

# Project Documentation: ShoppyGlobe – Backend API using Node.js & Express

## 1. Project Overview

This project is a complete **E-commerce Backend API** built using **Node.js, Express, and MongoDB Atlas**, with **JWT authentication and Cart functionality**.

**With this API, we can:**

- Register & Login Users securely
- Fetch products from database
- Add products to cart
- Update quantity in cart
- Remove product from cart
- Get user's entire cart

All cart routes are **protected and require JWT authentication**.

Smart features used:

- Password encryption (bcrypt)
- Token-based authentication (JWT)
- Error-handling middleware
- MongoDB Atlas cloud database

## 2. Middleware Overview

### a) Authentication Middleware

This middleware protects cart routes. It verifies the token sent in the request header.

- ✓ If the token is valid, it allows the request to proceed
- ✗ If invalid/missing, it returns **401 Unauthorized**

**Why useful?**

- Ensures only logged-in users can add/view cart items.
- Prevents unauthorized access to private data.

### b) Error Handling Middleware

There are two error handlers:

**notFoundHandler**

Used when a route does not exist.

"Route /unknown not found"

## globalErrorHandler

Catches internal errors and prevents application crash.

## Why is it important?

- Returns clean error messages
- Sends proper status codes like 500, 404, 400

## 3. API Routes Explanation

### User Authentication Routes

#### POST /api/register — Register a New User

- ✓ Validates name, email, password

The screenshot shows a Postman request to `http://localhost:3000/api/register`. The request method is POST. The JSON body contains:

```
{
  "name": "sravan Kumar",
  "email": "sravankumar@gmail.com",
  "password": "1234"
}
```

The response status is 201 Created, with a message: "User registered successfully". The response body is identical to the request body.

- ✓ Validates proper email format

The screenshot shows a Postman request to `http://localhost:3000/api/register`. The request method is POST. The JSON body contains:

```
{
  "name": "sravan Kumar",
  "email": "abcd",
  "password": "1234"
}
```

The response status is 400 Bad Request, with a message: "invalid email address".

- ✓ Hashes password before saving

- ✗ If email exists → 409 Conflict

The screenshot shows the Postman interface with the following details:

- Request:** POST to <http://localhost:3000/api/register>
- Body:** JSON content (copied from the previous screenshot):
 

```
{
        "name": "sravan Kumar",
        "email": "sravan@gmail.com",
        "password": "1234"
      }
```
- Response:** Status: 409 Conflict, Size: 34 Bytes, Time: 39 ms. The message is "Email already in use".
- Terminal:** PS D:\Internshala\Projects\ShoppyglobeBackend> node server.js  
server is running on PORT:3000  
DB Connected

## POST /api/login — Authenticate User

- ✓ Validates email & password
- ✓ Returns a JWT token on success

The screenshot shows the Postman interface with the following details:

- Request:** POST to <http://localhost:3000/api/login>
- Body:** JSON content (copied from the previous screenshot):
 

```
{
        "name": "sravan Kumar",
        "email": "sravan@gmail.com",
        "password": "1234"
      }
```
- Response:** Status: 200 OK, Size: 281 Bytes, Time: 239 ms. The message is "Login successful", the user object contains name and email, and the access token is provided.
- Terminal:** PS D:\Internshala\Projects\ShoppyglobeBackend> node server.js  
server is running on PORT:3000  
DB Connected

✗ If wrong password → 401 Unauthorized

The screenshot shows the Postman interface with the following details:

- Request:** POST to <http://localhost:3000/api/login>
- Body:** JSON content (copied from the previous screenshot):
 

```
{
        "name": "sravan",
        "email": "sravan@gmail.com",
        "password": "123skckc4"
      }
```
- Response:** Status: 401 Unauthorized, Size: 32 Bytes, Time: 236 ms. The message is "Incorrect password".
- Terminal:** PS D:\Internshala\Projects\ShoppyglobeBackend> node server.js  
server is running on PORT:3000  
DB Connected

## Product Routes

### GET /api/products — Fetch All Products

- ✓ Returns list of products stored in MongoDB Atlas
- ✓ Data was inserted directly in Atlas (not through API)

POSTMAN Screenshot showing a successful GET request to `http://localhost:3000/api/products`. The response status is 200 OK, size is 12.57 KB, and time is 37 ms. The response body contains a JSON object with a "products" array containing one item:

```

1  {
2    "products": [
3      {
4        "_id": "69268ea2fd53ed46af5d0da2",
5        "title": "Apple MacBook Air M2 13\""",
6        "description": "Ultralight laptop with Apple M2 chip, 8GB RAM, 256GB SSD and Retina display.",
7        "price": 104900,
8        "stockQuantity": 40,
9        "discountPercentage": 7,
10       "rating": 4.7,
11       "brand": "apple",
12       "category": "laptop",
13       "thumbnail": "https://images.unsplash.com/photo-1517316714731-489689fd1ca8"
14     }
15   ]
16 }
17
18

```

## GET /api/product/:id — Fetch Product by ID

- ✓ Fetches single product details using MongoDB \_id

POSTMAN Screenshot showing a successful GET request to `http://localhost:3000/api/product/69268ea2fd53ed46af5d0da2`. The response status is 200 OK, size is 493 Bytes, and time is 29 ms. The response body contains a JSON object with a "product" array containing one item:

```

1  {
2    "product": [
3      {
4        "_id": "69268ea2fd53ed46af5d0da2",
5        "title": "Apple MacBook Air M2 13\""",
6        "description": "Ultralight laptop with Apple M2 chip, 8GB RAM, 256GB SSD and Retina display.",
7        "price": 104900,
8        "stockQuantity": 40,
9        "discountPercentage": 7,
10       "rating": 4.7,
11       "brand": "apple",
12       "category": "laptop",
13       "thumbnail": "https://images.unsplash.com/photo-1517316714731-489689fd1ca8",
14       "images": [
15         "https://images.unsplash.com/photo-1518770660439-4636190af475",
16         "https://images.unsplash.com/photo-1518770660439-4636190af475"
17       ]
18     }
19   ]
20
21
22

```

✗ If ID not valid → 404 Not Found

POSTMAN Screenshot showing a 400 Bad Request error for an invalid product ID. The response status is 400 Bad Request, size is 31 Bytes, and time is 5 ms. The response body contains a JSON object with a "message" field:

```

1  {
2    "message": "Invalid productId"
3  }

```

## Cart Routes (Protected with Token)

**Note:-** Cart API cannot be tested without Backend Token (Login Needed)

## POST /api/cart — Add Item to Cart

- ✓ Creates new cart if user has no cart
- ✓ Adds product + quantity

```

Status: 200 OK  Size: 249 Bytes  Time: 208 ms
Response Headers Cookies Results Docs
1 {
2   "user": "6926babab56ef525aba9300",
3   "items": [
4     (
5       "product": "69268ea2fd53ed4af5d0d42",
6       "quantity": 1,
7       "id": "6926c0fcab568fs25aba930c"
8     )
9   ],
10  "id": "6926c0fcab568fs25aba930a",
11  "createdAt": "2025-11-26T08:57:32.944Z",
12  "updatedAt": "2025-11-26T08:57:33.041Z",
13  "__v": 1
14 }

```

- ✗ JWT not valid → 403 Forbidden

```

Status: 403 Forbidden  Size: 31 Bytes  Time: 6 ms
Response Headers Cookies Results Docs
1 {
2   "message": "Invalid JWT token"
3 }

```

- ✓ If product exists → updates quantity instead of duplicating

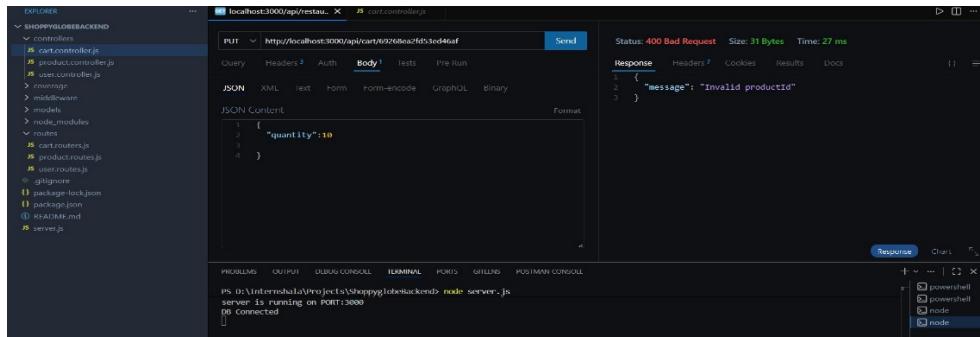
```

Status: 200 OK  Size: 250 Bytes  Time: 113 ms
Response Headers Cookies Results Docs
1 {
2   "id": "6926c0fcab568fs25aba930a",
3   "user": "6926babab56ef525aba9300",
4   "items": [
5     (
6       "product": "69268ea2fd53ed4af5d0d42",
7       "quantity": 10,
8       "id": "6926c0fcab568fs25aba930c"
9     )
10    ],
11    "createdAt": "2025-11-26T08:57:32.944Z",
12    "updatedAt": "2025-11-26T09:10:04.578Z",
13    "__v": 1
14  }

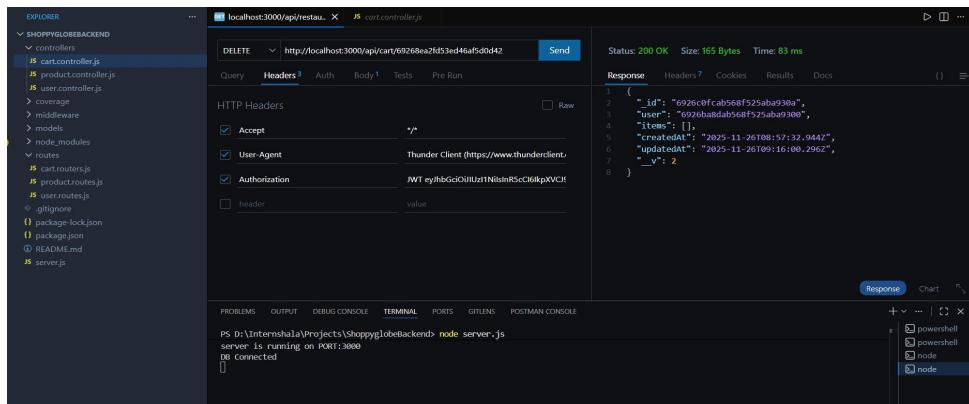
```

## PUT /api/cart — Update Product Quantity

- ✓ Modifies quantity for a specific product
- ✗ If product not found → 404



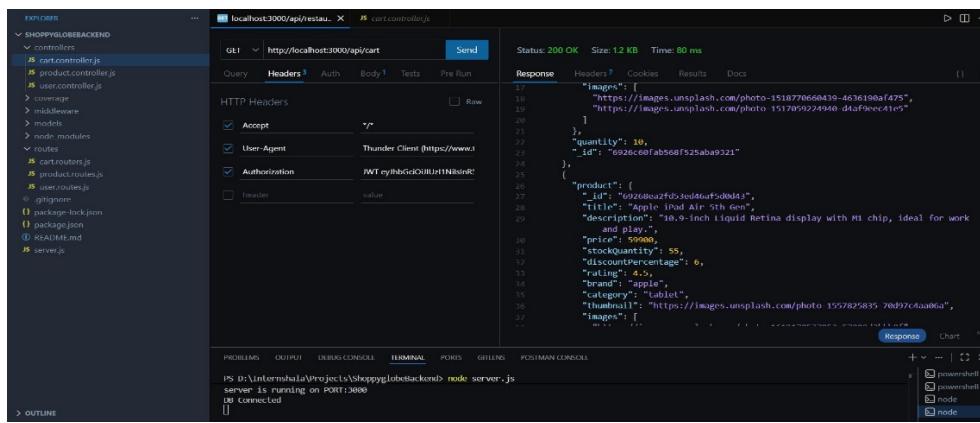
## DELETE /api/cart — Remove Product from Cart



- ✓ Removes product from user's cart

## GET /api/cart — Get User Cart with Product Details

- ✓ Returns products with populated details (name, price, img...)
- ✓ Requires Authorization Token



## 4. MongoDB Atlas Setup Summary

- ✓ Used **cloud database** instead of local MongoDB
- ✓ Allowed public access using IP 0.0.0.0/0

The screenshot shows the MongoDB Atlas interface under the 'Security' section. A modal window titled 'Add entries to your IP Access List' is open. It contains fields for 'IP Address' (0.0.0.0/0) and 'Description' (Sraovn db), along with 'Cancel' and 'Update Entry' buttons. Below this, a table shows the current IP Access List with one entry: 0.0.0.0/0 and Sraovn db. There are 'EDIT' and 'REMOVE' buttons for each row. At the bottom right of the modal is a 'Finish and Close' button.

- ✓ Created non-admin DB user for safety

## 5. Error Handling Summary

- ✗ If path not found → 404

The screenshot shows a Postman collection named 'SHOPPYGLOBEBACKEND'. A specific request is selected for the 'cart.controller.js' endpoint. The 'Send' button is highlighted. The 'Response' tab shows the error details: Status: 404 Not Found, Size: 103 Bytes, Time: 4 ms. The response body is a JSON object with 'success': false, 'error': 'Not Found', and 'message': 'The requested route 'GET /sdjknkdjsn' does not exist.' Below the interface, a terminal window shows the command 'node server.js' being run, indicating the application is running locally.

- No raw system errors are sent to client
- Proper status codes returned:
  - 400 (Bad Request)
  - 401 (Unauthorized)
  - 404 (Not Found)
  - 500 (Internal Server Error)

- ✓ Server never crashes even if an error occurs

## Final Result

successfully developed:

Feature	Status
User Auth + JWT	✓
Password Security (bcrypt)	✓

Feature	Status
Product API + MongoDB	✓
Protected Cart System	✓
Error handlers	✓
Cloud DB + Testing	✓

**ShoppyGlobe Backend is fully secure, scalable, and production-ready!**

Top of Form

Bottom of Form

Top of Form

Bottom of Form