ShoppyGlobe E-commerce API Documentation

The ShoppyGlobe E-commerce API is built using Node.js, Express.js, and MongoDB to provide backend functionalities for an e-commerce application. This project involves creating, reading, updating, and deleting (CRUD) data for products and cart items, implementing user authentication and authorization, and ensuring secure access to cart operations.

Key Features

1. **Node.js & Express Setup**
   - Routes:
     - **GET /products** - Fetch all products.
     - **GET /products/**
       - Fetch single product by ID.
     - **POST /cart** - Add product to cart.
     - **PUT /cart/**
       - Update product quantity in cart.
     - **DELETE /cart/**
       - Remove product from cart.
2. **MongoDB Integration**
   - Collections: `Products` (product data) and `Cart` (cart items).
   - CRUD operations for products and cart items.
3. **Error Handling & Validation**
   - Checks for invalid IDs and stock availability.
4. **Authentication & Authorization**
   - **JWT** for secure access.
   - Routes for **/register** and **/login**.
   - Protects cart routes to allow only logged-in users.
5. **Testing with ThunderClient**
   - Tested all routes with ThunderClient, ensuring correct responses and error handling.

Challenges

- **ID Comparisons**: Used `ObjectId` for consistency.
- **Authorization**: JWT for protecting cart routes.
- **Error Handling**: Custom error messages for better clarity.

Conclusion

The ShoppyGlobe API provides a secure, scalable backend for e-commerce, enabling product management, cart operations, and user authentication. Built with Node.js, Express, and MongoDB, it ensures reliability through structured error handling and route protection.

Folder Structure

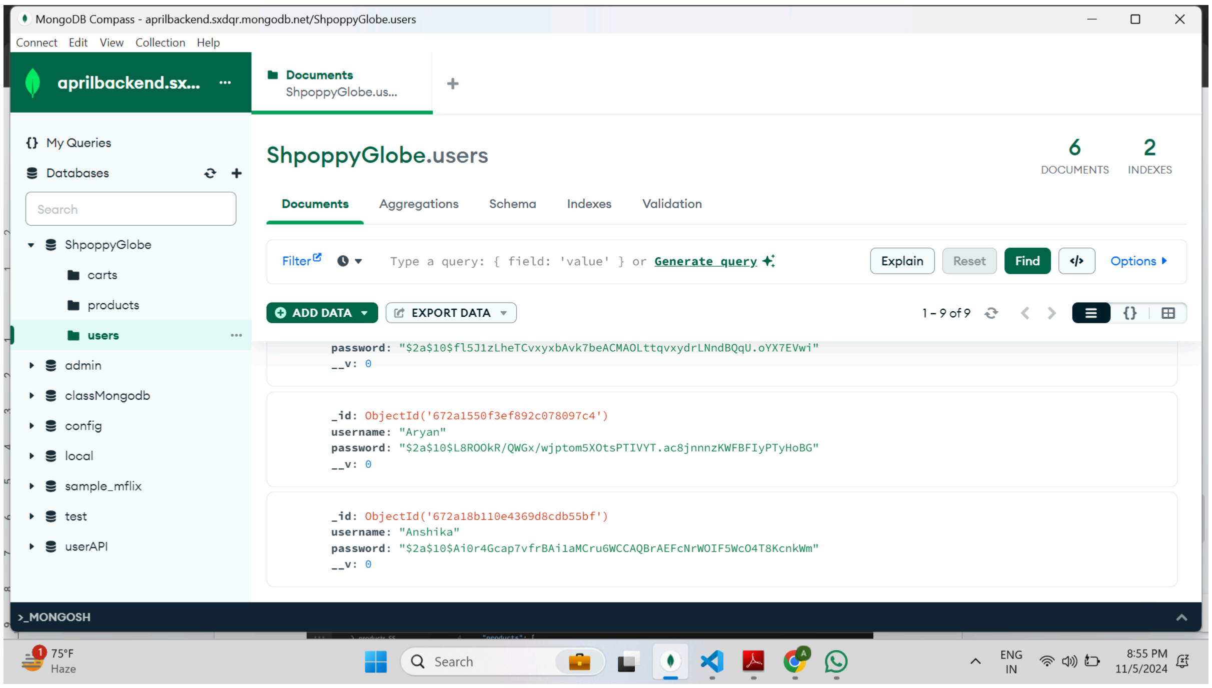
```
ShoppyGlobe/
├── controllers/
│   ├── User.js
│   ├── productController.js
│   └── cartController.js
│
├── models/
│   ├── userModel.js                # Defines user schema
│   ├── productModel.js             # Defines product schema
│   └── cartModel.js                # Defines cart schema
│
├── routes/
│   ├── User.js                     # Routes register, login)
│   ├── productRoutes.js            # Routes for product
│   └── cartRoutes.js               # Routes for cart operations
│
├── middleware/
│   └── authMiddleware.js   #JWT verification for protected routes
│
├── config/
│   └── db.js                       # MongoDB connection setup
│
├── .env                            # Environment variables
├── app.js                          # Main application
├── server.js                       # Server setup and startup file
└── README.md                       # Project documentation
```

GITHUB lINK :-

https://github.com/Ashmita2282/ShoppyGlobe-1

SCREENSHOT OF MONGODB

1. **Users Collections :** All the registered Users

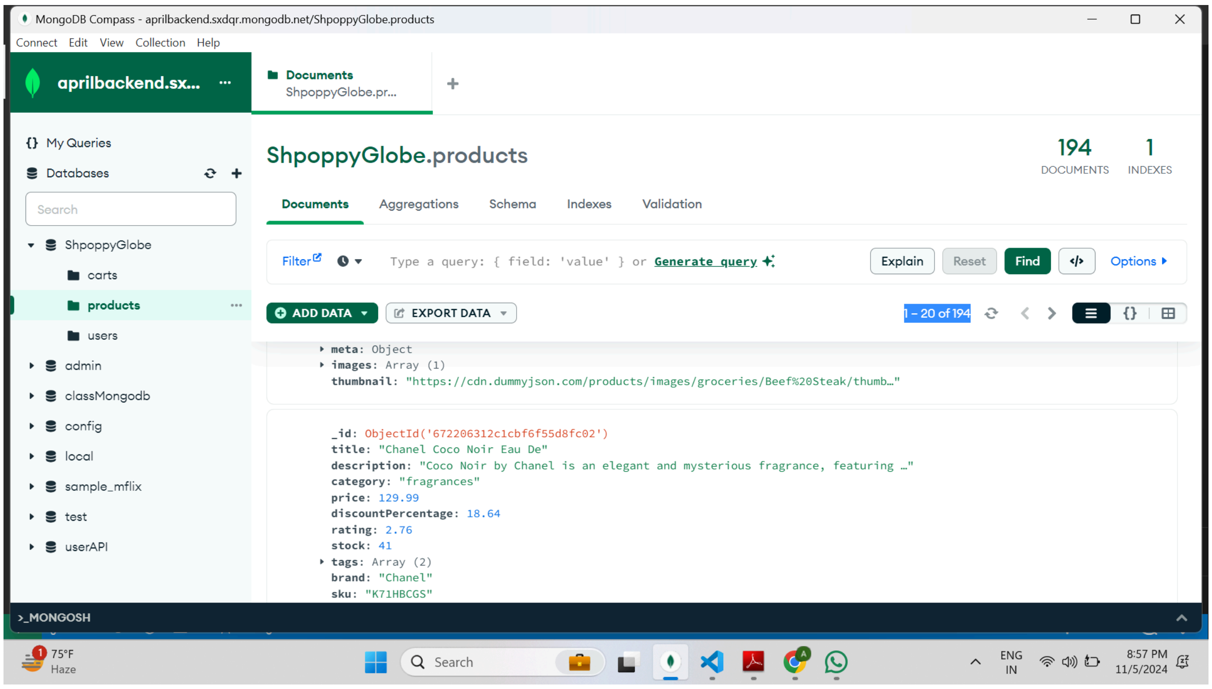


```
password: "$2a$10$fl5J1zLheTCvxyxbAvk7beACMAOLttqvxydrLNndBQqU.oYX7EVwi"
__v: 0

_id: ObjectId('672a1550f3ef892c078097c4')
username: "Aryan"
password: "$2a$10$L8ROOkR/QWGx/wjptom5XOtsPTIVYT.ac8jnnnzKWFBFIyPTyHoBG"
__v: 0

_id: ObjectId('672a18b110e4369d8cdb55bf')
username: "Anshika"
password: "$2a$10$Ai0r4Gcap7vfrBAi1aMCru6WCCAQBrAEFcNrWOIF5Wc04T8KcnkWm"
__v: 0
```

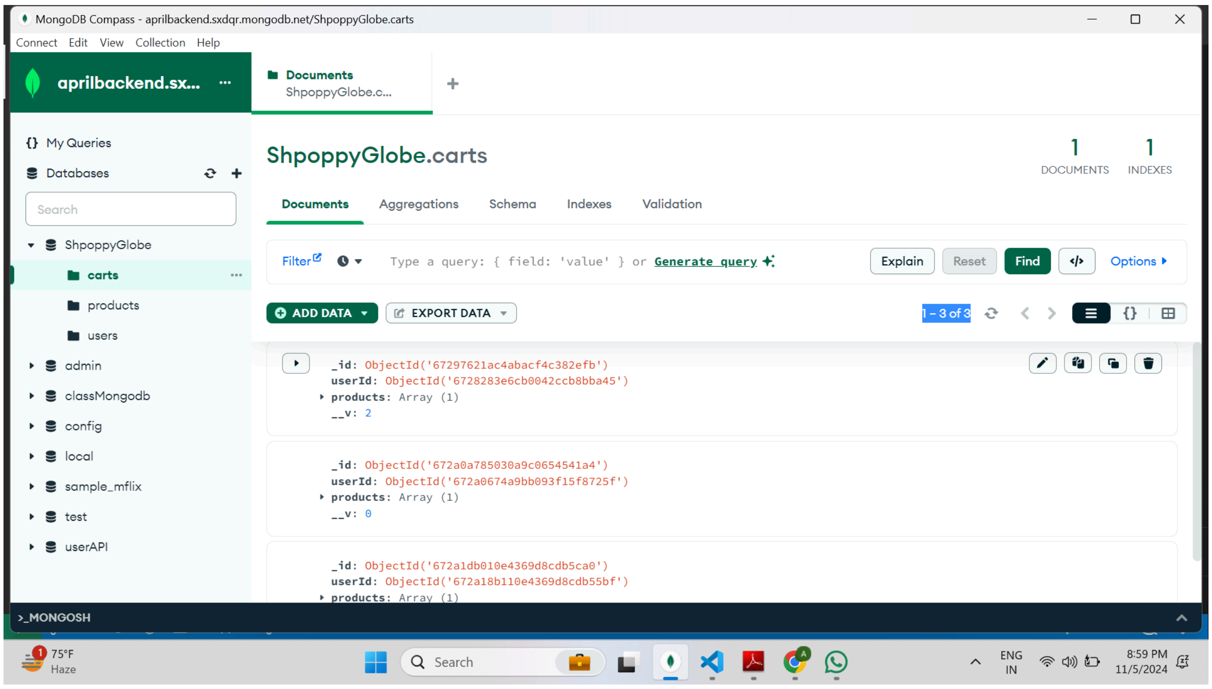
2. **Product Collections:** All the products



```
▸ meta: Object
▸ images: Array (1)
  thumbnail: "https://cdn.dummyjson.com/products/images/groceries/Beef%20Steak/thumb…"

  _id: ObjectId('672206312c1cbf6f55d8fc02')
  title: "Chanel Coco Noir Eau De"
  description: "Coco Noir by Chanel is an elegant and mysterious fragrance, featuring …"
  category: "fragrances"
  price: 129.99
  discountPercentage: 18.64
  rating: 2.76
  stock: 41
▸ tags: Array (2)
  brand: "Chanel"
  sku: "K71HBCGS"
```

3. **Cart Collection:** Stores all the items added to users' shopping carts, including product IDs, quantities, and user references.

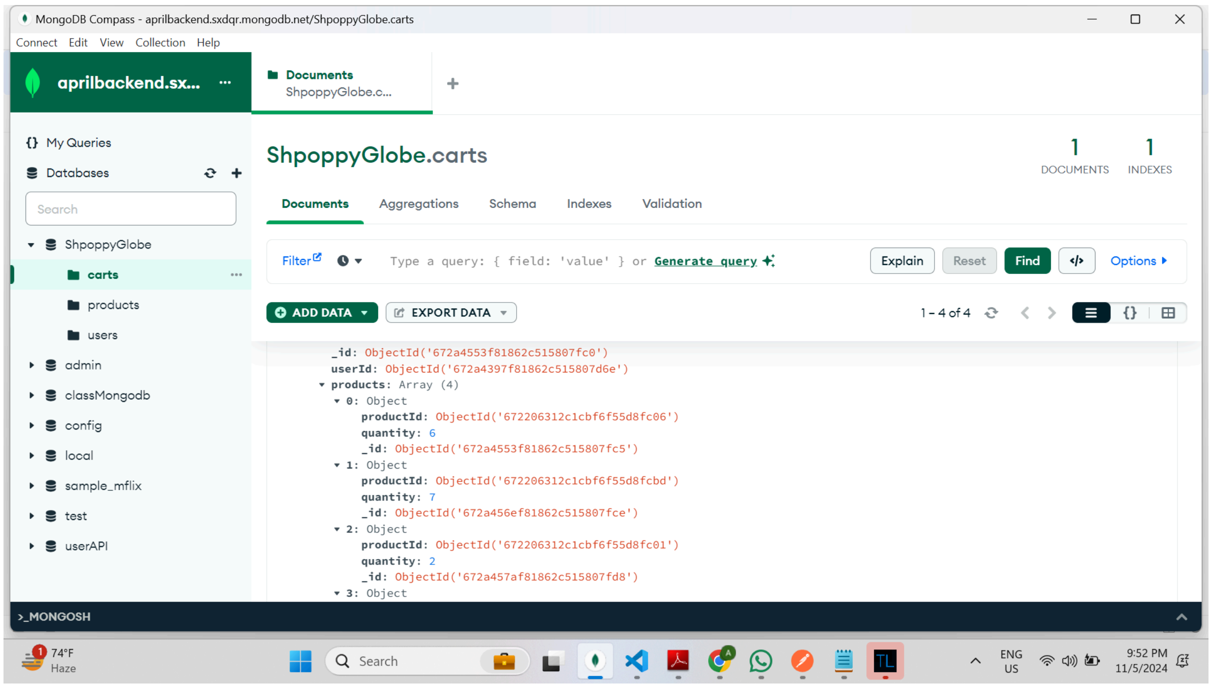

MongoDB Compass - aprilbackend.sxdqr.mongodb.net/ShpoppyGlobe.carts
Connect Edit View Collection Help
aprilbackend.sx...
Documents ShpoppyGlobe.c...
My Queries
Databases
ShpoppyGlobe
carts
products
users
admin
classMongodb
config
local
sample_mflix
test
userAPI
ShpoppyGlobe.carts
1 DOCUMENTS
1 INDEXES
Documents Aggregations Schema Indexes Validation
Explain Reset Find Options
ADD DATA
EXPORT DATA
1 – 3 of 3
>_MONGOSH

```
_id: ObjectId('67297621ac4abacf4c382efb')
userId: ObjectId('6728283e6cb0042ccb8bba45')
products: Array (1)
__v: 2

_id: ObjectId('672a0a785030a9c0654541a4')
userId: ObjectId('672a0674a9bb093f15f8725f')
products: Array (1)
__v: 0

_id: ObjectId('672a1db010e4369d8cdb5ca0')
userId: ObjectId('672a18b110e4369d8cdb55bf')
products: Array (1)
```

4. **Cart Collection:**Cart Item of a cart with User ID (`672a4397f81862c515807d6e`) having 4 products in cart with its quantity

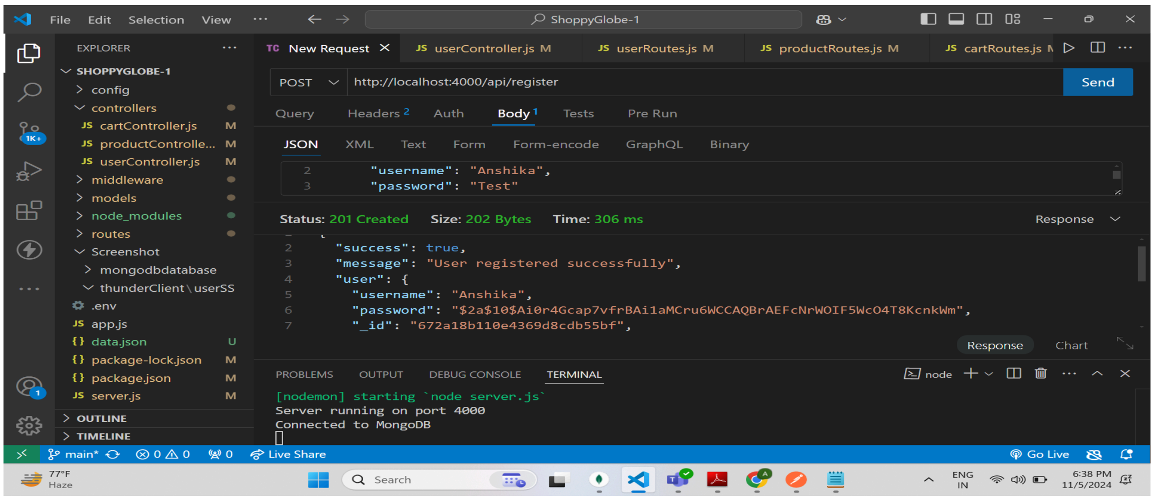

MongoDB Compass - aprilbackend.sxdqr.mongodb.net/ShpoppyGlobe.carts
Connect Edit View Collection Help
aprilbackend.sx...
Documents ShpoppyGlobe.c...
My Queries
Databases
ShpoppyGlobe
carts
products
users
admin
classMongodb
config
local
sample_mflix
test
userAPI
ShpoppyGlobe.carts
1 DOCUMENTS
1 INDEXES
Documents Aggregations Schema Indexes Validation
Explain Reset Find Options
ADD DATA
EXPORT DATA
1 – 4 of 4
>_MONGOSH

```
_id: ObjectId('672a4553f81862c515807fc0')
userId: ObjectId('672a4397f81862c515807d6e')
products: Array (4)
  0: Object
    productId: ObjectId('672206312c1cbf6f55d8fc06')
    quantity: 6
    _id: ObjectId('672a4553f81862c515807fc5')
  1: Object
    productId: ObjectId('672206312c1cbf6f55d8fcbd')
    quantity: 7
    _id: ObjectId('672a456ef81862c515807fce')
  2: Object
    productId: ObjectId('672206312c1cbf6f55d8fc01')
    quantity: 2
    _id: ObjectId('672a457af81862c515807fd8')
  3: Object
```

SCREENSHOT OF THUNDER

1. Users Collections :

a. Register User API



POST http://localhost:4000/api/register

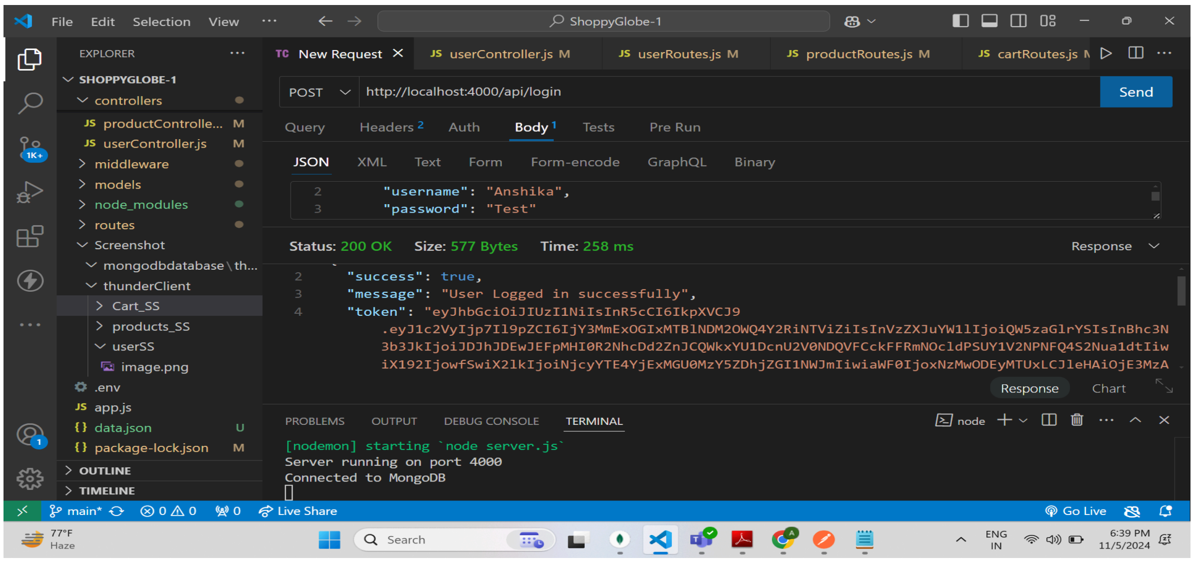
```
"username": "Anshika",
"password": "Test"
```

Status: 201 Created   Size: 202 Bytes   Time: 306 ms

```
  "success": true,
  "message": "User registered successfully",
  "user": {
    "username": "Anshika",
    "password": "$2a$10$Ai0r4Gcap7vfrBAi1aMCru6WCCAQBrAEFcNrWOIF5WcO4T8KcnkWm",
    "_id": "672a18b110e4369d8cdb55bf",
```

```
[nodemon] starting `node server.js`
Server running on port 4000
Connected to MongoDB
```

b. Login User API



POST http://localhost:4000/api/login

```
"username": "Anshika",
"password": "Test"
```

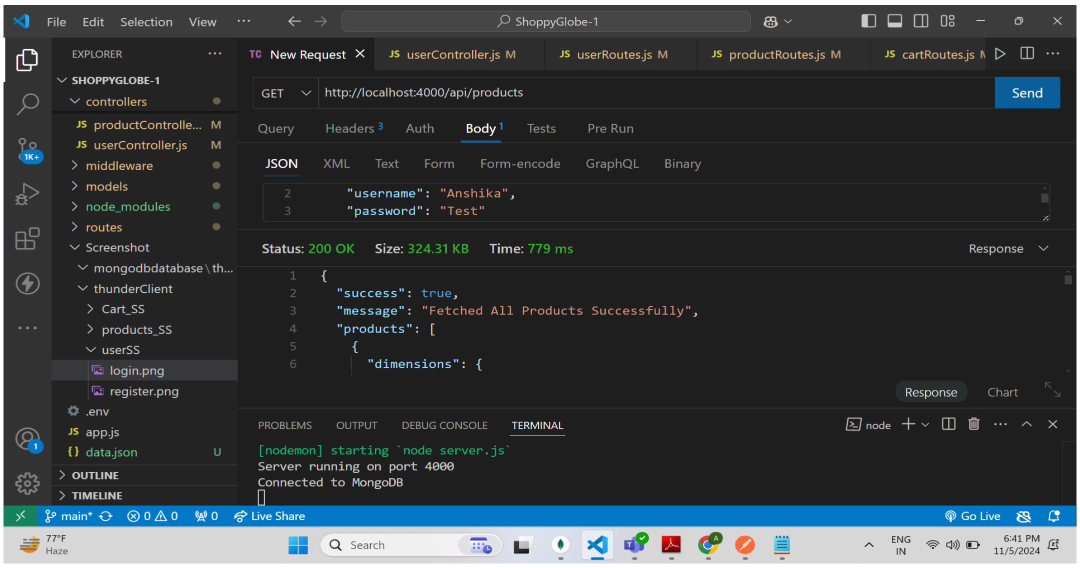
Status: 200 OK   Size: 577 Bytes   Time: 258 ms

```
  "success": true,
  "message": "User Logged in successfully",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
    .eyJ1c2VyIjp7Il9pZCI6IjY3MmExOGIxMTBlNDM2OWQ4Y2RiNTViZiIsInVzZXJuYW1lIjoiQW5zaGlrYSIsInBhc3N
    3b3JkIjoiJDJhJDEwJEFpMHI0R2NhcDd2ZnJCQWkxYU1DcnU2V0NDQVFCckFFRmNOcldPSUY1V2NPNFQ4S2Nua1dtIiw
    iX192IjowfSwiX2lkIjoiNjcyYTE4YjExMGU0MzY5ZDhjZGI1NWJmIiwiaWF0IjoxNzMwODEyMTUxLCJleHAiOjE3MzA
```

```
[nodemon] starting `node server.js`
Server running on port 4000
Connected to MongoDB
```

2. Product Collections:

a. **Get products**



```
GET  http://localhost:4000/api/products
```

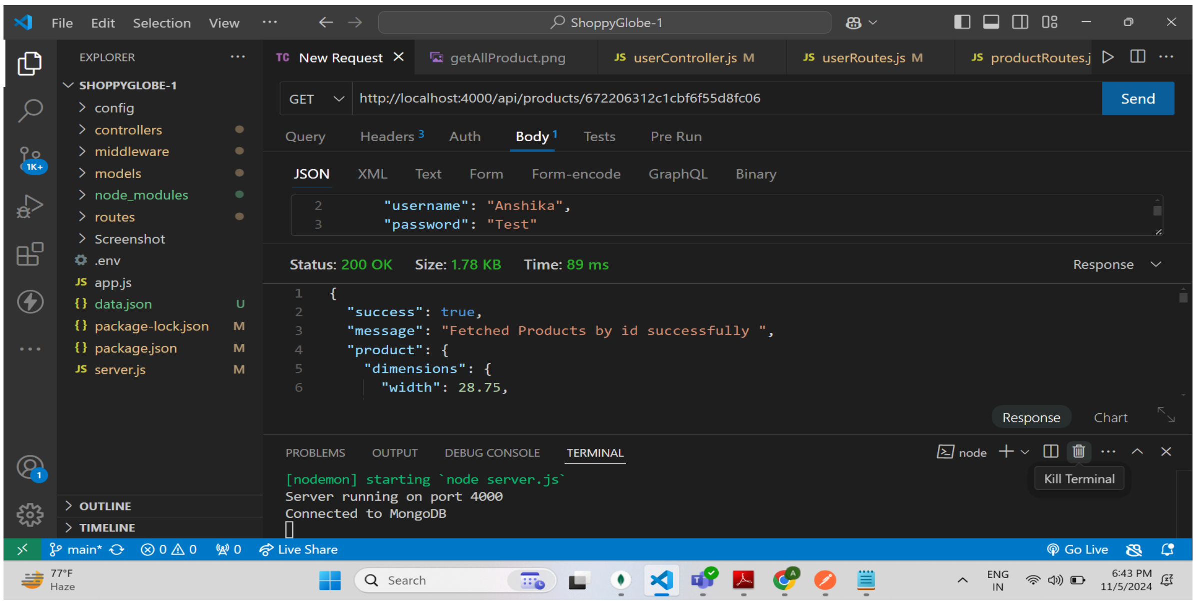
```
    "username": "Anshika",
    "password": "Test"
```

Status: 200 OK   Size: 324.31 KB   Time: 779 ms

```
{
  "success": true,
  "message": "Fetched All Products Successfully",
  "products": [
    {
      "dimensions": {
```

```
[nodemon] starting `node server.js`
Server running on port 4000
Connected to MongoDB
```

b. Get Product by id



```
GET  http://localhost:4000/api/products/672206312c1cbf6f55d8fc06
```

```
    "username": "Anshika",
    "password": "Test"
```

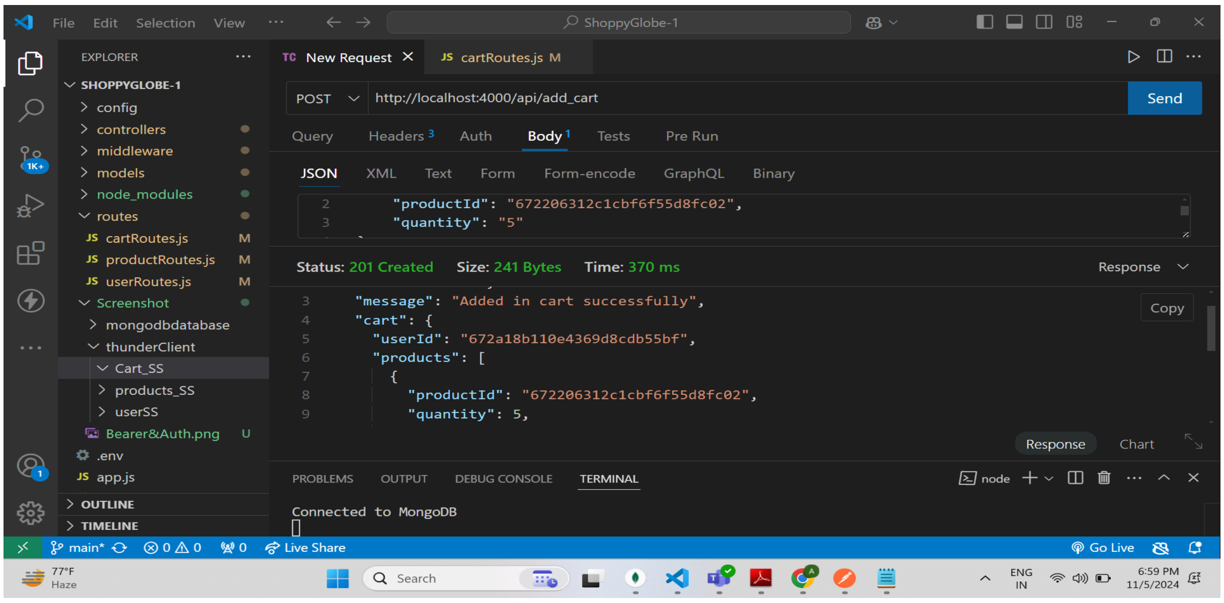
Status: 200 OK   Size: 1.78 KB   Time: 89 ms

```
{
  "success": true,
  "message": "Fetched Products by id successfully ",
  "product": {
    "dimensions": {
      "width": 28.75,
```

```
[nodemon] starting `node server.js`
Server running on port 4000
Connected to MongoDB
```

3. Cart Collection:

a. **POST :- Add item to the cart**



POST http://localhost:4000/api/add_cart

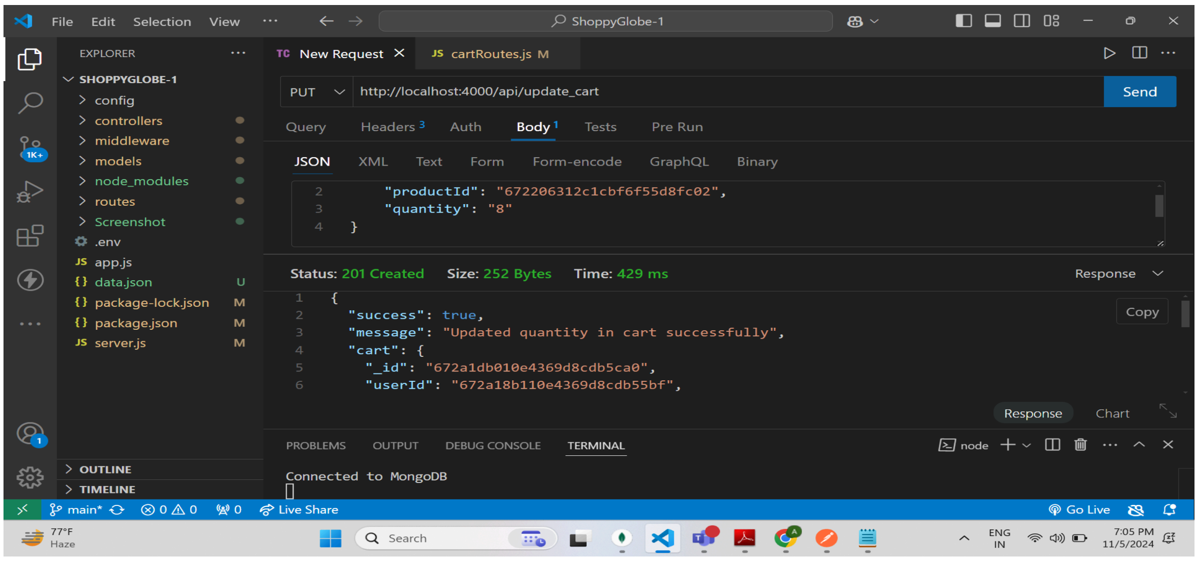
```
    "productId": "672206312c1cbf6f55d8fc02",
    "quantity": "5"
```

Status: 201 Created Size: 241 Bytes Time: 370 ms

```
  "message": "Added in cart successfully",
  "cart": {
    "userId": "672a18b110e4369d8cdb55bf",
    "products": [
      {
        "productId": "672206312c1cbf6f55d8fc02",
        "quantity": 5,
```

```
Connected to MongoDB
```

b. **PUT :-** Update Item quantity of the cart



PUT http://localhost:4000/api/update_cart

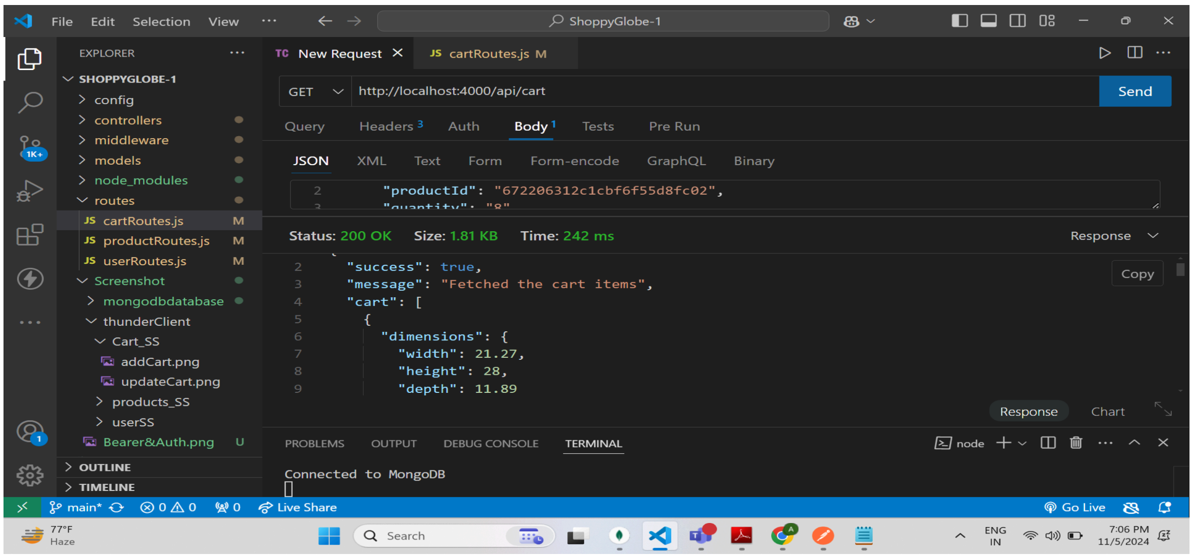
```
    "productId": "672206312c1cbf6f55d8fc02",
    "quantity": "8"
}
```

Status: 201 Created Size: 252 Bytes Time: 429 ms

```
{
  "success": true,
  "message": "Updated quantity in cart successfully",
  "cart": {
    "_id": "672a1db010e4369d8cdb5ca0",
    "userId": "672a18b110e4369d8cdb55bf",
```

```
Connected to MongoDB
```

c. **Get :- fetch the data of the cart of particular user**



```
GET    http://localhost:4000/api/cart
```

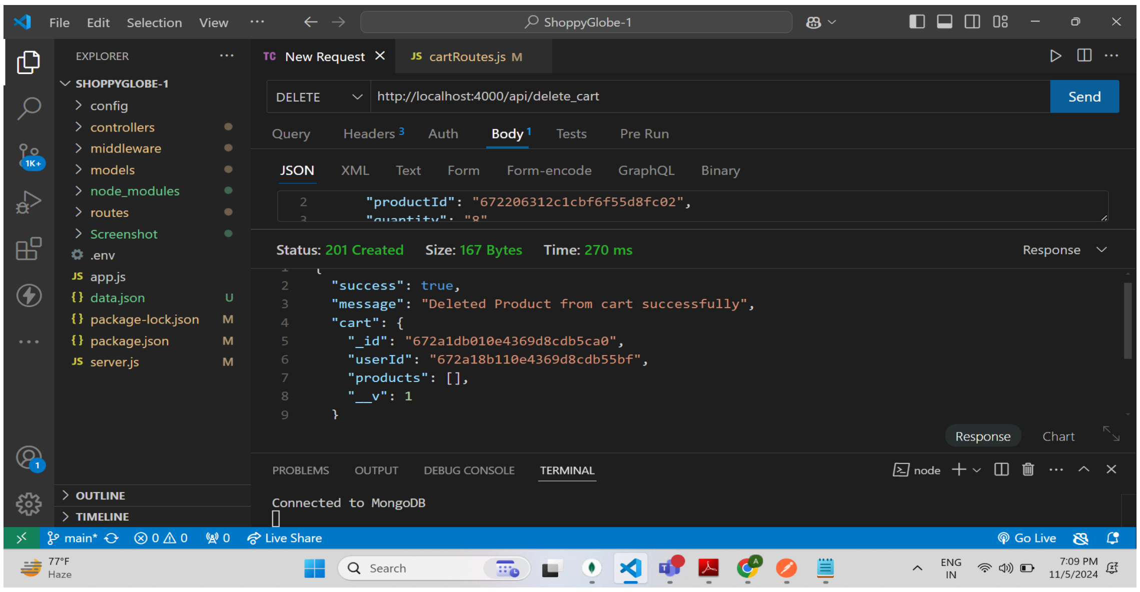
```
    "productId": "672206312c1cbf6f55d8fc02",
    "quantity": "8"
```

Status: 200 OK   Size: 1.81 KB   Time: 242 ms

```
  "success": true,
  "message": "Fetched the cart items",
  "cart": [
    {
      "dimensions": {
        "width": 21.27,
        "height": 28,
        "depth": 11.89
```

```
Connected to MongoDB
```

d. Delete :- Delete the Item from a cart of a user



```
DELETE    http://localhost:4000/api/delete_cart
```

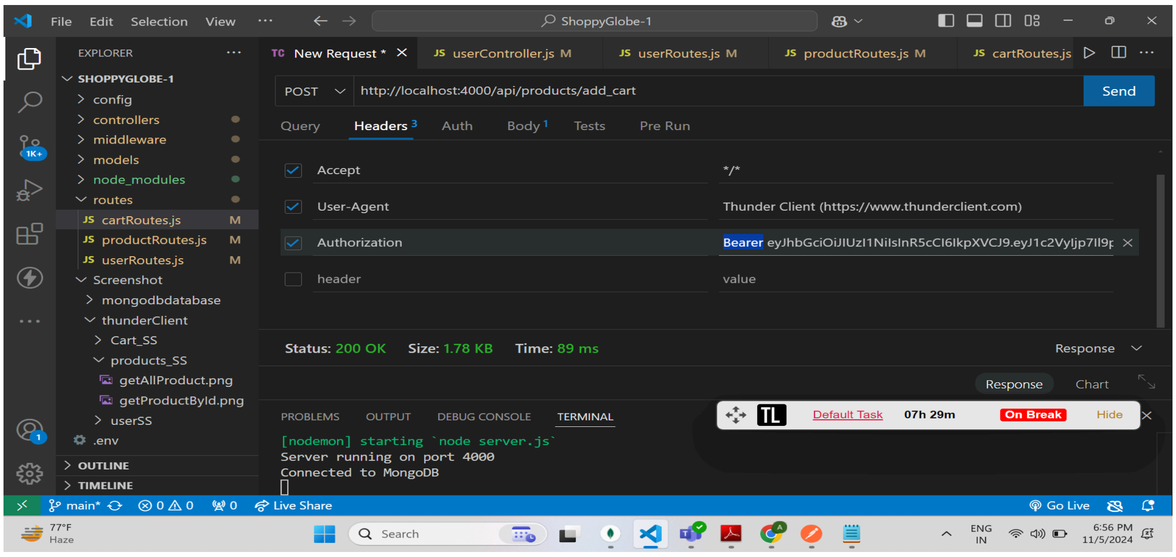
```
    "productId": "672206312c1cbf6f55d8fc02",
    "quantity": "8"
```

Status: 201 Created   Size: 167 Bytes   Time: 270 ms

```
  "success": true,
  "message": "Deleted Product from cart successfully",
  "cart": {
    "_id": "672a1db010e4369d8cdb5ca0",
    "userId": "672a18b110e4369d8cdb55bf",
    "products": [],
    "__v": 1
  }
```

```
Connected to MongoDB
```

4. Authentication Token and Bearer Example



POST http://localhost:4000/api/products/add_cart

Accept	*/*
User-Agent	Thunder Client (https://www.thunderclient.com)
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7Il9p
header	value

Status: 200 OK Size: 1.78 KB Time: 89 ms

```
[nodemon] starting `node server.js`
Server running on port 4000
Connected to MongoDB
```