Tạo môi trường, chọn tab environments ở bên trái màn hình Postman, click dấu cộng tạo một môi trường mới



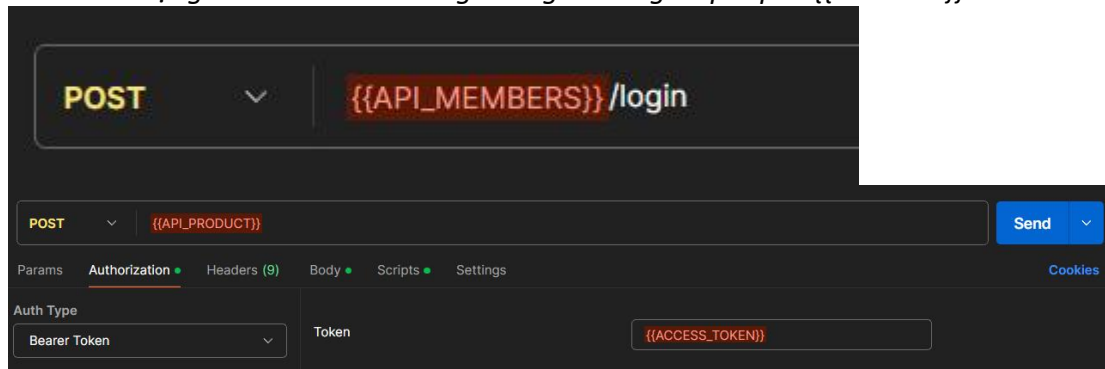
B2: Tạo các biến có thể tái sử dụng trong môi trường.



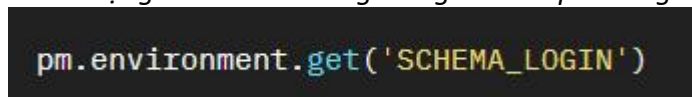
B3: Setup để các Request có thể sử dụng các biến môi trường. Trên thanh các tab làm việc, click chọn vào khu vực hiện No Environment -> chọn Environment của bạn



B4: Để sử dụng các biến môi trường chúng ta dùng cú pháp: `{{Tên Biến}}`



Để sử dụng biến môi trường trong test script chúng ta sử dụng:



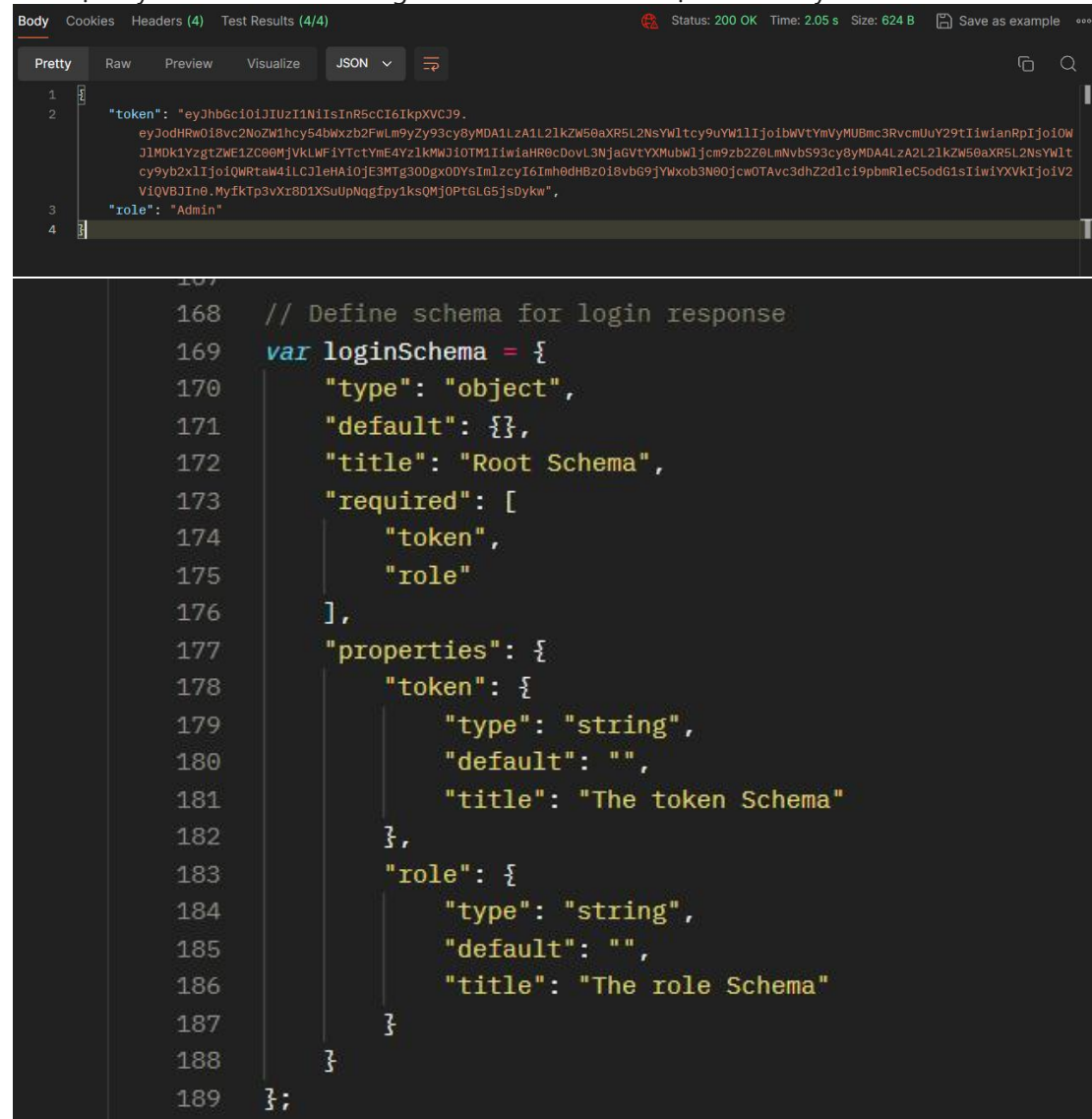
LƯU Ý: Các giá trị của các biến môi trường có thể tự điền bằng tay hoặc có thể tự động được gán, ví dụ một số cách gán tự động giá trị của biến môi trường:



III. Cách gán JSON Schema vào biến môi trường

- JSON Schema là tiêu chuẩn của tài liệu JSON mô tả cấu trúc và các yêu cầu của dữ liệu JSON của bạn

- Ví dụ: đây là trả về của hàm login và schema của dữ liệu trả về này



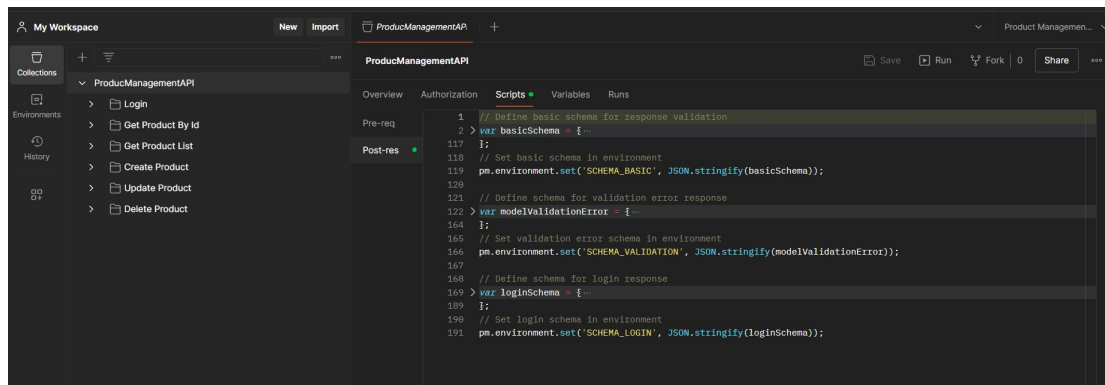
The image shows a web browser window displaying a JSON response from a login endpoint. The response is a JSON object with two properties: "token" and "role". The "token" value is a long alphanumeric string, and the "role" value is "Admin". Below the browser window, a code editor shows the JSON Schema definition for this response. The schema is defined as an object with two required properties: "token" (a string) and "role" (a string). The schema is named "Root Schema" and is used to validate the login response.

```
168 // Define schema for login response
169 var loginSchema = {
170     "type": "object",
171     "default": {},
172     "title": "Root Schema",
173     "required": [
174         "token",
175         "role"
176     ],
177     "properties": {
178         "token": {
179             "type": "string",
180             "default": "",
181             "title": "The token Schema"
182         },
183         "role": {
184             "type": "string",
185             "default": "",
186             "title": "The role Schema"
187         }
188     }
189 };
```

- Để biết các kiểm tra JSON Schema bạn có thể xem ở các test-case ví dụ ở dưới

- Bạn có thể sử dụng trang web sau để tạo JSON Schema dựa trên response có sẵn của bạn: [JSON Schema Tool](#)

- Sau khi bạn có JSON Schema, bạn không nên gán thủ công vào biến môi trường, nếu gán thủ công vào biến môi trường thì có thể sẽ dẫn đến lỗi. Cách gán JSON Schema vào biến môi trường tự động



- Để hiểu rõ hơn về cách test JSON Schema trong Postman bạn có thể xem ở đây: [Example 02 – JSON Schema Validation | Postman API Monitoring Examples | Postman API Network](#)

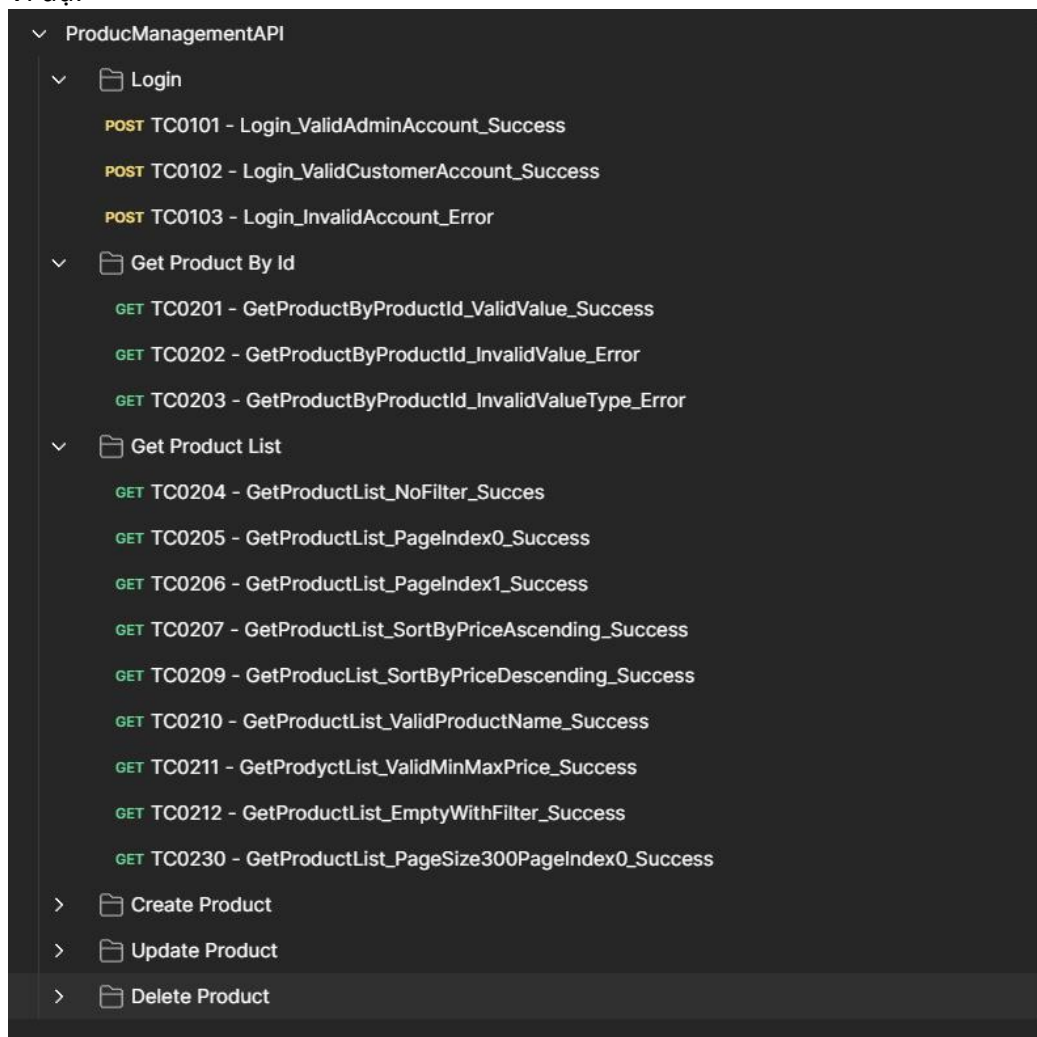
IV. Đặt tên test-case

*TC**## - [Chức năng]_[Kịch bản]_[Kết quả mong muốn]*

** : đại diện cho bảng dữ liệu tương ứng (01-Member, 02-Product)

: đánh số thứ tự

Ví dụ:



▼	ProducManagementAPI
>	Login
>	Get Product By Id
>	Get Product List
▼	Create Product
	POST TC0213 - CreateProduct_ValidCredentials_Success
	POST TC0214 - CreateProduct_ExistId_Failed
	POST TC0215 - CreateProduct_InvalidCategoryId_Error
	POST TC0216 - CreateProduct_InvalidPriceAndQuantity_Error
	POST TC0217 - CreateProduct_Unauthorized_Error
	POST TC0218 - CreateProduct_Unauthenticated_Error
▼	Update Product
	PUT TC0219 - UpdateProduct_ValidCredentials_Success
	PUT TC0220 - UpdateProduct_NotExistId_Error
	PUT TC0221 - UpdateProduct_InvalidCategoryId_Error
	PUT TC0222 - UpdateProduct_InvalidPriceAndQuantity_Error
	PUT TC0223 - UpdateProduct_Unauthorized_Error
	PUT TC0224 - UpdateProduct_Unauthenticated_Error
▼	Delete Product
	DEL TC0225 - DeleteProduct_ValidCredentials_Success
	DEL TC0226 - DeleteProduct_NotExistId_Error
	DEL TC0227 - DeleteProduct_InvalidValueType_Error
	DEL TC0228 - DeleteProduct_Unauthorized_Error
	DEL TC0229 - DeleteProduct_Unauthenticated_Error

LƯU Ý: Cách đặt tên hay đánh số cho test-case trên không phải là quy chuẩn cho tất cả, tên test-case sẽ có sự khác biệt nhất định giữa mỗi đơn vị, hay cá nhân,.. Nhưng đó sẽ là tiêu chuẩn mọi người nên hướng tới.

V. Ví dụ test-case của các chức năng

1. Login

The screenshot displays a REST client interface for a test case named "TC0101 - Login_ValidAdminAccount_Success". The method is POST, and the URL is `{{API_MEMBERS}}/login`. The interface includes tabs for Params, Authorization, Headers (8), Body, Scripts, and Settings. The Scripts tab is active, showing a pre-request script and a post-request script. The post-request script contains four assertions: checking the status code is 200, validating the response schema, checking the token is a non-empty string, and verifying the response has a role of 'Admin'. The Test Results panel at the bottom shows four passed assertions: "Login successfully", "Schema validation", "Token is a non-empty string", and "Response has a role 'Admin'".

```
1  var jsonData = pm.response.json();
2
3  // Check status
4  pm.test("Login successfully", function () {
5    pm.response.to.have.status(200); // Verify status code is 200
6  });
7
8  // Validate response schema
9  pm.test('Schema validation', () => {
10    pm.response.to.have.jsonSchema(JSON.parse(pm.environment.get('SCHEMA_LOGIN'))); // Validate JSON schema
11  });
12
13 // Check if token is a non-empty string
14 pm.test("Token is a non-empty string", function () {
15   pm.expect(jsonData.token).to.be.a("string").and.not.be.empty; // Verify token is a non-empty string
16   pm.environment.set("ACCESS_TOKEN", jsonData.token); // Store token in environment variable
17 });
18
19 // Check if the response has a role 'Admin'
20 pm.test("Response has a role 'Admin'", function () {
21   pm.expect(jsonData).to.have.property("role"); // Check for role property
22   pm.expect(jsonData.role).to.eql("Admin"); // Verify role is 'Admin'
23 });
24
```

Test Results (4/4)

- PASS Login successfully
- PASS Schema validation
- PASS Token is a non-empty string
- PASS Response has a role 'Admin'

2. Get Product By Id

HTTP

ProdcManagementAPI / Get Product By Id / TC0201 - GetProductByProductId_ValidValue_Success

GET

{{(API_PRODUCT)}/1

ParamsAuthorizationHeaders (6)BodyScriptsSettings

Pre-req

Post-res

```
1 var jsonData = pm.response.json();
2
3 // Check response status code
4 pm.test("Response status code is 200", function () {
5   pm.expect(jsonData.status).to.equal(200); // Verify status code is 200
6 });
7
8 // Validate response schema
9 pm.test('Schema validation', () => {
10   pm.response.to.have.jsonSchema(JSON.parse(pm.environment.get('SCHEMA_BASIC'))); // Validate JSON schema
11 });
12
13 // Check product details in response
14 pm.test("Response contains the correct product details", function () {
15   pm.expect(jsonData.data['product-id']).to.equal(1); // Verify product ID is 1
16   pm.expect(jsonData.data['category-name']).to.equal("Food"); // Verify category name is 'Food'
17   pm.expect(jsonData.data['product-name']).to.equal("Candy"); // Verify product name is 'Candy'
18   pm.expect(jsonData.data['weight']).to.equal("500g"); // Verify weight is '500g'
19   pm.expect(jsonData.data['unit-price']).to.equal(20000.0000); // Verify unit price is 20000.0000
20   pm.expect(jsonData.data['units-in-stock']).to.equal(19); // Verify units in stock is 19
21 });
22
23 // Check message in response
24 pm.test("Response contains the correct message", function () {
25   pm.expect(jsonData.message).to.equal("Get product succeed with id: 1"); // Verify message is correct
26 });
27
```

BodyCookiesHeaders (4)Test Results (4/4)

AllPassedSkippedFailed

PASS

Response status code is 200

PASS

Schema validation

PASS

Response contains the correct product details

PASS

Response contains the correct message

3. Get Product List

ProducManagementAPI / Get Product List / TC0207 - GetProductList_SortByPriceAscending_Success

GET{{(APL_PRODUCT)}}?sort_by_price=true&page_index=0

ParamsAuthorizationHeaders (6)BodyScriptsSettings

Query Params

<input type="checkbox"/>	Key	Value
<input type="checkbox"/>	name	
<input type="checkbox"/>	categories_id	
<input type="checkbox"/>	unit_price_min	
<input type="checkbox"/>	unit_price_max	
<input checked="" type="checkbox"/>	sort_by_price	true
<input type="checkbox"/>	descending	false
<input checked="" type="checkbox"/>	page_index	0
<input type="checkbox"/>	page_size	20
	Key	Value

ProducManagementAPI / Get Product List / TC0207 - GetProductList_SortByPriceAscending_Success

GET{{(APL_PRODUCT)}}?sort_by_price=true&page_index=0

ParamsAuthorizationHeaders (6)BodyScriptsSettings

Pre-req

Post-res

```
1  var jsonData = pm.response.json();
2
3  // Check if the status code is 200
4  pm.test("Status code is 200", function () {
5    pm.response.to.have.status(200);
6  });
7
8  // Validate response schema
9  pm.test('Schema validation', () => {
10    pm.response.to.have.jsonSchema(JSON.parse(pm.environment.get('SCHEMA_BASIC')));
11  });
12
13  // Check if the response is sorted by price in ascending order
14  pm.test("Response is sorted by price in ascending order", function () {
15    var previousPrice = 0;
16    jsonData.data.items.forEach(function(product) {
17      var unitPrice = product["unit-price"];
18      pm.expect(unitPrice).to.be.at.least(previousPrice); // Check if current price is greater than or equal to previous price
19      previousPrice = product["unit-price"];
20    });
21  });
22
23  // Check pagination information
24  pm.test("Response contains pagination information", function () {
25    pm.expect(jsonData).to.be.an('object');
26    var data = jsonData.data;
27    pm.expect(data).to.have.property('total-item-count').to.eql(222); // Verify total-item-count
28    pm.expect(data).to.have.property('page-size').to.eql(200); // Verify page-size
29    pm.expect(data).to.have.property('total-pages-count').to.eql(2); // Verify total-pages-count
30    pm.expect(data).to.have.property('page-index').to.eql(0); // Verify page-index
31    pm.expect(data).to.have.property('next').to.be.true; // Verify next
32    pm.expect(data).to.have.property('previous').to.be.false; // Verify previous
33  });
34
35  // Check product details
36  pm.test("Response contains the correct product details", function () {
37    var items = jsonData.data["items"];
38    pm.expect(items).to.be.an("array");
39    pm.expect(items.length).to.eql(200); // Check if there are 200 items
40
41    // Check details of first item
42    var firstItem = items[0];
43    pm.expect(firstItem["product-id"]).to.eql(88);
44    pm.expect(firstItem["category-id"]).to.eql(1);
45    pm.expect(firstItem["product-name"]).to.eql("Product88");
46    pm.expect(firstItem["category-name"]).to.eql("Food");
47    pm.expect(firstItem["weight"]).to.eql("Weight88");
48    pm.expect(firstItem["unit-price"]).to.eql(3.8700);
49    pm.expect(firstItem["units-in-stock"]).to.eql(34);
50  });
```



```

50
51 // Check details of middle item
52 var middleItem = items[2];
53 pm.expect(middleItem["product-id"]).to.eql(33);
54 pm.expect(middleItem["category-id"]).to.eql(1);
55 pm.expect(middleItem["product-name"]).to.eql("Product33");
56 pm.expect(middleItem["category-name"]).to.eql("Food");
57 pm.expect(middleItem["weight"]).to.eql("Weight33");
58 pm.expect(middleItem["unit-price"]).to.eql(7.7500);
59 pm.expect(middleItem["units-in-stock"]).to.eql(4);
60
61 // Check details of last item
62 var lastItem = items[items.length - 1];
63 pm.expect(lastItem["product-id"]).to.eql(150);
64 pm.expect(lastItem["category-id"]).to.eql(1);
65 pm.expect(lastItem["product-name"]).to.eql("Product150");
66 pm.expect(lastItem["category-name"]).to.eql("Food");
67 pm.expect(lastItem["weight"]).to.eql("Weight150");
68 pm.expect(lastItem["unit-price"]).to.eql(923.2400);
69 pm.expect(lastItem["units-in-stock"]).to.eql(36);
70 });
71

```

Body	Cookies	Headers (4)	Test Results (5/5)
<div> All Passed Skipped Failed </div>			
<div>PASS</div> Status code is 200			
<div>PASS</div> Schema validation			
<div>PASS</div> Response is sorted by price in ascending order			
<div>PASS</div> Response contains pagination information			
<div>PASS</div> Response contains the correct product details			

4. Create Product

ProductManagementAPI / Create Product / TC0213 - CreateProduct_ValidCredentials_Success

POST

{{API_PRODUCT}}

Params

Authorization

Headers (8)

Body

Scripts

Settings

Auth Type

Bearer Token

Token

{{ACCESS_TOKEN}}

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

HTTP ProdManagementAPI / Create Product / TC0213 - CreateProduct_ValidCredentials_Success

POST ▼ {{(API_PRODUCT)}}

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1  [
2  .. "product-id": 600,
3  .. "category-id": 1,
4  .. "product-name": "test post postman",
5  .. "weight": "100g",
6  .. "unit-price": 1000,
7  .. "units-in-stock": 10
8  ]
```

HTTP ProdManagementAPI / Create Product / TC0213 - CreateProduct_ValidCredentials_Success

POST ▼ {{(API_PRODUCT)}}

Params Authorization Headers (8) **Body** **Scripts** Settings

Pre-req

Post-res ●

```
1  var jsonData = pm.response.json();
2
3  // Check status code
4  pm.test("Status code is 200", function () {
5    pm.response.to.have.status(200); // Verify status code is 200
6  });
7
8  // Check JSON schema
9  pm.test('Schema validation', () => {
10    pm.response.to.have.jsonSchema(JSON.parse(pm.environment.get('SCHEMA_BASIC'))); // Validate JSON schema
11  });
12
13  // Check message in response
14  pm.test("Message is 'Add product Succeed'", function () {
15    pm.expect(jsonData.message).to.eql("Add product Succeed"); // Verify message
16  });
17
18  // Check data in response
19  pm.test("Response has correct result object", function () {
20    pm.expect(jsonData.data).to.have.property('product-id', 600); // Verify product ID
21    pm.expect(jsonData.data).to.have.property('category-id', 1); // Verify category ID
22    pm.expect(jsonData.data).to.have.property('product-name', 'test post postman'); // Verify product name
23    pm.expect(jsonData.data).to.have.property('weight', '100g'); // Verify weight
24    pm.expect(jsonData.data).to.have.property('unit-price', 1000); // Verify unit price
25    pm.expect(jsonData.data).to.have.property('units-in-stock', 10); // Verify units in stock
26  });
27
```

Body Cookies Headers (4) **Test Results (4/4)**

All Passed Skipped Failed ↺

PASS Status code is 200

PASS Schema validation

PASS Message is 'Add product Succeed'

PASS Response has correct result object

5. Update Product

ProductManagementAPI / Update Product / TC0219 - UpdateProduct_ValidCredentials_Success

PUT ▼ `{{(API_PRODUCT)}}/600`

Params Authorization Headers (8) Body Scripts Settings

Auth Type: Bearer Token ▼ Token: `{{(ACCESS_TOKEN)}}`

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

ProductManagementAPI / Update Product / TC0219 - UpdateProduct_ValidCredentials_Success

PUT ▼ `{{(API_PRODUCT)}}/600`

Params Authorization Headers (8) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☐ JSON ▼

```
1 {
2   "category-id": 1,
3   "product-name": "test update",
4   "weight": "100g",
5   "unit-price": 100,
6   "units-in-stock": 10
7 }
```

ProductManagementAPI / Update Product / TC0219 - UpdateProduct_ValidCredentials_Success

PUT ▼ `{{(API_PRODUCT)}}/600`

Params Authorization Headers (8) Body Scripts Settings

Pre-req

Post-res

```
1 var jsonData = pm.response.json();
2
3 // Check status code is 200
4 pm.test("Status code is 200", function () {
5   pm.response.to.have.status(200); // Verify status code is 200 (OK)
6 });
7
8 // Validate response schema
9 pm.test('Schema validation', () => {
10   pm.response.to.have.jsonSchema(JSON.parse(pm.environment.get('SCHEMA_BASIC'))); // Validate JSON schema
11 });
12
13 // Check success message in response
14 pm.test("Response contains expected values", function () {
15   pm.expect(jsonData).to.have.property('message', "Update product Success"); // Verify success message
16 });
17
18 // Check updated product details in response
19 pm.test("Response contains expected values", function () {
20   pm.expect(jsonData.data).to.have.property("category-id", 1); // Verify category ID
21   pm.expect(jsonData.data).to.have.property("product-name", "test update"); // Verify product name
22   pm.expect(jsonData.data).to.have.property("weight", "100g"); // Verify weight
23   pm.expect(jsonData.data).to.have.property("unit-price", 100); // Verify unit price
24   pm.expect(jsonData.data).to.have.property("units-in-stock", 10); // Verify units in stock
25 });
26
```

Body Cookies Headers (4) Test Results (4/4)

All Passed Skipped Failed ↺

PASS Status code is 200

PASS Schema validation

PASS Response contains expected values

PASS Response contains expected values

6. Delete Product

ProdManagementAPI / Delete Product / TC0225 - DeleteProduct_ValidCredentials_Success

DELETE {{API_PRODUCT}}/600

Params Authorization Headers (6) Body Scripts Settings

Auth Type

Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token {{ACCESS_TOKEN}}

ProdManagementAPI / Delete Product / TC0225 - DeleteProduct_ValidCredentials_Success

DELETE {{API_PRODUCT}}/600

Params Authorization Headers (6) Body Scripts Settings

Pre-req

Post-res

```
1 var jsonData = pm.response.json();
2
3 // Validate response schema
4 pm.test('Schema validation', () => {
5   pm.response.to.have.jsonSchema(JSON.parse(pm.environment.get('SCHEMA_BASIC'))); // Validate JSON schema
6 });
7
8 // Check status code is 200
9 pm.test("Status code is 200", function () {
10   pm.response.to.have.status(200); // Verify status code is 200 (OK)
11 });
12
13 // Check success message in response
14 pm.test("Response contains the correct message", function () {
15   pm.expect(jsonData.message).to.equal("Delete product Success"); // Verify success message
16 });
17
```

Body Cookies Headers (4) Test Results (3/3)

All Passed Skipped Failed

PASS Schema validation

PASS Status code is 200

PASS Response contains the correct message