

## PART 1: TEST PLAN DESIGN

### 1. Scope:

- *Backend (C# APIs)*
  - User Authentication:
    - Login validation with valid and invalid credentials.
    - Authentication token generation, validity and expiration tests.
  - Product Management:
    - CRUD (Create, Read, Update, Delete) operations for products.
    - Error handling for invalid product data.
    - Search tests using id.
  - Order Processing:
    - Order creation and verification.
    - Canceling and updating order status.
    - Verification of error handling in invalid order transactions.
- *Frontend (ReactJS)*
  - User Authentication and Dashboard:
    - Verification of login and logout functionalities.
    - Testing of data display and update in the user's control panel.
    - Testing of the responsiveness and accessibility of the user interface.
  - Product Listing and Order Processing:
    - Verification of product listing display and filtering.
    - Testing of order creation, update and cancellation workflows.
    - Testing of the responsiveness and accessibility of the user interface.

### 2. Objectives:

- **Software Quality:** Ensure that all critical backend and frontend functionalities work correctly and are robust to invalid input.
- **Usability:** Ensure that the user interface is accessible, easy to use and displays correctly on different devices and resolutions.

- **Security:** Verify that the authentication process is secure and that sensitive data is protected.
- **Reliability:** Ensure that the system adequately handles errors and provides clear feedback to the user.

### 3. Resources

- **Tools:**
  - Postman: For backend API testing.
  - Cypress: For automated frontend testing.
  - Git: For version control and code storage.
- **Environments:**
  - Local development environment: local server for backend testing.
  - Browser: Google Chrome for frontend testing.
  - Devices: Desktop, tablet and mobile computers for responsiveness testing.
- **Test data:**
  - Test users with different credentials (correct and incorrect).
  - Sample products with different attributes.
  - Test orders with different status and configurations.

### 4. Risks

- **Technical risks:**
  - Possible version incompatibilities between test tools and the development environment.
  - Errors in the test environment configuration that could affect test execution.
- **Mitigation strategies:**
  - Configuration and verification of controlled and well-documented environments before testing begins.
  - Use of a separate, replicable test environment to avoid interference with the production environment.
  - Running tests on multiple browsers and devices to ensure complete coverage.

## 5. Results

- Test Plan Document: It will contain the test strategy, scope, objectives, resources, risks and deliverables.
- Test Cases: A complete set of test cases designed to cover all the functionalities described in the scope.
- Test Execution Report: Detailed documentation of test results, including cases passed, failed, and any defects found.
- Automation Scripts: Automated test scripts for key frontend and backend tests, accompanied by execution instructions.

## PART 2: TEST CASE DEVELOPMENT

Instructions for use of the table:

- **ID:** Unique identifier of the test case, useful for tracking.
- **Test Case Title:** Brief description of the purpose of the test case.
- **Description:** More detailed explanation about what is being tested and why it is relevant.
- **Steps to Execute:** Detailed instructions on how to carry out the test.
- **Expected Result:** What should happen if the system works correctly.

### BACKEND (C# APIS):

ID	Test Case Title	Description	Steps to Execute	Expected Result
TC001	Login with Valid Credentials	Verify that users can successfully log in with valid credentials.	1. Send a POST request to /api/User/login with valid credentials.	The API responds with a valid authentication token and status code 200.
TC002	Login with Invalid Credentials	Validate that the system rejects login with incorrect credentials.	1. Send a POST request to /api/User/login with invalid credentials.	The API responds with an error message and status code 401.
TC003	Login with both fields empty.	Validate that the system does not authorize login with empty fields.	1. Send a POST request to /api/User/login without placing any credentials.	The API responds with an error message and status code 401.
TC004	Login with empty user field.	Validate that the system does not authorize login with an empty user field.	1. Send a POST request to /api/User/login without placing the user, but placing the password.	The API responds with an error message and status code 401.
TC005	Login with empty password field.	Validate that the system does not authorize login with an empty password field.	1. Send a POST request to /api/User/login without putting the	The API responds with an error message and status code 401.

			password, but putting the user.	
TC006	Create Product	Verify that the API allows to create a product with valid data.	1. Send a POST request to /api/Product with all required data.	The API responds with a success message and a 201 status code.
TC007	Consult Product List	Verify that the API allows consulting the list of products correctly.	1. Send a GET request to /api/Product.	The API responds with the complete list of products and a 200 status code.
TC008	Create Product with Existing ID	Validate that the API properly handles the creation of a product when the ID already exists.	1. Send a POST request to /api/Product using an ID that already exists.	The API responds with an error message and a 409 status code, indicating that the ID already exists.
TC009	Browse Product by ID	Verify that the API allows querying a specific product using its ID.	1. Send a GET request to /api/Product with a specific ID.	The API responds with the requested product details and a 200 status code.
TC010	Edit Product by ID	Verify that the API allows editing a specific product using its ID.	1. Send a PUT request to /api/Product with a specific ID and new data to update the product.	The API responds with a success message and a status code 200.
TC011	Delete Product by ID	Verify that the API allows you to delete a specific product using its ID.	1. Send a DELETE request /api/Product with a specific ID.	The API responds with a success message and a status code 200.
TC012	Product Upgrade with Nonexistent ID	Test system behavior when attempting to update a product that does not exist.	1. Send a PUT request to /api/Product with a product ID that does not exist.	The API responds with an error message and 404 status code.
TC013	Elimination of Product with Inexistent ID	Verify the behavior of the system when an attempt is made to delete a product that does not exist.	1. Send a PUT request to /api/Product with a product ID that does not exist.	The API responds with an error message and 404 status code.
TC014	View all orders	Verify that the API correctly returns all existing orders.	1. Send a GET request to /api/Order without additional parameters.	The API responds with a complete list of commands and a status code 200.
TC015	Search for an order using an existing ID.	Verify that the API returns a specific order when a valid ID is provided.	1. Send a GET request to the /api/Order with a valid order ID as parameter.	The API responds with the details of the specific order and a 200 status code.
TC016	Search an order for a non-existent ID.	Test the behavior of the API when requesting an order with an ID that does not exist.	1. Send a GET request to /api/Order with an order ID that does not exist.	The API responds with an error message and a 404 status code.

TC017	Search order with invalid ID	Validate that the API properly handles the request when an invalid ID (e.g., non-numeric) is provided.	1. Send a GET request to /api/Order with an invalid order ID.	The API responds with an error message and a status code 400 or 422.
TC018	Create order with valid data	Verify that the API allows to create an order with all valid data.	1. Send a POST request to /api/Order with valid values.	The API responds with a success message and a 201 status code, and the order is successfully saved.
TC019	Create order with empty productName	Validate that the API properly handles the creation of an order when the productName is empty.	1. Send a POST request to /api/Order with empty productName in the body.	The API responds with an error message and a status code 400 or 422.
TC020	Create order with quantity at 0	Verify that the API rejects the creation of an order when quantity is equal to 0, since it does not make sense to create an order without products.	1. Send a POST request to /api/Order with quantity equal to 0 in the body.	The API responds with an error message and a status code 400 or 422.
TC021	Create order with negative quantity	Validate that the API rejects the creation of an order when quantity is negative.	1. Send a POST request to /api/Order with negative quantity in the body.	The API responds with an error message and a status code 400 or 422.
TC022	Create order with empty status	Verify that the API properly handles the creation of an order when status is empty.	1. Send a POST request to /api/Order with empty status in the body.	The API responds with an error message and a status code 400 or 422.
TC023	Create order with duplicate ID	Validate that the API rejects the creation of an order when the id already exists.	1. Send a POST request to /api/Order with an id that already exists.	The API responds with an error message and a 409 status code, indicating ID duplication.
TC024	Create order with all empty fields	Validate that the API rejects the creation of an order when all fields are empty.	1. Send a POST request to /api/Order with all empty fields in the body.	The API responds with an error message and a status code 400 or 422.
TC025	Create order with incorrect data type	Verify that the API properly handles the creation of an order when sending an incorrect data type (e.g., quantity as string).	1. Send a POST request to /api/Order with an incorrect data type in one of the fields.	The API responds with an error message and a status code 400 or 422.
TC026	Update order with valid data (without changing the ID)	Verify that the API allows updating an existing order with all valid data except the id, which must not be editable.	1. Send a PUT request to /api/Order/{id} with a body that includes productName, quantity, and status with valid values.	The API responds with a success message and a status code 200 or 204, and the order is successfully updated in the database, without changing the id.

TC027	Update the ID of an order	Verify that the API does not allow to modify the id of an existing order.	1. Send a PUT request to /api/Order/{id} with a body that includes a different id than the current one.	The API responds with an error message and a status code 400 or 409, indicating that the id is not editable.
TC028	Update order with empty productName	Validate that the API properly handles the update of an order when the productName is empty.	1. Send a PUT request to /api/Order/{id} with empty productName in the body.	The API responds with an error message and a status code 400 or 422.
TC029	Update order with quantity at 0	Verify that the API rejects the update of an order when quantity equals 0	1. Send a PUT request to /api/Order/{id} with quantity at 0 in the body.	The API responds with an error message and a status code 400 or 422, indicating that quantity cannot be 0.
TC030	Update order with negative quantity	Validate that the API rejects the update of an order when quantity is negative.	1. Send a PUT request to /api/Order/{id} with negative quantity on the body.	The API responds with an error message and a status code 400 or 422.
TC031	Update the order with an empty status	Verify that the API properly handles the update of an order when status is empty.	1. Send a PUT request to /api/Order/{id} with empty status in the body.	The API responds with an error message and a status code 400 or 422.
TC032	Update order with non-existent ID	Test the behavior of the API when trying to update an order with an ID that does not exist.	1. Send a PUT request to /api/Order/{id} with an id that does not exist.	The API responds with an error message and a 404 status code.
TC033	Update order with all fields empty	Validate that the API rejects the update of an order when all fields are empty.	1. Send a PUT request to /api/Order/{id} with all empty fields in the body.	The API responds with an error message and a status code 400 or 422.
TC034	Update order with incorrect data type	Verify that the API properly handles the update of an order when an incorrect data type is sent (e.g., quantity as string).	1. Send a PUT request to /api/Order/{id} with an incorrect data type in one of the fields.	The API responds with an error message and a status code 400 or 422.
TC035	Delete order with existing ID	Verify that the API allows deleting an existing order with a valid order ID.	1. Send a DELETE request to /api/Order/{id} with a valid id in the URL.	The API responds with a success message and a status code 200 or 204, and the order is deleted.
TC036	Delete order with non-existent ID	Verify that the API properly handles the request when trying to delete an order with an ID that does not exist.	1. Send a DELETE request to /api/Order/{id} with an id that does not exist.	The API responds with an error message and a 404 status code, indicating that the command was not found.

TC037	Delete order with non-numeric ID	Verify that the API handles the request properly when trying to delete an order with a non-numeric ID.	1. Send a DELETE request to /api/Order/{id} with a non-numeric id in the URL (e.g., a string).	The API responds with an error message and a status code 400, indicating an invalid ID format.
TC038	Delete order with negative ID	Verify that the API properly handles the request when trying to delete an order with a negative ID	1. Send a DELETE request to /api/Order/{id} with a negative id in the URL.	The API responds with an error message and a status code 400, indicating an invalid ID format.
TC039	Delete order without authorization	Verify that the API properly handles the request when an attempt is made to delete an order without proper authorization.	1. Send a DELETE request to /api/Order/{id} without a valid authorization token or credentials.	The API responds with an error message and a 401 status code, indicating lack of authorization.

### Frontend (ReactJS)

ID	Test Case Title	Description	Steps to Execute	Expected Result
TC001	Login with Valid Credentials	Verify that users can successfully log in with valid credentials.	1. Navigate to the login page. 2. Enter a valid username and password. 3. Click on "Login".	The user is redirected to the home page and a welcome message is displayed.
TC002	Login with Invalid Credentials	Validate that the system rejects login with incorrect credentials.	1. Navigate to the login page. 2. Enter an invalid username and password. 3. Click on "Login".	An error message appears indicating that the credentials are incorrect.
TC003	Viewing the Product List	Verify that the product list is loaded correctly and shows updated information.	1. Navigate to the product page. 2. Verify that all available products are correctly displayed with their information.	The product list is successfully loaded and displays the correct information.
TC004	User Interface Responsibility	Ensure that the user interface fits correctly on different devices.	1. Accessing the application from a mobile device. 2. Navigate through the different sections of the application. 3. Verify the visualization on desktop, tablet and mobile.	The interface adapts adequately to different screen sizes.

## PART 3: TEST EXECUTION AND REPORTING

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

#### 1. TC001: Login with Valid Credentials - POST <http://localhost:5044/api/User/login>

The screenshot shows the Postman interface for a test case named "TC001 - Login with Valid Credentials". The request is a POST to `http://localhost:5044/api/User/login`. The body is in JSON format with the following content:

```
1 {
2   "username": "testuser",
3   "password": "password"
4 }
```

The response is a 200 OK status with a response time of 2 ms and a body size of 171 B. The response body is in JSON format and contains a token:

```
1 {
2   "token": "sampletoken"
3 }
```

#### 2. TC002: Login with invalid Credentials - POST

#### <http://localhost:5044/api/User/login>

The screenshot shows the Postman interface for a test case named "TC002 - Login with invalid Credentials". The request is a POST to `http://localhost:5044/api/User/login`. The body is in JSON format with the following content:


```
1 {
2   "username": "username",
3   "password": "wrongpassword"
4 }
```

The response is a 401 Unauthorized status with a response time of 14 ms and a body size of 331 B. The response body is in JSON format and contains error details:

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.6.2",
3   "title": "Unauthorized",
4   "status": 401,
5   "traceId": "00-c46a8e68e94078c9326c8855aacf7bee-d25aa619b0ca72ae-00"
6 }
```



### 3. TC003: Login with both fields empty POST <http://localhost:5044/api/User/login>

 Login / TC003 - Login with both fields empty.

POST

http://localhost:5044/api/User/login

Send

Params Auth Headers (11) **Body** Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "username": "",
3   "password": ""
4 }
```

Body 401 Unauthorized 5 ms 331 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2",
3   "title": "Unauthorized",
4   "status": 401,
5   "traceId": "00-d4d95ac5930f84c9118d2f4ccf9adca1-7d0b61040b1216eb-00"
6 }
```

#### 4. TC004: Login with empty user field POST <http://localhost:5044/api/User/login>

HTTP Login / TC004 - Login with empty user field Save Share

**POST** http://localhost:5044/api/User/login Send

Params Auth Headers (11) Body Scripts Settings Cookies

raw JSON Beautify

```
1 {
2   "username": "",
3   "password": "password"
4 }
```


Body 401 Unauthorized 5 ms 331 B Save as example



Pretty Raw Preview Visualize JSON Copy Search


```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2",
3   "title": "Unauthorized",
4   "status": 401,
5   "traceId": "00-525d2e103bf2904522ba725989ab4603-8cf194c953a2ca1c-00"
6 }
```

## 5. TC005: Login with empty password field POST


<http://localhost:5044/api/User/login>


 Login / TC005 - Login with empty password field



 Save  Share

POST 





http://localhost:5044/api/User/login




Send 

Params Auth Headers (11) Body  Scripts Settings Cookies

raw  JSON  Beautify

```
1 {
2   "username": "user",
3   "password": ""
4 }
```

Body   401 Unauthorized 8 ms 331 B  Save as example 

Pretty Raw Preview Visualize JSON   

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.2",
3   "title": "Unauthorized",
4   "status": 401,
5   "traceId": "00-fa972c50bfb3ca60eafa6e47d4567729-62b655f7045bc8ff-00"
6 }
```

## 6. TC006: Create Product POST <http://localhost:5044/api/Product>

[HTTP](#) Product / TC006 - Create Product Save Share

POST

http://localhost:5044/api/Product

Send

Params Auth Headers (9) **Body** Scripts Settings Cookies

raw JSON Beautify

```
1 {
2   "id": 0,
3   "name": "Product1",
4   "price": 1
5 }
```

Body 201 Created 9 ms 236 B Save as example

Pretty Raw Preview Visualize JSON Copy Search

```
1 {
2   "id": 0,
3   "name": "Product1",
4   "price": 1
5 }
```

## 7. TC007: Consult Product List GET <http://localhost:5044/api/Product>

Product / TC007 - Consult Product List

Save

Share

GET

http://localhost:5044/api/Product

Send

Params Auth Headers (9) Body • Scripts Settings

Cookies

### Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body



200 OK

9 ms

224 B



Save as example

...

Pretty

Raw

Preview


Visualize

JSON



```
1  [
2    {
3      "id": 0,
4      "name": "Product1",
5      "price": 1
6    },
7    {
8      "id": 1,
9      "name": "Product2",
10     "price": 10
11   }
12 ]
```

## 8. TC008: Create Product with Existing ID POST <http://localhost:5044/api/Product>

 Product / TC008 - Create Product with Existing ID Save Share

POST

http://localhost:5044/api/Product

Send

Params Auth Headers (9) **Body** Scripts Settings Cookies

raw

JSON



Beautify

```
1 {
2   "id": 0,
3   "name": "Product3",
4   "price": 30
5 }
```

Body 201 Created 7 ms 237 B Save as example

Pretty Raw Preview Visualize

JSON

```
1 {
2   "id": 0,
3   "name": "Product3",
4   "price": 30
5 }
```

## 9. TC009: Browse Product by ID GET <http://localhost:5044/api/Product/{Id}>

GET TC009 - Browse Product + ⌵ No environment ⌵

Product / TC009 - Browse Product by ID Save ⌵ Share

GET ⌵  Send ⌵

Params Auth Headers (9) Body ● Scripts Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body ⌵ 200 OK 14 ms 185 B Save as example ⋮

Pretty Raw Preview Visualize JSON ⌵ ⌵ 🔍

```
1 {  
2   "id": 1,  
3   "name": "Product2",  
4   "price": 10  
5 }
```

## 10.TC010: Edit Product by ID PUT <http://localhost:5044/api/Product/{Id}>

PUT TC010 - Edit Product by

+

⌵

No environment ⌵

HTTP

Product / TC010 - Edit Product by ID

Save ⌵

Share

PUT ⌵

http://localhost:5044/api/Product/1

Send ⌵

Params

Auth

Headers (9)

Body ●

Scripts

Settings

Cookies

raw ⌵

JSON ⌵

Beautify

```
1 {
2   "id": 0,
3   "name": "Product1Edit",
4   "price": 100
5 }
```


Body ⌵



🌐 204 No Content 2 ms 81 B


Save as example ⋮




## 11.TC011: Delete Product by ID DELETE <http://localhost:5044/api/Product/{Id}>


 Product / TC011 - Delete Product by ID Copy

 Save 



 Share

DELETE 

http://localhost:5044/api/Product/0

Send 


Params Auth Headers (7) Body Scripts Settings




raw  JSON 

Cookies

Beautify

1

Body 

 204 No Content 6 ms 81 B  Save as example 

## 12. TC012: Product Upgrade with Nonexistent ID PUT

<http://localhost:5044/api/Product/{Id}>

HTTP Product / TC012 - Product Upgrade with Nonexistent ID Save Share

PUT ▼  Send ▼

Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 0,
3   "name": "Product1Edit",
4   "price": 100
5 }
```

Body ▼ 404 Not Found 4 ms 325 B Save as example ⋮

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-c0a7c83aa1f0e6d7ed9d144eee210d84-a060a9dea086d6b3-00"
6 }
```

## 13. TC013: Elimination of Product with Inexistent ID DELETE

<http://localhost:5044/api/Product/{Id}>

HTTP Product / TC013 - Elimination of Product with Inexistent ID Save Share

DELETE ▼  Send ▼

Params Auth Headers (7) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify


1

Body ▼ 404 Not Found 9 ms 325 B Save as example ⋮

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-bc41396a12cc64c3aaef88f98d35919-747aeabc6aa919f5-00"
6 }
```

#### 14.TC014: View all orders GET <http://localhost:5044/api/Order>

 Orders / TC014 - View All Orders Save Share

GET ⌵

http://localhost:5044/api/Order



Send ⌵

Params Auth Headers (7) Body Scripts Settings Cookies

none ⌵


This request does not have a body

Body ⌵ 200 OK 4 ms 270 B Save as example ⋮

Pretty Raw Preview Visualize JSON ⌵  

```
1  [
2    {
3      "id": 1,
4      "productName": "string1",
5      "quantity": 0,
6      "status": "string"
7    },
8    {
9      "id": 2,
10     "productName": "",
11     "quantity": 0,
12     "status": "string"
13   }
14 ]
```

#### 15.TC015: Search for an order using an existing ID GET <http://localhost:5044/api/Order/{id}>

 Orders / TC015 - Search for an order using an existing ID Save Share

GET ⌵

http://localhost:5044/api/Order/1



Send ⌵

Params Auth Headers (7) Body Scripts Settings Cookies

none ⌵

This request does not have a body

Body ⌵ 200 OK 6 ms 211 B Save as example ⋮

Pretty Raw Preview Visualize JSON ⌵  

```
1  {
2    "id": 1,
3    "productName": "string1",
4    "quantity": 0,
5    "status": "string"
6  }
```

## 16.TC016: Search an order for a non-existent ID GET

<http://localhost:5044/api/Order/{id}>

HTTP Orders / TC016 - Search an order for a non-existent ID. Save Share

GET ▼  Send ▼

Params Auth Headers (7) Body Scripts Settings Cookies

none ▼

This request does not have a body

Body ▼ 404 Not Found 5 ms 325 B Save as example ⋮

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-c450f4ae5df0487b9ed6d22102c688a9-0fc8c3a39565d35e-00"
6 }
```

## 17.TC017: Search order with invalid ID GET <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC017 - Search order with invalid ID Save Share

GET ▼  Send ▼

Params Auth Headers (7) Body Scripts Settings Cookies

none ▼

This request does not have a body

Body ▼ 400 Bad Request 11 ms 408 B Save as example ⋮

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     "id": [
7       "The value 'test' is not valid."
8     ]
9   },
10  "traceId": "00-f0d923848c3bffa1f4068e46e1a49c7-8c0ff40e7b390a83-00"
11 }
```

## 18.TC018: Create order with valid data POST <http://localhost:5044/api/Order>

HTTP Orders / TC018 - Create order with valid data Save Share

POST ▼  Send ▼

Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 10,
3   "productName": "Product1",
4   "quantity": 10,
5   "status": "Created"
6 }
```

Body ▼ 201 Created 6 ms 266 B Save as example ⋮

Pretty Raw Preview Visualize JSON ▼ ↔ 📄 🔍

```
1 {
2   "id": 10,
3   "productName": "Product1",
4   "quantity": 10,
5   "status": "Created"
6 }
```

## 19.TC019: Create order with empty productName POST <http://localhost:5044/api/Order>

HTTP Orders / TC019 - Create order with empty productName Save Share

POST ▼  Send ▼

Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 11,
3   "productName": "",
4   "quantity": 10,
5   "status": "Created"
6 }
```

Body ▼ 201 Created 9 ms 258 B Save as example ⋮

Pretty Raw Preview Visualize JSON ▼ ↔ 📄 🔍

```
1 {
2   "id": 11,
3   "productName": "",
4   "quantity": 10,
5   "status": "Created"
6 }
```

## 20. TC020: Create order with quantity at 0 POST <http://localhost:5044/api/Order>

HTTP Orders / TC020 - Create order with quantity at 0 Save Share

POST ▼  Send ▼

Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 12,
3   "productName": "Product12",
4   "quantity": 0,
5   "status": "Created"
6 }
```

Body ▼ 201 Created 5 ms 266 B Save as example ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 📄 🔍

```
1 {
2   "id": 12,
3   "productName": "Product12",
4   "quantity": 0,
5   "status": "Created"
6 }
```

## 21. TC021: Create order with negative quantity POST <http://localhost:5044/api/Order>

HTTP Orders / TC021 - Create order with negative quantity Save Share

POST ▼  Send ▼

Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 13,
3   "productName": "Product13",
4   "quantity": -5,
5   "status": "Created"
6 }
```

Body ▼ 201 Created 4 ms 267 B Save as example ⋮

Pretty Raw Preview Visualize JSON ▼ ≡ 📄 🔍

```
1 {
2   "id": 13,
3   "productName": "Product13",
4   "quantity": -5,
5   "status": "Created"
6 }
```

## 22. TC022: Create order with empty status POST <http://localhost:5044/api/Order>

HTTP Orders / TC022 - Create order with empty status Save Share

**POST** ▼  Send ▼

Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ **JSON** ▼ Beautify

```
1 {
2   "id": 14,
3   "productName": "Product14",
4   "quantity": 3,
5   "status": ""
6 }
```

Body ▼ 201 Created 4 ms 259 B Save as example ⋮

Pretty Raw Preview Visualize **JSON** ▼ ≡ 📄 🔍

```
1 {
2   "id": 14,
3   "productName": "Product14",
4   "quantity": 3,
5   "status": ""
6 }
```

## 23. TC023: Create order with duplicate ID POST <http://localhost:5044/api/Order>

HTTP Orders / TC023 - Create order with duplicate ID Save Share

**POST** ▼  Send ▼

Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ **JSON** ▼ Beautify

```
1 {
2   "id": 14,
3   "productName": "OrderDuplicate",
4   "quantity": 30,
5   "status": "Created"
6 }
```

Body ▼ 201 Created 2 ms 272 B Save as example ⋮

Pretty Raw Preview Visualize **JSON** ▼ ≡ 📄 🔍

```
1 {
2   "id": 14,
3   "productName": "OrderDuplicate",
4   "quantity": 30,
5   "status": "Created"
6 }
```

## 24.TC024: Create order with all empty fields POST <http://localhost:5044/api/Order>

HTTP Orders / TC024 - Create order with all empty fields Save Share

**POST** http://localhost:5044/api/Order Send

Params Auth Headers (9) **Body** Scripts Settings Cookies

raw **JSON** Beautify

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

Body 400 Bad Request 4 ms 508 B Save as example

Pretty Raw Preview Visualize **JSON** Copy Search

```
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      "' ' is an invalid start of a value. Path: $.id | LineNumber: 1 |
11      BytePositionInline: 8."
12    ]
13   },
14   "traceId": "00-8260b821818d7aaf63806d2d63e2695b-8464e8bfc418573b-00"
```



## 25.TC025: Create order with incorrect data type POST

<http://localhost:5044/api/Order>

[HTTP](#) Orders / TC025 - Create order with incorrect data type Save Share

POST

http://localhost:5044/api/Order

Send

Params Auth Headers (9) **Body** Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "id": 17,
3   "productName": "Product17",
4   "quantity": zero,
5   "status": "Created"
6 }
```

Body 400 Bad Request 5 ms 521 B Save as example

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     "order": [
7       "The order field is required."
8     ],
9     "$.quantity": [
10      "'z' is an invalid start of a value. Path: $.quantity | LineNumber: 3 |
11      BytePositionInLine: 14."
12    ]
13  },
14   "traceId": "00-4958731f7e024327786e87430b4cf2b0-ec8289c14f40a02f-00"
15 }
```

**26.TC026:** Update order with valid data (without changing the ID) PUT <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC026 - Update order with valid data (without changing the ID) Save Share

PUT ▼  Send ▼

Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 10,
3   "productName": "Product17Update",
4   "quantity": 100,
5   "status": "Complete"
6 }
```

Body ▼ 204 No Content 5 ms 81 B Save as example ⋮

**27.TC027:** Update the ID of an order PUT <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC027 - Update the ID of an order Save Share

PUT ▼  Send ▼

Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 170,
3   "productName": "Product17Update",
4   "quantity": 100,
5   "status": "Created"
6 }
```

Body ▼ 204 No Content 3 ms 81 B Save as example ⋮

**28.TC028:** Update order with empty productName PUT <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC028 - Update order with empty productName Save Share

PUT ▼  Send ▼

Overview Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 10,
3   "productName": "",
4   "quantity": 100,
5   "status": "Created"
6 }
```

Body ▼ 204 No Content 12 ms 81 B Save as example ⋮

**29.TC029:** Update order with quantity at 0 PUT <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC029 - Update order with quantity at 0 Save Share

PUT ▼  Send ▼

Overview Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 10,
3   "productName": "Product17Update",
4   "quantity": 0,
5   "status": "Created"
6 }
```

Body ▼ 204 No Content 5 ms 81 B Save as example ⋮

### 30. TC030: Update order with negative quantity PUT

<http://localhost:5044/api/Order/{id}>

HTTP Orders / TC030 - Update order with negative quantity Save Share

PUT ▼  Send ▼

Overview Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": -10,
3   "productName": "Product17Update",
4   "quantity": -20,
5   "status": "Created"
6 }
```

Body ▼ 204 No Content 5 ms 81 B Save as example ⋮

### 31. TC031: Update the order with an empty status PUT

<http://localhost:5044/api/Order/{id}>

HTTP Orders / TC031 - Update the order with an empty status Save Share

PUT ▼  Send ▼

Overview Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": -10,
3   "productName": "Product17Update",
4   "quantity": 20,
5   "status": ""
6 }
```

Body ▼ 204 No Content 5 ms 81 B Save as example ⋮

### 32. TC032: Update order with non-existent ID PUT

<http://localhost:5044/api/Order/{id}>

[HTTP](#) Orders / TC032 - Update order with non-existent ID Save Share

PUT

http://localhost:5044/api/Order/444

Send

Overview Params Auth Headers (9) **Body** Scripts Settings Cookies

raw

JSON

Beautify

```
1 {
2   "id": 444,
3   "productName": "Product444Update",
4   "quantity": 20,
5   "status": "Created"
6 }
```

Body 404 Not Found 5 ms 325 B Save as example

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": "00-3d461e0b209853482ffffafe0a8f1909b-48fd8ce26e45b24e-00"
6 }
```

### 33.TC033: Update order with all fields empty PUT

<http://localhost:5044/api/Order/{id}>

[HTTP](#) Orders / TC033 - Update order with all fields empty Save Share

PUT ▼

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

Send ▼

Overview Params Auth Headers (9) **Body** ● Scripts Settings Cookies

raw ▼

JSON ▼

Beautify

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

Body ▼ 400 Bad Request 5 ms 522 B Save as example ⋮

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      "' ' is an invalid start of a value. Path: $.id | LineNumber: 1 |
11      BytePositionInLine: 8."
12    ]
13  },
14   "traceId": "00-6cc0ce5e982c13a4740b1a5d974726e6-98523ab7ed5cb11d-00"
15 }
```

### 34. TC034: Update order with incorrect data type PUT <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC034 - Update order with incorrect data type Save Share

PUT ▼  Send ▼

Overview Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   "id": 10,
3   "productName": "OrderUpdate",
4   "quantity": "test",
5   "status": "Complete"
6 }
```

Body ▼ 400 Bad Request 7 ms 554 B Save as example ⋮

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     "$.quantity": [
10      "The JSON value could not be converted to System.Int32. Path: $.
11      quantity | LineNumber: 3 | BytePositionInLine: 20."
12    ]
13  },
14  "traceId": "00-e95381f4613eb1ae0bd2aaefc04cc98a-c2337e69e5275c8a-00"
15 }
```

### 35. TC035: Delete order with existing ID DELETE <http://localhost:5044/api/Order/{id}>

HTTP Orders / TC035 - Delete order with existing ID Save Share

DELETE ▼  Send ▼

Overview Params Auth Headers (7) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1

Body ▼ 204 No Content 10 ms 81 B Save as example ⋮

### 36. TC036: Delete order with non-existent ID DELETE

<http://localhost:5044/api/Order/{id}>

[HTTP](#) Orders / TC036 - Delete order with non-existent ID DELETE Save Share

DELETE ▼

http://localhost:5044/api/Order/100

Send ▼

Overview Params Auth Headers (7) Body Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1

Body ▼ 404 Not Found 5 ms 325 B Save as example ⋮

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-ee04bcc66eb6148ef50ca3811e3f911d-f7ecf5de394da4c0-00"  
6 }
```

### 37. TC037: Delete order with non-numeric ID DELETE

<http://localhost:5044/api/Order/{id}>

[HTTP](#) Orders / TC037 - Delete order with non-numeric ID Save Share

DELETE ▼

http://localhost:5044/api/Order/test

Send ▼

Overview Params Auth Headers (7) Body Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1

Body ▼ 400 Bad Request 6 ms 408 B Save as example ⋮

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     "id": [  
7       "The value 'test' is not valid."  
8     ]  
9   },  
10  "traceId": "00-9d085b45563fc47d6863e55cb33b7e68-f6c70c5ca8954835-00"  
11 }
```



### 38. TC038: Delete order with negative ID DELETE

<http://localhost:5044/api/Order/{id}>

HTTP Orders / TC038 - Delete order with negative ID Save Share

DELETE ▼  Send ▼

Overview Params Auth Headers (7) Body Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1

Body ▼ 404 Not Found 5 ms 325 B Save as example ⋮

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": "00-255b01bf246ef377cb710062486f9091-4fd55f9ccea31b99-00"
6 }
```

### 39. TC039: Delete order without authorization DELETE

<http://localhost:5044/api/Order/{id}>

HTTP Orders / TC039 - Delete order without authorization Save Share

DELETE ▼  Send ▼

Overview Params Auth Headers (7) Body Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1

Body ▼ 204 No Content 6 ms 81 B Save as example ⋮

Pretty Raw Preview Visualize Text ▼ ⌵ 📄 🔍

```
1 |
```

## 2. Documentation of Results:

- Results Format:
  - Test Case ID:** Identification of the test case.
  - Test Case Title:** Title of the test case to be executed.

- **Result:** Pass if the test case has been successful, Fail if it does not meet expectations.
- **Comments:** Any additional details, such as if there were unexpected behaviors or if the result was not as expected.

Test Case ID	Test Case Title	Result	Comments
TC001	Login with Valid Credentials	Pass	Authentication was successful and a token was received in the response body.
TC002	Login with Invalid Credentials	Pass	The system correctly rejected the authentication and returned a 401 (Unauthorized) error.
TC003	Login with both fields empty.	Pass	The system correctly returned a 401 (Unauthorized) error when attempting to log in with both fields empty.
TC004	Login with empty user field.	Pass	The system correctly returned a 401 (Unauthorized) error when trying to start the system.
TC005	Login with empty password field.	Pass	The system correctly returned a 401 (Unauthorized) error when attempting to log in with only the password field.
TC006	Create Product	Pass	The system successfully created the product with the data provided and returned a status 201 along with the details of the product created.
TC007	Consult Product List	Pass	The system successfully returned a list of products with status 200. The list contains the products with their respective id, name, and price.
TC008	Create Product with Existing ID	Fail	The system allowed the creation of a product with an ID that already existed, which is incorrect. A status 201 was received when a status 409 is expected.

TC009	Browse Product by ID	Pass	The product query with ID 1 was successful. A status 200 was received.
TC010	Edit Product by ID	Pass	The edition of the product with ID 0 was successful. A status 204 was received, indicating that the operation was completed successfully with no additional content.
TC011	Delete Product by ID	Pass	The deletion of the product with the specified ID was successful. A status 204 was received, indicating that the operation was completed successfully with no additional content.
TC012	Product Upgrade with Nonexistent ID	Pass	Product update with a non-existent ID was handled correctly. A 404 status was received.
TC013	Elimination of Product with Inexistent ID	Pass	The removal of the product with a non-existent ID was handled correctly. A 404 status was received.
TC014	View all orders	Pass	The consultation of all products was successful. A status 200 was received.
TC015	Search for an order using an existing ID.	Pass	The issue of the product with the specified ID was successful. A status 200 was received.
TC016	Search an order for a non-existent ID.	Pass	The request to access a non-existent resource was handled correctly. A 404 status was received.
TC017	Search order with invalid ID	Pass	The request to access a resource with an invalid ID was handled correctly. A status 400 was received.
TC018	Create order with valid data	Pass	The creation of the order with all valid data

			was successful. A status 201 was received.
TC019	Create order with empty productName	Fail	Order creation with empty productName was not handled correctly. A 201 status was received, indicating that the order was created even though the productName was empty, which should not be allowed.
TC020	Create order with quantity at 0	Fail	The creation of the order with quantity equal to 0 was not handled correctly. A status 201 was received, indicating that the order was created even though the quantity was 0.
TC021	Create order with negative quantity	Fail	The creation of the order with negative quantity was not handled correctly. A 201 status was received, indicating that the order was created even though the quantity was -5.
TC022	Create order with empty status	Fail	The creation of the order with an empty status was not handled correctly. A status 201 was received, indicating that the order was created even though the status was empty.
TC023	Create order with duplicate ID	Fail	The creation of the order with an empty status was not handled correctly. A status 201 was received, indicating that the order was created even though the status was empty.
TC024	Create order with all empty fields	Pass	The API correctly handled the creation of an order with all fields empty. A status 400 was received,

			indicating that there were validation errors.
TC025	Create order with incorrect data type	Pass	The API correctly handled the creation of an order with all fields empty. A status 400 was received, indicating that there were validation errors.
TC026	Update order with valid data (without changing the ID)	Pass	The API allowed the order to be updated with all valid data. A status 204 was received,
TC027	Update the ID of an order	Fail	The API allowed the modification of the order ID, which is not the expected behavior, since the order ID should not be modifiable. A status 204 was received, indicating that the operation was successful, which contradicts the ID immutability restriction.
TC028	Update order with empty productName	Fail	The API allowed the order update even though the productName was empty, which is not the expected behavior. A status 204 was received, indicating that the operation was successful, which contradicts the validation that should prevent updates with empty fields.
TC029	Update order with quantity at 0	Fail	The API allowed the order update even though quantity was 0, which is not the expected behavior. A status 204 was received, indicating that the operation was successful, which contradicts the validation that should prevent updates with quantity equal to 0.
TC030	Update order with negative quantity	Fail	The API allowed the order to be updated

			even though quantity was negative, which is not the expected behavior. A status 204 was received.
TC031	Update the order with an empty status	Fail	The API allowed the order update even though the status was empty, which is not the expected behavior. A 204 status was received, indicating that the operation was successful, which contradicts the validation that should prevent updates with empty status.
TC032	Update order with non-existent ID	Pass	The API responded correctly with a 404 status, indicating that the ID was not found, which is the expected behavior.
TC033	Update order with all fields empty	Pass	The system returned a status 400 with validation error messages.
TC034	Update order with incorrect data type	Pass	The system correctly returned a 400 status code with a validation error, indicating that the quantity field could not be converted to an integer value.
TC035	Delete order with existing ID	Pass	The system successfully updated the order and returned a 204 status code, indicating that the update was successful.
TC036	Delete order with non-existent ID	Pass	The system returned a 404 status code with a "Not Found" message, which is the expected behavior when attempting to update an order that doesn't exist.
TC037	Delete order with non-numeric ID	Pass	The API responded with a status 400, indicating that the value provided is invalid, which is the expected behavior.

TC038	Delete order with negative ID	Fail	The API responded with a 404 status, which is not correct. A 400 status was expected for a negative ID.
TC039	Delete order without authorization	Fail	The API responded with a 204 status, which is not the expected behavior. A 401 status was expected.

### 3. Defect Reporting:

- Defect Details:
  - Defect ID:** A unique identifier for each defect.
  - Associated Test Case ID:** Identify in which test case the defect occurs.
  - Description:** Clear and concise explanation of the defect found.
  - Steps to Replicate:** Detailed step-by-step to replicate the defect.
  - Actual Result:** What happens when the scenario is run.
  - Expected Result:** What should happen according to the requirements.
  - Severity:** Classification of the defect (e.g., Critical, Major, Minor).

Defect ID	Associated Test Case ID	Description	Steps to Replicate	Actual Result	Expected Result	Severity
DEF001	TC003, TC004, TC005	The system does not display a specific error message for empty fields in the login.	1. Access the login page. 2. Leave both fields empty, only fill in the user name, or only fill in the password. 3. Click on "Login".	The system returns a 401 (Unauthorized) error.	The system should display a specific message indicating that all fields are mandatory.	Minor
DEF002	TC008	The system allows the creation of a product with an ID that already exists, which should not be allowed.	1. Send a POST request to /api/Product with an existing product ID. 2. Observe the system response	The system returns a 201 status.	The system should return an error with a 409 status and a message indicating that the ID already exists.	Major (as it allows the creation of duplicates, which can cause problems in product management).

			which is a code 201.			
DEF003	TC019	The API allows the creation of an order with empty productName.	1. Send a POST request to /api/Order with empty productName.	A status 201 was received and the order was created with empty productName.	The API should reject the order creation and return a status 400 or 422, indicating that productName cannot be empty.	Major
DEF004	TC020	The API allows the creation of an order with quantity equal to 0.	1. Send a POST request to /api/Order with quantity equal to 0 and all other data valid.	A 201 status was received and the order was created with quantity equal to 0.	The API should reject the order creation and return a status 400 or 422, indicating that quantity cannot be 0.	Major
DEF005	TC021	API allows the creation of an order with negative quantity	1. Send a POST request to /api/Order with the amount in negative and all other valid data.	A 201 status was received and the order was created with negative quantity.	The API should reject the order creation and return a status 400 or 422, indicating that quantity cannot be negative.	Major
DEF006	TC022	The API allows the creation of an order with empty status.	1. Send a POST request to /api/Order with valid productName, quantity greater than zero and empty status.	A 201 status was received and the order was created with an empty status.	The API should reject the order creation and return a status 400 or 422.	Major
DEF007	TC028	The API allows the update of an order with empty	1. Send a PUT request to /api/Order/{id} with an empty	The API responded with a 204 status, indicating that the	The API should reject the update and return an error with a	Major



		productName.	productName in the body.	operation was successful.	status of 400 or 422, indicating that productName cannot be empty.	
DEF008	TC029	The API allows the update of an order with quantity equal to 0.	1. Send a PUT request to /api/Order/{id} with quantity equal to 0 in the body.	The API responded with a 204 status, indicating that the operation was successful.	The API should reject the update and return an error with a status of 400 or 422, indicating that quantity cannot be 0.	Major
DEF009	TC030	The API allows the update of an order with negative quantity.	1. Send a PUT request to /api/Order/{id} with negative quantity in the body.	The API responded with a 204 status, indicating that the operation was successful.	The API should reject the update and return an error with a status of 400 or 422, indicating that quantity cannot be negative.	Major
DEF010	TC031	The API allows the update of an order with empty status.	1. Send a PUT request to /api/Order/{id} with empty status in the body.	The API responded with a 204 status, indicating that the operation was successful.	The API should reject the update and return an error with a status of 400 or 422, indicating that status cannot be empty.	Major
DEF011	TC038	The API does not correctly validate a negative ID when trying to delete an order.	Send a DELETE request to /api/Order/-1	The API returns a 404 status.	The API should return a 400 status.	Major
DEF012	TC039	The API does not correctly handle the deletion of an	Send a DELETE request to /api/Order/{id} without a valid	The API returns a 204 status.	The API should return a 401 status.	Critical - This defect is critical because it allows sensitive

		unauthorize d order, allowing the operation with a 204 status.	authorizatio n token.			operations to be performed without proper authorizatio n, which can compromise the security of the system.
--	--	---	--------------------------	--	--	--