

# ShoppyGlobe RESTful API - Test Case Documentation

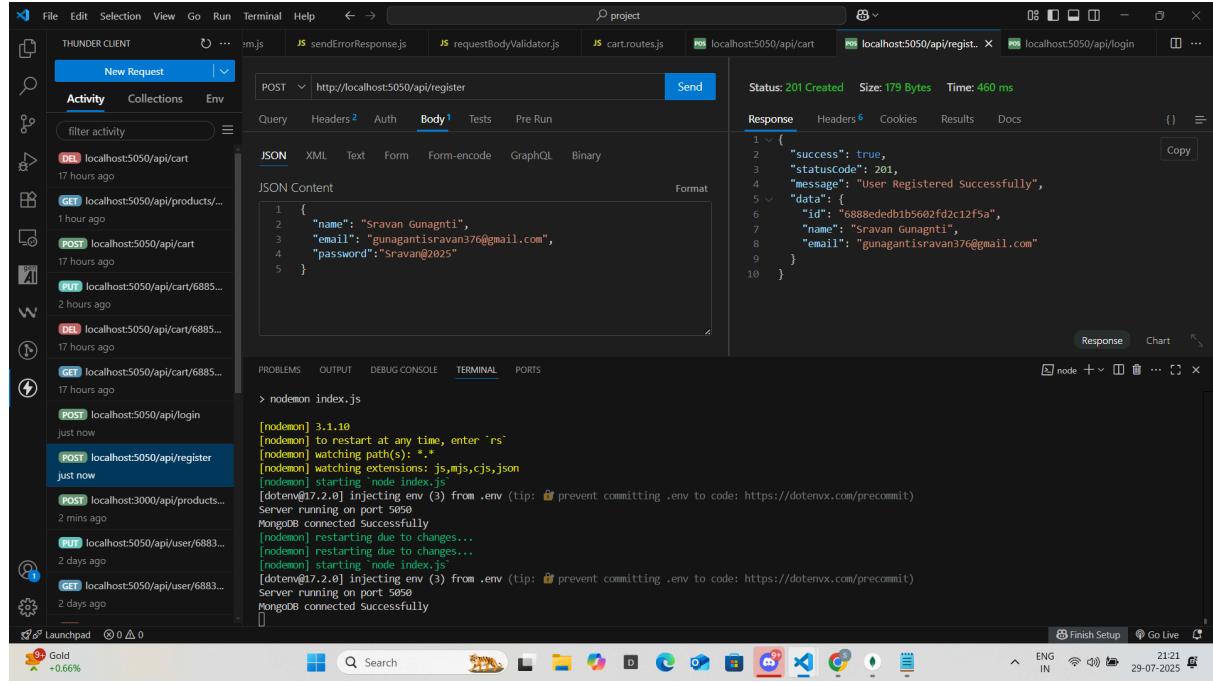
## Overview

This document includes all possible test cases for the REST API routes developed for the ShoppyGlobe backend project. The test cases are organized based on the route and functionality. Each section includes the HTTP method, endpoint, description, request body (if applicable), expected status, and expected response message.

## User Authentication

### 1. Register - Successful

- **POST /api/register**
- **Description:** Registers a new user
- **Expected:** **201 Created**, JWT token and user data returned



The screenshot shows the Thunder Client application interface. In the top navigation bar, the 'Body' tab is selected. Below it, the 'JSON' tab is active. The JSON content pane contains the following data:

```
1  {
2   "name": "Sravan Gunagni",
3   "email": "gunagantisravan376@gmail.com",
4   "password": "Sravan@2025"
5 }
```

In the top right corner of the main window, there is a status message: "Status: 201 Created Size: 179 Bytes Time: 460 ms". The bottom right corner of the application window shows the system tray with icons for battery level (99%), signal strength, and the date/time (29-07-2025).

## 2. Register - Missing Fields

- **Expected:** 400 Bad Request, "Missing Required Fields"

The screenshot shows the Thunder Client interface. On the left, the activity log lists various API calls. In the center, a new request is being prepared for `http://localhost:5050/api/register`. The `Body` tab is selected, showing the following JSON content:

```
1 {
2   "email": "gunagantisravan376@gmail.com",
3   "password": "Sravan@2025"
4 }
```

The response pane on the right shows a 400 Bad Request status with the following JSON error message:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Missing Required Fields",
6     "message": "Name, Email, Password are required. Please enter all the fields to register"
7   }
8 }
```

## 3. Register - Invalid Email

- **Expected:** 400 Bad Request, "Invalid Email"

The screenshot shows the Thunder Client interface. The activity log on the left includes a recent `POST localhost:5050/api/register` call. In the center, a new request is being prepared for `http://localhost:5050/api/register`. The `Body` tab is selected, showing the following JSON content:

```
1 {
2   "name": "Sravan Gunaganti",
3   "email": "gunagantisravanmail.com",
4   "password": "Sravan"
5 }
```

The response pane on the right shows a 400 Bad Request status with the following JSON error message:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Invalid Email",
6     "message": "Please enter a valid email address. Please ensure it follows the standard format like: example@domain.com"
7   }
8 }
```

A warning message at the bottom right states: "Warning: The free version will no longer support collections starting August 3rd, 2025." and "Source: Thunder Client".

## 4. Register - Weak Password

- **Expected: 400 Bad Request, "Invalid Password"**

Status: 400 Bad Request Size: 220 Bytes Time: 112 ms

```

1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {},
5   "title": "Invalid Password",
6   "message": "Password must start with an uppercase letter, include
              lowercase letters, numbers, special characters, and be at least
              6 characters long."
7 }
8

```

## 5. Register - Email Already Exists

- **Expected: 409 Conflict, "Email already registered"**

Status: 409 Conflict Size: 152 Bytes Time: 513 ms

```

1 {
2   "success": false,
3   "statusCode": 409,
4   "error": {},
5   "title": "Email already registered",
6   "message": "A user with this email already exists. Please login
               instead."
7 }
8

```

## 6. Login - Successful

- **POST /api/login**
- **Expected:** 200 OK, data and JWT returned

The screenshot shows the Thunder Client interface. A new request is being made to `http://localhost:5050/api/login`. The body contains the following JSON:

```
[{"email": "gunagantisravan376@gmail.com", "password": "Sravan@2025"}]
```

The response status is 201 Created, with a size of 367 Bytes and a time of 194 ms. The response body is a JSON object containing a success message, status code, access token, and user data.

```
{"success": true, "statusCode": 201, "message": "User Registered Successfully", "accesstoken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 .eyJpZC16IjY4ODhlZGVkyfjNTYwMzkhmtxMny1YSisiMhdC1MzgwNDM 2HSwizXhIjoxNzUzOkWnWYxfQ .1mf3QhyIPWQnOaAjyzhy1692NUixAOXCSD8LiOwc", "data": { "id": "688edeb1b5602fd2c12f5a", "name": "Sravan Gunagni", "email": "gunagantisravan376@gmail.com" }}
```

The terminal pane shows the output of the `nodemon index.js` command, indicating successful node.js execution and MongoDB connection.

## 7. Login - Invalid Credentials

- **Expected:** 401 Unauthorized, "Invalid credentials"

The screenshot shows the Thunder Client interface. A new request is being made to `http://localhost:5050/api/login`. The body contains the following JSON:

```
[{"email": "gunagantisravan376@gmail.com", "password": "Sravan@20"}]
```

The response status is 401 Unauthorized, with a size of 125 Bytes and a time of 333 ms. The response body is a JSON object indicating failure, an error code, and a message.

```
{"success": false, "statusCode": 401, "error": { "title": "Invalid Credentials", "message": "Incorrect password. Please try again.."} }
```

The terminal pane shows the output of the `nodemon index.js` command, indicating successful node.js execution and MongoDB connection.

## 8. Login - Missing Fields

- **Expected:** 400 Bad Request, "Missing Required Fields"

The screenshot shows the Thunder Client interface. On the left, the activity log lists various API requests. In the center, a new request is being sent to `http://localhost:5050/api/login`. The body contains the following JSON:

```
1 {
2   "email": "gunagantisravan173@gmail.com"
3 }
```

The response on the right shows a 400 Bad Request status with the message: "Email, Password are required. Please enter all the fields to login".

## Product Management

### 9. Add Product - Successful

- **POST /api/product**
- **Headers:** JWT required
- **Expected:** 201 Created, Product data returned

The screenshot shows the Thunder Client interface. On the left, the activity log lists various API requests. In the center, a new request is being sent to `http://localhost:5050/api/product`. The body contains the following JSON:

```
1 {
2   "name": "Keyboard",
3   "description": "Mechanical keyboard With Backlight Keyword",
4   "price": 3000,
5   "stock": 20
6 }
```

The response on the right shows a 201 Created status with the message: "Product Created Successfully". The response body includes the created product data:

```
1 {
2   "success": true,
3   "statusCode": 201,
4   "message": "Product Created Successfully",
5   "data": {
6     "name": "Keyboard",
7     "description": "Mechanical keyboard With Backlight Keyword",
8     "price": 3000,
9     "stock": 20,
10    "_id": "6888f04fb1b5602fd2c1301f",
11    "__v": 0
12  }
13 }
```

At the bottom, the terminal shows node.js logs indicating the application is running and MongoDB is connected.

## 10. Add Product - Missing Fields

- **Expected:** 400 Bad Request, "Missing Required Fields"

The screenshot shows the Thunder Client interface. On the left, a sidebar lists various API interactions. In the center, a main panel displays a POST request to `http://localhost:5050/api/product`. The request body is set to JSON and contains the following data:

```
1 {
2   "name": "Essence Mascara Lash Princess",
3   "description": "The Essence Mascara Lash Princess is a popular mascara known for its volumizing and lengthening effects. Achieve dramatic lashes with this long-lasting and cruelty-free formula.",
4   "stock": 99
5 }
```

The response on the right indicates a **400 Bad Request** with a size of 151 bytes and a time of 455 ms. The response body is:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Missing Required Fields",
6     "message": "Name, description, price , stock are required to add product"
7   }
8 }
```

## 11. Add Product - Invalid Stock Type

- **Expected:** 400 Bad Request, "Invalid Stock Input"

The screenshot shows the Thunder Client interface. On the left, a sidebar lists various API interactions. In the center, a main panel displays a POST request to `http://localhost:5050/api/product`. The request body is set to JSON and contains the following data:

```
1 {
2   "name": "Essence Mascara Lash Princess",
3   "description": "The Essence Mascara Lash Princess is a popular mascara known for its volumizing and lengthening effects. Achieve dramatic lashes with this long-lasting and cruelty-free formula.",
4   "price": 9.99,
5   "stock": 9.7
6 }
```

The response on the right indicates a **400 Bad Request** with a size of 135 bytes and a time of 928 ms. The response body is:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Invalid Stock Input",
6     "message": "stock is required and must be a positive number."
7   }
8 }
```

## 12. Bulk Product Insert - Successful

- **POST /api/products/bulk**
- **Headers:** JWT required
- **Request Body:** Valid array of products
- **Expected:** 201 Created, products inserted

The screenshot shows the Thunder Client interface. In the top navigation bar, 'THUNDER CLIENT' is selected. Below it, there are tabs for 'New Request', 'localhost:3000/api/produ...', 'localhost:5050/api/produ...', 'New Request', 'products.json', 'dummy.json', and 'localhost:5050/api/produ...'. The main area shows a 'New Request' dialog for a 'POST' to 'http://localhost:5050/api/products/bulk'. The 'Headers' tab is active, containing 'Accept: \*/\*', 'User-Agent: Thunder Client (https://www.thunderclient.co)', and 'Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5CjI6IkpxVC19.e'. The 'Body' tab shows a JSON payload: [ { "name": "Essence Mascara Lash Princess", "description": "The Essence Mascara Lash Princess is a popular mascara known for its volumizing and lengthening effects. Achieve dramatic lashes with this long-lasting and cruelty-free formula.", "price": 9.99, "stock": 99 } ]. The 'Response' tab shows a successful 201 Created response with a size of 49.82 KB and a time of 577 ms. The response body is identical to the one in the 'Body' tab. The 'Terminal' tab at the bottom shows node.js logs indicating the server is running and MongoDB is connected.

## 13. Get All Products

- **GET /api/products**
- **Expected:** 200 OK, product array returned

The screenshot shows the Thunder Client interface. In the top navigation bar, 'THUNDER CLIENT' is selected. Below it, there are tabs for 'controller copy.js', 'localhost:5050/api/produ...', 'localhost:5050/api/user/6...', 'New Request', 'New Request', 'credentials', and 'localhost:5050/api/produ...'. The main area shows a 'New Request' dialog for a 'GET' to 'http://localhost:5050/api/products'. The 'Query Parameters' tab is active, showing an empty 'parameter' field. The 'Response' tab shows a successful 200 OK response with a size of 6.51 KB and a time of 528 ms. The response body is a JSON array of products, identical to the one in the previous screenshot. The 'Terminal' tab at the bottom shows node.js logs indicating the server is running and MongoDB is connected.

## 14. Get Product by ID - Valid ID

- **GET /api/products/:id**
- **Expected:** 200 OK, product object returned

```
1 {
2   "success": true,
3   "statusCode": 200,
4   "data": {
5     "id": "6888f04fb1b5602fd2c1301f",
6     "name": "Keyboard",
7     "description": "Mechanical Keyboard With Backlight Keyword",
8     "price": 3000,
9     "stock": 20
10   }
11 }
```

## 15. Get Product by ID - Non existing product

- **Expected:** 404 Not Found, "Product Not Found"

```
1 {
2   "success": false,
3   "statusCode": 404,
4   "error": {
5     "title": "Product Not Found",
6     "message": "No product found with ID: 6888f04fb1b5602fd2c1301f"
7   }
8 }
```

## 16. Update Product - Valid Update

- **PUT /api/products/:id**
- **Headers:** JWT required
- **Expected:** 200 OK, updated product

The screenshot shows the Thunder Client interface. In the top navigation bar, 'THUNDER CLIENT' is selected. Below it, a 'New Request' dropdown is open, showing the URL 'http://localhost:5050/api/products/6888f04fb1b5602fd2c1301f'. The 'Body' tab is active, containing the following JSON payload:

```
1 {
  2   "name": "Mechanical Keyboard",
  3   "description": "Mechanical keyboard With Backlight Keyword",
  4   "price": 3010,
  5   "stock": 15
  6 }
```

The response panel shows a status of 200 OK, size of 236 Bytes, and time of 27 ms. The response body is identical to the payload above. The bottom terminal window shows node.js logs indicating successful connection and MongoDB setup.

## 17. Update Product - Not Found

- **Expected:** 404 Not Found, "Product Not Found"

The screenshot shows the Thunder Client interface. In the top navigation bar, 'THUNDER CLIENT' is selected. Below it, a 'New Request' dropdown is open, showing the URL 'http://localhost:5050/api/products/688872331823e1f958b7321b'. The 'Body' tab is active, containing the following JSON payload:

```
1 {
  2   "name": "Calvin Klein CK One",
  3   "description": "CK One by Calvin Klein is a classic unisex fragrance, known for its fresh and clean scent. It's a versatile fragrance suitable for everyday wear.",
  4   "price": 49.99,
  5   "stock": 25
  6 }
```

The response panel shows a status of 404 Not Found, size of 145 Bytes, and time of 816 ms. The response body is a JSON object with 'success': false, 'statusCode': 404, and an 'error' field containing a message about the product not being found.

## 18. Delete Product - Valid

- **DELETE /api/products/:id**
- **Expected:** 200 OK, deleted message

```
HTTP Headers
Accept: */*
User-Agent: Thunder Client (https://www.thunderclient.co)
Authorization: JWT eyJhbGciOiJIUzI1NiIsInR5cI6IkpxCj9.c
header: value

Response
Status: 200 OK Size: 236 Bytes Time: 391 ms
Headers: Content-Type: application/json; charset=utf-8
Content-Length: 236
Date: Mon, 29 Jul 2025 21:48:17 GMT
Connection: keep-alive
Set-Cookie: session=6888f04fb1b5602fd2c1301f; path=/; secure; HttpOnly
{
  "success": true,
  "statusCode": 200,
  "message": "Product Deleted Successfully",
  "data": {
    "id": "6888f04fb1b5602fd2c1301f",
    "name": "Mechanical Keyboard",
    "description": "Mechanical keyboard With Backlight Keyword",
    "price": 3010,
    "stock": 15,
    "__v": 0
  }
}
```

## 19. Delete Product - Not Found

- **Expected:** 404 Not Found, "Product Not Found"

```
Query Parameters
parameter: value

Response
Status: 404 Not Found Size: 145 Bytes Time: 417 ms
Headers: Content-Type: application/json; charset=utf-8
Content-Length: 145
Date: Mon, 29 Jul 2025 21:47:29 GMT
Connection: keep-alive
Set-Cookie: session=6889bf0cbAAF2c183421f729; path=/; secure; HttpOnly
{
  "success": false,
  "statusCode": 404,
  "error": {
    "title": "Product Not Found",
    "message": "No product found with ID: 6889bf0cbAAF2c183421f729 to delete"
  }
}
```

# Cart Management

## 20. Add to Cart - New Item

- **POST /api/cart**
- **Headers:** JWT required
- **Expected:** 200 OK, populated cart

The screenshot shows the Thunder Client interface. A POST request is made to `http://localhost:5050/api/cart`. The response status is 200 OK, size is 381 Bytes, and time is 98 ms. The response body contains a JSON object representing a new cart item:

```
17     },
18     {
19       "productId": {
20         "_id": "688902e9461771ab374e4b2b",
21         "name": "Tennis Ball",
22         "description": "The Tennis Ball is a standard ball used in the sport of tennis. It is designed for bouncing and hitting with tennis rackets during matches or practice sessions.",
23         "price": 6.99,
24         "stock": 28,
25         "__v": 0
26       },
27       "quantity": 1,
28       "_id": "688904dc461771ab374e4b38"
29     },
30   ],
31   "createdAt": "2025-07-29T17:20:41.510Z",
32   "updatedAt": "2025-07-29T17:29:00.157Z",
33   "__v": 1
34 }
35 }
```

## 21. Add to Cart - Product Not Found

- **Expected:** 404 Not Found, "Product Not Found"

The screenshot shows the Thunder Client interface. A POST request is made to `http://localhost:5050/api/cart`. The response status is 404 Not Found, size is 150 Bytes, and time is 591 ms. The response body is a JSON object indicating the error:

```
1  {
2    "success": false,
3    "statusCode": 404,
4    "error": {
5      "title": "Product Not Found",
6      "message": "No product found with ID: 6888f61cbc92c7e91895f470 to add to cart"
7    }
8 }
```

## 22. Add to Cart - Invalid ProductId

- **Expected:** 400 Bad Request, "Invalid Product ID"

The screenshot shows the Thunder Client interface. On the left, the activity log lists several API calls, including a recent POST to `localhost:5050/api/cart`. In the center, a new request is being built with a POST method to `http://localhost:5050/api/cart`. The body contains the JSON object `{"productId": "6888f61cbc92c"}`. The response pane shows a status of 400 Bad Request with the message: 

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Invalid Product ID",
6     "message": "The provided product ID '6888f61cbc92c' is not a valid MongoDB ObjectId."
7   }
8 }
```

A warning message in the bottom right corner states: "Warning: The free version will no longer support collections starting August 3rd, 2025." The system tray at the bottom shows a battery level of 188%.

## 23. Get User Cart

- **GET /api/cart**
- **Expected:** 200 OK, populated cart

The screenshot shows the Thunder Client interface. On the left, the activity log lists several API calls, including a recent GET to `localhost:5050/api/cart`. In the center, a new request is being built with a GET method to `http://localhost:5050/api/cart`. The response pane shows a status of 200 OK with the message: 

```
1 {
2   "success": true,
3   "statusCode": 200,
4   "data": {
5     "_id": "688902e9461771ab374e4b2a",
6     "userId": "688adedeb1b5602fd2c12f5a",
7     "products": [
8       {
9         "productId": {
10           "_id": "6888f61cbc92c7e91895f4d8",
11           "name": "Rolex Cellini Moonphase",
12           "description": "The Rolex Cellini Moonphase is a masterpiece of horology, featuring a moon phase complication and exquisite design. It reflects Rolex's commitment to precision and elegance.",
13           "price": 12999.99,
14           "stock": 36
15       },
16       {
17         "productId": {
18           "_id": "6888f61cbc92c7e91895f50d",
19           "name": "Tennis Ball",
20           "description": "The Tennis Ball is a standard ball used in the sport of tennis. It is designed for bouncing and"
21       }
22     ]
23 }
```

A warning message in the bottom right corner states: "Warning: The free version will no longer support collections starting August 3rd, 2025." The system tray at the bottom shows a battery level of 28% and the date 29-07-2025.

## 24. Update Cart Item - Valid Quantity

- **PUT /api/cart**
- **Request Body:**
- **Expected: 200 OK, Cart Updated Successfully**

The screenshot shows the Thunder Client interface. In the left sidebar, there's a list of activity logs. In the main area, a request is being made to `PUT http://localhost:5050/api/cart`. The body contains a JSON object with `"productId": "6888f61cbc92c7e91895f4d8"` and `"quantity": 4`. The response status is **200 OK**, size is **199 Bytes**, and time is **333 ms**. The response body is a JSON object indicating success, status code 200, message "Cart Updated SuccessFully", and a data array containing two items, each with a product ID, name, description, price, stock, and quantity.

```
1 {
2   "success": true,
3   "statusCode": 200,
4   "message": "Cart Updated SuccessFully",
5   "data": [
6     {
7       "_id": "6888f61cbc92c7e91895f4d8",
8       "userId": "6888beddb1b5602fd2c12f5a",
9       "products": [
10         {
11           "productId": {
12             "_id": "6888f61cbc92c7e91895f4d8",
13             "name": "Rolex Cellini Moonphase",
14             "description": "The Rolex Cellini Moonphase is a masterpiece of horology, featuring a moon phase complication and exquisite design. It reflects Rolex's commitment to precision and elegance.",
15             "price": 12999.99,
16             "stock": 36
17           },
18           "quantity": 4,
19           "_id": "6888f61cbc92c7e91895f4d8"
20         },
21         {
22           "productId": {
23             "_id": "6888f61cbc92c7e91895f50d",
24             "name": "Tennis Ball",
25             "description": "The Tennis Ball is a standard ball used in"
26         }
27     ]
28 }
```

## 25. Update Cart Item - Missing Quantity

- **Expected: 400 Bad Request, "Missing Quantity"**

The screenshot shows the Thunder Client interface. In the left sidebar, there's a list of activity logs. In the main area, a request is being made to `PUT http://localhost:5050/api/cart`. The body contains a JSON object with `"productId": "6888f61cbc92c7e91895f479"`. The response status is **400 Bad Request**, size is **141 Bytes**, and time is **11 ms**. The response body indicates failure with a status code of 400, an error message, and a title "Missing Quantity".

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Missing Quantity",
6     "message": "Quantity must be provided for updating items in the cart"
7   }
8 }
```

## 26. Update Cart Item - Invalid Quantity

- **Expected:** 400 Bad Request, "Invalid Quantity"

The screenshot shows the Thunder Client interface. On the left, a sidebar lists various API endpoints with their methods, URLs, and timestamps. In the center, a request builder window is open for a PUT request to `http://localhost:5050/api/cart`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "productId": "6888f61cbc92c7e91895f485",
3   "quantity": 2.5
4 }
```

The response panel on the right shows a status of 400 Bad Request, with a size of 132 Bytes and a time of 397 ms. The response body is:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Invalid Quantity",
6     "message": "Quantity must be a positive integer (minimum 1)."
7   }
8 }
```

## 26. Update Cart Item - Stock exceeded

- **Expected:** 400 Bad Request, "Stock Limit Exceeded"

The screenshot shows the Thunder Client interface. On the left, a sidebar lists various API endpoints. In the center, a request builder window is open for a PUT request to `http://localhost:5050/api/cart`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "productId": "6888f61cbc92c7e91895f485",
3   "quantity": 65
4 }
```

The response panel on the right shows a status of 400 Bad Request, with a size of 139 Bytes and a time of 433 ms. The response body is:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Stock Limit Exceeded",
6     "message": "Current quantity in cart: 100, stock available: 64."
7   }
8 }
```

## 27. Update Cart Item - Non existing product

- **Expected:** 404 Not Found, "Product Not Found"

The screenshot shows the Thunder Client interface. A PUT request is being made to `http://localhost:5050/api/cart`. The JSON body contains:

```
1 {
2   "productId": "6888f61cbc92c7e91895f4d8",
3   "quantity": 4
4 }
```

The response status is 404 Not Found, with a message: "No product found with ID '6888f61cbc92c7e91895f4d8' in cart".

## 28. Remove Item from Cart

- **DELETE /api/cart/:productId**
- **Expected:** 200 OK, Deleted cart item successfully

The screenshot shows the Thunder Client interface. A DELETE request is being made to `http://localhost:5050/api/cart/6888f61cbc92c7e91895f4d8`. The response status is 200 OK, indicating the cart item was successfully deleted.

## 29. Remove Item from Cart - Invalid ProductID

- **Expected:** 400 Bad Request, "Invalid ProductId"

The screenshot shows the Thunder Client interface. In the left sidebar, there is a list of activity logs. In the main panel, a new request is being prepared for `DELETE http://localhost:5050/api/cart/6888f61cbc`. The Headers tab is selected, showing the following configuration:

Header	Value
Accept	/*
User-Agent	Thunder Client (https://www.thunderclient.co)
Authorization	JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.e
header	value

The Response tab shows the following JSON output:

```
1 {
2   "success": false,
3   "statusCode": 400,
4   "error": {
5     "title": "Invalid productid",
6     "message": "The provided product ID '6888f61cbc' is not a valid MongoDB ObjectId."
7   }
8 }
```

## 30. Clear Entire Cart

- **DELETE /api/cart**
- **Expected:** 200 OK, "Cart cleared"

The screenshot shows the Thunder Client interface. In the left sidebar, there is a list of activity logs. In the main panel, a new request is being prepared for `DELETE http://localhost:5050/api/cart`. The Headers tab is selected, showing the following configuration:

Header	Value
Accept	/*
User-Agent	Thunder Client (https://www.thunderclient.co)
Authorization	JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.e
header	value

The Response tab shows the following JSON output:

```
1 {
2   "success": true,
3   "statusCode": 200,
4   "message": "Cart for user ID 6889beccbaaf2c183421f721 deleted successfully.",
5   "data": []
6 }
```

# Error & Misc Testing

## 31. Access Protected routes without token

- **Expected:** 401 Unauthorized, "Access token missing"

The screenshot shows the Thunder Client interface. In the main panel, a POST request to `http://localhost:5050/api/cart` is being processed. The response status is **401 Unauthorized**, with a size of **113 Bytes** and a time of **375 ms**. The response body contains the following JSON:

```
1 {
2     "success": false,
3     "statusCode": 401,
4     "error": {
5         "title": "Unauthorized",
6         "message": "Access token missing or malformed"
7     }
8 }
```

The left sidebar lists various API requests made by the client, including GET, PUT, and DELETE methods on different endpoints like `/invalidroute`, `/cart`, and `/products`.

## 33. Invalid Route Access

- **GET /invalidroute**
- **Expected:** 404 Not Found, "Invalid Route"

The screenshot shows the Thunder Client interface. A GET request to `http://localhost:5050/invalidroute` is being processed. The response status is **404 Not Found**, with a size of **132 Bytes** and a time of **110 ms**. The response body contains the following JSON:

```
1 {
2     "success": false,
3     "statusCode": 404,
4     "error": {
5         "title": "Invalid Route",
6         "message": "The requested route '/invalidroute' does not exist."
7     }
8 }
```

The left sidebar lists various API requests made by the client, including GET, PUT, and DELETE methods on different endpoints like `/invalidroute`, `/cart`, and `/products`.

## 34. Expired Token

- Expected: 403 Forbidden, "Token Expired"

The screenshot shows the Thunder Client interface. In the left sidebar, there's a list of activity logs. In the main panel, a new request is being sent to `http://localhost:5050/api/cart` using the `DELETE` method. The Headers tab is selected, showing the following configuration:

Header	Value
Accept	/*
User-Agent	Thunder Client (https://www.thunderclient.co)
Authorization	JWT eyJhbGciOiUzI1Ni...C9.e
header	value

The response pane shows a status of 403 Forbidden with the message: "Your token has expired. Please log in again."

## 35. Invalid Token Format

- Expected: 403 Forbidden, "Invalid Token"

The screenshot shows the Thunder Client interface. In the left sidebar, there's a list of activity logs. In the main panel, a new request is being sent to `http://localhost:5050/api/cart` using the `POST` method. The Headers tab is selected, showing the following configuration:

Header	Value
Accept	/*
User-Agent	Thunder Client (https://www.thunderclient.co)
Authorization	JWT fcghijkl
header	value

The response pane shows a status of 403 Forbidden with the message: "Access token is invalid. Please log in again."

## 36. Invalid Json

- **Expected: 400 Bad Request, "Invalid Json"**

The screenshot shows the Thunder Client interface. On the left, the activity log lists various API requests. In the center, a POST request to `http://localhost:5050/api/login` is selected. The 'Body' tab is active, showing the JSON content:

```
1  {
2     "success": false,
3     "statusCode": 400,
4     "error": {
5         "title": "Invalid JSON",
6         "message": "The JSON payload is malformed. Please check your request body syntax."
7     }
8 }
```

The response panel on the right indicates a **Status: 400 Bad Request**, **Size: 149 Bytes**, and **Time: 13 ms**. The response body is identical to the one shown in the body tab.