

ShoppyGlobe Backend API Testing Documentation

This document provides step-by-step instructions and expected results for testing all required API routes using ThunderClient . Screenshots of the requests, responses, and MongoDB structure must be added below the relevant sections for submission compliance.

GitHub Repository Link: [ecommerceShoppyglobe at main · bhavanishankar7075/ecommerceShoppyglobe](https://github.com/bhavanishankar7075/ecommerceShoppyglobe)

Base API URL: <http://localhost:5000/api>

I. MongoDB Database Structure

The project uses MongoDB with the Mongoose ODM to store all persistent application data.

1. Products Collection Setup

Objective: Show the collection schema structure and initial product data insertion.

Proof: Screenshot showing the fields (**name**, **price**, **description**, **stock**, **thumbnail**, **category**) and at least one document inserted into the **products** collection.

2. Cart Collection Structure

Objective: Show how the cart structure links a user to an array of items. The cart document must be user-specific (**user ObjectId reference**) and contain item details (**product ObjectId reference**, **quantity**).

Proof: Screenshot showing the Cart collection schema with the user reference field and the structure of the items array.

II. Authentication & Authorization Routes

These routes secure the application by implementing JWT-based authentication.

1. User Registration (POST /api/register)

Route: POST /api/register

Objective: Register a new user and receive a JWT token.

Expected Status: 201 Created

Body:

```
{  
  "email": "testuser@example.com",  
  "password": "password123"  
}
```

Expected Response: JSON object containing success: true and the generated token.

Screenshot:

The screenshot shows the Thunder Client interface. In the top navigation bar, the URL is set to `http://localhost:5000/api/register`. The main area displays a successful `POST` request with a status of `201 Created`, size of `276 Bytes`, and time of `352 ms`. The `Body` tab shows the JSON response:

```
1 {
2     "message": "User registered successfully.",
3     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
... .eyJpZC16Ijy9Mmtzzjxkjvjkryjb0zTV1zEyyzj0kS1sImhdCI6MTc2NDQ0MjAwM
1wIZXmWjjoxtzyWNTIANMzfo_1ldwM
... -ECVoyxx8LMm6SUjs7Fqow",
4     "userId": "692b3f91b0db84e5eea2c2d1",
5     "success": true
6 }
```

2. User Login (POST /api/login)

Route: POST /api/login

Objective: Authenticate the user and receive a new JWT token.

Expected Status: 200 OK

Body: (Use the same email/password used for registration)

```
{
    "email": "testuser@example.com",
    "password": "password123"
}
```

Expected Response: JSON object containing **success: true** and the **token**. (The token must be saved for all subsequent Cart requests)

Screenshot:

The screenshot shows the Thunder Client interface. In the top navigation bar, the URL is set to `http://localhost:5000/api/login`. The main area displays a successful `POST` request with a status of `200 OK`, size of `264 Bytes`, and time of `198 ms`. The `Body` tab shows the JSON response:

```
1 {
2     "message": "Login successful.",
3     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
... .eyJpZC16Ijy9SMztzzjxkjvjkryjb0zTV1zEyyzj0kS1sImhdCI6MTc2NDQ0MjEzM
1wIZXmWjjoxtzyWNTIANMzfo_1ldwM
... -OpnvYBKMGEKF3tu8mfbxmZ3GQ07h3C_WPYI",
4     "userId": "692b3f91b0db84e5eea2c2d1",
5     "success": true
6 }
```

III. Product Routes (Public Access)

These routes are public and allow any client (including the unauthenticated frontend) to retrieve product information.

1. Fetch All Products (GET /api/products)

Route: GET /api/products

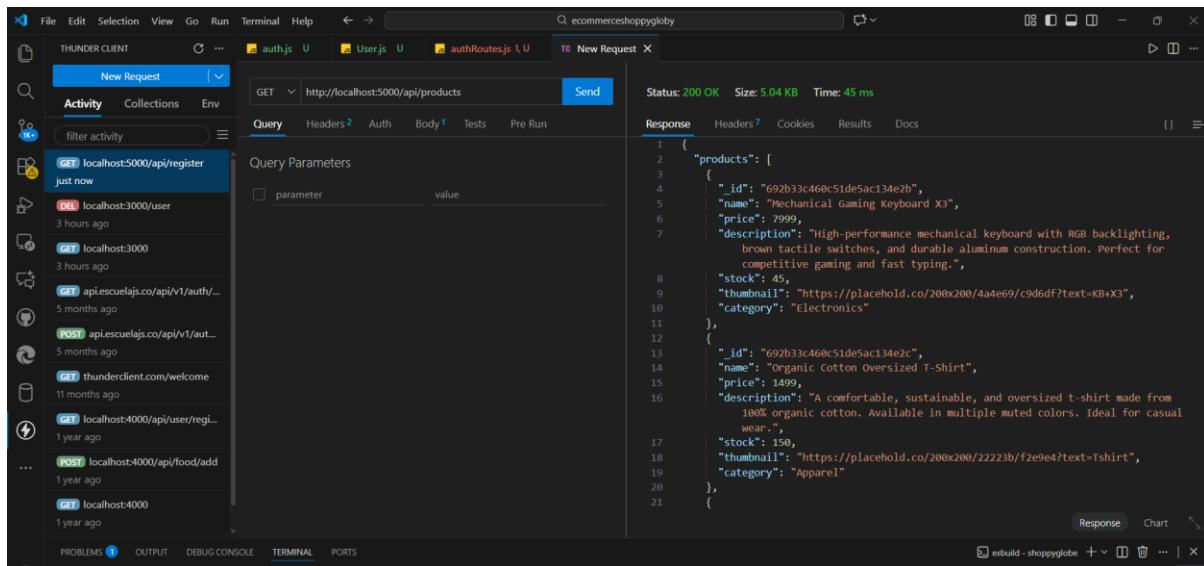
Objective: Retrieve the full list of products stored in MongoDB.

Expected Status: 200 OK

Headers: None required.

Expected Response: JSON object containing the **products** array.

Screenshot:



```
1  {
2    "products": [
3      {
4        "_id": "692b33c460c51de5ac134e2b",
5        "name": "Mechanical Gaming Keyboard X3",
6        "price": 7999,
7        "description": "High-performance mechanical keyboard with RGB backlighting, brown tactile switches, and durable aluminum construction. Perfect for competitive gaming and fast typing.",
8        "stock": 45,
9        "thumbnail": "https://placehold.co/200x200/4a4e69/c9d6df?text=KB+X3",
10       "category": "Electronics"
11     },
12     {
13       "_id": "692b33c460c51de5ac134e2c",
14       "name": "Organic Cotton Oversized T-Shirt",
15       "price": 1499,
16       "description": "A comfortable, sustainable, and oversized t-shirt made from 100% organic cotton. Available in multiple muted colors. Ideal for casual wear.",
17       "stock": 150,
18       "thumbnail": "https://placehold.co/200x200/22223b/f2e9e4?text=Tshirt",
19       "category": "Apparel"
20     }
21   ]
```

2. Fetch Single Product (GET /api/products/:id)

Route: GET /api/products/692b340f60c51de5ac134e30

Objective: Retrieve details for a single product.

Expected Status: 200 OK

Headers: None required.

Expected Response: JSON object containing the **single product object**.

Screenshot:

```

1 {
2   "product": {
3     "_id": "692b340f60c51de5ac13d30",
4     "name": "High-Resolution 4K LED Monitor",
5     "price": 28500,
6     "description": "27-inch 4K UHD monitor with HDR support and slim bezels. Ideal for professional graphic design and high-end gaming.",
7     "stock": 10
8     "thumbnail": "https://placeholder.co/200x200/4F7C73/E9E9E9?text=4K+Monitor",
9     "category": "Electronics"
10   },
11   "success": true
12 }

```

IV. Protected Cart Routes (CRUD Operations)

Crucial Requirement: All routes in this section require the JWT token to be included in the request header.

Header Required for ALL Cart Requests:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJpZCI6IjY5MmlzMjZlYjBkYjg0ZTVlZWYzE1YSlsImhdCI6MTc2NDQ0MjM5NSwiZXhwIjoxNzY0NTI4Nzk1fQ.zSLlgVaUlo1pyhSWqPFxdi8x9ROMKoDBuISHaDOqp54

1. Get Cart Items (GET /api/cart)

Route: GET /api/cart

Objective: Retrieve the current cart contents for the authenticated user.

Expected Status: 200 OK

Initial Expected Response:

{"cart": {"items": []}, "success": true} (If the user's cart is empty).

Screenshot:

```

1 {
2   "cart": {
3     "_id": "692b343eb0db84e5eea2c169",
4     "user": "692b326eb0db84e5eea2c15a",
5     "items": [],
6     "y": 22
7   },
8   "success": true
9 }

```

2. Add Product to Cart (POST /api/cart)

Route: POST /api/cart

Objective: Add a product (e.g., quantity 2) to the user's cart.

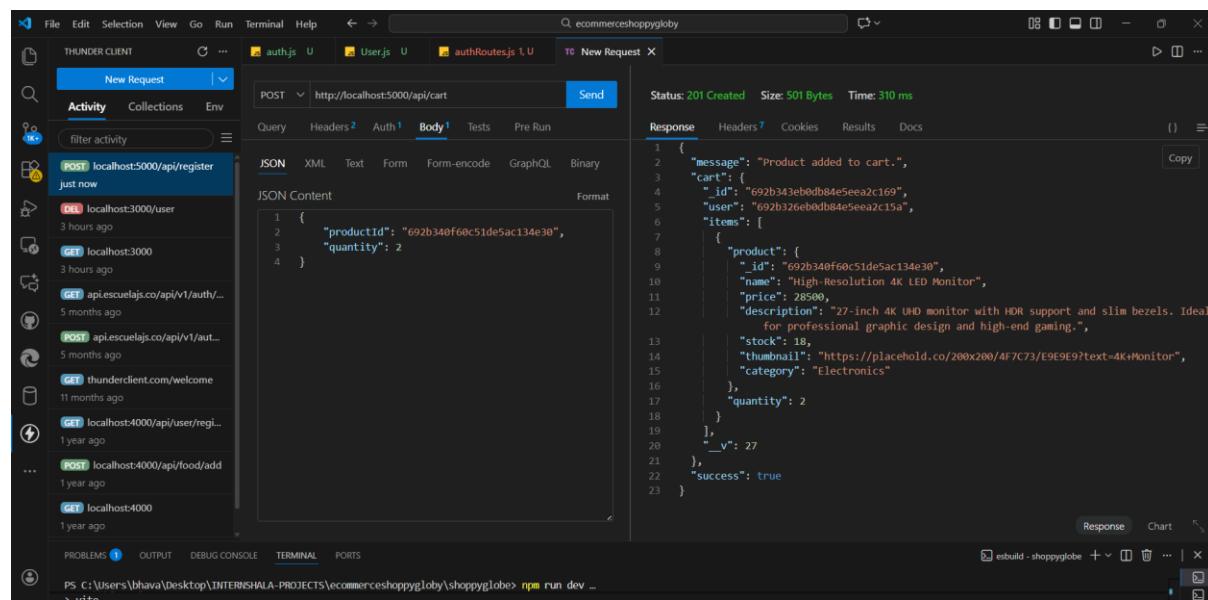
Expected Status: 201 Created

Body:

```
{  
  "productId": "692b340f60c51de5ac134e30",  
  "quantity": 2  
}
```

Expected Response: JSON object containing the updated cart with the new item.

Screenshot:



3. Update Cart Quantity (PUT /api/cart/:productId)

Route: PUT /api/cart/692b340f60c51de5ac134e30

Objective: Change the quantity of Product 1 (already in the cart) to 5.

Expected Status: 200 OK

Body:

```
{  
  "quantity": 5  
}
```

Expected Response: JSON object containing the updated cart showing the new quantity.

Screenshot:

The screenshot shows the Thunder Client interface. In the top bar, it says "ecommercehoppyglob". The main area has a "New Request" button, a dropdown for "PUT http://localhost:5000/api/cart/692b340f60c51de5ac134e30", and a "Send" button. Below this, there are tabs for "Query", "Headers", "Auth", "Body", "Tests", and "Pre Run". The "Body" tab is selected and contains a JSON payload:

```
1 {
2     "message": "Cart quantity updated.",
3     "cart": {
4         "id": "692b343eb0db84e5eea2c169",
5         "user": "692b326e00db84e5eea2c15a",
6         "items": [
7             {
8                 "product": {
9                     "id": "692b340f60c51de5ac134e30",
10                    "name": "High-Resolution 4K LED Monitor",
11                    "price": 2850,
12                    "description": "27-inch 4K UHD monitor with HDR support and slim bezels. Ideal for professional graphic design and high-end gaming.",
13                    "stock": 18,
14                    "thumbnail": "https://placeholder.co/200x200/4F7C73/E9E9E9?text=4K+Monitor",
15                    "category": "Electronics"
16                },
17                "quantity": 5
18            }
19        ],
20        "v": 27
21    },
22    "success": true
23 }
```

The response section shows a status of "200 OK", size of "501 Bytes", and time of "305 ms". The response body is identical to the one sent.

4. Remove Product from Cart (DELETE /api/cart/:productId)

Route: DELETE /api/cart/692b340f60c51de5ac134e30

Objective: Remove Product 1 entirely from the cart.

Expected Status: 200 OK

Body: None.

Expected Response: JSON object containing the updated cart with the item removed.

Screenshot:

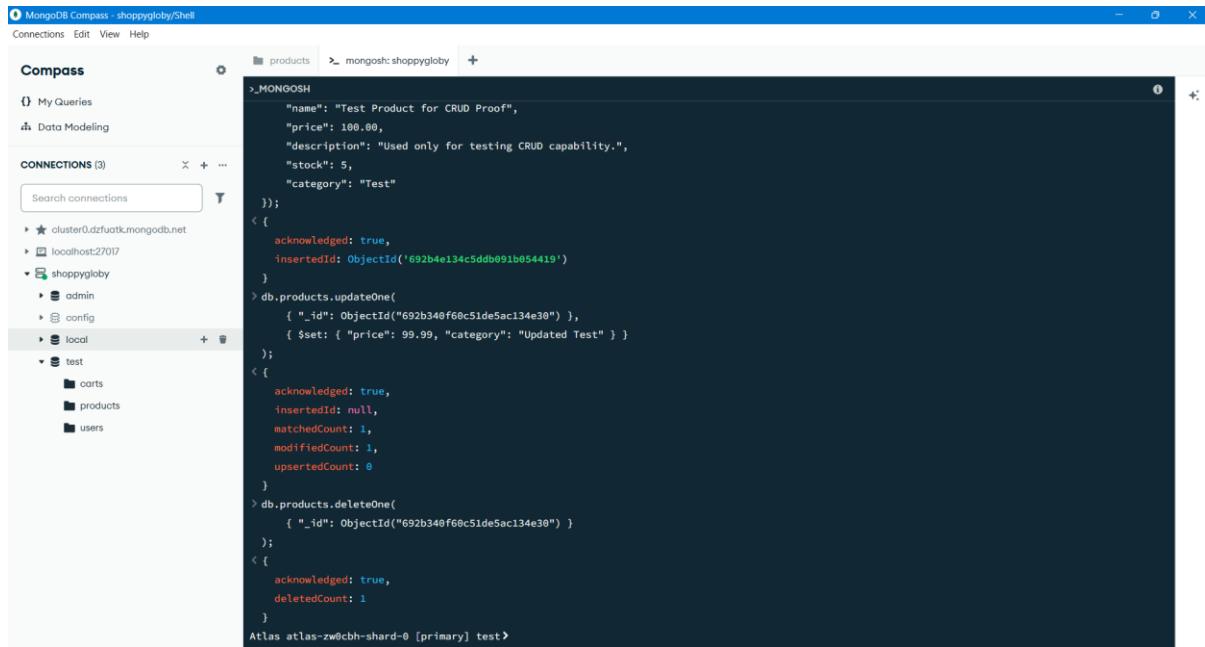
The screenshot shows the Thunder Client interface. In the top bar, it says "ecommercehoppyglob". The main area has a "New Request" button, a dropdown for "DELETE http://localhost:5000/api/cart/692b340f60c51de5ac134e30", and a "Send" button. Below this, there are tabs for "Query", "Headers", "Auth", "Body", "Tests", and "Pre Run". The "Query Parameters" section is expanded, showing a single parameter "v" with a value of "28". The "Body" tab is selected and shows no content.

The response section shows a status of "200 OK", size of "151 Bytes", and time of "281 ms". The response body is:

```
1 {
2     "message": "Product removed from cart.",
3     "cart": {
4         "id": "692b343eb0db84e5eea2c169",
5         "user": "692b326e00db84e5eea2c15a",
6         "items": [],
7         "v": 28
8     },
9     "success": true
10 }
```

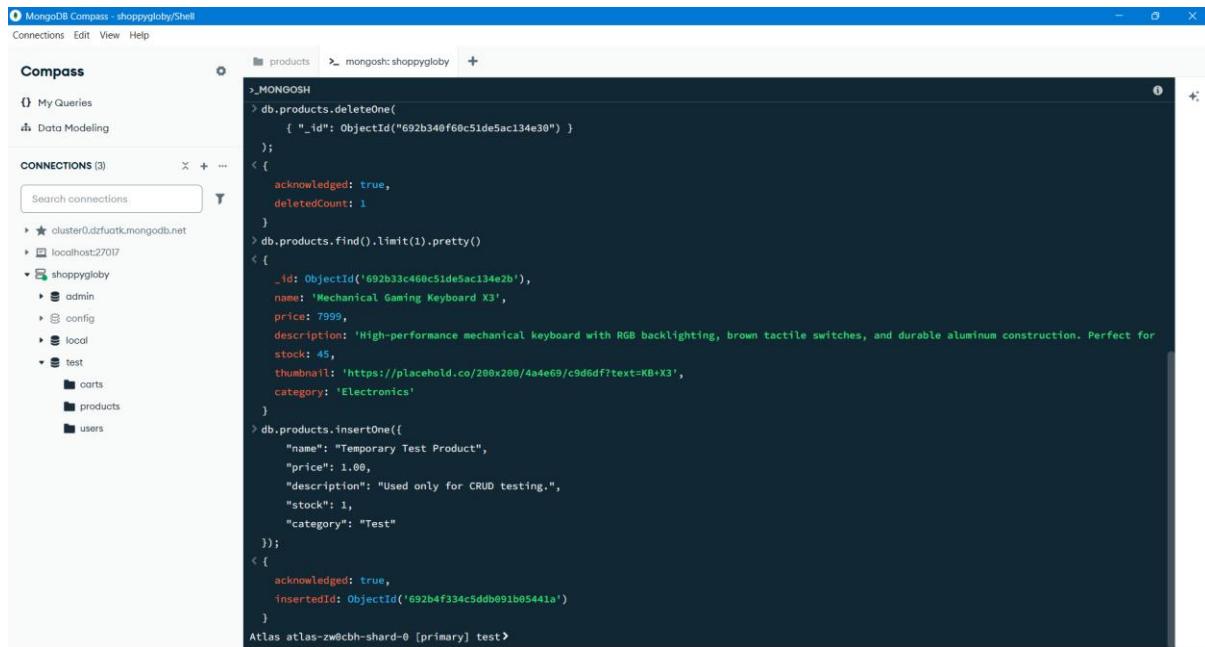
Implement CRUD operations on MongoDB collections

ALL CRUD OPERATION ON THE CART



```
> MONGOSH
> db.products.updateOne(
  { "_id": ObjectId("692b340f60c51de5ac134e30") },
  { $set: { "price": 99.99, "category": "Updated Test" } }
);
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.products.deleteOne(
  { "_id": ObjectId("692b340f60c51de5ac134e30") }
);
< {
  acknowledged: true,
  deletedCount: 1
}
Atlas atlas-zw@cbh-shard-0 [primary] test>
```

CRUD OPERATIONS ON THE PRODUCTLIST



```
> MONGOSH
> db.products.deleteOne(
  { "_id": ObjectId("692b340f60c51de5ac134e30") }
);
< {
  acknowledged: true,
  deletedCount: 1
}
> db.products.find().limit(1).pretty()
< {
  _id: ObjectId('692b33c460c51de5ac134e2b'),
  name: 'Mechanical Gaming Keyboard X3',
  price: 7999,
  description: 'High-performance mechanical keyboard with RGB backlighting, brown tactile switches, and durable aluminum construction. Perfect for',
  stock: 45,
  thumbnail: 'https://placeholder.co/200x200/4a4e69/c9d6df?text=KB+X3',
  category: 'Electronics'
}
> db.products.insertOne({
  "name": "Temporary Test Product",
  "price": 1.00,
  "description": "Used only for CRUD testing.",
  "stock": 1,
  "category": "Test"
});
< {
  acknowledged: true,
  insertedId: ObjectId('692b4f334c5ddb091b05441a')
}
Atlas atlas-zw@cbh-shard-0 [primary] test>
```

MongoDB Compass - shoppygloby/Shell

Connections Edit View Help

Compass

My Queries Data Modeling

CONNECTIONS (3)

Search connections

cluster0.dzfuo7k.mongodb.net

localhost:27017

shoppygloby

- admin
- config
- local
- test
 - carts
 - products
 - users

MONGOSH

```
> db.products.insertOne({
  "name": "Temporary Test Product",
  "price": 1.00,
  "description": "Used only for CRUD testing.",
  "stock": 1,
  "category": "Test"
});

< {
  acknowledged: true,
  insertedId: ObjectId('692b4f334c5ddb091b05441a')
}

> db.products.updateOne(
  { "_id": ObjectId("692b4f334c5ddb091b05441a") },
  { $set: { "price": 99.99, "stock": 99 } }
);

< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

> db.products.deleteOne(
  { "_id": ObjectId("692b4f334c5ddb091b05441a") }
);

< {
  acknowledged: true,
  deletedCount: 1
}
```