

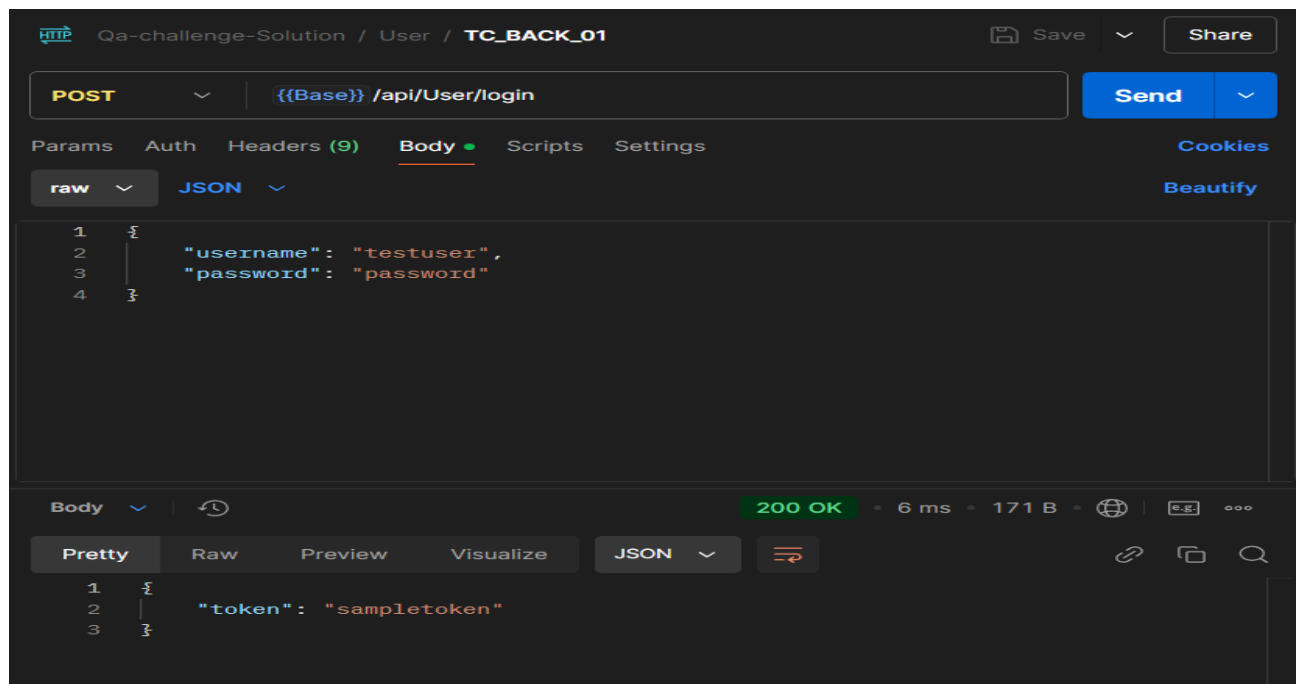
PART 3: Test execution and reporting

1. Test Execution – Backend (C# APIS):

TC_BACK_01: Login with Valid Credentials

Expected result: The API responds with a valid authentication token and status code 200.

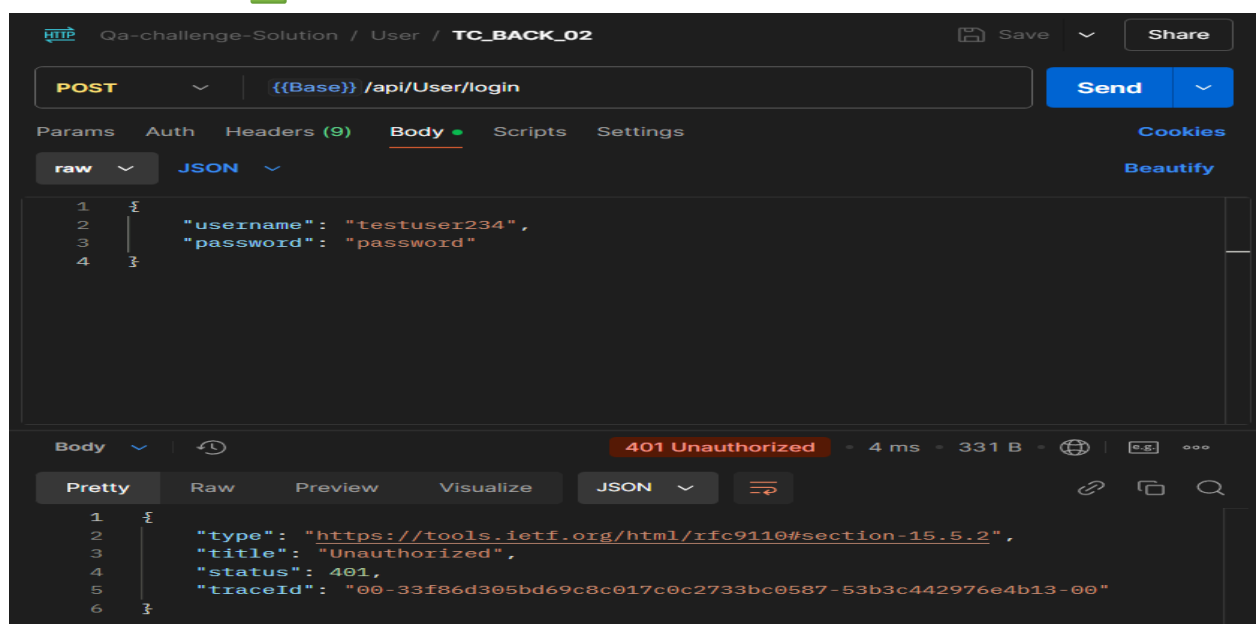
Test result: Pass ✓



TC_BACK_02: Login with invalid user

Expected result: The API responds with an error message and status code 401.

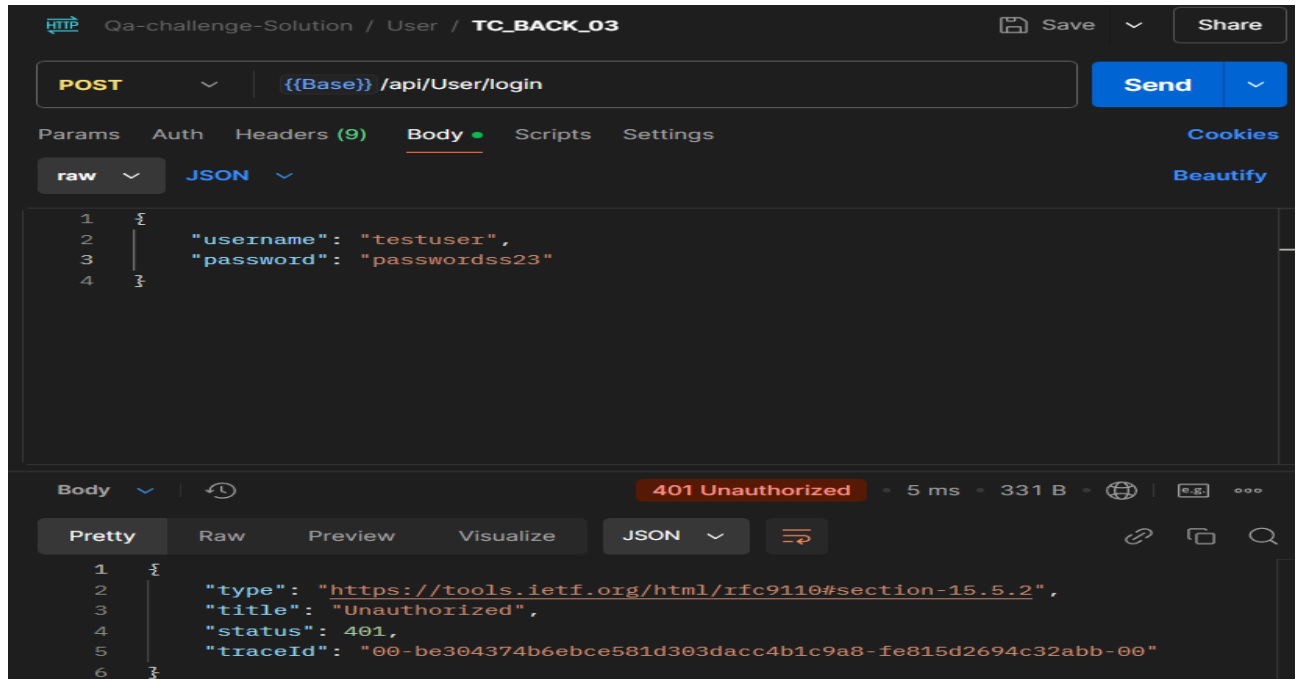
Test result: Pass ✓



TC_BACK_03: Login with invalid password

Expected result: The API responds with an error message and status code 401.

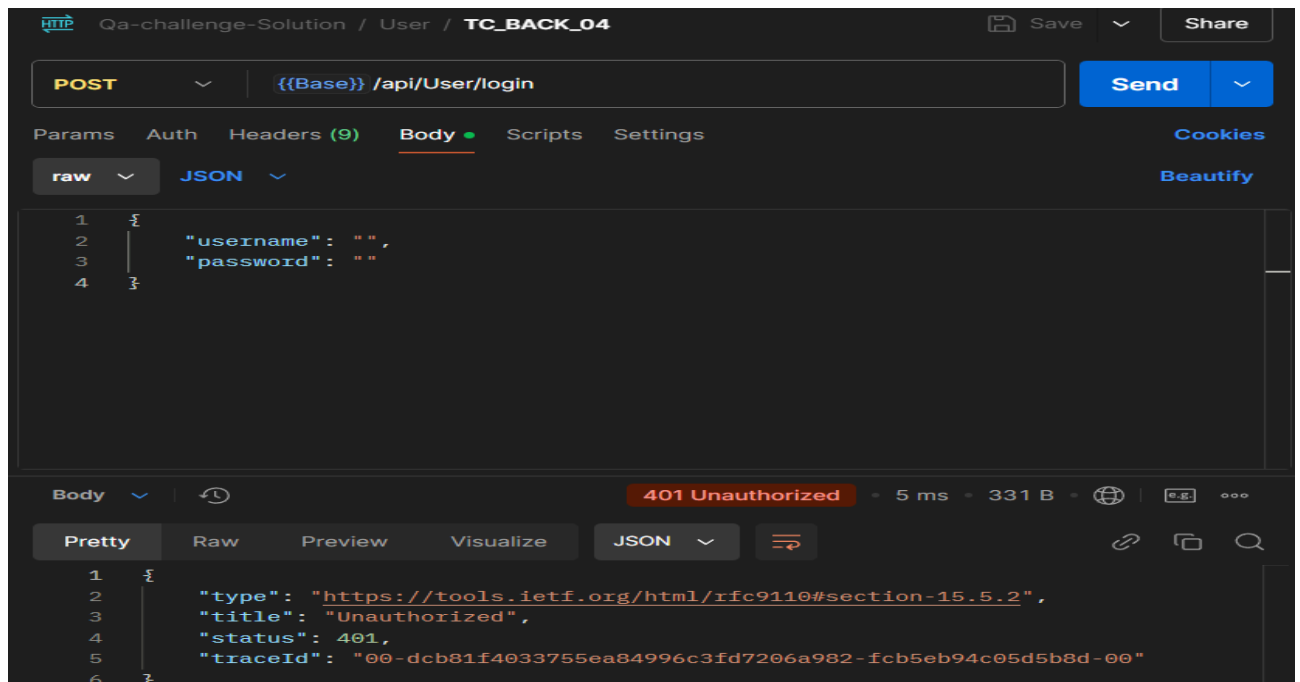
Test result: Pass ✓



TC_BACK_04: Login with both user and password fields empty

Expected result: The API responds with an error message and status code 401.

Test result: Pass ✓



TC_BACK_05: Login with empty user field

Expected result: The API responds with an error message and status code 401.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named "TC_BACK_05". The request is a POST to the endpoint `{{Base}}/api/User/login`. The request body is a JSON object with an empty `username` field and a `password` field. The response is a 401 Unauthorized status with a response time of 5 ms and a body size of 331 B. The response body is a JSON object containing error details.

```
POST {{Base}}/api/User/login
```

```
{
  "username": "",
  "password": "password"
}
```

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2",
  "title": "Unauthorized",
  "status": 401,
  "traceId": "00-f8e104adf882b845c43300a185139e82-cc1b51c9e581f120-00"
}
```

TC_BACK_06: Login with empty password field

Expected result: The API responds with an error message and status code 401.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named "TC_BACK_06". The request is a POST to the endpoint `{{Base}}/api/User/login`. The request body is a JSON object with a `username` field set to "testuser" and an empty `password` field. The response is a 401 Unauthorized status with a response time of 4 ms and a body size of 331 B. The response body is a JSON object containing error details.

```
POST {{Base}}/api/User/login
```

```
{
  "username": "testuser",
  "password": ""
}
```

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2",
  "title": "Unauthorized",
  "status": 401,
  "traceId": "00-6f49f198bf94a9a0a1349e68cd528c03-20f6874c3d165279-00"
}
```

TC_BACK_07: Create Product

Expected result: The API responds with a success message and status code 201.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named 'TC_BACK_07'. The method is 'POST' and the URL is '{{Base}} /api/Product'. The request body is a JSON object:

```
{  "id": 22,  "name": "Product789",  "price": 10}
```

. The response status is '201 Created' with a response time of 2 ms and a body size of 241 B. The response body is a JSON object:

```
{  "id": 22,  "name": "Product789",  "price": 10}
```

.

TC_BACK_08: Consult Product List

Expected result: The API responds with the complete list of products and status code 200.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named 'TC_BACK_08'. The method is 'GET' and the URL is '{{Base}} /api/Product'. The response status is '200 OK' with a response time of 3 ms and a body size of 264 B. The response body is a JSON array of three product objects:

```
[  {    "id": 1,    "name": "Product1",    "price": 10  },  {    "id": 2,    "name": "Product2",    "price": 10  },  {    "id": 3,    "name": "Product3",    "price": 100  }]
```

.

TC_BACK_09: Create Product with Existing ID

Expected result: The API responds with an error message and status code 409, indicating ID conflict.

Test result: Fail ❌

The screenshot shows a REST client interface for a POST request to `{{Base}}/api/Product`. The request body is a JSON object: `{ "id": 3, "name": "Product4", "price": 150 }`. The response status is `201 Created`, which is incorrect for this test case. A red arrow points to the `"name": "Product4"` field in the response body.


```
1 {
2   "id": 3,
3   "name": "Product4",
4   "price": 150
5 }
```

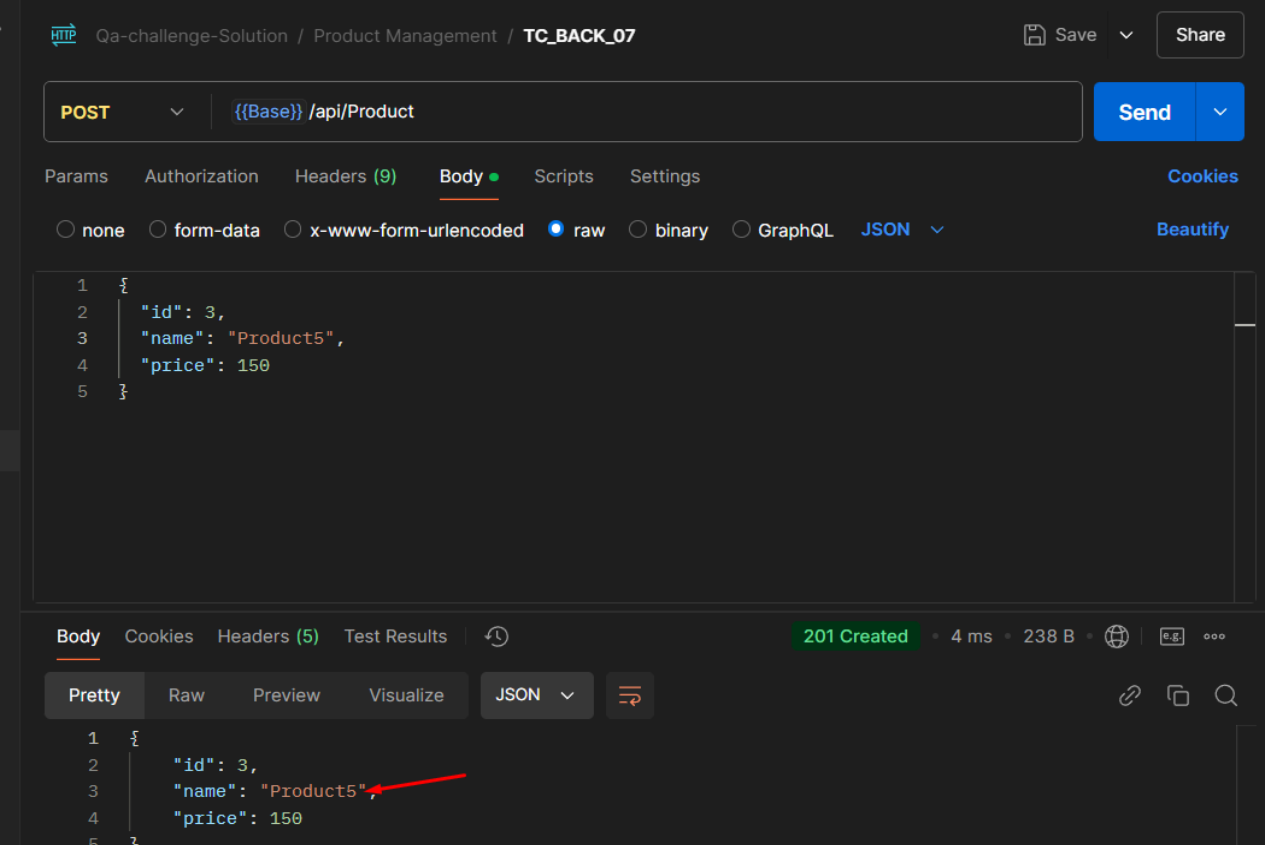
The screenshot shows a REST client interface for a POST request to `{{Base}}/api/Product`. The request body is a JSON object: `{ "id": 3, "name": "Product5", "price": 150 }`. The response status is `201 Created`, which is incorrect for this test case. A red arrow points to the `"name": "Product5"` field in the response body.

```
1 {
2   "id": 3,
3   "name": "Product5",
4   "price": 150
5 }
```

TC_BACK_10:Edit Product by ID

Expected result: The API responds with a success message and status code 204

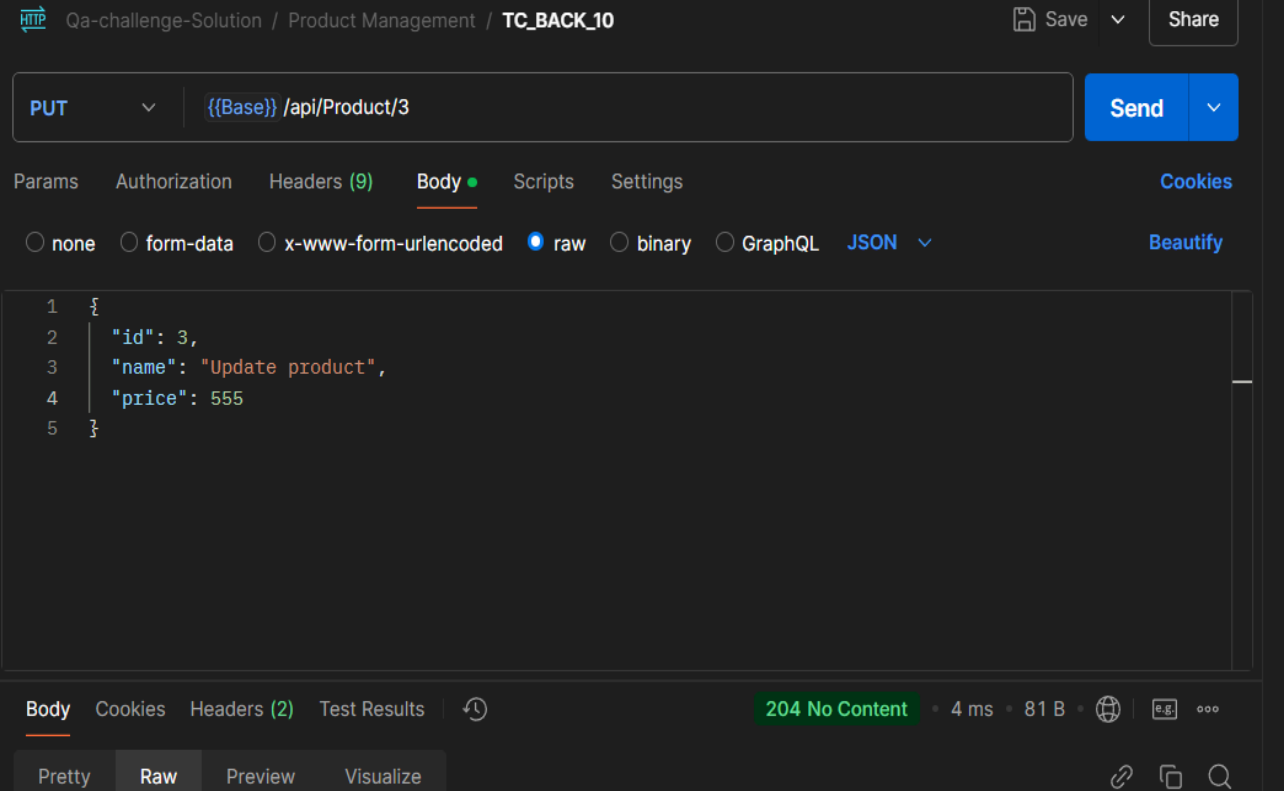
Test result: Pass .



The screenshot shows the Postman interface for a test case named TC_BACK_07. The URL is `{{Base}} /api/Product` and the method is **POST**. The **Body** tab is selected, showing a JSON payload:

```
{  "id": 3,  "name": "Product5",  "price": 150}
```

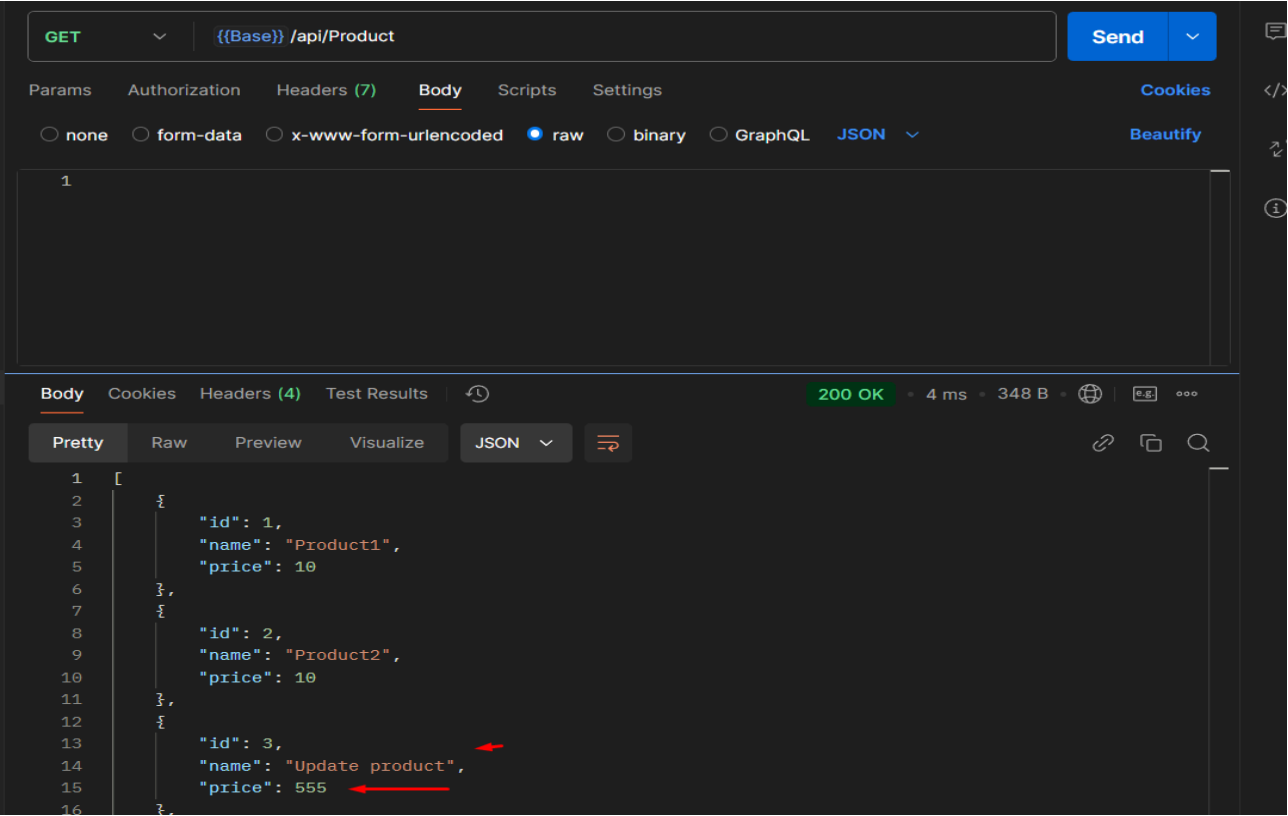
. The status bar at the bottom indicates a successful response: **201 Created** with a response time of 4 ms and a body size of 238 B. The **Body** tab is also expanded, showing the same JSON payload in a pretty-printed view.



The screenshot shows the Postman interface for a test case named TC_BACK_10. The URL is `{{Base}} /api/Product/3` and the method is **PUT**. The **Body** tab is selected, showing a JSON payload:


```
{  "id": 3,  "name": "Update product",  "price": 555}
```

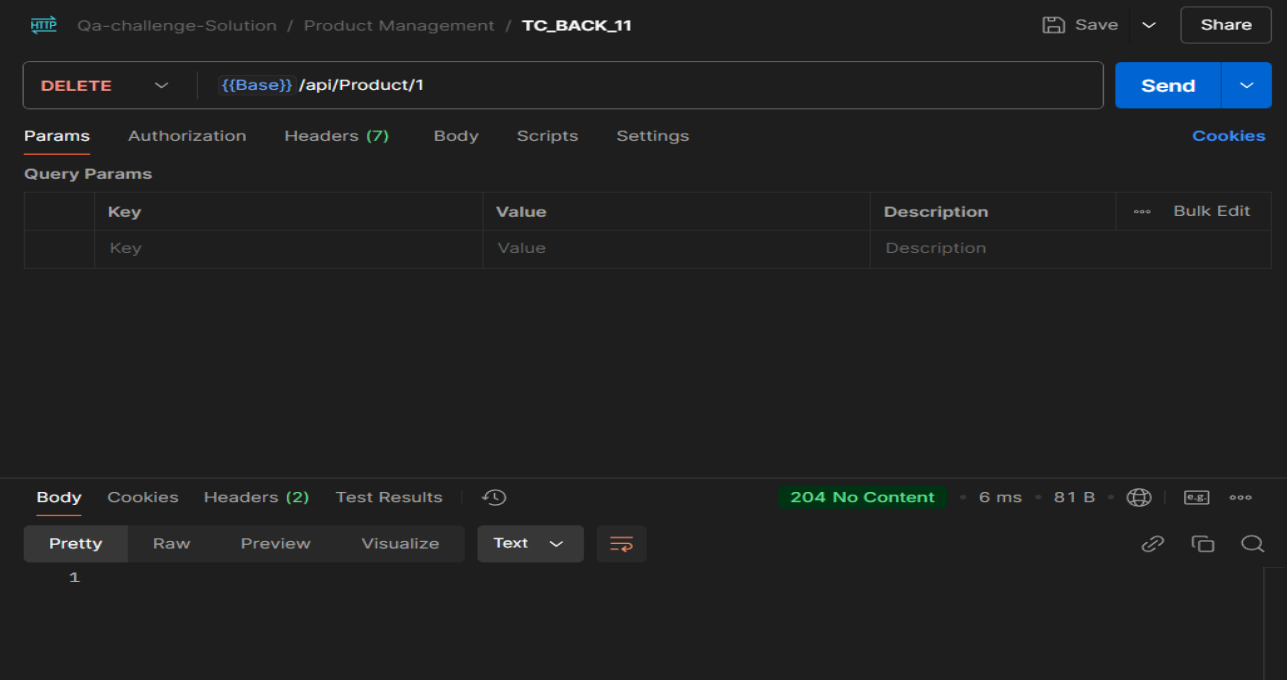
. The status bar at the bottom indicates a successful response: **204 No Content** with a response time of 4 ms and a body size of 81 B. The **Body** tab is also expanded, showing the same JSON payload in a pretty-printed view.



TC_BACK_11:Delete Product by ID

Expected result: The API responds with a success message and status code 204

Test result: Pass 



TC_BACK_12: Consult Product by ID

Expected result: The API responds with a success message and status code 200.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named 'TC_BACK_12'. The request is a GET to the endpoint '{{Base}} /api/Product/3'. The response is a 200 OK status with a response time of 4 ms and a body size of 192 B. The response body is displayed in JSON format:

```
{
  "id": 3,
  "name": "Update product",
  "price": 555
}
```

TC_BACK_13: Create Order with Invalid Quantity

Expected result: The API responds with an error message and status code 400 or 422.

Test result: Fail ✗

The screenshot shows a REST client interface for a test case named 'TC_BACK_13'. The request is a POST to the endpoint '{{Base}} /api/Order'. The request body is a JSON object with an invalid quantity of -4:

```
{
  "id": 1,
  "productName": "NameOrder1",
  "quantity": -4,
  "status": "Pending"
}
```

The response is a 201 Created status with a response time of 5 ms and a body size of 266 B. The response body is displayed in JSON format:

```
{
  "id": 1,
  "productName": "NameOrder1",
  "quantity": -4,
  "status": "Pending"
}
```


TC_BACK_14: Create Order with Valid Data

Expected result: The API responds with a success message and status code 201.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named TC_BACK_13. The request is a POST to the endpoint `{{Base}}/api/Order`. The body is a JSON object with the following data: `{ "id": 1, "productName": "NameOrder1", "quantity": 30, "status": "Pending" }`. The response status is **201 Created**, with a response time of 12 ms and a body size of 266 B. The response body is displayed in a pretty-printed JSON format, matching the request body.

```
1 {
2   "id": 1,
3   "productName": "NameOrder1",
4   "quantity": 30,
5   "status": "Pending"
6 }
```

Body: **201 Created** • 12 ms • 266 B • [Globe] [e.g.] [More]

Pretty Raw Preview Visualize JSON [More]

```
1 {
2   "id": 1,
3   "productName": "NameOrder1",
4   "quantity": 30,
5   "status": "Pending"
6 }
```

TC_BACK_15: Update Order with Valid Data

Expected result: The API responds with a success message and status code 200 or 204.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named TC_BACK_15. The request is a PUT to the endpoint `{{Base}}/api/Order/1`. The body is a JSON object with the following data: `{ "id": 1, "productName": "NameOrder1", "quantity": 500, "status": "In progress" }`. The response status is **204 No Content**, with a response time of 7 ms and a body size of 81 B. The response body is displayed in a pretty-printed text format, showing the number 1.

```
1 {
2   "id": 1,
3   "productName": "NameOrder1",
4   "quantity": 500,
5   "status": "In progress"
6 }
```

Body: **204 No Content** • 7 ms • 81 B • [Globe] [e.g.] [More]

Pretty Raw Preview Visualize Text [More]

```
1
```

TC_BACK_16: Delete Order with Non-Existent IDs

Expected result: The API responds with an error message and status code 404.

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named TC_BACK_16. The request is a DELETE method to the endpoint `{{Base}} /api/Order/999`. The response is a 404 Not Found status with a response time of 5 ms and a body size of 325 B. The response body is displayed in JSON format:

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",
  "title": "Not Found",
  "status": 404,
  "traceId": "00-41dbfd641ff04cc9544a5c964a43f5db-840815501a28e785-00"
}
```

TC_BACK_17: Delete Order with Non-Numeric ID

Expected result: The API responds with an error message and status code 400

Test result: Pass ✓.

The screenshot shows a REST client interface for a test case named TC_BACK_17. The request is a DELETE method to the endpoint `{{Base}} /api/Order/testdeleteID`. The response is a 400 Bad Request status with a response time of 19 ms and a body size of 416 B. The response body is displayed in JSON format:

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "id": [
      "The value 'testdeleteID' is not valid."
    ]
  },
  "traceId": "00-d592ef6b3c1e3bd8b545bc2aae802de1-fcb8b2f285b1c40a-00"
}
```

TC_BACK_18: View all orders

Expected result: The API responds with a complete list of commands and a status code 200.

Test result: Pass ✓

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

- Header:** HTTP icon, breadcrumb "Qa-challenge-Solution / Orders / TC_BACK_18", "Save" button, and "Share" button.
- Request Bar:** Method "GET", dropdown arrow, and URL "{{Base}} /api/Order/". A "Send" button is on the right.
- Tabs:** "Params", "Auth", "Headers (7)", "Body", "Scripts", "Settings", and "Cookies".
- Query Params Table:**

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		
- Response Bar:** "Body" tab, refresh icon, status "200 OK", and metrics "3 ms", "358 B", "Globe icon", "E.g. icon", and "More icon".
- Response Format:** "Pretty" (selected), "Raw", "Preview", "Visualize", "JSON" dropdown, and "Syntax Highlight" icon.
- Response Body:** A JSON array of three order objects.

```
1  [  
2    {  
3      "id": 1,  
4      "productName": "NameOrder1",  
5      "quantity": -4,  
6      "status": "Pending"  
7    },  
8    {  
9      "id": 2,  
10     "productName": "NameOrder2",  
11     "quantity": 50,  
12     "status": "Complete"  
13   },  
14   {  
15     "id": 3,  
16     "productName": "NameOrder3",  
17     "quantity": 60,  
18     "status": "Complete"  
19   }  
20 ]
```

TC_BACK_19: Search for an order using an existing ID.
Expected result: The API responds with the details of the specific order and a 201 status code
Test result: Pass✔

HTTP

Qa-challenge-Solution / Orders / TC_BACK_19

Save

Share

DELETE

{{Base}} /api/Order/2

Send

Params

Auth

Headers (7)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body

204 No Content

3 ms

81 B

e.g.

Pretty

Raw

Preview

Visualize

Text

1

HTTP

Qa-challenge-Solution / Orders / TC_BACK_19

Save

Share

DELETE

{{Base}} /api/Order/2

Send

Params

Auth

Headers (7)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body

404 Not Found

5 ms

325 B

e.g.

Pretty

Raw

Preview

Visualize

JSON

1 {

2 "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",

3 "title": "Not Found",

4 "status": 404,

5 "traceId": "00-400c3ae92bafd3cf31cb9a1a072e69a8-c9ffd4b604a0ff77-00"

6 }

TC_BACK_20: Create order with empty productName

Expected result: The API responds with an error message and a status code 400 or 422.

Test result: Fail ❌

The screenshot shows a REST client interface for a POST request to `{{Base}} /api/Order/`. The request body is a JSON object: `{ "id": 3, "productName": "", "quantity": 680, "status": "create" }`. The response status is `201 Created`, with a response time of 5 ms and a body size of 256 B. The response body is displayed in JSON format, showing the same object as the request.

TC_BACK_21: Create order with quantity at 0

Expected result: The API responds with an error message and a status code 400 or 422

Test result: Pass ✅

The screenshot shows a REST client interface for a POST request to `{{Base}} /api/Order/`. The request body is a JSON object: `{ "id": 6, "productName": "testProduct", "quantity": "", "status": "create" }`. The response status is `400 Bad Request`, with a response time of 13 ms and a body size of 540 B. The response body is displayed in JSON format, showing an error message: `{ "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1", "title": "One or more validation errors occurred.", "status": 400, "errors": { "order": ["The order field is required."], "$.quantity": ["The JSON value could not be converted to System.Int32. Path: $.quantity | LineNumber: 3 | BytePositionInLine: 16."] }, "traceId": "00-088c260aa7b49c46626beb73b2b9ad76-4c8b2511b48eba24-00" }`

TC_BACK_22: Create order with empty status

Expected result: The API responds with an error message and a status code 400 or 422

Test result: Fail ❌

The screenshot shows a REST client interface for a test case named TC_BACK_22. The request is a POST to the endpoint `{{Base}} /api/Order/`. The request body is a JSON object with the following fields: `"id": 8`, `"productName": "testProduc"`, `"quantity": "14"`, and `"status": ""`. The response status is `201 Created`, with a response time of 5 ms and a size of 259 B. The response body is a JSON object with the same fields as the request, but the `"status"` field is empty. The test result is marked as 'Fail'.

```
1 {
2   "id": 8,
3   "productName": "testProduc",
4   "quantity": "14",
5   "status": ""
6 }
```

```
1 {
2   "id": 8,
3   "productName": "testProduc",
4   "quantity": 14,
5   "status": ""
6 }
```

TC_BACK_23: Create order with duplicate ID

Expected result: The API responds with an error message and a 409 status code, indicating ID duplication.

Test result: Fail ❌

The screenshot shows a REST client interface for a test case named TC_BACK_23. The request is a POST to the endpoint `{{Base}} /api/Order/`. The request body is a JSON object with the following fields: `"id": 3`, `"productName": "test112"`, `"quantity": 680`, and `"status": "create"`. The response status is `201 Created`, with a response time of 3 ms and a size of 263 B. The response body is a JSON object with the same fields as the request, but the `"status"` field is empty. The test result is marked as 'Fail'.

```
1 {
2   "id": 3,
3   "productName": "test112",
4   "quantity": 680,
5   "status": "create"
6 }
```

```
1 {
2   "id": 3,
3   "productName": "test112",
4   "quantity": 680,
5   "status": "create"
6 }
```

GET



{{Base}} /api/Order/

Params

Authorization

Headers (7)

Body

Scripts

Settings

Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (4)

Test Results



Pretty

Raw

Preview

Visualize

JSON



```
8      {
9          "id": 3,
10         "productName": "",
11         "quantity": 680,
12         "status": "create"
13     },
14     {
15         "id": 8,
16         "productName": "testProduc",
17         "quantity": 14,
18         "status": ""
19     },
20     {
21         "id": 8,
22         "productName": "testProduc",
23         "quantity": 14,
24         "status": "test1"
25     },
26     {
27         "id": 3,
28         "productName": "test112",
29         "quantity": 680
```

TC_BACK_24: Create order with all empty fields

Expected result: The API responds with an error message and a status code 400 or 422

Test result: Pass ✓

The screenshot shows a REST client interface for a test case named TC_BACK_24. The request is a POST to the endpoint `{{Base}}/api/Order/` with a raw JSON body:

```
1 {
2   "id": "",
3   "productName": "",
4   "quantity": 0,
5   "status": ""
6 }
```

The response is a 400 Bad Request with the following JSON body:

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
3   "title": "One or more validation errors occurred.",
4   "status": 400,
5   "errors": {
6     "order": [
7       "The order field is required."
8     ],
9     "$.id": [
10      "The JSON value could not be converted to System.Int32. Path: $.id | LineNumber: 1 | BytePositionInLine: 10."
11    ]
12  },
13   "traceId": "00-c88d50099ba71d9df40dc54cc8d8838c-e690b9bce1224be1-00"
14 }
```

TC_BACK_25: Update the ID of an order

Expected result: The API responds with an error message and a status code 400 or 409, indicating that the id is not editable.

Test result: Fail ✗

The screenshot shows a REST client interface for a test case named TC_BACK_25. The request is a PUT to the endpoint `{{Base}}/api/Order/1` with a raw JSON body:

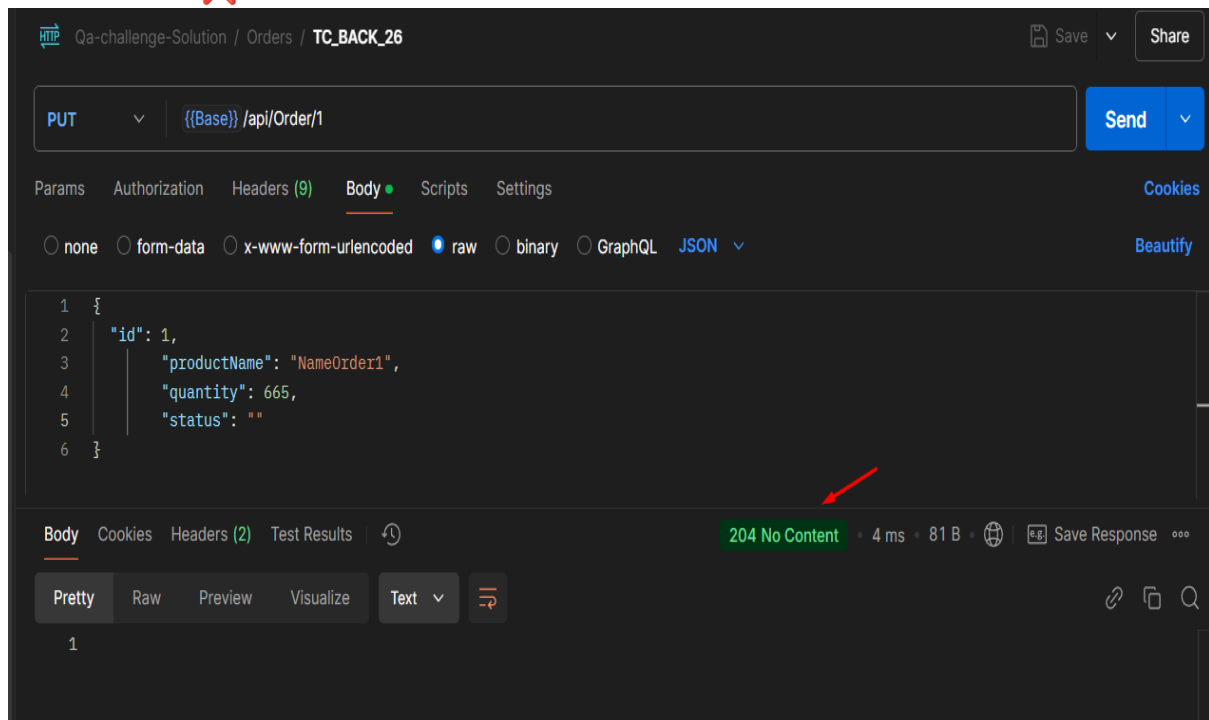
```
1 {
2   "id": 25,
3   "productName": "NameOrder1",
4   "quantity": 665,
5   "status": "Pending"
6 }
```

The response is a 204 No Content.

TC_BACK_26: Update the order with an empty status

Expected result: The API responds with an error message and a status code 400 or 422.

Test result: Fail ❌



HTTP Qa-challenge-Solution / Orders / TC_BACK_26

PUT {{Base}}/api/Order/1

Send

Params Authorization Headers (9) Body Scripts Settings Cookies

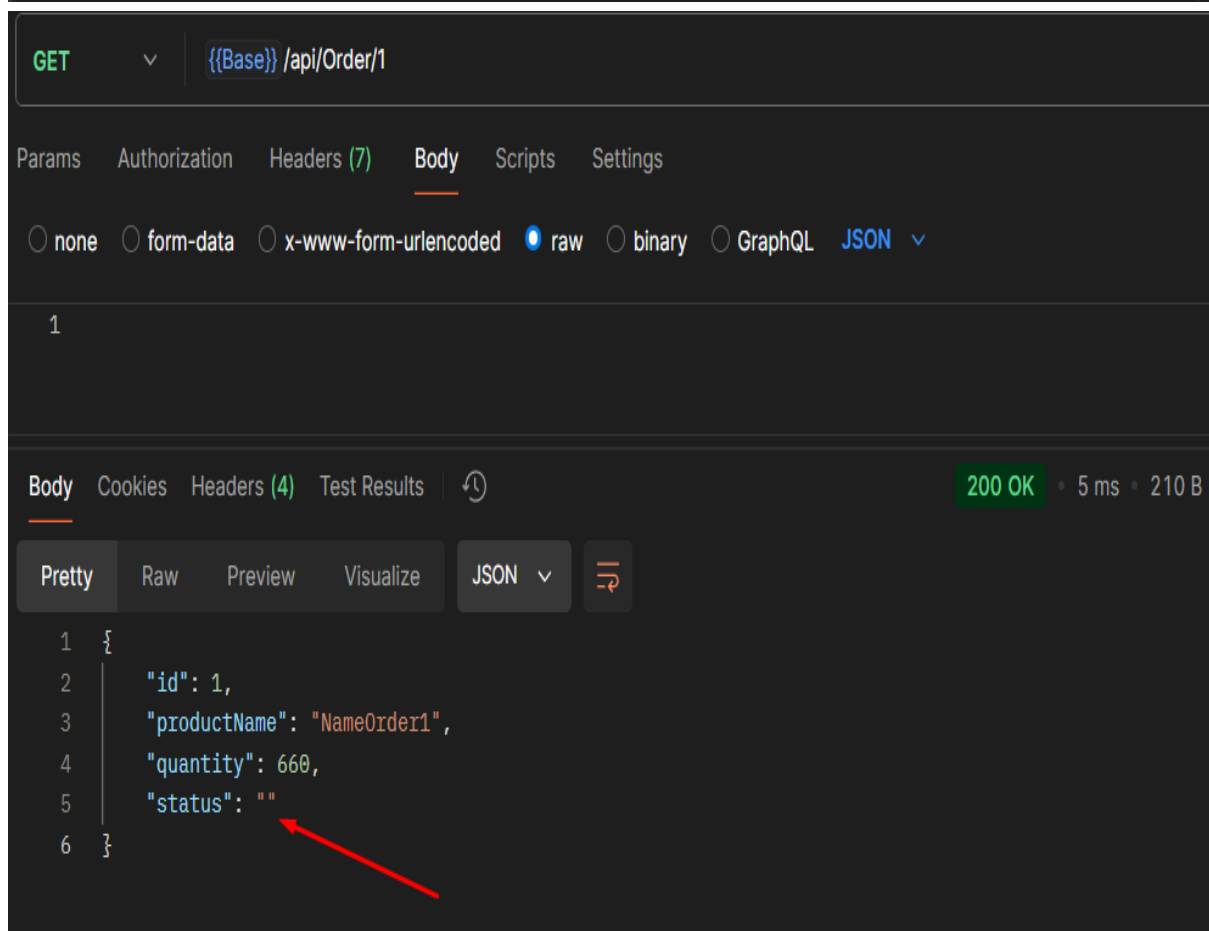
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "id": 1,
3   "productName": "NameOrder1",
4   "quantity": 665,
5   "status": ""
6 }
```

Body Cookies Headers (2) Test Results 204 No Content 4 ms 81 B Save Response

Pretty Raw Preview Visualize Text

1



GET {{Base}}/api/Order/1

Params Authorization Headers (7) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body Cookies Headers (4) Test Results 200 OK 5 ms 210 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "productName": "NameOrder1",
4   "quantity": 660,
5   "status": ""
6 }
```

TC_BACK_27: Prevent deletion of existing order

Expected result: The API responds with an error message indicating that the deletion is not allowed, and returns a status code 403 or 405

Test result: Fail ❌

HTTP Qa-challenge-Solution / Orders / TC_BACK_27 Save Share

DELETE {{Base}}/api/Order/1 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (2) Test Results 204 No Content 4 ms 81 B Save Response

Pretty Raw Preview Visualize Text

1

GET {{Base}}/api/Order/1 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

1

Body Cookies Headers (4) Test Results 404 Not Found 3 ms 325 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",
3   "title": "Not Found",
4   "status": 404,
5   "traceId": "00-0b5e52ca1ff690483e419c165673fe29-ce0ab1898b9e985c-00"
6 }
```

TC_BACK_28: Update order with empty productName

Expected result: The API responds with an error message and a status code 400 or 422

Test result: Fail ❌

Qa-challenge-Solution / Orders / TC_BACK_28

PUT {{Base}}/api/Order/23

Params Authorization Headers (9) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "id": 13,
3   "productName": "2",
4   "quantity": 667,
5   "status": "Complete"
6 }
```

Body Cookies Headers (2) Test Results 204 No Content 3 ms 81 B Save Response

Pretty Raw Preview Visualize Text

1

GET {{Base}}/api/Order/23

Params Authorization Headers (7) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

1

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 23,
3   "productName": "NameOrder123",
4   "quantity": 777,
5   "status": "Complete"
6 }
```

TC_BACK_29: Update order with negative quantity

Expected result: The API responds with an error message and a status code 400 or 422.

Test result: Fail ❌

Qa-challenge-Solution / Orders / TC_BACK_29

PUT {{Base}}/api/Order/23

Body: raw

```
1 { "id": 23,
2   "productName": "NameOrder123",
3   "quantity": -44,
4   "status": "testC"
5 }
```

Body Cookies Headers (2) Test Results

204 No Content · 3 ms · 81 B

Pretty Raw Preview Visualize Text

1

GET {{Base}}/api/Order/23

Body: raw

```
1 {
2   "id": 23,
3   "productName": "NameOrder123",
4   "quantity": 777,
5   "status": "testC"
6 }
```

Body Cookies Headers (4) Test Results

200 OK · 2 ms · 218 B

Pretty Raw Preview Visualize JSON

1 {
2 "id": 23,
3 "productName": "NameOrder123",
4 "quantity": 777,
5 "status": "testC"
6 }

TC_BACK_30: Update order with all fields empty

Expected result: The API responds with an error message and a status code 400 or 422

Test result: Pass ✅

Qa-challenge-Solution / Orders / TC_BACK_30

PUT {{Base}}/api/Order/23

Body: raw

```
1 {
2   "id": "",
3   "productName": "",
4   "quantity": "",
5   "status": ""
6 }
```

Body Cookies Headers (4) Test Results

400 Bad Request · 3 ms · 542 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
3   "title": "One or more validation errors occurred.",
4   "status": 400,
5   "errors": {
6     "updatedOrder": [
7       "The updatedOrder field is required."
8     ],
9     "$.id": [
10      "The JSON value could not be converted to System.Int32. Path: $.id | LineNumber: 1 | BytePositionInLine: 12."
11    ]
12  },
13   "traceId": "00-14e9bba0e4d5687d302c97dba81b89ad-5c7d47683880ebd-00"
```

Frontend (ReactJS)

TC_FRONT_01: Login with Valid Credentials

Expected result: The user is redirected to the Dashboard page and a welcome message is displayed.

Test result: Fail ❌

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

Login

testuser Login

Logged in with token: sampletoken

TC_FRONT_02: Login with Invalid Credentials

Expected result: An error message appears indicating that the credentials are incorrect.

Test result: Fail ❌

← → ↻ ⓘ localhost:3000/login

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

Login

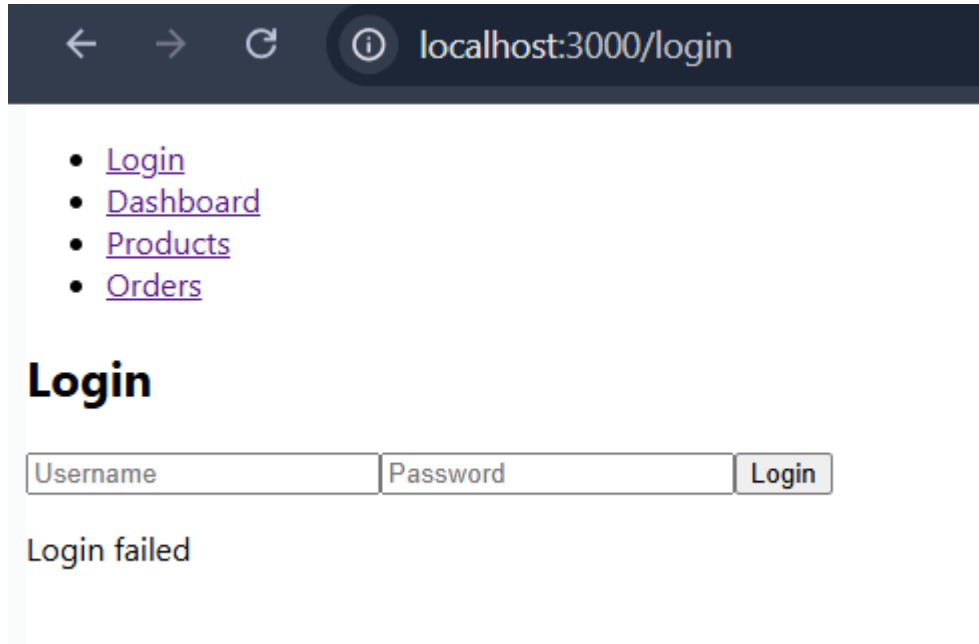
testuser Login

Login failed

TC_FRONT_03: Login with Empty Username and Password

Expected result: Verify that the login form prevents submission when both the username and password fields are empty.

Test result: Fail ❌



• [Login](#)

• [Dashboard](#)

• [Products](#)

• [Orders](#)

Login

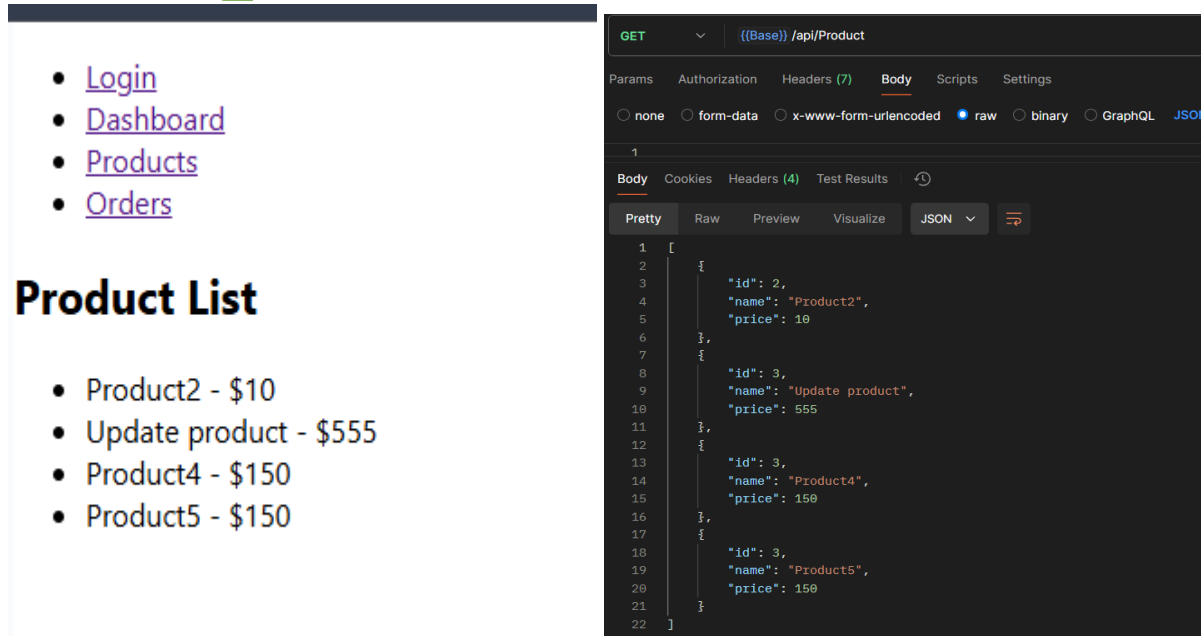
Username Password Login

Login failed

TC_FRONT_04: View Product List

Expected result: The product list page displays all available products with accurate information

Test result: Pass ✅



• [Login](#)

• [Dashboard](#)

• [Products](#)

• [Orders](#)

Product List

- Product2 - \$10
- Update product - \$555
- Product4 - \$150
- Product5 - \$150

```
GET {{Base}} /api/Product
Params  Authorization  Headers (7)  Body  Scripts  Settings
none  form-data  x-www-form-urlencoded  raw  binary  GraphQL  JSON
1
Body  Cookies  Headers (4)  Test Results
Pretty  Raw  Preview  Visualize  JSON
1  [
2    {
3      "id": 2,
4      "name": "Product2",
5      "price": 10
6    },
7    {
8      "id": 3,
9      "name": "Update product",
10     "price": 555
11   },
12   {
13     "id": 3,
14     "name": "Product4",
15     "price": 150
16   },
17   {
18     "id": 3,
19     "name": "Product5",
20     "price": 150
21   }
22 ]
```

TC_FRONT_05: Navigation between Sections

Expected result: The user is redirected to the correct section, and the content is displayed as expected.

Test result: Pass ✓

localhost:3000/dashboard

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

Dashboard

Welcome to the user dashboard!

localhost:3000/orders

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

Order List

- NameOrder3 - 60 - Pending
- 680 - create
- testProduc - 14 - Pending
- testProduc - 14 - test1
- test112 - 680 - create
- test112 - 680 - create
- testedit id - 14 - test1
- NameOrder1 - 660 - Complete
- NameOrder123 - 777 - testC

localhost:3000/products

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

Product List

- Product2 - \$10
- Update product - \$555
- Product4 - \$150
- Product5 - \$150


localhost:3000/login

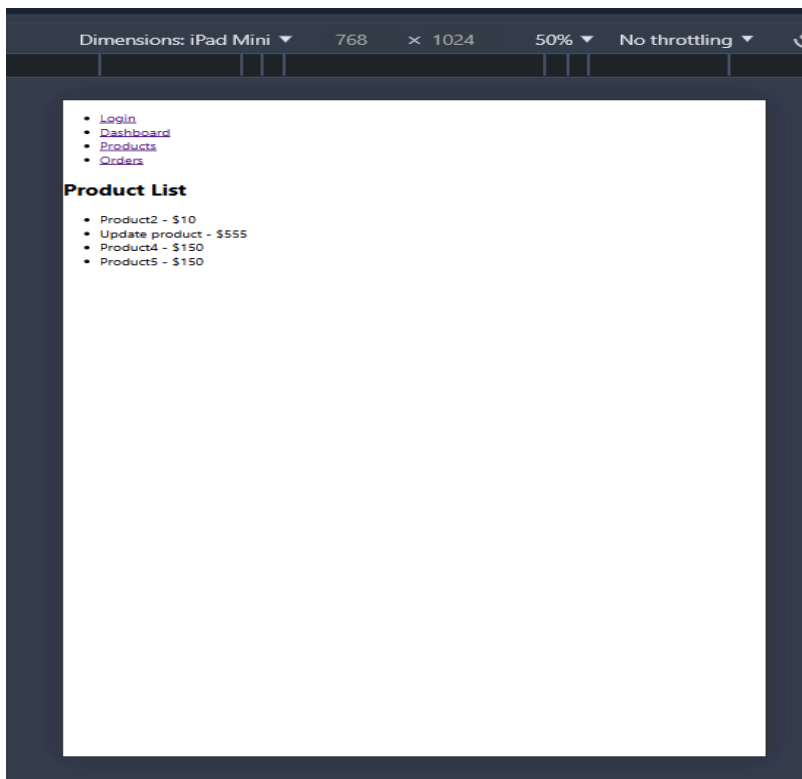
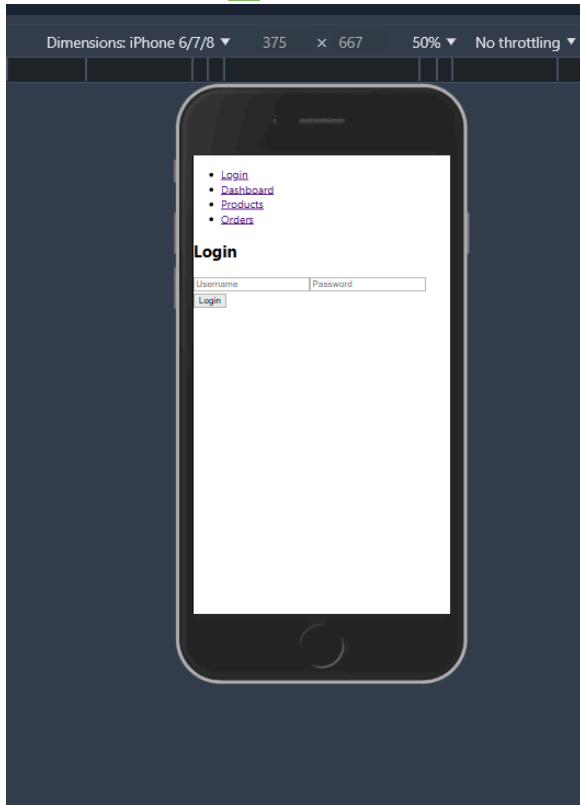
- [Login](#)
- [Dashboard](#)
- [Products](#)
- [Orders](#)

Login

TC_FRONT_06: Responsive Design

Expected result: The layout adjusts properly for desktop, tablet, and mobile screen sizes.

Test result: Pass 















TC_FRONT_07: Logout Functionality










Expected result: The user is logged out, their session is cleared, and they are redirected to the login page. Any attempt to access protected pages without logging in redirects back to the login page.

Test result: Fail ❌


2. Document Results

Test case ID	Title	Result	Comments
TC_BACK_01	Login with Valid Credentials	Pass ✓	Successful authentication. Token received
TC_BACK_02	Login with invalid user	Pass ✓	Authentication failed and a 401 Unauthorized error was returned
TC_BACK_03	Login with invalid password	Pass ✓	A 401 Unauthorized error was returned for a login attempt with invalid password
TC_BACK_04	Login with both user and password fields empty	Pass ✓	A 401 Unauthorized error was returned for a login attempt with empty credentials
TC_BACK_05	Login with empty user field	Pass ✓	The system attempted to initialize but failed authentication, resulting in a 401 Unauthorized error
TC_BACK_06	Login with empty password field	Pass ✓	The system attempted to initialize but failed authentication, resulting in a 401 Unauthorized error

TC_BACK_07	Create Product	Pass 	The system attempted to initialize but failed authentication, resulting in a 401 Unauthorized error
TC_BACK_08	Consult Product List	Pass 	
TC_BACK_09	Create Product with Existing ID	Fail 	It is possible to assign the same ID to multiple products.
TC_BACK_10	Edit Product by ID	Pass 	Code 201, it is possible to edit products by id
TC_BACK_11	Delete Product by ID	Pass 	Code 201, it is possible to delete products by id
TC_BACK_12	Consult Product by ID	Pass 	Code 201, it is possible to search products by id
TC_BACK_13	Create Order with Invalid Quantity	Fail 	It is possible to assign orders with negative quantities
TC_BACK_14	Create Order with Valid Data	Pass 	Orders are created correctly using the required/valid data.
TC_BACK_15	Update Order with Valid Data	Pass 	Orders are Updated correctly using the required/valid data
TC_BACK_16	Delete Order with Non-Existent ID	Pass 	If the ID parameter is not the one set then it is not possible to search, it must be of numerical type.
TC_BACK_17	Delete Order with Non-Numeric ID	Pass 	we cannot delete orders that do not have a numerical id
TC_BACK_18	View all orders	Pass 	It is possible to list the orders with a call, status code 200

TC_BACK_19	Search for an order using an existing ID.	Pass 	It is possible to search for orders by ID
TC_BACK_20	Create order with empty productName	Fail 	The system allows you to create orders without the product name assigned to it.
TC_BACK_21	Create order with quantity at 0	Pass 	The system does not allow to create orders with a quantity of 0 products.
TC_BACK_22	Create order with empty status	Fail 	The system allows you to create orders without assigning a status.
TC_BACK_23	Create order with duplicate ID	Fail 	It is possible to create orders with the same ID
TC_BACK_24	Create order with all empty fields	Pass 	The API responds with an error message and a status code 400
TC_BACK_25	Update the ID of an order	Fail 	Although it is not possible to modify an order ID, there is no clear error message from the Api. But there is a status 204
TC_BACK_26	Update the order with an empty status	Fail 	It is possible to empty the status of an order and leave it blank.
TC_BACK_27	Prevent deletion of existing order	Fail 	Eliminating an order can create gaps in financial and inventory records, which could make it difficult to reconcile accounts and generate accurate reports.

TC_BACK_28	Update order with empty productName	Fail ❌	It is not possible to modify the product name in the order, but a 204 status code is displayed, instead of a message indicating that it is not possible or a 404 code.
TC_BACK_29	Update order with negative quantity	Fail ❌	It is not possible to update to negative quantities, but neither is a 404 or 422 status code visible. Only a 204 code is visible.
TC_BACK_30	Update order with all fields empty	Pass ✅	It is not possible to leave the order fields and the status code is as expected.
TC_FRONT_01	Login with Valid Credentials	Fail ❌	We are not redirected to the dashboard once we start, also the token is visible from the ui, this should not happen.
TC_FRONT_02	Login with Invalid Credentials	Fail ❌	There are no messages to let the user know that the credentials are incorrect.
TC_FRONT_03	Login with Empty Username and Password	Fail ❌	There is no menu to tell the user why he/she cannot log in.
TC_FRONT_04	View Product List	Pass ✅	Product details are visible from the product section.
TC_FRONT_05	Navigation between Sections	Pass ✅	By making use of the sections we are redirected to them.
TC_FRONT_06	Responsive Design	Pass ✅	The site adapts to the devices

TC_FRONT_07	Logout Functionality	Pass 	The user is unable to log out of the platform correctly. When trying to perform this action, there are several problems that prevent the successful logo
-------------	----------------------	--	--

3.Defect Reporting

Defect 1: Create Product with Existing ID

Test Case ID: TC_BACK_09

Description: The system allows assigning the same ID to multiple products, which compromises the integrity of the product catalog.

Steps to Reproduce:

1. Attempt to create a product with an ID that already exists in the system.
2. Submit the request.
3. Check the response.

Actual Result: The system permits the creation of products with duplicate IDs.

Expected Result: The system should return an error message indicating that the product ID already exists, with an appropriate status code.

Severity: High

Priority: Medium

Defect 2: Create Order with Invalid Quantity

Test Case ID: TC_BACK_13

Description: It is possible to create orders with negative quantities, which violates business rules.

Steps to Reproduce:

1. Attempt to create an order with a negative quantity value.
2. Submit the request.
3. Check the response.

Actual Result: The system accepts negative quantities for orders.

Expected Result: The system should reject the request and return an error message indicating that the quantity must be a positive number.

Severity: Critical

Priority: High

Defect 3: Login with Valid Credentials (Frontend)

Test Case ID: TC_FRONT_01

Description: After logging in with valid credentials, the user is not redirected to the dashboard. Additionally, the authentication token is visible on the UI, which is a security risk.

Steps to Reproduce:

1. Open the login page.
2. Enter valid credentials.
3. Click the login button.
4. Observe the behavior of the UI and inspect for token visibility.

Actual Result: The user is not redirected to the dashboard, and the token is exposed in the UI.

Expected Result: The user should be redirected to the dashboard upon successful login, and the token should not be visible on the UI.

Severity: Critical

Priority: High

Defect 4: Login with Invalid Credentials (Frontend)

Test Case ID: TC_FRONT_02

Description: There are no error messages to inform the user that their credentials are incorrect, leading to a poor user experience.

Steps to Reproduce:

1. Open the login page.
2. Enter invalid credentials.
3. Click the login button.
4. Observe the response on the UI.

Actual Result: The system fails to display an error message.

Expected Result: The system should display a clear error message indicating that the credentials are incorrect.

Severity: Medium

Priority: High

Defect 5: Logout Functionality

Test Case ID: TC_FRONT_07

Description: Users are unable to log out correctly. Multiple issues prevent the successful completion of the logout action, such as persistent session tokens and redirect errors.

Steps to Reproduce:

1. Log in to the application.
2. Attempt to log out by clicking the logout button.
3. Observe the system behavior.

Actual Result: Logout functionality fails, and session persistence issues occur.

Expected Result: The system should terminate the session, remove tokens, and redirect the user to the login page.

Severity: High

Priority: Medium