

## 1. Scope of the Test Plan

### Frontend (ReactJS)

- User Interface (UI): Verify that all visual components (such as buttons, forms, lists, etc.) are functional and styled correctly.
- User Authentication: Validate the login, registration, and credential verification flow.
- Dashboard: Ensure that the display of product and order data works correctly.
- Product and Order Lists: Verify that products and orders are loaded correctly from the backend, displayed correctly in the UI, and that actions (edit, delete, add) work as expected.
- Form Validations: Validate required fields and restrictions (such as email format, password length, etc.).
- UI Performance: Verify load times and overall performance.

### Backend (C# API) with postman

- User Authentication: Verify login functionality and error handling for incorrect or missing credentials.
- Product Management: Ensure that RESTful APIs for creating, reading, updating, and deleting products (CRUD) are implemented correctly.
- Order Management: Verify that APIs for handling creating, updating, deleting, and viewing orders are working correctly.
- Error Handling and Validations: Ensure that the backend correctly handles invalid input, errors, and exceptions (e.g. missing or incorrect product data).
- User Authorization: Check that protected routes are properly restricted and that only authenticated users can access certain APIs.
- Integration and Communication: Verify that communication between the frontend and backend (via APIs) is seamless and data is properly synchronized.

## 2. Testing Process Objectives

The testing process will encompass both frontend and backend functionalities. The primary types of tests to be conducted include functional, integration, accessibility, validation, and API tests.

### Validation

The following pages will be analyzed: Home, Login, Dashboard, Products and Orders page.

For these pages, manual test cases will be implemented to validate the content and functionality of the website as well as the integration with the API.

The designed tests will be automated using Cypress and Postman and swagger for the connection to the API during backend tests.

### **Backend**

The following entities will be analyzed: User, Product, Order.

For each part, tests were designed to validate the functionality, defining

**User:** Login user

- User with correct password
- Incorrect user and password
- Both fields empty
- Incorrect user and correct password

**Product:** Add product, update product data, delete a product, show all products.

**Orders:** Create a new order, update an order, delete an order, show all orders.

### **Resources:**

The resources required to carry out the execution of the Test Plan will be the following:

- Environment: Windows 10, internet access, project running locally (API and web app).
- Tools: Dotnet 8, Node, Java 17, Maven. VS Code, Postman.
- Data set: access to test data to include in Postman Runners ("Data set for Postman - JSON files" folder).
- Browsers: Chrome, Edge, Electron.

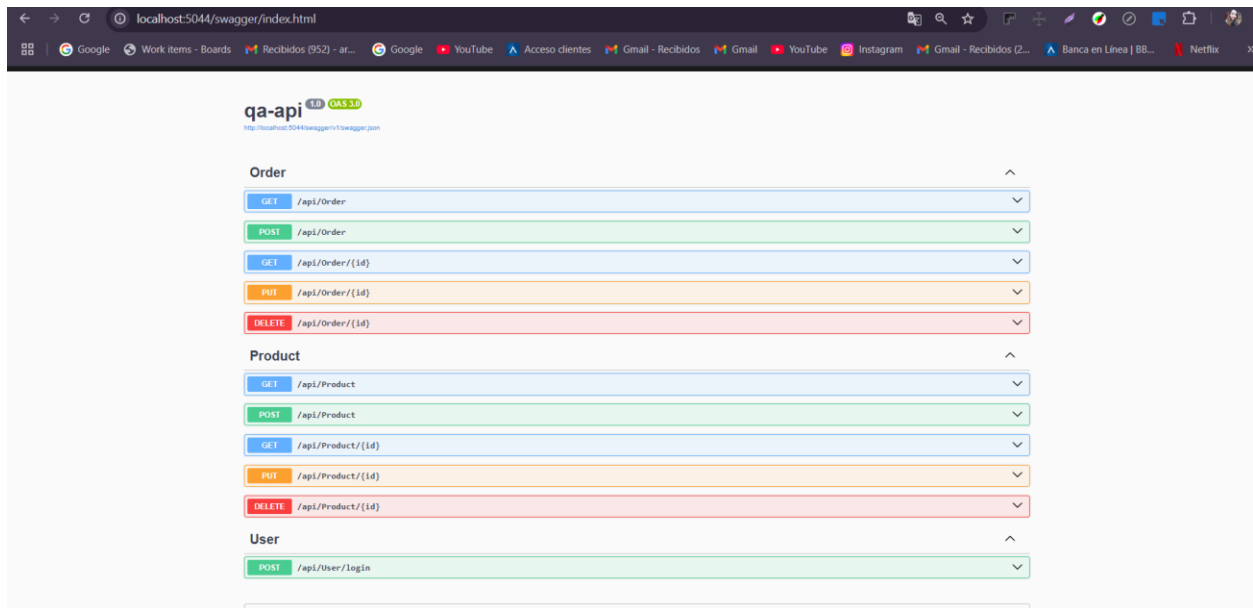
### **Classification of severity**

According to its importance and margin of attention:

- Critical: the failure found is in an essential functionality, or prevents some process in its entirety.
- High: the failure detected in a main process.
- Medium: the failure detected does not limit the operation of other processes, but is relevant.
- Low: the detected bug does not directly influence a functionality.
- Very low: the bug found does not influence a functionality, mainly visual errors.

### **Link repository**

# TEST CHALLENGE ORIANA ARMAS



- Interoperability Verification: Validate that communication between the frontend (ReactJS) and the backend (C# API) is efficient and accurate.
- User Experience (UX) Validation: Verify that the user interface is intuitive, accessible, and free of visual glitches.
- Specific Objectives for the Frontend
  - Verify that UI elements are interactive and functional.
  - Verify that form validations (such as login and product creation) are implemented correctly.
  - Ensure that navigation between different dashboard pages works correctly.
- Specific Objectives for the Backend
  - Validate that the APIs correctly implement CRUD functionalities for products and orders.
  - Ensure that authentication and authorization handling works correctly to prevent unauthorized access.

## Manual testing test case

### LOGIN

1. User and password correct

Response the app validates correctly and displays the token.

Reproduction steps:

1. Enter the user
2. Enter the password
3. Click on the login button

## 4. System responses

- [Login](#)
- [Dashboard](#)
- [Products](#)
- [Orders](#)

### Login

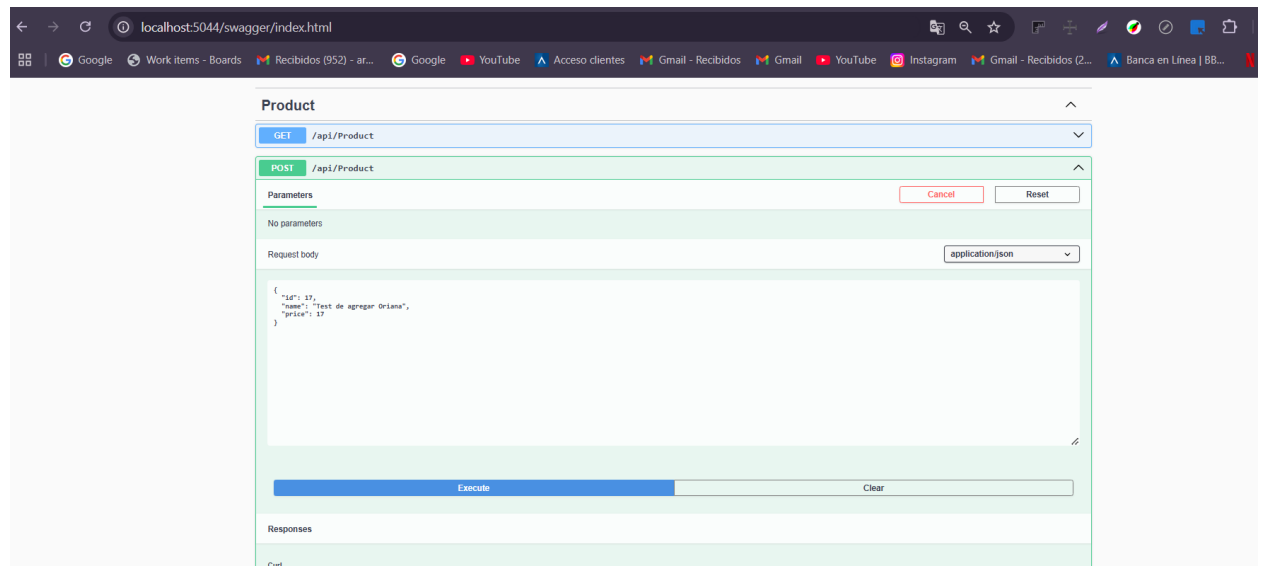
testuser	*****	Login
----------	-------	-------

Logged in with token: sampletoken

When running the tests with POSTMAN Products an array appears with the data entered manually, but initially the application had empty values, it means that I have passed this data through the backend, data has been added, modified and deleted.

### Product

#### Add PRODUCT FOR API WITH METHOD POST CASE #1



# TEST CHALLENGE ORIANA ARMAS

## Result

localhost:3000/products

- Login
- Dashboard
- Products
- Orders

**Product List**

- Test a ver si cambia - \$80
- prueba - \$300
- prueba - \$300
- pruebaaa - \$10
- pruebaaa - \$10
- pruebaaa - \$10
- Test de agregar producto - \$1000
- Test de agregar producto - \$1000**
- Test de agregar Oriana - \$17**

## CASE 2: PUT MODIFY PRODUCT FOR API WITH METHOD PUT

200 OK No links

GET /api/Product/{id}

PUT /api/Product/{id}

Parameters

Name	Description
id	required Integer(\$int32) (path)

Request body

application/json

```
{  "id": 17,  "name": "test de ori modificado",  "price": 10}
```

Execute Clear

Responses

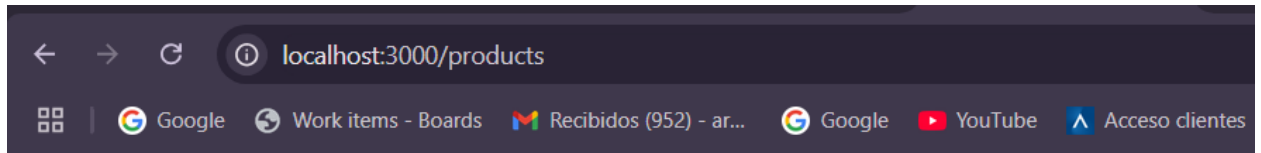
Curl

```
curl -X 'PUT' \
```

Presione 'AV PAG' en el teclado para tomar una captura.

## Result

## TEST CHALLENGE ORIANA ARMAS

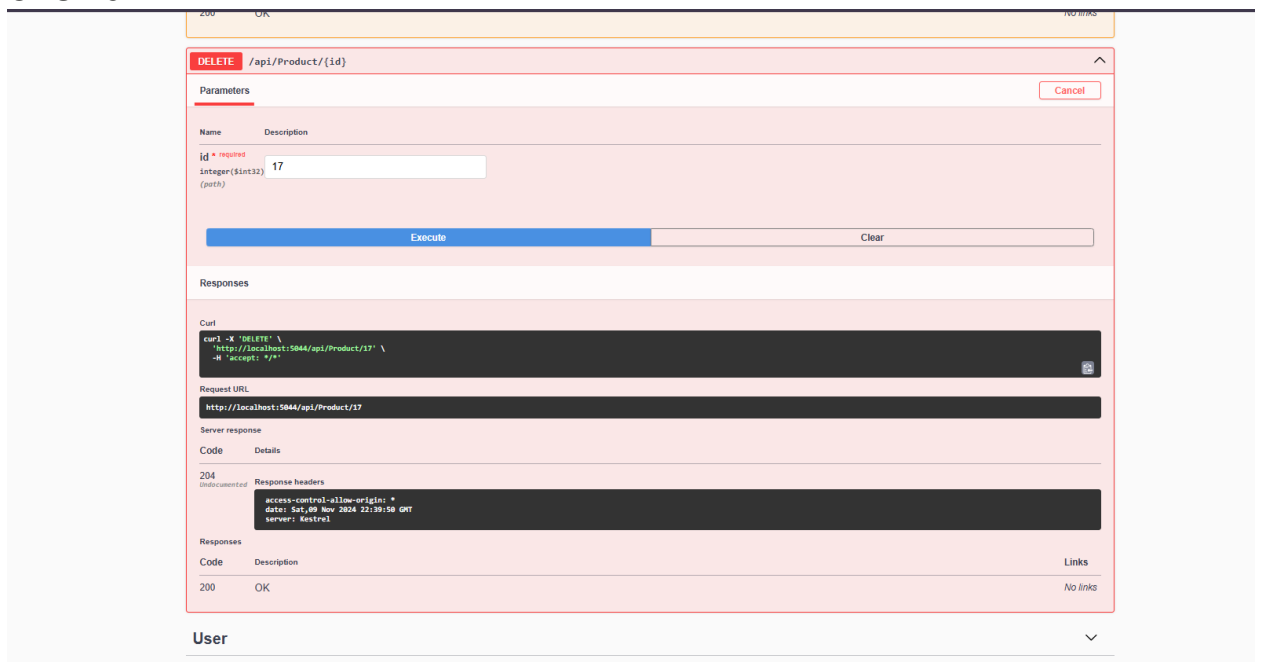


- [Login](#)
- [Dashboard](#)
- [Products](#)
- [Orders](#)

### Product List

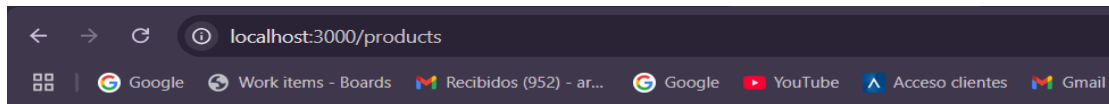
- Test a ver si cambia - \$80
- prueba - \$300
- prueba - \$300
- pruebaaaa - \$10
- pruebaaaa - \$10
- pruebaaaa - \$10
- Test de agregar producto - \$1000
- Test de agregar producto - \$1000
- Test de Ori modificado - \$10

### PRODUCT FOR API WITH METHOD DELETE CASE 3 DELETE:



Result:

# TEST CHALLENGE ORIANA ARMAS



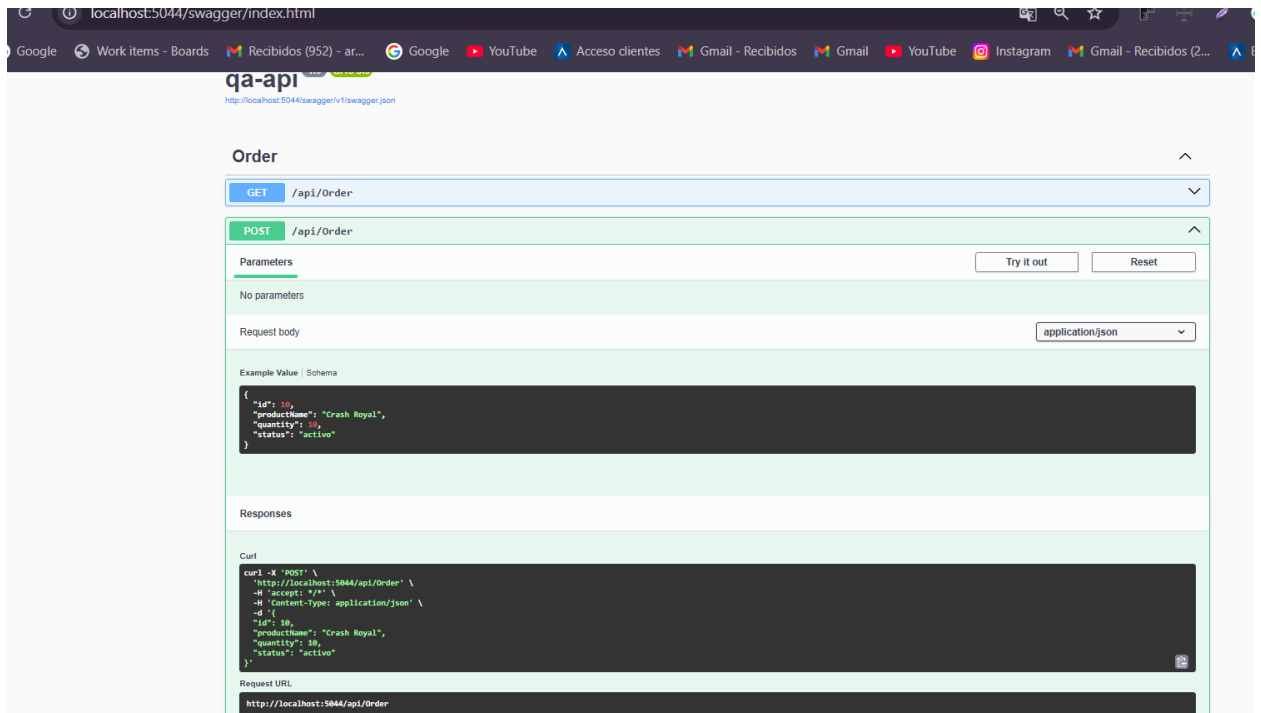
- [Login](#)
- [Dashboard](#)
- [Products](#)
- [Orders](#)

## Product List

- Test a ver si cambia - \$80
- prueba - \$300
- prueba - \$300
- pruebaaaa - \$10
- pruebaaaa - \$10
- pruebaaaa - \$10
- Test de agregar producto - \$1000
- Test de agregar producto - \$1000

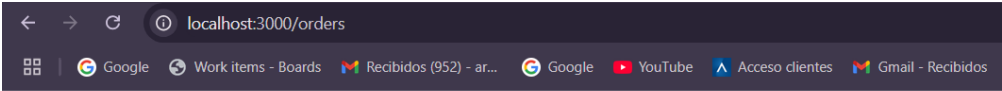
## CASE 1

## CREATE ORDER FOR API WITH METHOD POST



RESULT:

# TEST CHALLENGE ORIANA ARMAS

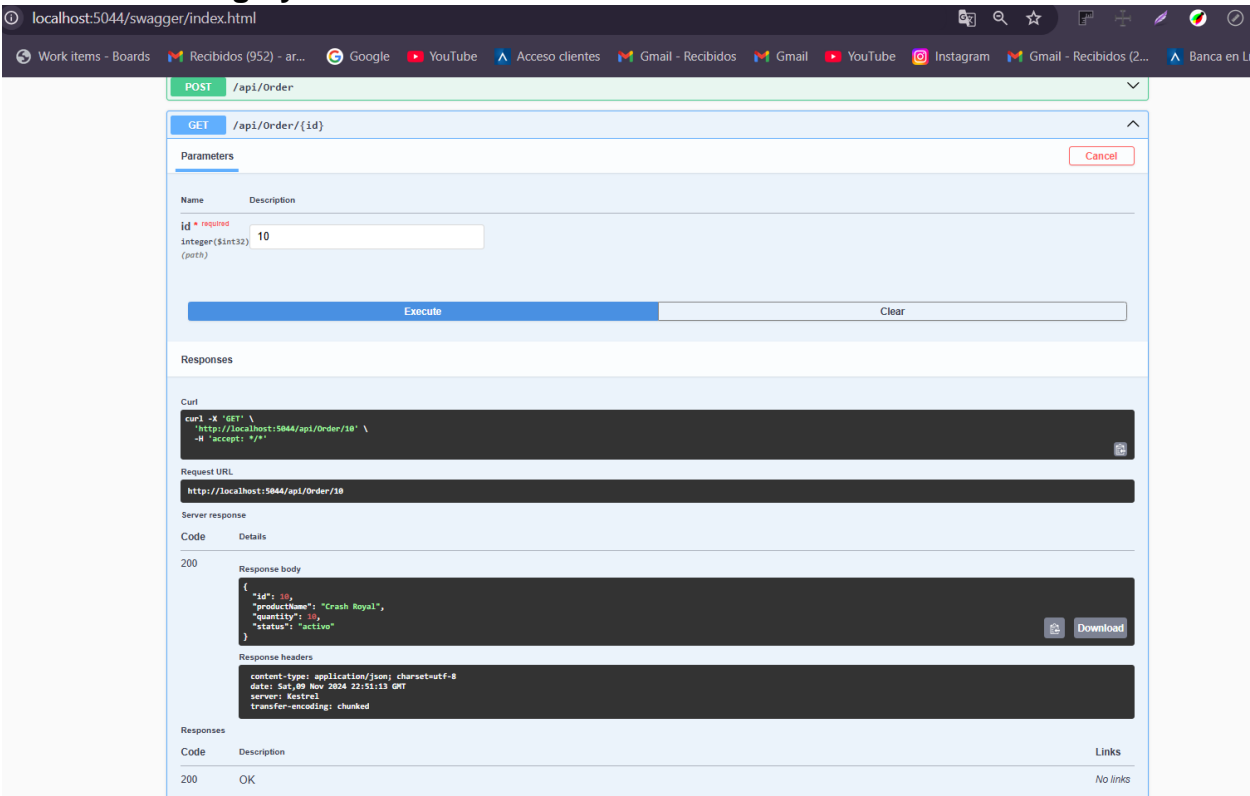


- [Login](#)
- [Dashboard](#)
- [Products](#)
- [Orders](#)

## Order List

- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- string - 0 - string
- Crash Royal - 10 - activo

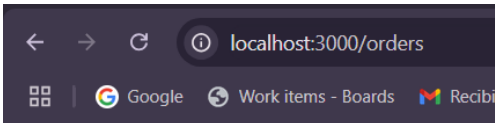
## CASE 2 Looking by ID



## CASE 3 : PUT



# TEST CHALLENGE ORIANA ARMAS



- [Login](#)
- [Dashboard](#)
- [Products](#)
- [Orders](#)

## Order List

- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- Galletas - 10 - compra
- string - 0 - string
- Crash Royal - 10 - VENDIDO

## CASE 4: DELETE

DELETE /api/Order/{id}

Parameters

Name	Description
id * required	
Integer (\$int32)	10
(path)	

Execute

Clear

Responses

Curl

curl -X 'DELETE' \n "http://localhost:5044/api/Order/10" \n -H 'accept: \*/\*' \n

Request URL

http://localhost:5044/api/Order/10

Server response

Code	Details
204	Response headers
Undocumented	access-control-allow-origin: * \n date: Sat, 09 Nov 2024 22:56:06 GMT \n server: Kestrel

Responses

Code	Description	Links
200	OK	No links

Product

## USERS CASE 1: POST ADD

# TEST CHALLENGE ORIANA ARMAS

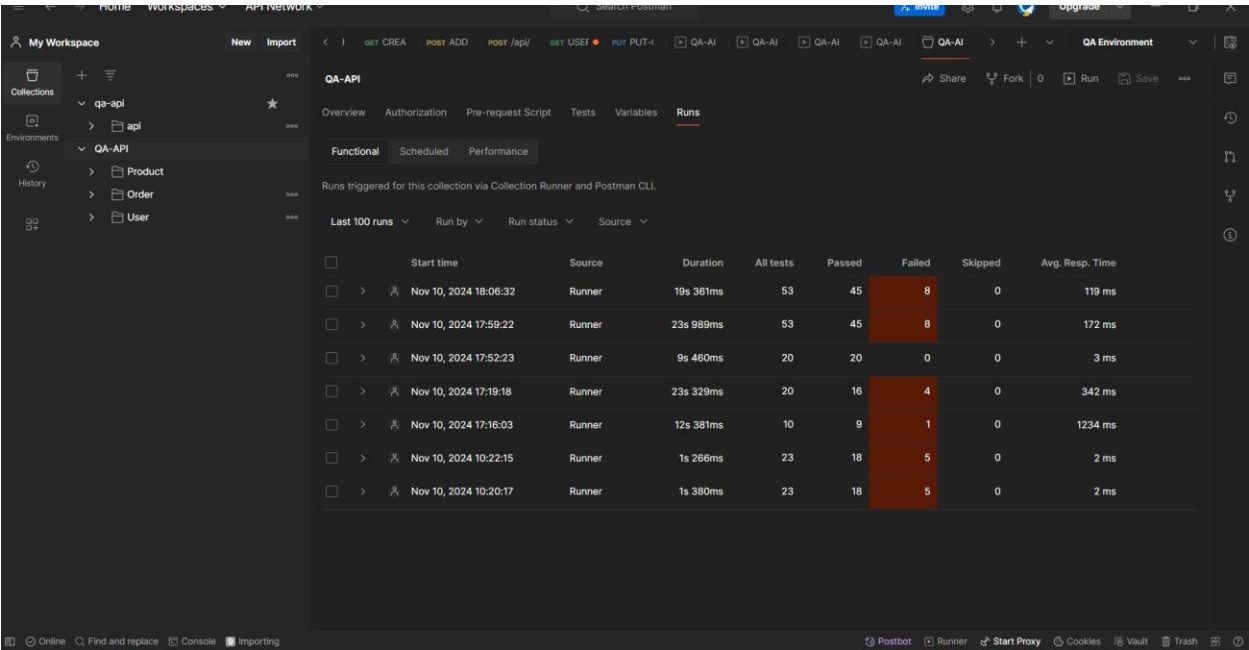
The image shows a Swagger UI interface for a REST API. The top navigation bar includes links for 'Order', 'Product', and 'User'. The 'User' section is expanded, showing a 'POST /api/User/login' endpoint. The 'Parameters' tab is selected, showing no parameters. The 'Request body' tab is also selected, showing a JSON body: `{ "username": "testuser", "password": "password" }`. The 'Responses' tab is selected, showing a 200 status code with a JSON response: `{ "token": "sampletoken" }`. The 'Curl' tab shows the corresponding curl command: `curl -X POST \ "http://localhost:5044/api/user/login" \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{ "username": "testuser", "password": "password" }`. The 'Request URL' is `http://localhost:5044/api/user/login`. The 'Server response' is shown with a 200 status code and the response body: `{ "token": "sampletoken" }`.

If any data is included that is not in the database it will look like this

The image shows a Swagger UI interface for a REST API. The top navigation bar includes links for 'Order', 'Product', and 'User'. The 'User' section is expanded, showing a 'POST /api/User/login' endpoint. The 'Parameters' tab is selected, showing no parameters. The 'Request body' tab is also selected, showing a JSON body: `{ "username": "ORIANA", "password": "password" }`. The 'Responses' tab is selected, showing a 401 status code with a JSON response: `{ "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2", "title": "Unauthorized", "status": 401, "traceId": "00-d6d1f84d637278157a0e9381cafb2a97-8cf392d8ad8dc0e-00" }`. The 'Curl' tab shows the corresponding curl command: `curl -X POST \ "http://localhost:5044/api/user/login" \ -H 'accept: */*' \ -H 'Content-Type: application/json' \ -d '{ "username": "ORIANA", "password": "password" }`. The 'Request URL' is `http://localhost:5044/api/user/login`. The 'Server response' is shown with a 401 status code and the response body: `{ "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2", "title": "Unauthorized", "status": 401, "traceId": "00-d6d1f84d637278157a0e9381cafb2a97-8cf392d8ad8dc0e-00" }`. The 'Response headers' are shown: `access-control-allow-origin: * content-type: application/problem+json; charset=utf-8 date: Sat, 09 Nov 2024 23:05:28 GMT server: Kestrel transfer-encoding: chunked`. The 'Responses' table shows a 200 status code with the description 'OK'.

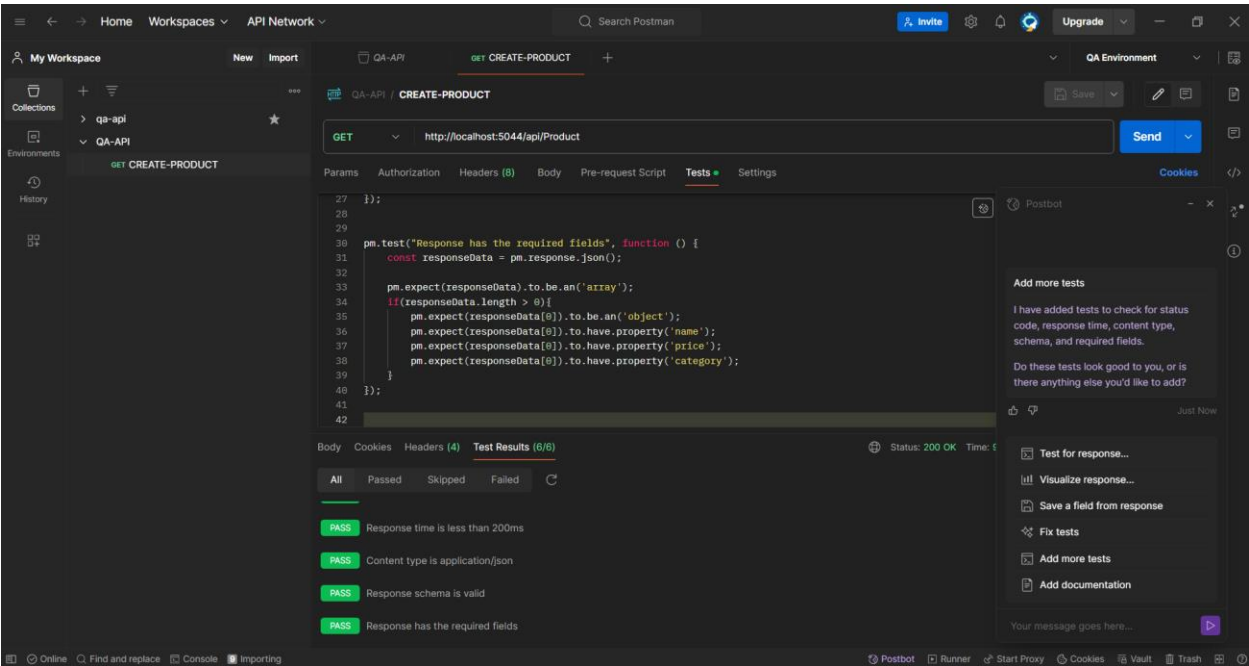
Postman testing

Execute the file



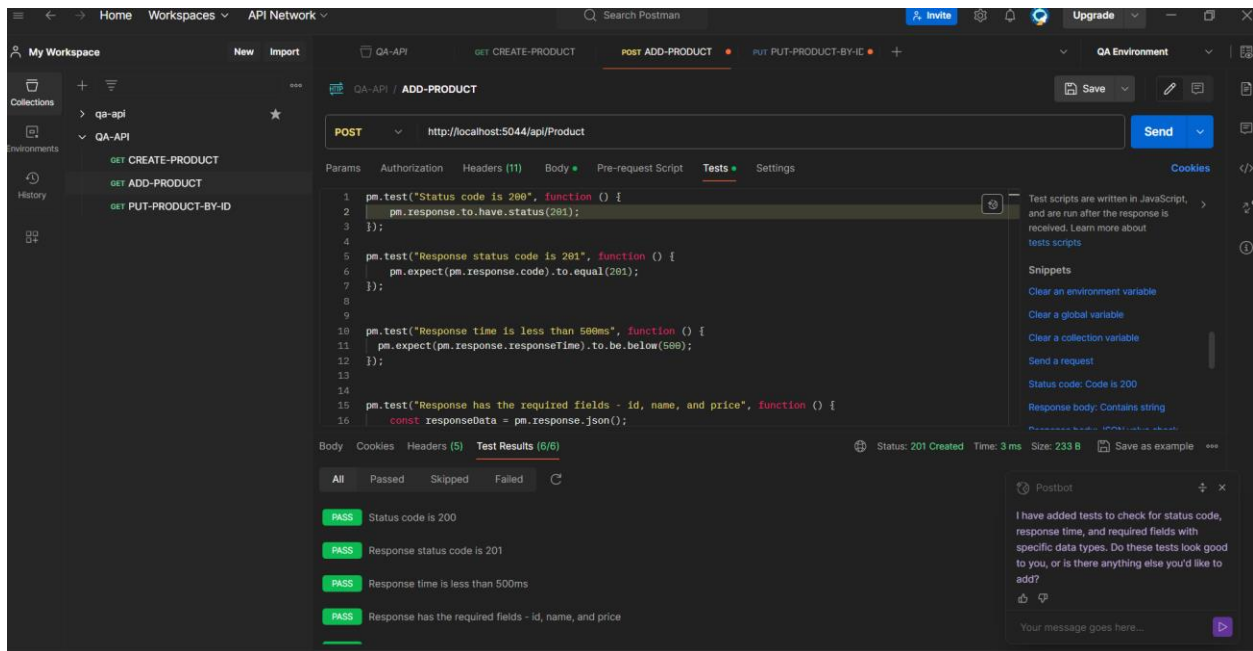
Evidence

Create Product

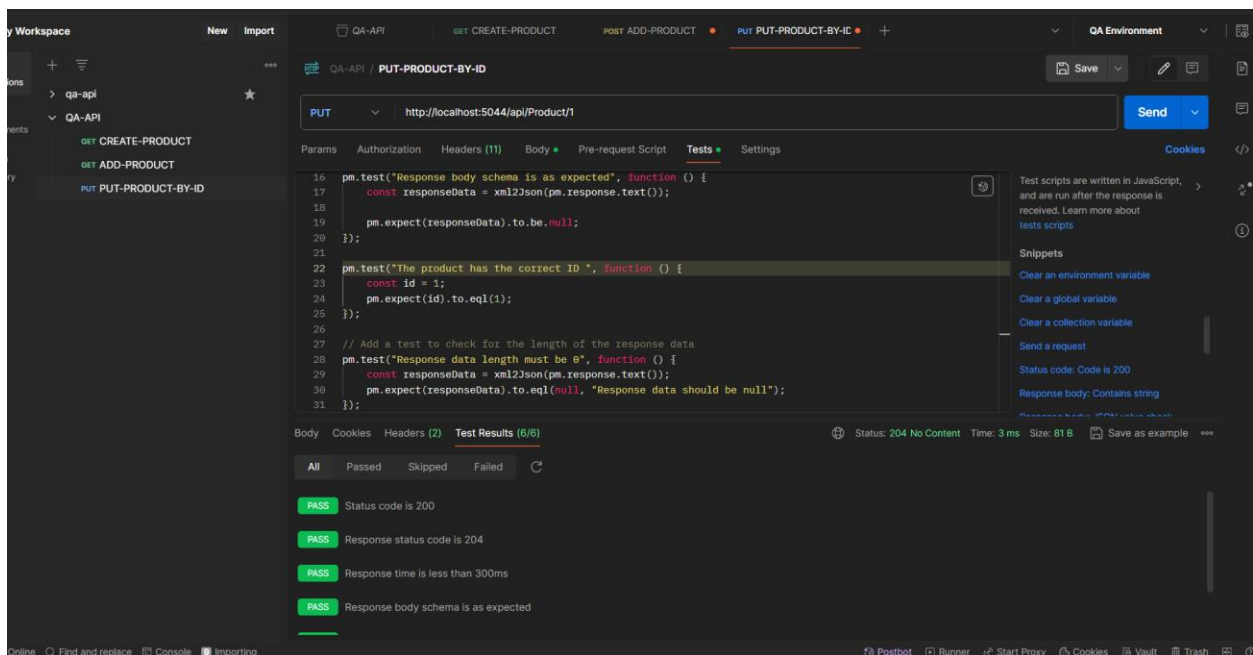


Add Product

# TEST CHALLENGE ORIANA ARMAS

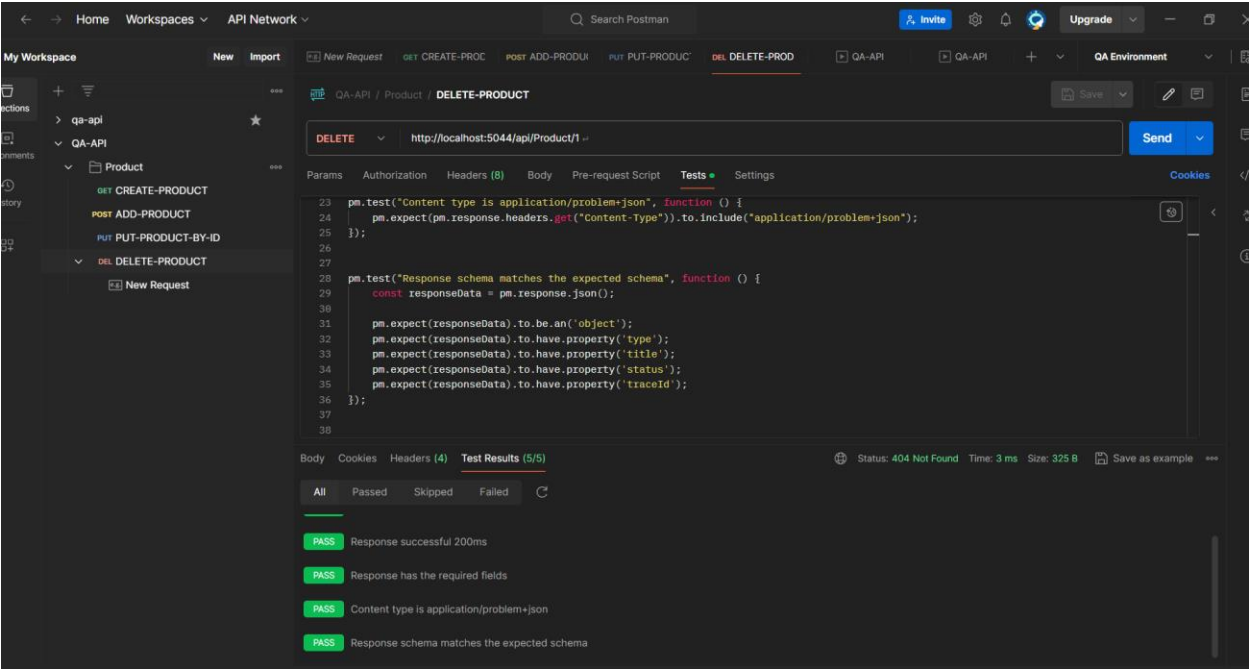


## PUT-BY ID Product

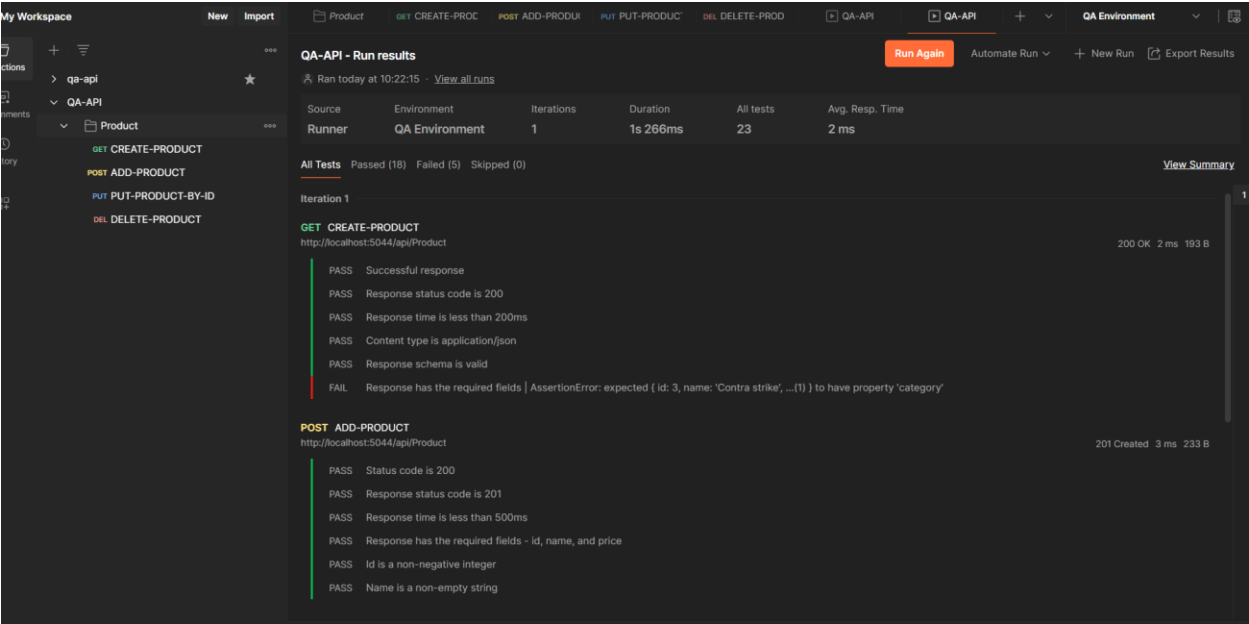


## DELETE Product

# TEST CHALLENGE ORIANA ARMAS



## RUN PRODUCT TEST CASE



# TEST CHALLENGE ORIANA ARMAS

## GET ORDER

The screenshot shows the Postman interface for a GET request to `http://localhost:5044/api/Order`. The request is part of a collection named 'QA-API' under the 'Order' folder. The 'Tests' tab is active, displaying the following JavaScript test scripts:

```
1 pm.test("Response status code is 200", function () {
2   pm.expect(pm.response.code).toEqual(200);
3 });
4
5
6 pm.test("Response has the correct content type - application/json", function () {
7   pm.expect(pm.response.headers.get("Content-Type")).toContain("application/json");
8 });
9
10
11 pm.test("Response is an array", function () {
12   const responseData = pm.response.json();
13   pm.expect(responseData).toBeAn('array');
14 });
15
16
```

The 'Test Results' section shows four passed tests:

- PASS Response status code is 200
- PASS Response has the correct content type - application/json
- PASS Response is an array
- PASS Response time is within an acceptable range

The overall status is 200 OK, with a response time of 3 ms and a size of 150 B.

## POST ADD - ORDER

The screenshot shows the Postman interface for a POST request to `http://localhost:5044/api/Order`. The request is part of a collection named 'QA-API' under the 'Order' folder. The 'Tests' tab is active, displaying the following JavaScript test scripts:

```
15 pm.expect(pm.response.responseTime).toBeBelow(500);
16 });
17
18
19 pm.test("The desired order exists - Puzzle", function () {
20   const responseData = xml2Json(pm.response.text());
21   if (responseData !== null) {
22     pm.expect(responseData.id).toEqual(2); // Verificar el ID
23     pm.expect(responseData.productName).toEqual("Puzzle"); // Verificar el nombre del producto
24     pm.expect(responseData.quantity).toEqual(100); // Verificar la cantidad
25     pm.expect(responseData.status).toEqual("Agotado");
26   }
27 });
28
29 pm.test("The desired order exists - Pizza", function () {
30   const responseData = xml2Json(pm.response.text());

```

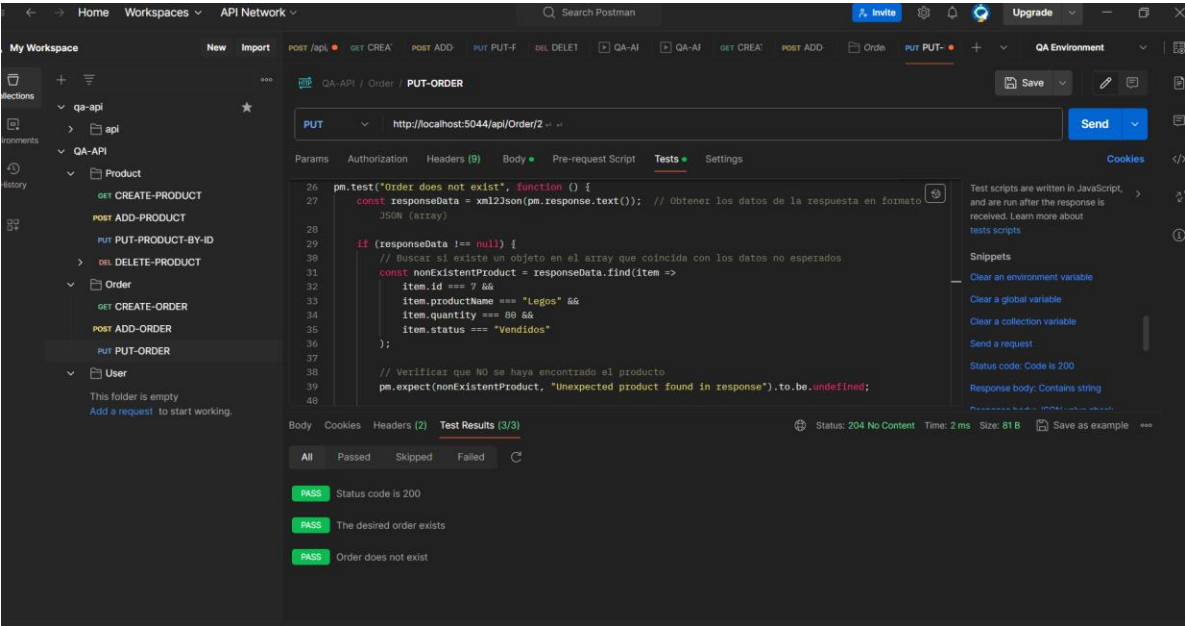
The 'Test Results' section shows four passed tests:

- PASS The desired order exists - Puzzle
- PASS The desired order exists - Pizza
- PASS Exist the order - Legos
- PASS Order does not exist

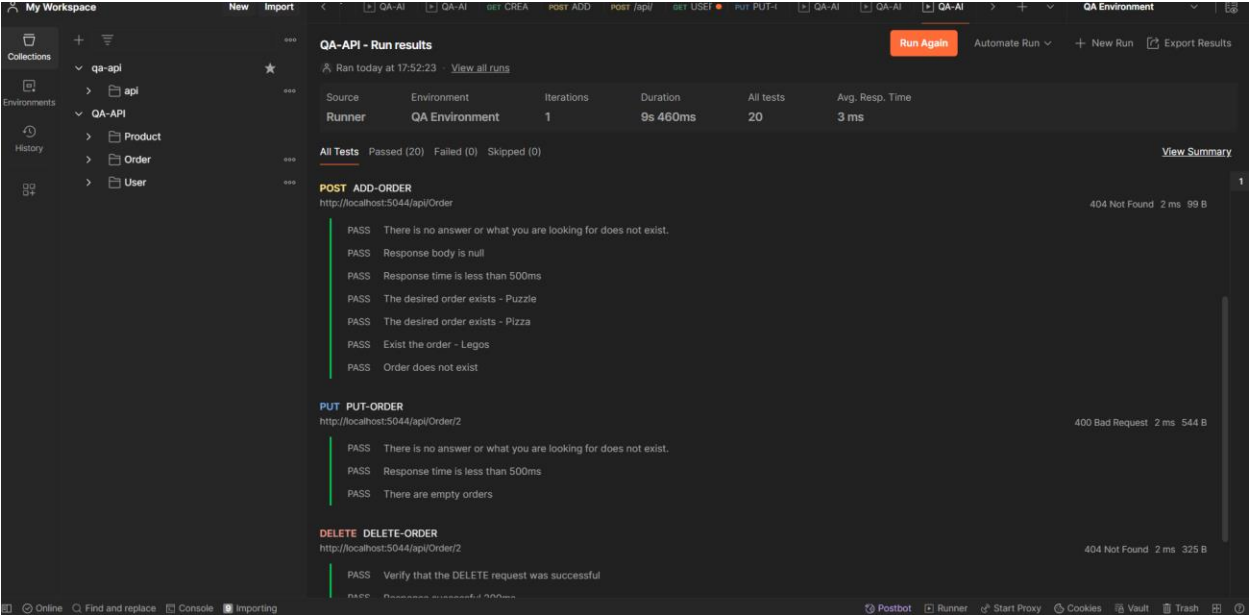
The overall status is 404 Not Found, with a response time of 6 ms and a size of 99 B.

# TEST CHALLENGE ORIANA ARMAS

## PUT ORDER

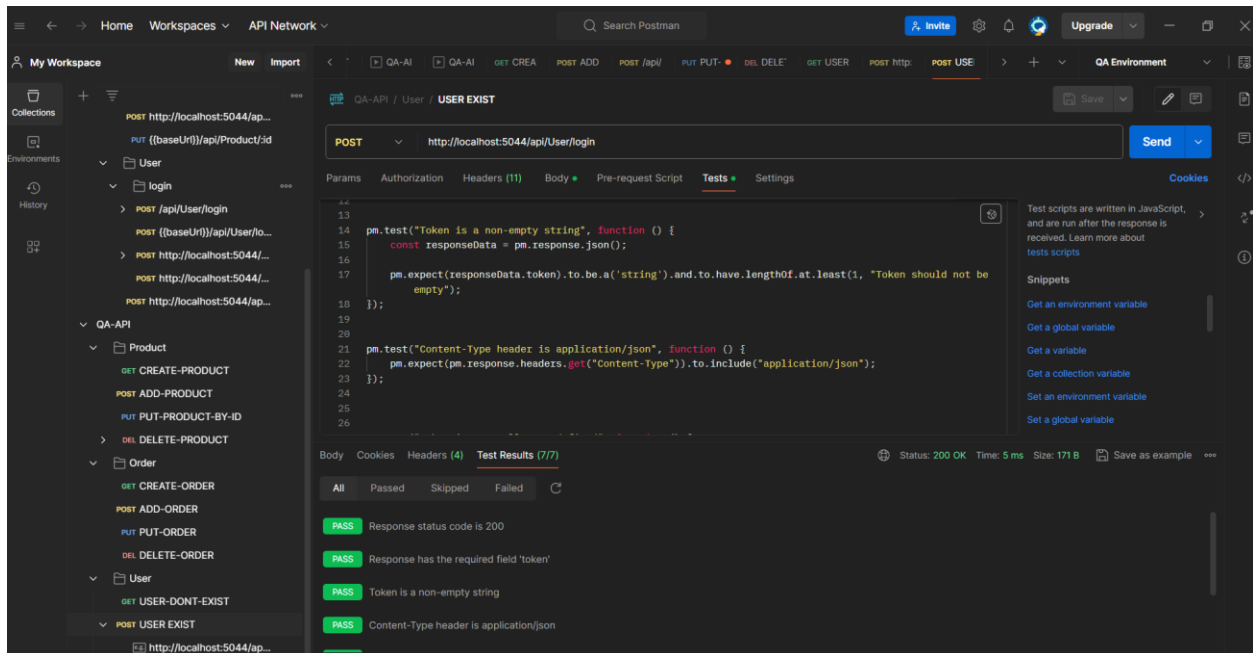


## RUN ORDER

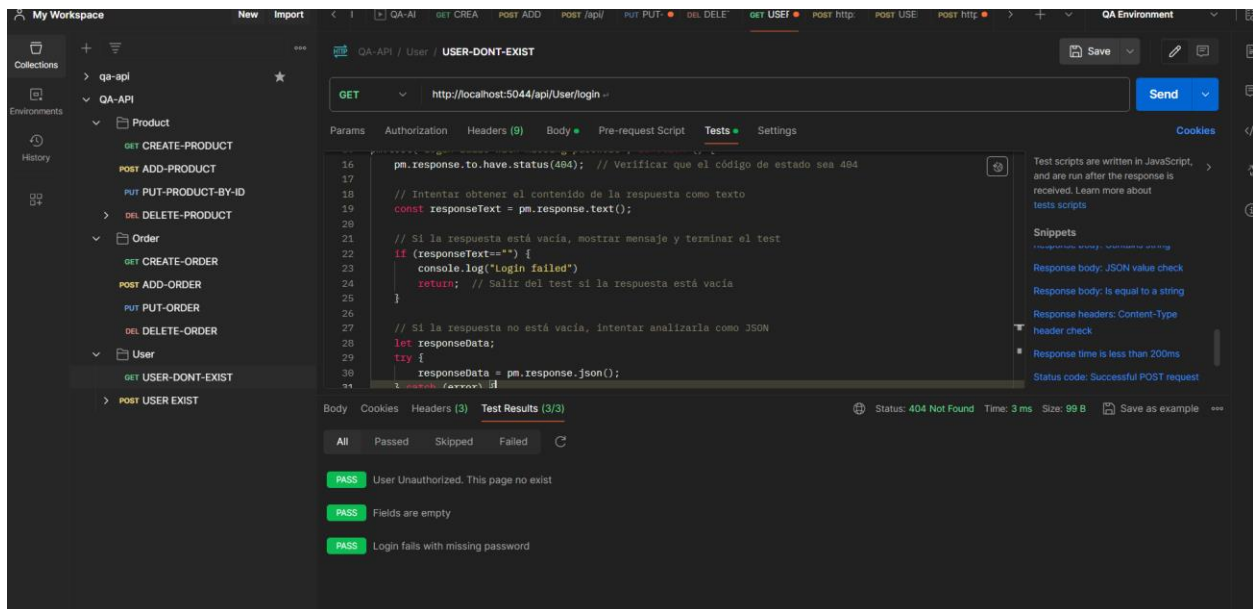


## USER EXIST

# TEST CHALLENGE ORIANA ARMAS



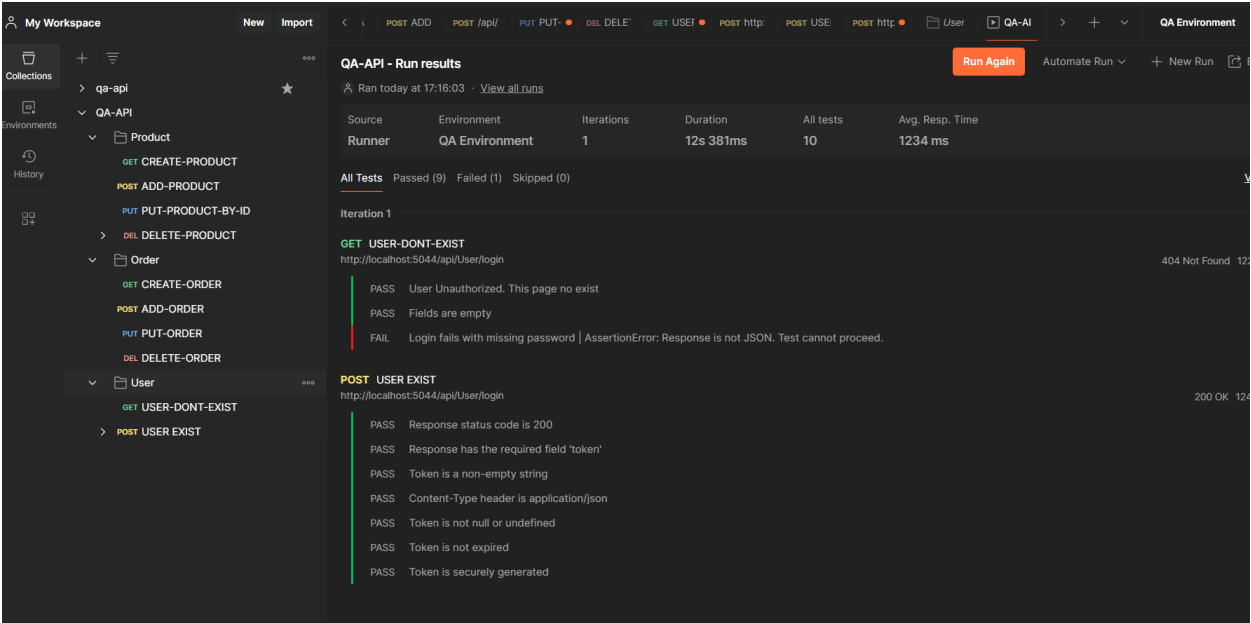
## USER DON'T EXIST



## RUN TEST



# TEST CHALLENGE ORIANA ARMAS



## ANALISYS

When executing the complete program, some elements have failed, either due to test rendering issues that have taken longer, or in some cases the page or the requested information has not been obtained. 8 test cases have failed

## EVIDENCE:

# TEST CHALLENGE ORIANA ARMAS

QA-API - Run results

Run today at 17:59:22 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA Environment	1	23s 989ms	53	172 ms

All Tests Passed (45) **Failed (8)** Skipped (0) [View Summary](#)

Iteration 1

**GET CREATE-PRODUCT**  
http://localhost:5044/api/Product 200 OK 595 ms 193 B

- FAIL Response time is less than 200ms | AssertionError: expected 595 to be below 200
- FAIL Response has the required fields | AssertionError: expected { id: 3, name: 'Contra strike', ...(1) } to have property 'category'

**POST ADD-PRODUCT**  
http://localhost:5044/api/Product 201 Created 553 ms 233 B

- FAIL Response time is less than 500ms | AssertionError: expected 553 to be below 500

**DELETE DELETE-PRODUCT**  
http://localhost:5044/api/Product/1 204 No Content 2 ms 81 B

- FAIL Verify that the DELETE request was successful | AssertionError: expected 204 to equal 404
- FAIL Response has the required fields | JSONError: No data, empty input at 1:1 \*
- FAIL Content type is application/problem+json | AssertionError: the given combination of arguments (undefined and string) is invalid for this assertion. You can use an array, a ...
- FAIL Response schema matches the expected schema | JSONError: No data, empty input at 1:1 \*

**GET USER-DONT-EXIST**  
http://localhost:5044/api/User/login 404 Not Found 2 ms 99 B

- FAIL Login fails with missing password | AssertionError: Response is not JSON. Test cannot proceed.

So 45 test cases have passed

QA-API - Run results

Run today at 17:59:22 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	QA Environment	1	23s 989ms	53	172 ms

All Tests **Passed (45)** Failed (8) Skipped (0) [View Summary](#)

Iteration 1

**GET CREATE-PRODUCT**  
http://localhost:5044/api/Product 200 OK 595 ms 193 B

- PASS Successful response
- PASS Response status code is 200
- PASS Content type is application/json
- PASS Response schema is valid

**POST ADD-PRODUCT**  
http://localhost:5044/api/Product 201 Created 553 ms 233 B

- PASS Status code is 200
- PASS Response status code is 201
- PASS Response has the required fields - id, name, and price
- PASS Id is a non-negative integer
- PASS Name is a non-empty string

**PUT PUT-PRODUCT-BY-ID**  
http://localhost:5044/api/Product/1 204 No Content 1 ms 81 B

- PASS Status code is 200