

APIs with Node.js & Express.js for ShoppyGlobe

- **POST /register**
 - Register a new user

The screenshot shows the VS Code interface with a REST client tab open for the endpoint `http://localhost:5000/register`. The request is a POST with a JSON body:

```
1 {
2   "name": "Aslam",
3   "email": "aslambhati@gmail.com",
4   "password": "12345"
5 }
```

The response is a 201 Created status with a JSON body:

```
1 {
2   "_id": "6894d0c793468205584325a4",
3   "name": "Aslam",
4   "email": "aslambhati@gmail.com",
5   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4OTRkMGM3OTM0NjgyMDU1ODQzMjVhNCIsImhhdCI6MTc1NDU4MzIzOSwiZXhwIjozNzU0NTg0MTM5fQ.NKfcbk-FktFaKj45U1pzEqTy38t0cAA1pkDv940frdg"
6 }
```

The terminal at the bottom shows the server running at port 5000 and the database connected successfully.

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure with the `shoppyglobe` database selected, containing collections `carts`, `products`, and `users`. The main panel shows the `users` collection with 2 documents. The first document is displayed:

```
_id: ObjectId('6894d0c793468205584325a4')
name: "Aslam"
email: "aslambhati@gmail.com"
password: "$2b$10$CbsE1aEXARa8B3MV/5KbResKd0LPo3oLWqgtgtVctUIaWiMxldVRjw"
createdAt: 2025-08-07T16:13:59.822+00:00
updatedAt: 2025-08-07T16:13:59.822+00:00
__v: 0
```

➤ POST /login

- Authenticate user and return a JWT token.

The screenshot shows the VS Code interface with a REST client request and response. The request is a POST to `http://localhost:5000/login` with a JSON body containing email and password. The response is a 200 OK status with a JSON body containing user details and a JWT token.

```
POST http://localhost:5000/login
```

Status: 200 OK Size: 262 Bytes Time: 113 ms

Response

```
{
  "_id": "6894d0c793468205584325a4",
  "name": "Aslam",
  "email": "aslamhathi@gmail.com",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4OTRkMGM3OTM0NjgyMDU1ODQzMjVhNCIsIm1hdCI6MTc1NDU4NDE1MywiZXBwIjozNzU0NTg1MDUzfQ.mK50iLMaIdJdm3F26ERaWb7n1t4C38qdOWqmtfGcg"
}
```

Terminal output:

```
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed
```

The screenshot shows the MongoDB Compass interface. A query is executed on the `users` collection, returning a single document. The document contains user details and a JWT token.

localhost:27017 > shoppyglobe > users

Documents 0 Aggregations Schema Indexes 2 Validation

{email: "aslamhathi@gmail.com"}

Generate query Explain Reset Find

ADD DATA EXPORT DATA UPDATE DELETE

25 1-1 of 1

```
{
  "_id": ObjectId('6894d0c793468205584325a4'),
  "name": "Aslam",
  "email": "aslamhathi@gmail.com",
  "password": "$2b$10$CbsE1aEXARa8B3MV/5KbResKd0LPo3oLWqtgtVctUIaWiMxLDVRjW",
  "createdAt": "2025-08-07T16:13:59.822+00:00",
  "updatedAt": "2025-08-07T16:13:59.822+00:00",
  "__v": 0
}
```

- **GET /products**
- Fetch all products list.

VS Code interface showing a REST client request to `GET http://localhost:5000/products`. The response is a JSON array of two product objects. The terminal shows the server is running at port 5000, DB is connected, and seeding is completed.

```
GET http://localhost:5000/products
```

Status: 200 OK Size: 9.04 KB Time: 27 ms

Response

```
{
  "message": "List of all products",
  "products": [
    {
      "_id": "6894d8dd6ed239e5f7f99fec",
      "name": "Essence Mascara Lash Princess",
      "price": 9.99,
      "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.",
      "stock": 99,
      "__v": 0,
      "createdAt": "2025-08-07T16:48:29.655Z",
      "updatedAt": "2025-08-07T16:48:29.655Z"
    },
    {
      "_id": "6894d8dd6ed239e5f7f99fed",
      "name": "Eyeshadow Palette with Mirror",
      "price": 19.99,
      "description": "The Eyeshadow Palette with Mirror offers a versatile range of eyeshadow shades for creating stunning eye looks. With a built-in mirror, it's convenient for on-the-go makeup application.",
      "stock": 34,
      "__v": 0
    }
  ]
}
```

Server running at port:5000
DB connected successfully
Seeding completed

MongoDB Compass interface showing the `products` collection in the `shoppyglobe` database. The collection contains three documents:

```
{
  "_id": ObjectId('6894d899904faff19465f140'),
  "name": "Essence Mascara Lash Princess",
  "price": 9.99,
  "description": "The Essence Mascara Lash Princess is a popular mascara known for its v...",
  "stock": 99,
  "__v": 0,
  "createdAt": 2025-08-07T16:47:21.825+00:00,
  "updatedAt": 2025-08-07T16:47:21.825+00:00
}
```

```
{
  "_id": ObjectId('6894d899904faff19465f141'),
  "name": "Eyeshadow Palette with Mirror",
  "price": 19.99,
  "description": "The Eyeshadow Palette with Mirror offers a versatile range of eyeshado...",
  "stock": 34,
  "__v": 0,
  "createdAt": 2025-08-07T16:47:21.826+00:00,
  "updatedAt": 2025-08-07T16:47:21.826+00:00
}
```

```
{
  "_id": ObjectId('6894d899904faff19465f142'),
  "name": "Powder Canister",
  "price": 14.99,
  "description": "The Powder Canister is a finely milled setting powder designed to set ..."
}
```

➤ GET /products/:id

- Fetch details of a single product by MongoDB ObjectId

The screenshot shows the VS Code interface with a REST client tab open. The request is a GET call to `http://localhost:5000/products/6894d8dd6ed239e5f7f99fed`. The response is a JSON object with the following details:

```
{
  "_id": "6894d8dd6ed239e5f7f99fed",
  "name": "Eyeshadow Palette with Mirror",
  "price": 19.99,
  "description": "The Eyeshadow Palette with Mirror offers a versatile range of eyeshadow shades for creating stunning eye looks. With a built-in mirror, it's convenient for on-the-go makeup application.",
  "stock": 34,
  "__v": 0,
  "createdAt": "2025-08-07T16:48:29.657Z",
  "updatedAt": "2025-08-07T16:48:29.657Z"
}
```

The status is 200 OK, size is 386 Bytes, and time is 11 ms. The bottom status bar shows "Server running at port:5000", "DB connected successfully", and "Seeding completed".

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure with `shoppyglobe` selected, containing `products` and `users` collections. The main panel shows the `products` collection with 30 documents. A query filter `{name: 'Eyeshadow Palette with Mirror'}` is applied. The document details are as follows:

```
{
  "_id": ObjectId('6894d8dd6ed239e5f7f99fed'),
  "name": "Eyeshadow Palette with Mirror",
  "price": 19.99,
  "description": "The Eyeshadow Palette with Mirror offers a versatile range of eyeshado...",
  "stock": 34,
  "__v": 0,
  "createdAt": "2025-08-07T16:48:29.657+00:00",
  "updatedAt": "2025-08-07T16:48:29.657+00:00"
}
```

- **POST /products/**
- Add a new product in the list.

The screenshot shows a VS Code editor with a REST client interface. The request is a POST to `http://localhost:5000/products/` with a JSON body. The response is a 201 Created status with a JSON body containing product details.

Request:

```
POST http://localhost:5000/products/
{
  "name": "Lenovo V15 Laptop",
  "price": 99.99,
  "description": "Lenovo V series laptop with 500 GB SSD, 16 GB RAM, Integrated Graphics with Anti-Glare screen.",
  "stock": 32
}
```

Response:

```
{
  "message": "Product added successfully",
  "product": {
    "name": "Lenovo V15 Laptop",
    "price": 99.99,
    "description": "Lenovo V series laptop with 500 GB SSD, 16 GB RAM, Integrated Graphics with Anti-Glare screen.",
    "stock": 32,
    "_id": "6894dfbff4cc0ea5aa18b21d",
    "createdAt": "2025-08-07T17:17:51.389Z",
    "updatedAt": "2025-08-07T17:17:51.389Z",
    "__v": 0
  }
}
```

The terminal at the bottom shows the server running on port 5000 and the database connected successfully.

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure, with the `products` collection selected. The main area displays a list of products with their details.

Products List:

Product Name	Price	Stock	Created At	Updated At
Kiwi	2.49	99	2025-08-07T17:08:57.865+00:00	2025-08-07T17:08:57.865+00:00
Lenovo V15 Laptop	99.99	32	2025-08-07T17:17:51.389+00:00	2025-08-07T17:17:51.389+00:00

➤ PUT /products/:id

- Update a product by MongoDB ObjectId

The screenshot shows the VS Code interface with a REST client tab open. The request is a PUT to `http://localhost:5000/products/6894dfbff4cc0ea5aa18b21d`. The request body is a JSON object representing a product update. The response is a 200 OK status with a JSON body indicating the product was updated successfully.

Request:

```
PUT http://localhost:5000/products/6894dfbff4cc0ea5aa18b21d
```

Request Body (JSON):

```
{
  "name": "Lenovo V15 Laptop",
  "price": 109.99,
  "description": "Lenovo V series laptop with 500 GB SSD, 16 GB RAM, Integrated Graphics with Anti-Glare screen & Ryzen 7 processor.",
  "stock": 39
}
```

Response:

```
{
  "message": "Product updated successfully",
  "updatedProduct": {
    "_id": "6894dfbff4cc0ea5aa18b21d",
    "name": "Lenovo V15 Laptop",
    "price": 109.99,
    "description": "Lenovo V series laptop with 500 GB SSD, 16 GB RAM, Integrated Graphics with Anti-Glare screen & Ryzen 7 processor.",
    "stock": 39,
    "createdAt": "2025-08-07T17:17:51.389Z",
    "updatedAt": "2025-08-07T17:32:56.884Z",
    "__v": 0
  }
}
```

Terminal Output:

```
[nodemon] starting `node server.js`
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed
```

```
_id: ObjectId('6894dfbff4cc0ea5aa18b21d')
name : "Lenovo V15 Laptop"
price : 109.99
description : "Lenovo V series laptop with 500 GB SSD, 16 GB RAM, Integrated Graphics..."
stock : 39
createdAt : 2025-08-07T17:17:51.389+00:00
updatedAt : 2025-08-07T17:32:56.884+00:00
__v : 0
```

➤ DELETE /products/:id

- Delete an individual product by its objectId.

The screenshot shows a VS Code editor with a REST client tab open. The request is a DELETE method to the URL `http://localhost:5000/products/6894dda9f4cc0ea5aa18b1ff`. The response is a 200 OK status with a body containing `{ "message": "Product deleted successfully" }`. The terminal at the bottom shows the server running on port 5000 and the database seeded successfully.

```
DELETE http://localhost:5000/products/6894dda9f4cc0ea5aa18b1ff
```

Status: 200 OK Size: 42 Bytes Time: 5 ms

```
{
  "message": "Product deleted successfully"
}
```

```
[nodemon] starting `node server.js`
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed
```

The screenshot shows the MongoDB Compass interface. The query is `{ name: "Eyeshadow Palette with Mirror" }`. The results section shows "No results" with the message "Try modifying your query to get results." The left sidebar shows the database structure with collections like admin, config, local, shoppyglobe, carts, products, and users.

```
{ name: "Eyeshadow Palette with Mirror" }
```

No results

Try modifying your query to get results.

➤ POST /cart

- Add a new product to the shopping cart.

The screenshot shows the VS Code interface with a REST client request and response. The request is a POST to `http://localhost:5000/cart` with a JSON body:

```
{
  "productId": "6894dda9f4cc0ea5aa18b200",
  "quantity": 2
}
```

The response is a 201 Created status with a 264 Byte size and 10 ms time. The response body is:

```
{
  "message": "Item successfully added to the Cart",
  "newItem": {
    "user": "6894d0c793468205584325a4",
    "product": "6894dda9f4cc0ea5aa18b200",
    "quantity": 2,
    "_id": "6894e89cf4cc0ea5aa18b22a",
    "createdAt": "2025-08-07T17:55:40.835Z",
    "updatedAt": "2025-08-07T17:55:40.835Z",
    "__v": 0
  }
}
```

The terminal at the bottom shows the following output:

```
[nodemon] starting `node server.js`
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed
[]
```

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure with the following collections:

- cluster0.43xbxf.mongodb.net
- localhost:27017
 - admin
 - config
 - local
 - shoppyglobe
 - carts
 - products
 - users

The main panel shows the 'carts' collection in the 'shoppyglobe' database. The 'Documents' tab is selected, showing a single document:

```
{
  "_id": ObjectId('6894e89cf4cc0ea5aa18b22a'),
  "user": ObjectId('6894d0c793468205584325a4'),
  "product": ObjectId('6894dda9f4cc0ea5aa18b200'),
  "quantity": 2,
  "createdAt": 2025-08-07T17:55:40.835+00:00,
  "updatedAt": 2025-08-07T17:55:40.835+00:00,
  "__v": 0
}
```


➤ GET /cart

- Fetch the list of all cart items.

The screenshot shows the Thunder Client interface. The top bar indicates the request method is GET and the URL is http://localhost:5000/cart. The status is 200 OK, size is 1.05 KB, and time is 21 ms. The Headers tab is selected, showing Accept: */*, User-Agent: Thunder Client (https://www.thunderclient.com), and Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

The Response tab shows the following JSON data:

```
1 {
2   "message": "List of all cart items",
3   "cartItems": [
4     {
5       "_id": "6894e89cf4cc0ea5aa18b22a",
6       "user": "6894d0c793468205584325a4",
7       "product": {
8         "_id": "6894dda9f4cc0ea5aa18b200",
9         "name": "Powder Canister",
10        "price": 14.99,
11        "description": "The Powder Canister is a finely milled setting powder
12        designed to set makeup and control shine. With a lightweight and
13        translucent formula, it provides a smooth and matte finish.",
14        "stock": 89,
15        "__v": 0,
16        "createdAt": "2025-08-07T17:08:57.865Z",
17        "updatedAt": "2025-08-07T17:08:57.865Z"
18      },
19      "quantity": 2,
20      "createdAt": "2025-08-07T17:55:40.835Z",
21      "updatedAt": "2025-08-07T17:55:40.835Z",
22      "__v": 0
23    }
24  ]
25 }
```

The bottom terminal shows the following output:

```
[nodemon] starting `node server.js`
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed
```

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure with the following collections:

- cluster0.43xbxfl.mongodb.net
- localhost:27017
 - admin
 - config
 - local
 - shoppyglobe
 - carts**
 - products
 - users

The main panel shows the carts collection in the shoppyglobe database. The Documents tab is selected, showing 2 documents. The query bar contains the text: Type a query: { field: 'value' } or Generate query. The first document is:

```
{
  "_id": ObjectId('6894e89cf4cc0ea5aa18b22a'),
  "user": ObjectId('6894d0c793468205584325a4'),
  "product": ObjectId('6894dda9f4cc0ea5aa18b200'),
  "quantity": 2,
  "createdAt": 2025-08-07T17:55:40.835+00:00,
  "updatedAt": 2025-08-07T17:55:40.835+00:00,
  "__v": 0
}
```

The second document is:

```
{
  "_id": ObjectId('6894e9a6f4cc0ea5aa18b22f'),
  "user": ObjectId('6894d0c793468205584325a4'),
  "product": ObjectId('6894dfbff4cc0ea5aa18b21d'),
  "quantity": 1,
  "createdAt": 2025-08-07T18:00:06.396+00:00,
  "updatedAt": 2025-08-07T18:00:06.396+00:00,
  "__v": 0
}
```

➤ PUT /cart/:id

- Update cart item by ObjectId.

The screenshot shows the VS Code interface with a REST client request and response. The request is a PUT to `http://localhost:5000/cart/6894dda9f4cc0ea5aa18b200`. The response is a 200 OK status with a JSON body: `{ "message": "Item successfully updated", "quantity": 4 }`. The terminal at the bottom shows the server running on port 5000 and connected to a database.

```
PUT http://localhost:5000/cart/6894dda9f4cc0ea5aa18b200
Send

Status: 200 OK Size: 52 Bytes Time: 25 ms

Response
1 {
2   "message": "Item successfully updated",
3   "quantity": 4
4 }
```

```
[nodemon] starting `node server.js`
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed
```

The screenshot shows the MongoDB Atlas interface. The left sidebar displays the database structure, including the `carts` collection. The main area shows the details of the `carts` collection, including the `_id`, `user`, `product`, `quantity`, `createdAt`, `updatedAt`, and `__v` fields.

```
cluster0.43xbxfl.mongodb.net
localhost:27017
  admin
  config
  local
  shoppyglobe
  carts
  products
  users
```

The screenshot shows the MongoDB Atlas interface with the `carts` collection. The top bar includes buttons for `ADD DATA`, `EXPORT DATA`, `UPDATE`, and `DELETE`. The data table shows two documents. The first document has a quantity of 4, and the second document has a quantity of 1. The `createdAt` and `updatedAt` fields are shown in ISO 8601 format.

```
ADD DATA EXPORT DATA UPDATE DELETE 25 1 - 2 of 2
```

```
_id: ObjectId('6894e89cf4cc0ea5aa18b22a')
user: ObjectId('6894d0c793468205584325a4')
product: ObjectId('6894dda9f4cc0ea5aa18b200')
quantity: 4
createdAt: 2025-08-07T17:55:40.835+00:00
updatedAt: 2025-08-07T18:14:42.609+00:00
__v: 0
```

```
_id: ObjectId('6894e9a6f4cc0ea5aa18b22f')
user: ObjectId('6894d0c793468205584325a4')
product: ObjectId('6894dfbff4cc0ea5aa18b21d')
quantity: 1
createdAt: 2025-08-07T18:00:06.396+00:00
updatedAt: 2025-08-07T18:00:06.396+00:00
__v: 0
```

➤ DELETE /cart/:id

- Remove cart item by ObjectId.

The screenshot shows the VS Code interface with a REST client request and response. The request is a DELETE method to the endpoint `http://localhost:5000/cart/6894dda9f4cc0ea5aa18b200`. The response is a 200 OK status with a 39-byte body and a 7ms time. The response body is a JSON object: `{ "message": "Item successfully removed" }`. The terminal shows the server running at port 5000 and the database connected successfully.

```
DELETE http://localhost:5000/cart/6894dda9f4cc0ea5aa18b200
```

Status: 200 OK Size: 39 Bytes Time: 7 ms

```
{
  "message": "Item successfully removed"
}
```

nodemon starting `node server.js`
[dotenv@17.0.1] injecting env (4) from .env - [tip] encrypt with dotenvx: https://dotenvx.com
Server running at port:5000
DB connected successfully
Seeding completed

The screenshot shows the MongoDB Compass interface. The left sidebar shows the database structure with the `carts` collection selected. The main panel shows a single document in the `carts` collection. The document contains the following fields:

- `_id`: ObjectId('6894e9a6f4cc0ea5aa18b22f')
- `user`: ObjectId('6894d0c793468205584325a4')
- `product`: ObjectId('6894dfbff4cc0ea5aa18b21d')
- `quantity`: 1
- `createdAt`: 2025-08-07T18:00:06.396+00:00
- `updatedAt`: 2025-08-07T18:00:06.396+00:00
- `__v`: 0

Error Handling

➤ POST /login with invalid credentials

- 400 Bad Request & Response with “Invalid credentials”

The screenshot shows the Thunder Client interface. The request is a POST to `http://localhost:5000/login`. The body is a JSON object: `{ "email": "anonymous@gmail.com", "password": "anonymous" }`. The response status is **400 Bad Request**, with a size of 33 Bytes and a time of 11 ms. The response body is: `{ "message": "Invalid credentials" }`.

➤ GET, POST, PUT, DELETE /cart with invalid id

- 401 Unauthorized & Response with “jwt expired”.

The screenshot shows the Thunder Client interface. The request is a POST to `http://localhost:5000/cart`. The headers are: `Accept: */*`, `User-Agent: Thunder Client (https://www.thunderclient.com)`, and `Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...`. The response status is **401 Unauthorized**, with a size of 25 Bytes and a time of 29 ms. The response body is: `{ "message": "jwt expired" }`.

➤ POST /cart without required fields

- Validate inputs
- 400 Bad Request with error message

The screenshot shows a REST client interface with the following details:

- Request:** POST to `http://localhost:5000/cart`. The body is JSON: `{ "quantity": 1 }`.
- Status:** 400 Bad Request. Size: 36 Bytes. Time: 31 ms.
- Response:**

```
1 {
2   "message": "ProductId not provided"
3 }
```

➤ GET, PUT, DELETE /invalid_routes

- 404 Not Found with error message

The screenshot shows a REST client interface with the following details:

- Request:** GET to `http://localhost:5000/anythingggg`.
- Status:** 404 Not Found. Size: 1.03 KB. Time: 4 ms.
- Response:**

```
1 {
2   "message": "Not Found - /anythingggg",
3   "stack": "Error: Not Found - /anythingggg\n    at notFound (file:///E:/htdocs\n/NodeJS/Shoppyglobe_backend/middlewares/errorMiddleware.js:4:17)\n    at\nLayer.handleRequest (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\lib\\layer\n.js:152:17)\n    at trimPrefix (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:342:13)\n    at E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:297:9\n    at processParams (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:582:12)\n    at next (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:291:5)\n    at E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:688:15\n    at next (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:276:14)\n    at Function.handle (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:186:3)\n    at router (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:186:3)\n    at router (E\n:\\htdocs\\NodeJS\\Shoppyglobe_backend\\node_modules\\router\\index.js\n:186:3)"
```