

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: <https://github.com/Chakriroy11/shoppyglobe>

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.

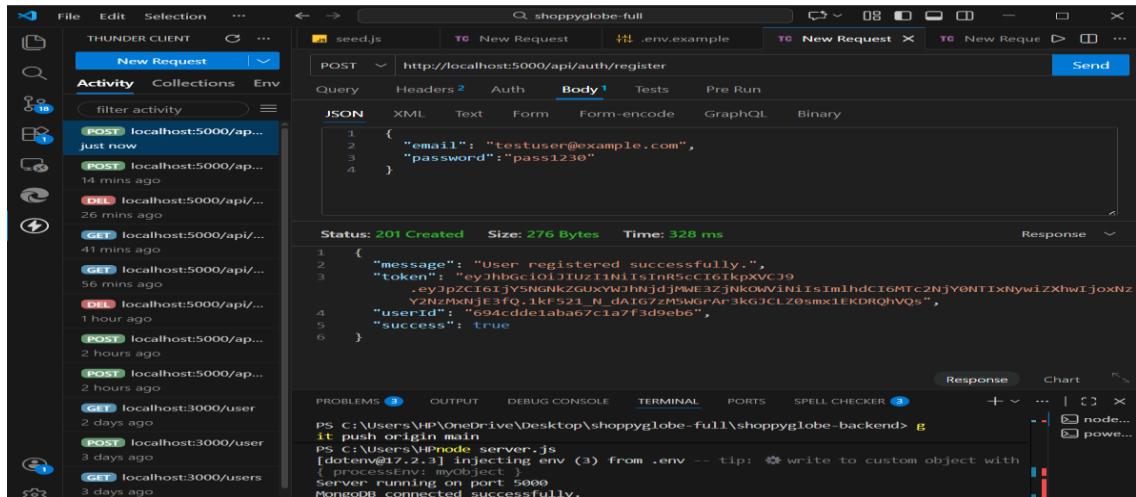
Expected Status: 201 Created

Body:

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

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

Screenshot:



2. User Login (POST /api/login)

Route: POST /api/login

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

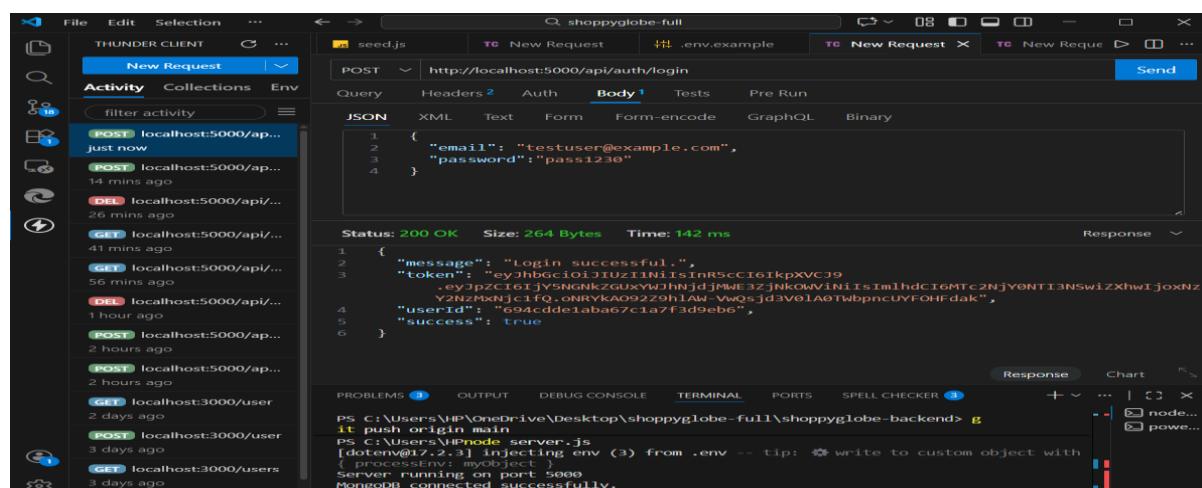
Expected Status: 200 OK

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:



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

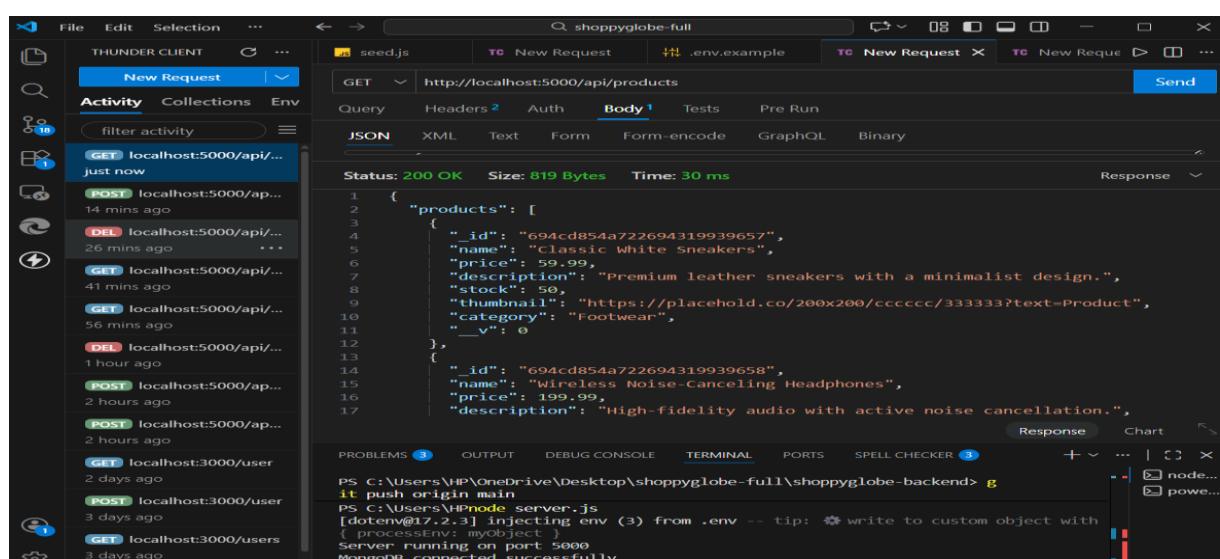
Objective: Retrieve the full list of products stored in MongoDB.

Expected Status: 200 OK

Headers: None required.

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

Screenshot:



The screenshot shows the Thunder Client interface. On the left, there's a sidebar with activity logs and a file tree. In the center, a request is being viewed for a GET request to `http://localhost:5000/api/products`. The response status is 200 OK, size is 819 Bytes, and time is 30 ms. The response body is a JSON array of products:

```
1  {
2     "products": [
3         {
4             "_id": "694cd854a722694319939657",
5             "name": "Classic White Sneakers",
6             "price": 59.99,
7             "description": "Premium leather sneakers with a minimalist design.",
8             "stock": 50,
9             "thumbnail": "https://placehold.co/200x200/cccccc/333333?text=Product",
10            "category": "Footwear",
11            "__v": 0
12        },
13        {
14            "_id": "694cd854a722694319939658",
15            "name": "Wireless Noise-Canceling Headphones",
16            "price": 199.99,
17            "description": "High-fidelity audio with active noise cancellation."
18        }
19    ]
20}
```

At the bottom, the terminal shows the command to start the Node.js server: `node server.js`.

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

Route: GET /api/products/692b340f60c51de5ac134e30

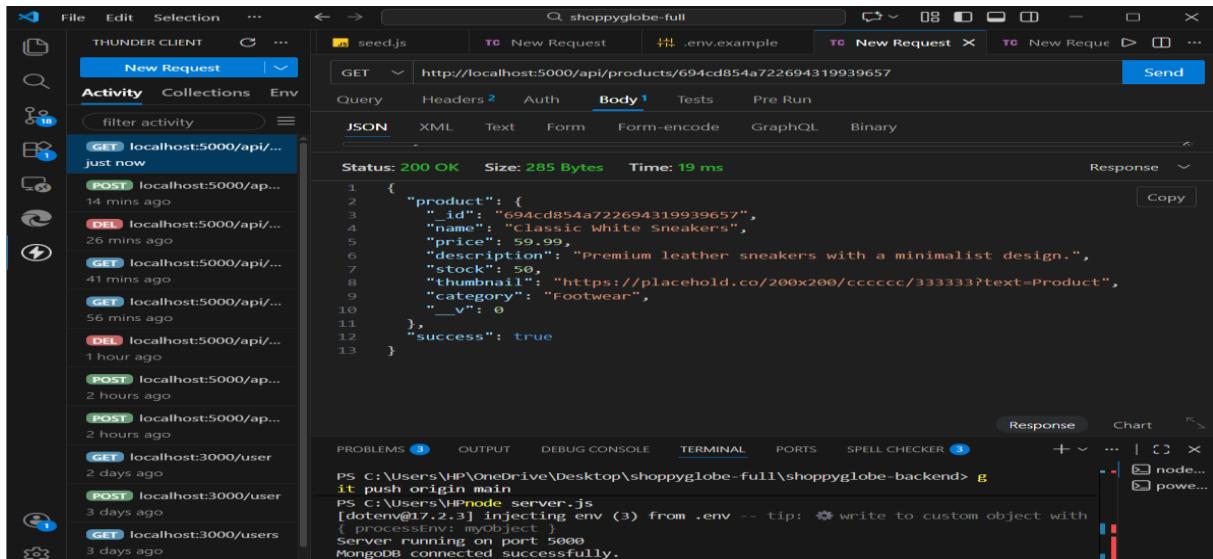
Objective: Retrieve details for a single product.

Expected Status: 200 OK

Headers: None required.

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

Screenshot:



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

eyJhbGciOiJIUzI1NilsInR5cCl6IkpxVCJ9.eyJpZCI6IjY5MmlzMjZlYjBkYjg0ZTVlZWeyYzE1YSIsImlhCl6MTc2NDQ0MjM5NSwiZXhwIjoxNzY0NTI4Nzk1fQ.zSLlgVaUlo1pyhSWqPFxdI8x9ROMKoDBuLSHaDOqp54

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

Route: GET /api/cart

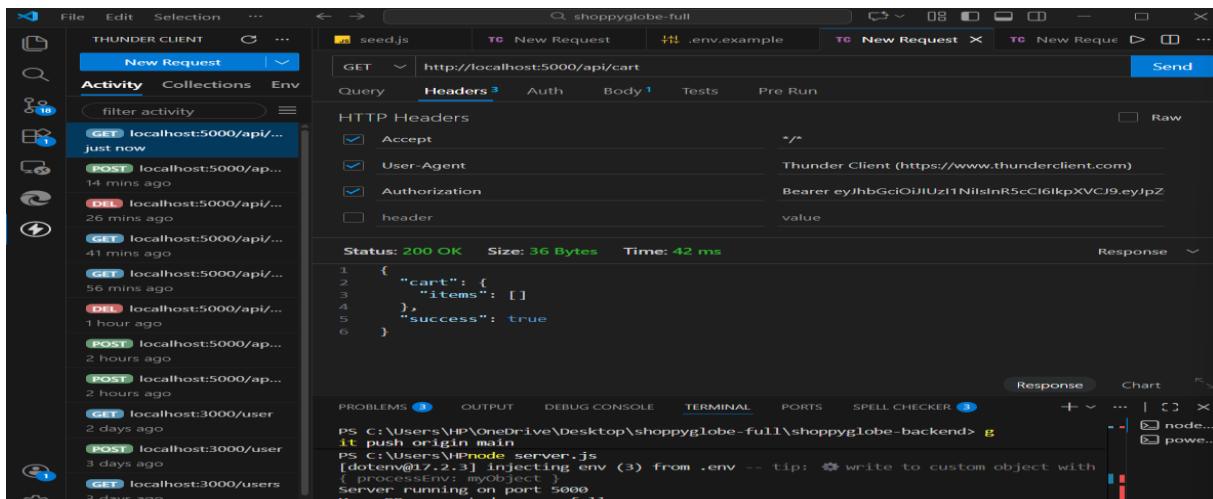
Objective: Retrieve the current cart contents for the authenticated user.

Expected Status: 200 OK

Initial Expected Response:

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

Screenshot:



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.

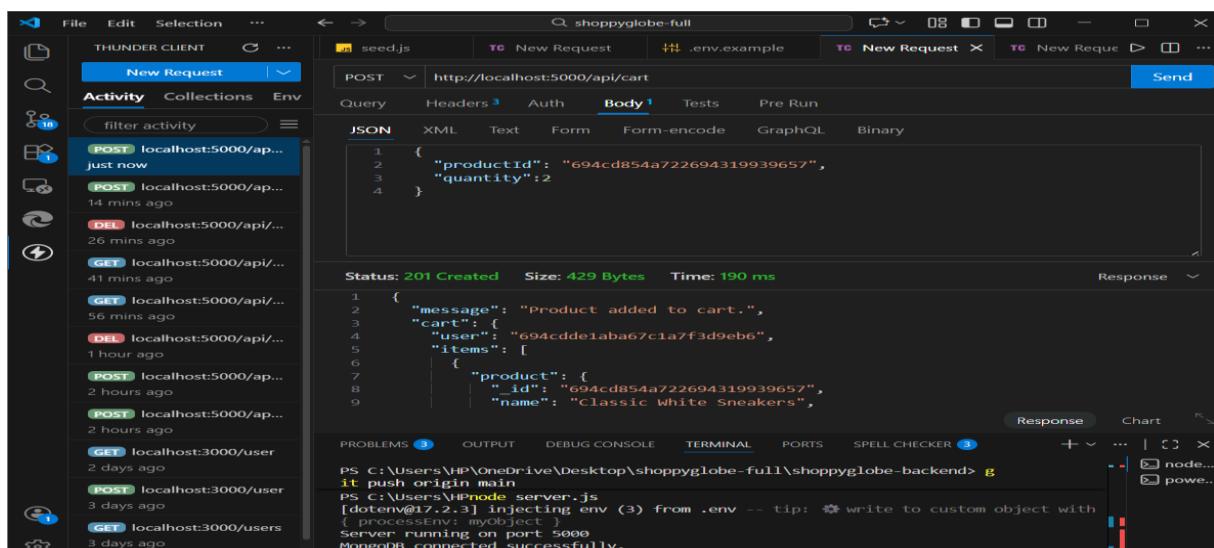
Expected Status: 201 Created

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.

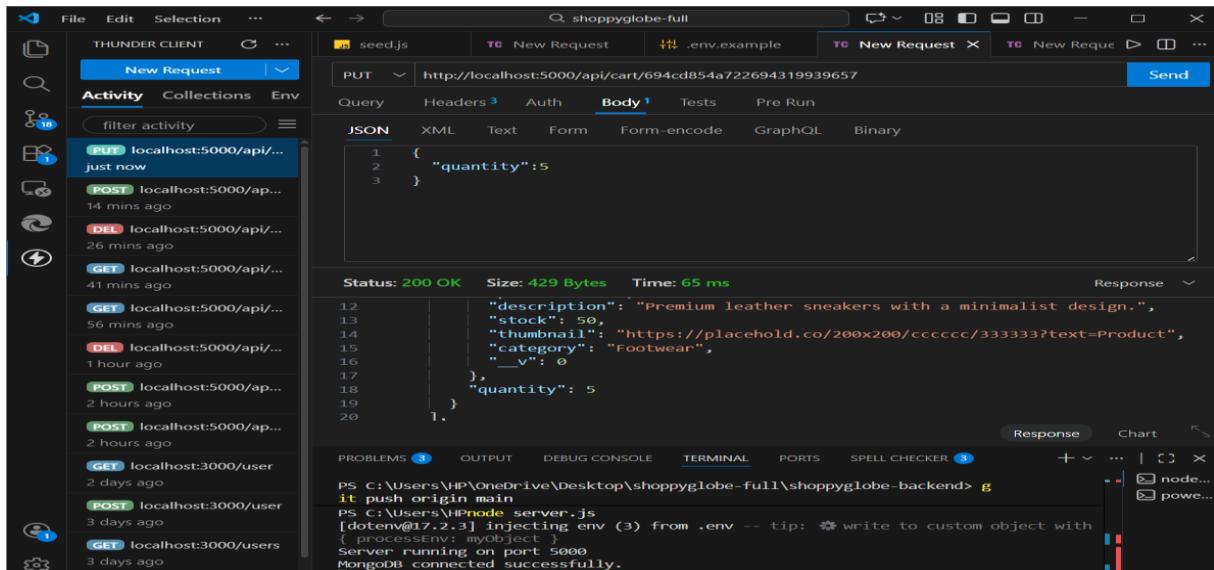
Expected Status: 200 OK

Body:

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

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

Screenshot:



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

Route: DELETE /api/cart/692b340f60c51de5ac134e30

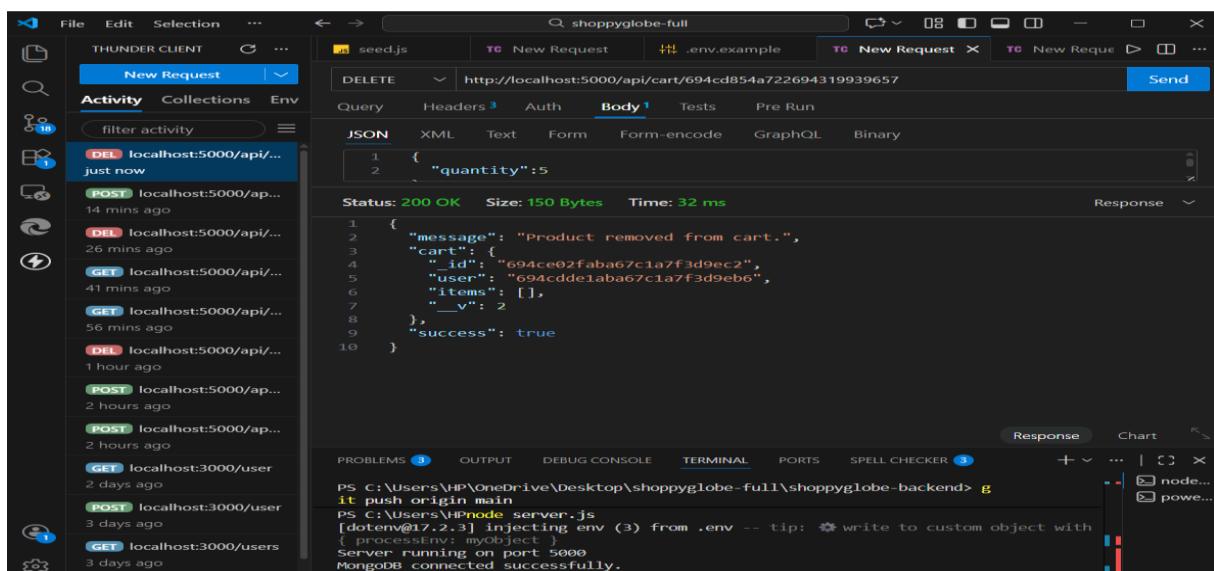
Objective: Remove Product 1 entirely from the cart.

Expected Status: 200 OK

Body: None.

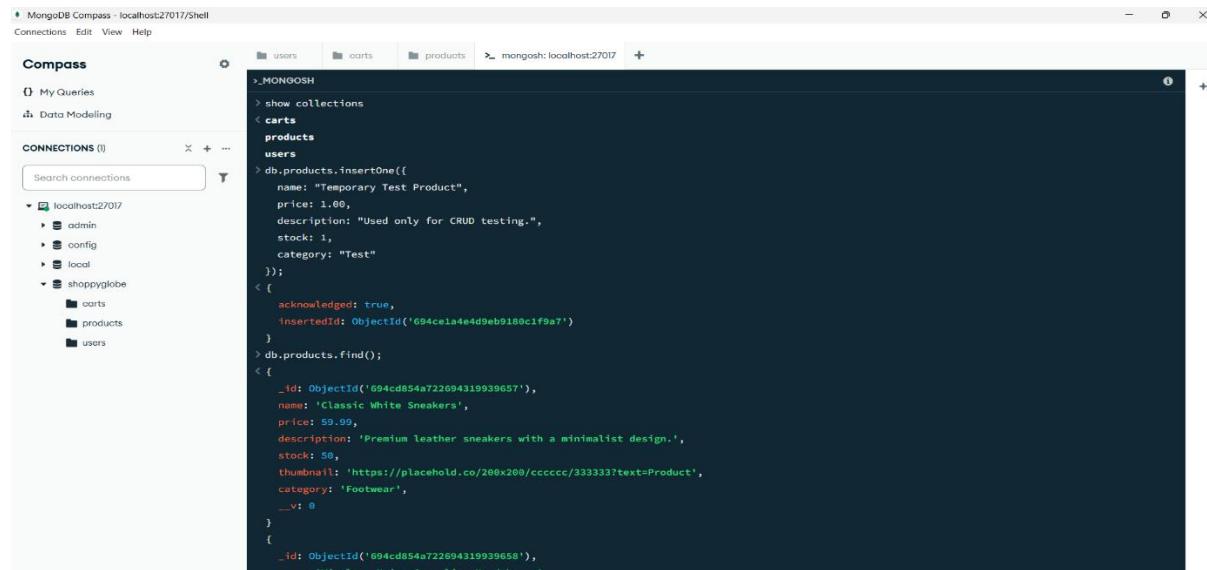
Expected Response: JSON object containing the updated cart with the item removed.

Screenshot:



Implement CRUD operations on MongoDB collections

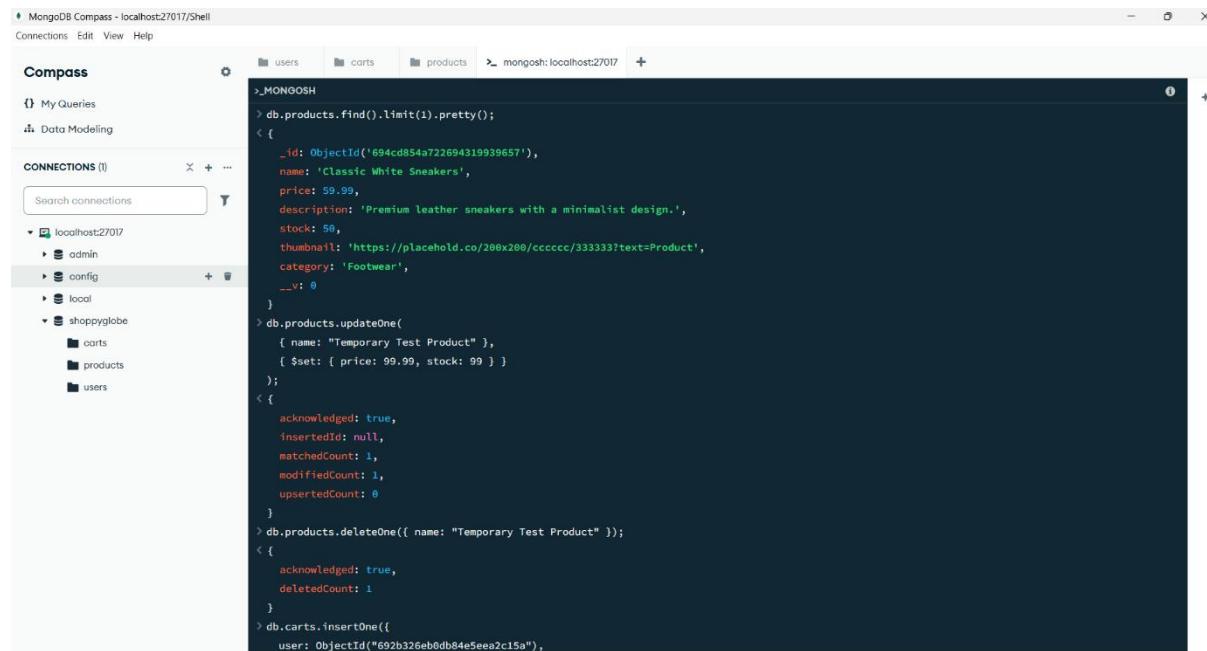
ALL CRUD OPERATION ON THE CART



MongoDB Compass - localhost:27017/Shell

```
>_MONGOSH
> show collections
< carts
  products
  users
> db.products.insertOne({
  name: "Temporary Test Product",
  price: 1.00,
  description: "Used only for CRUD testing.",
  stock: 1,
  category: "Test"
});
< {
  acknowledged: true,
  insertedId: ObjectId('694ce1a4e4d9eb9180c1f9a7')
}
> db.products.find();
< [
  {
    _id: ObjectId('694cd854a722694319939657'),
    name: 'Classic White Sneakers',
    price: 59.99,
    description: 'Premium leather sneakers with a minimalist design.',
    stock: 50,
    thumbnail: 'https://placehold.co/200x200/cccccc/33333?text=Product',
    category: 'Footwear',
    __v: 0
  },
  {
    _id: ObjectId('694cd854a722694319939658'),
    name: 'Vintage Denim Jeans',
    price: 29.99,
    description: 'Washable denim jeans with a classic fit and subtle distressing.',
    stock: 100,
    thumbnail: 'https://placehold.co/200x200/999999/33333?text=Product',
    category: 'Clothing',
    __v: 0
  }
]
```

CRUD OPERATIONS ON THE PRODUCTLIST



MongoDB Compass - localhost:27017/Shell

```
>_MONGOSH
> db.products.find().limit(1).pretty();
< [
  {
    _id: ObjectId('694cd854a722694319939657'),
    name: 'Classic White Sneakers',
    price: 59.99,
    description: 'Premium leather sneakers with a minimalist design.',
    stock: 50,
    thumbnail: 'https://placehold.co/200x200/cccccc/33333?text=Product',
    category: 'Footwear',
    __v: 0
  }
]
> db.products.updateOne(
  { name: "Temporary Test Product" },
  { $set: { price: 99.99, stock: 99 } }
);
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.products.deleteOne({ name: "Temporary Test Product" });
< {
  acknowledged: true,
  deletedCount: 1
}
> db.carts.insertOne({
  user: ObjectId("692b326eb0db84e5eea2c15a"),
  items: [
    {
      product: ObjectId('694cd854a722694319939657'),
      quantity: 1
    }
  ]
});
```

MongoDB Compass - localhost:27017/Shell

Connections Edit View Help

Compass

{} My Queries

DATA Modeling

CONNECTIONS ()

Search connections

localhost:27017

- admin
- config
- local
 - carts
 - products
 - users
- shoppyglobe
 - carts
 - products
 - users

>_MONGOSH

```
>]
  });
<{
  acknowledged: true,
  insertedId: ObjectId('694ce247e4d9eb9180c1f9a8')
}
> db.carts.updateOne(
  { user: ObjectId("692b326eb0db84e5ea2c15a") },
  { $set: { "items.0.quantity": 10 } }
);
<{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.carts.updateOne(
  { user: ObjectId("692b326eb0db84e5ea2c15a") },
  { $set: { items: [] } }
);
<{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
shoppyglobe>|
```