



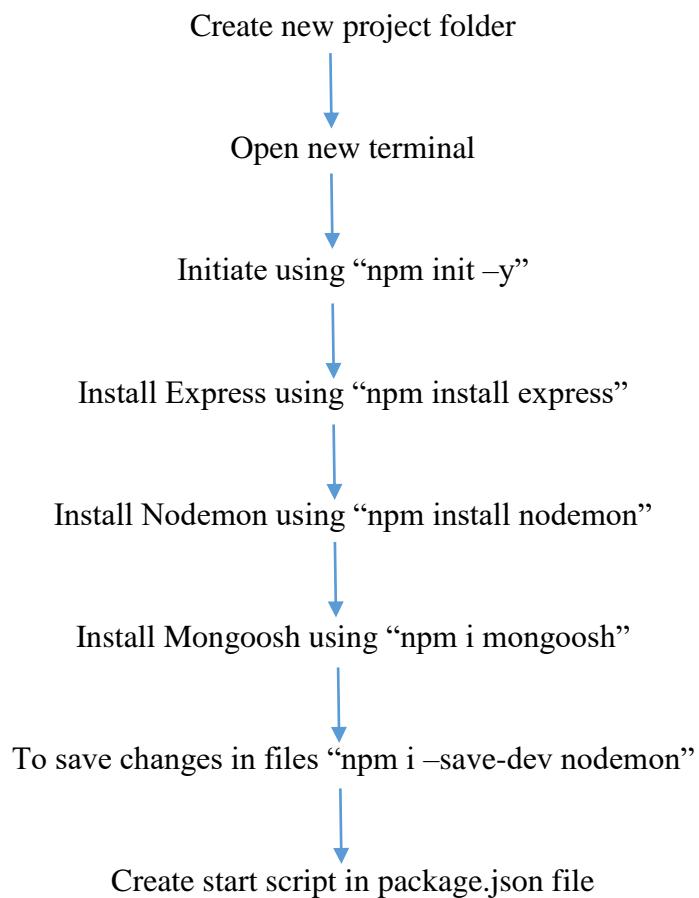
Project

Build APIs with Node.js and Express.js for Shoppyglobe E-commerce

Objective:

Create the backend for the ShoppyGlobe application using Node.js, Express, and MongoDB.

Project setup:



Create a server:

The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure under "SHOPPY-APP". The "Backend" folder contains "node_modules", "package-lock.json", "package.json", and "JS server.js".
- CODE EDITOR**: The "JS server.js" file is open, showing the following code:

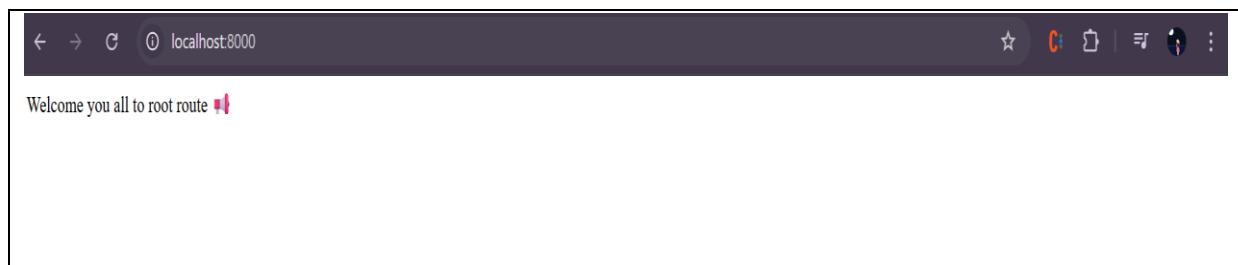
```
1 // importing dependencies needed files
2
3 import express from "express"
4
5 const app=express();
6
7 // create a server
8 const PORT = 5050;
9 app.listen(PORT,()=>{
10   console.log(` Server is running on ${PORT}`)
11 })
12 }
```
- TERMINAL**: Shows the output of the nodemon command, indicating the server is running on port 5050 and restarting due to changes.

To connect middleware and initial route:

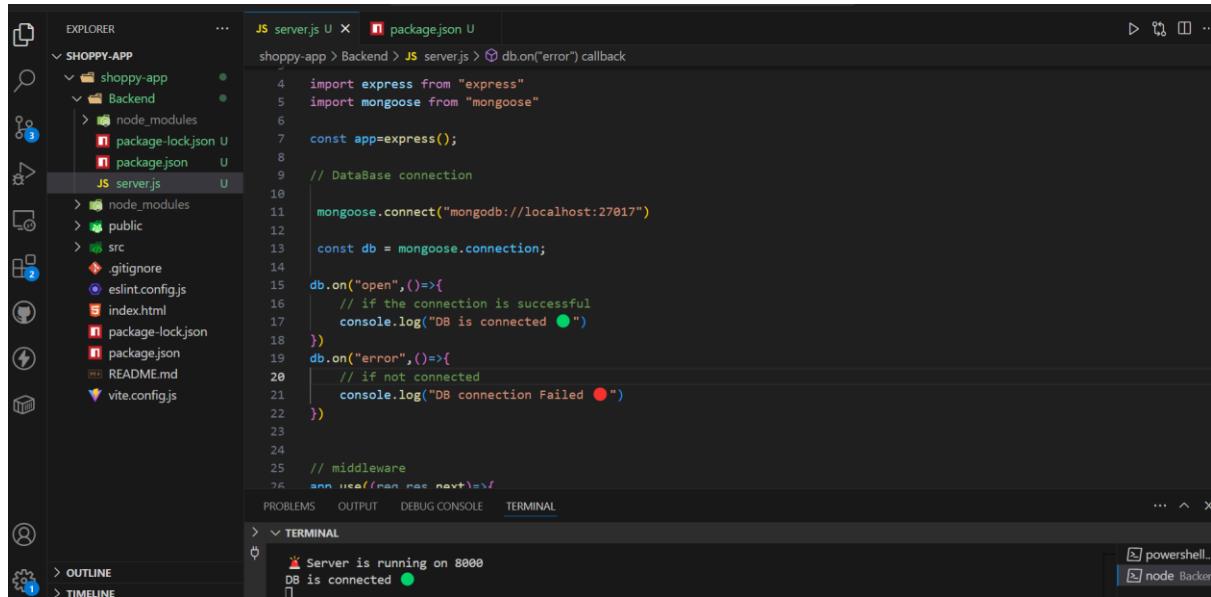
The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure under "SHOPPY-APP". The "Backend" folder contains "node_modules", "package-lock.json", "package.json", and "JS server.js".
- CODE EDITOR**: The "JS server.js" file is open, showing the following code:

```
3
4 import express from "express"
5
6 const app=express();
7
8 // middleware
9 app.use((req,res,next)=>{
10   console.log("Middleware is connected");
11   next();
12 });
13
14 // basic route defining
15
16 app.get('/',(req,res)=>{
17   res.send("Welcome you all to root route 🎉");
18 })
19
20 // create a server
21 const PORT = 8000;
22 app.listen(PORT,()=>{
23   console.log(` Server is running on ${PORT}`)
24 })
25 }
```
- TERMINAL**: Shows the output of the nodemon command, indicating the middleware is connected.



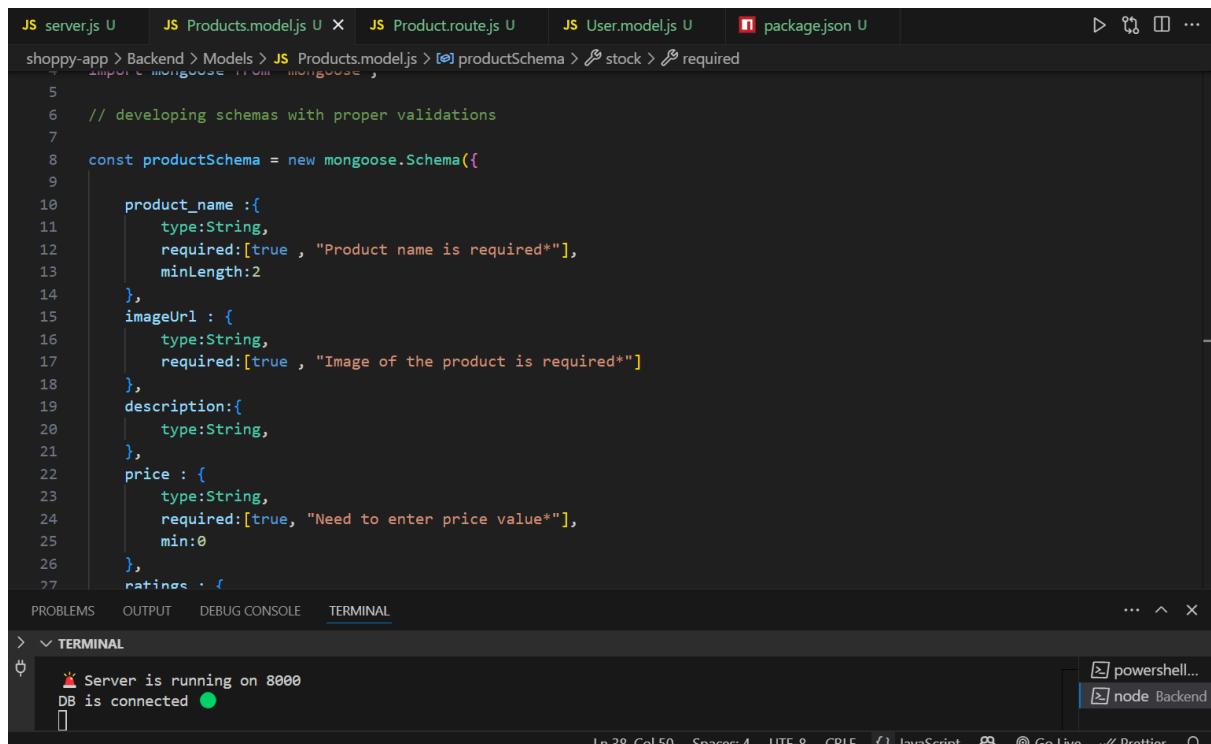
Connect with MongoDB:



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "SHOPPY-APP".
- Code Editor:** The file "server.js" is open, displaying code for connecting to MongoDB. It includes a "DataBase connection" section where the database URL "mongodb://localhost:27017" is specified. It also includes event listeners for "open" and "error" events.
- Terminal:** Shows the output of running the application. It displays the message "Server is running on 8000" and "DB is connected".

Created a model for products and defining schemas with proper validations



The screenshot shows the VS Code interface with the following details:

- Code Editor:** The file "Products.model.js" is open, showing the definition of a "productSchema" using the Mongoose Schema API. The schema includes fields for "product_name", "imageUrl", "description", "price", and "ratings". Each field has its type, validation rules, and a required flag.
- Terminal:** Shows the output of running the application. It displays the message "Server is running on 8000" and "DB is connected".

Exported products model

```
JS server.js U JS Products.model.js U X JS Product.route.js U JS User.model.js U package.json U
shoppy-app > Backend > Models > JS Products.model.js > [!] productSchema > ⚠ stock > ⚠ required
  8   const productSchema = new mongoose.Schema({
  27     ratings : {
  31       default:0
  32     },
  33     category :{
  34       type:String
  35     },
  36     stock : {
  37       type:Number,
  38       required:[true, "Stock value is required!"]
  39     }
  40   })
  41
  42 })
  43
  44 // creating a model
  45
  46 const ProductsModel= mongoose.model('Products', productSchema);
  47
  48 // exporting
  49 export default ProductsModel;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ... ^ X

> V TERMINAL

⚠ Server is running on 8000 DB is connected

powershell... node Backend

CRUD operations for products

Create new product

- 1) I have defined the route to create new product – app.post("/api/product",createProductData)
- 2) I have written logic in controller part
- 3) Tested with Thunder-Client extension

```
JS server.js U JS Products.model.js U JS Product.route.js U JS product.controller.js U X New Request
shoppy-app > Backend > Controller > JS product.controller.js > ⚡ createProductData
  1   import ProductsModel from "../Models/Products.model.js";
  2
  3
  4
  5   export async function createProductData(req,res){
  6
  7     try{
  8       // destructure process
  9       let {product_name,imageUrl,ratings,price,category,description,stock} = req.body;
 10       let newProduct = await ProductsModel.create({product_name,imageUrl,ratings,price,category,description,stock});
 11     // if successfully created a product
 12     return res.status(201).json({"New product is created":newProduct});
 13   }
 14   // facing any error means
 15   catch(err){
 16     return res.status(404).json({err:"Something went wrong while try to create new product ▲"})
 17   }
 18 }
 19 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

THUNDER CLIENT

New Request | ...

Activity Collections Env

filter activity

POST localhost:8000/api/pr... just now

POST localhost:5050/api/res... 9 hours ago

POST localhost:5050/api/lo... 10 hours ago

GET localhost:5050/api/rest... 11 hours ago

GET localhost:5050/api/rest... 11 hours ago

POST localhost:5050/api/res... 14 hours ago

GET localhost:3030/api/book 4 days ago

DEL localhost:3030/api/users 6 days ago

JS server.js U JS Products.model.js U JS Product.route.js U JS product.controller.js U TO New Request X

POST http://localhost:8000/api/product Send

Status: 201 Created Size: 407 Bytes Time: 69 ms

Response Headers Cookies Results Docs ()

```

1  {
2    "New product is created": {
3      "product_name": "Mamaearth Daily Glow Face Cream",
4      "imageUrl": "https://rukminim3.flixcart.com/image/1042
/971/xif0q/moisturizer-cream/c/m/h/-original
-imagzzm5ds6umfbfd.jpeg?q=60&crop=false",
5      "description": "Daily Glow Face Cream for Bright Skin
With Vitamin C & Turmeric",
6      "price": "300 INR",
7      "ratings": 4.2,
8      "category": "Beauty_products",
9      "stock": 50,
10     "_id": "68597f16d98750997ca46448",
11     "__v": 0
12   }
13 }
```

Response Chart

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Middleware is connected

powershell...

node-Postman...

Response in MongoDB compass

MongoDB Compass - localhost:27017/test.products

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (6)

localhost:27017

- admin
- config
- local
 - startup.log
- products
- movies
- test
 - books
 - products
 - restaurants
 - users

localhost:27017

localhost:27017 > test > products

Documents 1 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or Generate query

Find Options

ADD DATA EXPORT DATA UPDATE DELETE

```

_id: ObjectId('68597f16d98750997ca46448')
product_name : "Mamaearth Daily Glow Face Cream"
imageUrl : "https://rukminim3.flixcart.com/image/1042/971/xif0q/moisturizer-cream/c/m/h/-original
-imagzzm5ds6umfbfd.jpeg?q=60&crop=false"
description : "Daily Glow Face Cream for Bright Skin With Vitamin C & Turmeric"
price : "300 INR"
ratings : 4.2
category : "Beauty_products"
stock : 50
__v : 0

```

Fetch the list of products

- Created route for fetching data – “app.get(“/api/products”,fetchProductList)
- Wrote logic in controller part
- Check with Thunder-client extension

A screenshot of the Visual Studio Code interface. The top navigation bar shows 'shoppy-app > Backend > Controller > JS product.controller.js > fetchProductList > [data]'. The main editor area contains the following code:

```
21
22 // 2) to fetch products data from DB
23
24 export async function fetchProductList(req,res){
25   try{
26     let data=await ProductsModel.find();
27     // if there is no product details in DB
28     if(!data){
29       return res.status(500).json({message:"No product's data is found in DataBase"})
30     }
31
32     return res.status(201).json(data);
33
34
35   }
36   // common error handling
37   catch(err){
38     return res.status(500).json({err:"Something went wrong while try to fetch products list from DB ▲"})
39   }
40 }
```

The bottom status bar indicates 'In 26. Col 24. Spaces: 4. UTE-8. CRLF. (JavaScript)'. The terminal below shows the following output:

```
default in Mongoose 7. Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server is running on 8000
DB is connected
Middleware is connected
```

Thunder client response

A screenshot of the Thunder client interface. The top navigation bar shows 'GET http://localhost:8000/api/products Send'. The main interface has tabs for 'Query', 'Headers 2', 'Auth', 'Body 1', 'Tests', and 'Pre Run'. The 'Response' tab is selected, showing the following JSON data:

```
1 [
2   {
3     "_id": "68597f16d98750997ca46448",
4     "product_name": "Mamaearth Daily Glow Face Cream",
5     "imageUrl": "https://rukminim3.flixcart.com/image/1042/971/xif0q
      /moisturizer-cream/c/m/h/-original-imagzzm5ds6umfbdb.jpeg?q
      =60&crop=false",
6     "description": "Daily Glow Face Cream for Bright Skin With
      Vitamin C & Turmeric",
7     "price": "300 INR",
8     "ratings": 4.2,
9     "category": "Beauty_products",
10    "stock": 50,
11    "__v": 0

```

The bottom status bar indicates 'Status: 201 Created Size: 821 Bytes Time: 62 ms'. The terminal below shows the same connection and server status as the VS Code screenshot.

Update Data

- Defining route to update from registered data – “/api/product/:id”,updateProductData
 - I have written a logic in controller part
 - Check using thunder-client extension

shoppo-app > Backend > Controller > `product.controller.js` > `updateProductData`

```
43 // 3) Update product details
44
45 export async function updateProductData(req,res){
46     // try and catch method
47     try{
48         // find id of the product
49         const _id= req.params.id;
50         // update
51         let updatedData = await ProductsModel.findByIdAndUpdate( _id, req.body,{new:true});
52         // if there is no product to update in DB
53         if(!updatedData){
54             return res.status(404).json({message:"No product's data is found in DataBase"})
55         }
56
57         return res.status(201).json({"Updated data":updatedData})
58     }
59     // common error |
60     catch(err){
61         return res.status(500).json({err:"Something went wrong while try to update product's data list from DB 🔞"})
62     }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

> < TERMINAL

(node:41604) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default in Mongoose 7. Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)

🔥 Server is running on 8000

DB is connected

Response in Thunder-client

The screenshot shows the Postman interface with the following details:

- Request URL:** `/localhost:8000/api/product/6859865b1fb38fbcb410ed4d`
- Status:** 201 Created
- Size:** 457 Bytes
- Time:** 164 ms
- Response Body:**

```
Updated data: {
  "_id": "6859865b1fb38fbcb410ed4d",
  "product_name": "Moisturiser for Face & Body",
  "imageUrl": "https://rukminim3.flixcart.com/image/1042/971/xif0q
               /moisturizer-cream/f/5/v/150-hydra-refresh-ultra-light-gel
               -zero-oil-moisturizer-cream-original-imah2rcnfbehz5ap.jpeg?q
               =60&crop=false",
  "description": "Hydra Refresh Ultra-Light Gel, Oil Free
                  Moisturiser for Face & Body",
  "price": "230 INR",
  "ratings": 3.8,
  "category": "Beauty_products",
  "stock": 23,
  "v": 0}
```
- Body Content:**

```
1 {
2
3   "ratings":3.8 ,
4   "price":"230 INR"
5
6
7 }
```

Response in MongoDB compass

localhost:27017 > test > products Open MongoDB shell

Documents (1) Aggregations Schema Indexes (1) Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options ▾

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#) 25 ▾ 1-2 of 2

```
imageUrl : "https://rukminim3.flixcart.com/image/1042/971/xif0q/moisturizer-cream/..."
description : "Daily Glow Face Cream for Bright Skin With Vitamin C & Turmeric"
price : "300 INR"
ratings : 4.2
category : "Beauty_products"
stock : 50
__v : 0

_id: ObjectId('6859865b1fb38fbcb410ed4d')
product_name : "Moisturiser for Face & Body"
imageUrl : "https://rukminim3.flixcart.com/image/1042/971/xif0q/moisturizer-cream/..."
description : "Hydra Refresh Ultra-Light Gel, Oil Free Moisturiser for Face & Body"
price : "230 INR"
ratings : 3.8
category : "Beauty_products"
stock : 23
__v : 0
```

Delete the Product

- Created the route to perform delete the product – “/api/product/:id:deleteProduct
- Defining the logic in controller part
- Check with thunder-client

```
JS server.js U JS Products.model.js U JS Product.route.js U JS product.controller.js U X TC New Request
shoppy-app > Backend > Controller > JS product.controller.js > ⚡ deleteProduct > ✓ "Deleted Successfully"
66 // 4) delete the product
67
68 export async function deleteProduct(req,res){
69   // try and catch method
70   try{
71     // find id of the product
72     const _id= req.params.id;
73     // delete
74     let deletedData = await ProductsModel.findByIdAndDelete( _id);
75     // if there is no product to delete in DB
76     if(!deletedData){
77       return res.status(404).json({message:"No product's data is found in DataBase"})
78     }
79
80     return res.status(201).json(["Deleted Successfully":deletedData])
81   }
82   // common error
83   catch(err){
84     return res.status(500).json({err:"Something went wrong while try to delete product's data from DB ▲"})
85   }
86
87 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

> ✘ TERMINAL

DB is connected

powershell... node Backend

Thunder - client response

```
JS server.js U JS Products.model.js U JS Product.route.js U JS product.controller.js U X TC New Request X
Status: 201 Created Size: 355 Bytes Time: 30 ms
Response Headers 6 Cookies Results Docs
1 {
2   "Deleted Successfully": {
3     "_id": "6859f82e295de4f5cb59711b",
4     "product_name": "Moisturiser for Face & Body",
5     "imageUrl": "https://rukminim3.flixcart.com/image/1042/971/xif0q/moisturizer-cream/...",
6     "description": "Hydra Refresh Ultra-Light Gel, Oil Free Moisturiser for Face & Body",
7     "price": "230 INR",
8     "ratings": 3.8,
9     "category": "Beauty_products",
10    "stock": 23,
11    "__v": 0
12  }
13 }
```

Query Headers 2 Auth Body 1 Tests Pre Run

Query Parameters

parameter value

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

> ✘ TERMINAL

POST /api/product 201 - 62ms
Middleware is connected
DELETE /api/product/6859f82e295de4f5cb59711b 201 - 18ms

powershell... node Backend

MongoDB compass

localhost:27017 > test > products Open MongoDB shell

Documents 1 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#) 25 1-1 of 1

```
_id: ObjectId('68597f16d98750997ca46448')
product_name: "Mamaearth Daily Glow Face Cream"
imageUrl: "https://rukminim3.flixcart.com/image/1042/971/xif0q/moisturizer-cream/..."
description: "Daily Glow Face Cream for Bright Skin With Vitamin C & Turmeric"
price: "300 INR"
ratings: 4.2
category: "Beauty_products"
stock: 50
__v: 0
```

CRUD operations for Cart

Defined schemas and model for cart

The screenshot shows the code editor with the file `Cart.model.js` open. The code defines a schema for a `Cart` document. It includes a `user` field (a reference to a `User` document) and an array of `items`. Each item has a `product` (a reference to a `Product` document) and a `quantity` (a required number). The `quantity` must be at least 1.

```
const cartSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  items: [
    {
      product: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Product",
        required: true
      },
      quantity: {
        type: Number,
        required: true,
        min: 1
      },
    }
  ]
});
```

The terminal below shows the message `DB seeded`.

Create new cart item

- I have defined the route to create new cart item – “/api/cart”,`createCartItem`
- Created schema and model for cart items
- Defined the logic in controller part
- Check with thunder-client

The screenshot shows the code editor with the file `Cart.controller.js` open. The `addToCart` function takes a `req` and `res` object. It retrieves the user ID from the request, extracts the product ID and quantity from the body, and then tries to find an existing cart for the user. If no cart is found, it creates a new one with the given item. If the cart already exists, it checks if the product is already in the cart. If it is, it updates the quantity; otherwise, it adds the new item. The terminal shows the message `DB seeded`.

```
export async function addToCart(req,res){
  const userId = req.user._id;
  const { productId, quantity } = req.body;

  try{
    let cart = await CartModel.findOne({ user: userId });

    if (!cart) {
      // Create new cart
      cart = new CartModel({
        user: userId,
        items: [{ product: productId, quantity }]
      });
    } else{
      // Check if product already in cart
      const itemIndex = cart.items.findIndex(
        (item) => item.product.toString() === productId
      );

      if (itemIndex > -1) {
        // Update quantity
        cart.items[itemIndex].quantity += quantity;
      }
    }
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}
```

The terminal shows the message `DB seeded`.

Response in Thunder-client

The screenshot shows the Thunder-client application interface. At the top, there are tabs for Product.route.js U, JS Cart.routes.js U, TC New Request X, TC localhost:8000/api/regist.., JS Cart.model.js U, JS Cart.controller.js U, and several others. The main area has a 'POST' dropdown and a URL input field containing 'http://localhost:8000/api/cart'. A 'Send' button is to the right. Below this, there are tabs for Query, Headers 3, Auth, Body 1 (which is selected), Tests, and Pre Run. Under 'Body 1', there are tabs for JSON (selected), XML, Text, Form, Form-encode, GraphQL, and Binary. The 'JSON Content' section contains the following JSON:

```
1 {
2   "productId": "685a1496c84404a36c4ea60c",
3   "quantity": 3
4 }
```

To the right, the 'Response' tab is selected, showing the API response details: Status: 200 OK, Size: 214 Bytes, Time: 74 ms. The response body is displayed as:

```
1 {
2   "message": "Product added to cart",
3   "cart": {
4     "_id": "685c2585fd66ba46318b6130",
5     "user": "685aca9d875f2eacdbc75a6d",
6     "items": [
7       {
8         "product": "685a1496c84404a36c4ea60c",
9         "quantity": 6,
10        "_id": "685c2585fd66ba46318b6131"
11      }
12    ],
13    "__v": 0
14 }
```

Below the response, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab shows the command executed: POST /api/cart 200 - 47ms, followed by the MongoDB document returned:

```
_id: new ObjectId("685aca9d875f2eacdbc75a6d"),
fullName: 'Hemalatha Thangavel',
email: 'hema2810@gmail.com',
password: '$2b$10$F7UDkmadL31Jw.lm8uVyVOJF04gBsJREA/VTBQBIXHgzD2QXmAxzy',
createdAt: 2025-06-24T15:56:13.979Z,
__v: 0
} user
```

Response in MongoDB compass

The screenshot shows the MongoDB Compass interface. The connection is set to 'localhost:27017 > test > carts'. There is a 'Documents' tab with 1 item, an 'Aggregations' tab, a 'Schema' tab, an 'Indexes' tab with 1 index, and a 'Validation' tab. On the right, there is a button for 'Open MongoDB shell'. Below these tabs, there are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. To the right of the document list, there are buttons for 'Explain', 'Reset', 'Find', 'Options', and a search bar. The document list shows one document with the following details:

```
_id: ObjectId('685c2585fd66ba46318b6130')
user: ObjectId('685aca9d875f2eacdbc75a6d')
items: Array (1)
  0: Object
    product: ObjectId('685a1496c84404a36c4ea60c')
    quantity: 3
    _id: ObjectId('685c2585fd66ba46318b6131')
__v: 0
```

PUT/cart/

Update the quantity of a product in the cart

The screenshot shows the VS Code interface. The top navigation bar has tabs for Product.route.js, Cart.routes.js, New Request, localhost:8000/api/register..., Cart.model.js, Cart.controller.js, and others. The main editor area displays the code for the updateCart method in Cart.controller.js. The terminal below shows the server is running on port 8000 and the database is connected.

```
46 // 2)PUT method
47
48 export async function updateCart(req, res) {
49   const userId = req.user._id;
50   // give req in body
51   const { productId, quantity } = req.body;
52   // try and catch method
53   try {
54     const cart = await CartModel.findOne({ user: userId });
55
56     if (!cart) {
57       return res.status(404).json({ message: "Cart not found" });
58     }
59
60     const itemIndex = cart.items.findIndex(
61       (item) => item.product.toString() === productId
62     );
63
64     if (itemIndex === -1) {
65       return res.status(404).json({ message: "Product not in cart" });
66     }
67   }
```

TERMINAL

```
Server is running on 8000
DB is connected
DB seeded
```

Response in thunder-client

The screenshot shows the Thunder Client interface. A POST request is being made to `http://localhost:8000/api/cart`. The request body is JSON, containing a product ID and quantity. The response status is 200 OK, and the response body is a JSON object detailing the updated cart item.

POST `http://localhost:8000/api/cart` Send

Status: 200 OK Size: 215 Bytes Time: 134 ms

Query Headers³ Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1  {
2    "productId": "685a1496c84404a36c4ea60c",
3    "quantity": 0
4  }
```

Response Headers⁶ Cookies Results Docs

```
1  {
2    "message": "Product added to cart",
3    "cart": {
4      "_id": "685c2585fd66ba46318b6130",
5      "user": "685aca9d875f2eacdbc75a6d",
6      "items": [
7        {
8          "product": "685a1496c84404a36c4ea60c",
9          "quantity": 20,
10         "_id": "685c2585fd66ba46318b6131"
11       }
12     ],
13     "__v": 0
14   }
```

TERMINAL

```
__v: 0
} user
POST /api/cart 200 - 70ms
```

Response in MongoDB compass

The screenshot shows the MongoDB Compass interface. The left sidebar displays a tree view of connections and databases. Under the 'localhost:27017' connection, the 'test' database is selected, showing collections like 'products', 'movies', 'restaurants', and 'users'. The 'carts' collection is currently selected and expanded, showing its documents. The main panel displays the first document in the 'carts' collection:

```
_id: ObjectId('685c2585fd66ba46318b6130')
user: ObjectId('685aca9d875f2eacdbc75a6d')
item: Array (1)
  0: Object
    product: ObjectId('685a1496c84404a36c4ea60c')
    quantity: 20
    _id: ObjectId('685c2585fd66ba46318b6131')
    --v : 0
```

DELETE/cart/ :

Remove a product from the cart.

The screenshot shows a code editor with several tabs open. The active tab is `Cart.controller.js`, which contains the following code:

```
81 // DELETE/cart/
82
83
84 export async function deleteCartItem(req,res){
85   const userId = req.user._id;
86   const { productId } = req.params;
87   // try and catch method
88   try {
89     const cart = await CartModel.findOne({ user: userId });
90
91     if (!cart) {
92       return res.status(404).json({ message: "Cart not found" });
93     }
94
95     cart.items = cart.items.filter(
96       (item) => item.product.toString() !== productId
97     );
98     // save the cart
99     await cart.save();
}
```

Below the code editor is a terminal window showing the application's startup logs:

```
(node:30424) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default in Mongoose 7. Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
⚠️ Server is running on 8000
DB is connected
DB seeded
```

Response in thunder-client

The screenshot shows the Thunder Client interface with the following details:

- Method: `DELETE`
- URL: `http://localhost:8000/api/cart/685a1496c84404a36c4e`
- Status: `200 OK`
- Size: `40 Bytes`
- Time: `220 ms`

The request body is empty, and the response body is:

```
{ "message": "Item removed from cart" }
```

Below the main interface is a terminal window showing the application's startup logs:

```
(node:30424) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default in Mongoose 7. Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
⚠️ Server is running on 8000
DB is connected
DB seeded
```

Authentication of User

- Created schema for users in Models part with proper validations
- Developed a model for schema and performed default export

User Schema

The screenshot shows the User model schema defined in `User.model.js`. The schema includes fields for `fullName`, `email`, and `password`, each with specific validation rules like required fields and minLength. The `password` field also includes a regular expression for email format validation.

```
1 import mongoose from 'mongoose';
2
3 // created schema for users
4
5 const userSchema = new mongoose.Schema({
6   fullName: {
7     type: String,
8     required: [true, "User name is required*"],
9     minlength: 3,
10    },
11   },
12   email: {
13     type: String,
14     required: [true, "Email is required*"],
15     unique: true,
16     match: [/^[\w+@\w+\.\w+$/], "Invalid email format"],
17   },
18   password: {
19     type: String,
20     required: [true, "Password is required*"],
21     minlength: 3,
22   },
23 }
```

The terminal below shows the application is running on port 8000, connected to the database, and seeded.

```
> 🚀 Server is running on 8000
DB is connected
DB seeded
```

The screenshot shows the `userModel` definition in `User.model.js`. It exports the `userModel` object which is a Mongoose model based on the `userSchema`.

```
6 const userSchema = new mongoose.Schema({
7   password: {
8     type: String,
9     minlength: 3,
10    },
11   },
12   createdAt: {
13     type: Date,
14     default: Date.now,
15   },
16 });
17
18 const userModel = mongoose.model("User", userSchema);
19
20 export default userModel;
```

Controller part for users

Registration of users

- I have created route to post user details for registration – “/api/register”, registerUser.
- Installed bcrypt using “npm i bcrypt” for hashing the password.

The screenshot shows a code editor with multiple tabs at the top: JS server.js, JS user.controller.js, package.json, JS User.model.js, JS User.routes.js, JS Seed.js, and JS proc. The main content area displays the following JavaScript code:

```
7 // for register the user if anyone newly visit my shoppy app
8 export function registerUser(req,res){
9     // defining req body
10    const {fullName,email,password}=req.body;
11    // find email already exist or not
12    userModel.findOne({email})
13    .then((data)=>{
14        if(data){
15            return res.status(400).json({message:"The Email is already exist in DB."})
16        }
17
18        const newUser = new userModel({
19            fullName,
20            email,
21            // hashing the password using bcrypt
22            password: bcrypt.hashSync(password, 10)
23        })
24
25        newUser.save()
26        .then((data)=>{
27            return res.status(201).json({"Successfully Registered":data});
28        })
29        .catch((err)=>[
30            return res.status(500).json({err:"Error while trying to register"});
31        ])
32    })
33}
```

Response in Thunder-Client

The screenshot shows the Thunder-Client interface with the following details:

- Method: POST
- URL: http://localhost:8000/api/register
- Status: 201 Created
- Size: 245 Bytes
- Time: 123 ms

The "Body" tab is selected, showing the JSON content sent in the request:

```
1 {
2   "fullName": "Hemalatha Thangavel",
3   "email": "hema2810@gmail.com",
4   "password": "1234"
5 }
```

The "Response" tab shows the JSON response received:

```
1 {
2   "Successfully Registered": {
3     "fullName": "Hemalatha Thangavel",
4     "email": "hema2810@gmail.com",
5     "password": "$2b$10$F7UDkmadL31Jw.1m8uVvVOJF04gBsjR
6         /VT8QBiXHzD2QXmAxzy",
7     "_id": "685aca9d875f2eacdbc75a6d",
8     "createdAt": "2025-06-24T15:56:13.979Z",
9     "__v": 0
10   }
11 }
```

Login by user

- For login the user,I have created the route in model part – “/api/login”,loginUser.
- I have written the logic to login the page in controller part.
- Validate the email and password with proper error handling.
- Validate the password using bcrypt.
- Check with thunder-client.

The screenshot shows a code editor with several tabs: server.js, user.controller.js, New Request, and User.routes.js. The user.controller.js tab is active, displaying the following code:

```
39 export function loginUser(req,res){
40   const {email,password}=req.body;
41   // find email already exist or not
42   userModel.findOne({email})
43   .then((data)>{
44     // if user data not register
45     if(!data){
46       return res.status(404).json({message:"The Email is not registered yet! Kindly go and register!"})
47     }
48
49     let validPassword=bcrypt.compareSync(password ,data.password);
50     // password checking
51     if(!validPassword){
52       return res.status(403),json({message:"Incorrect Password!"})
53     }
54     res.status(201).send({
55       user:[
56         {
57           email:data.email,
58           password:data.password,
59           fullName:data.fullName
59         }
59       ]
59     })
59   })
59 }
```

Below the code editor is a terminal window showing the application starting up:

```
Middleware is connected
POST /api/login 201 - 121ms
[nodemon] restarting due to changes...
[nodemon] starting 'node server.js'
```

Response in Thunder-Client

The screenshot shows the Thunder-Client interface with the following details:

- Method: POST
- URL: http://localhost:8000/api/login
- Status: 201 Created
- Size: 146 Bytes
- Time: 94 ms

The Body tab shows the JSON payload sent to the server:

```
1 {
2
3   "email": "hema2810@gmail.com",
4   "password": "1234"
5 }
```

The Response tab shows the JSON response received from the server:

```
1 {
2   "user": {
3     "email": "hema2810@gmail.com",
4     "password": "$2b$10$F7UDkmadL31Jw.lm8uVyVOJF04gBsjREAVTBQBIXHgzD2QXmAXzy",
5     "fullName": "Hemalatha Thangavel"
6   }
7 }
```

Below the main interface is a terminal window showing the application starting up:

```
DB seeded
Middleware is connected
POST /api/login 201 - 82ms
Middleware is connected
POST /api/login 201 - 84ms
```

Error handling

If password is wrong-Email is correct

The screenshot shows a POST request to `http://localhost:8000/api/login`. The JSON body contains:

```
1 {
2   "email": "hema2810@gmail.com",
3   "password": "123"
4 }
```

The response status is **403 Forbidden**, size **33 Bytes**, and time **133 ms**. The response body is:

```
1 {
2   "message": "Incorrect Password!"
3 }
```

The terminal output shows the server restarting due to changes and starting node server.js. It also shows the connection to the database and middleware being connected. The final line shows the POST /api/login 403 - 95ms.

If email is incorrect - password is correct

The screenshot shows a POST request to `http://localhost:8000/api/login`. The JSON body contains:

```
1 {
2   "email": "hemaa2810@gmail.com",
3   "password": "1234"
4 }
```

The response status is **404 Not Found**, size **70 Bytes**, and time **20 ms**. The response body is:

```
1 {
2   "message": "The Email is not registered yet! Kindly go and register!"
3 }
```

The terminal output shows the server restarting due to changes and starting node server.js. It also shows the connection to the database and middleware being connected. The final lines show the POST /api/login 403 - 95ms and POST /api/login 404 - 7ms.

Authorization

- JSON web token is installed using “npm i jsonwebtoken”
- I have created a access_token

The screenshot shows the Postman interface. A POST request is made to `http://localhost:8000/api/login`. The JSON body contains:

```
1 {
2
3   "email": "hema2810@gmail.com",
4   "password": "1234"
5 }
```

The response status is 201 Created, size is 260 Bytes, and time is 240 ms. The response body is:

```
1 {
2   "user": {
3     "email": "hema2810@gmail.com",
4     "fullName": "Hemalatha Thangavel"
5   },
6   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJpZCI6IjY4NWFjYTlkODc1ZjJlYWNkYmM3NWE2ZCIsImIhdCI
6MTc1MDgyODgxMCwiZXhwIjoxNzUwODMyNDEwfQ
.FHRmaDh43KdwftmCvt1DbUDs5BXc40WtnJSz4iqqqNw"
7 }
```

In the terminal, it shows the server is running on port 8000, DB is connected, and DB seeded.

Verify token

The screenshot shows the `verifyToken.js` file in VS Code. It contains a middleware function for jwt verification:

```
1 // importing dependencies and files
2 import jwt from "jsonwebtoken"
3 import userModel from "../Models/User.model.js";
4
5 // middleware
6 export function verifyToken(req,res,next){
7   if(req.headers && req.headers.authorization && req.headers.authorization.split(" ")[0]==="JWT")
8   {
9     // jwt method to verify the token
10    jwt.verify(req.headers.authorization.split(" ")[1], 'secretKey',
11
12    function(err, verifiedToken) {
13      // if facing any error-error message
14      if(err){
15        return res.status(404).json({message:"Invalid JWT token"})
16      }
17      // console the verified token
18      console.log(verifiedToken, 'verifiedToken')
19
20      userModel.findById(verifiedToken.id)
21      .then((user)=>{
```

Response in Thunder-Client

The screenshot shows the Thunder-Client interface with a successful HTTP request. The request details are as follows:

- Method: GET
- URL: http://localhost:8000/api/products
- Status: 201 Created
- Size: 370.93 KB
- Time: 93 ms

The Headers tab shows the following configuration:

- Accept: */*
- User-Agent: Thunder Client (https://www.thunderclient.net)
- Authorization: JWT eyJhbGciOiJUzI1NiIsInR5

The Response tab displays the JSON response for a product creation:

```
1 [
2   {
3     "_id": "68597f16d98750997ca46448",
4     "product_name": "Mamearth Daily Glow Face Cream",
5     "imageUrl": "https://rukminim3.flixcart.com/image/1042/971/xif0q/moisturizer-cream-c/m/h/-original-imagzzm5ds6umfbdb.jpeg?q=60&crop=false",
6     "description": "Daily Glow Face Cream for Bright Skin With Vitamin C & Turmeric",
7     "price": "300 INR",
8     "ratings": 4.2,
9     "category": "Beauty_products",
10    "stock": 50,
11    "v": A
12  }
```

The Terminal tab shows the verification token output:

```
{ id: '685aca9d875f2eacdbc75a6d', iat: 1750828810, exp: 1750832410 } verifiedToken
{
  _id: new ObjectId("685aca9d875f2eacdbc75a6d"),
  fullName: 'Hemalatha Thangavel',
  email: 'hema2810@gmail.com',
  password: '$2b$10$F7UDkmadL31Jw.lm8uVyVOJF04gBsjREA/VTBQBIXHgzD2QXmAXzy',
  createdAt: 2025-06-24T15:56:13.979Z,
  __v: 0
} user
```

The screenshot shows the Thunder-Client interface with a successful HTTP request. The request details are as follows:

- Method: POST
- URL: http://localhost:8000/api/product
- Status: 201 Created
- Size: 486 Bytes
- Time: 63 ms

The Headers tab shows the following configuration:

- Accept: */*
- User-Agent: Thunder Client (https://www.thunderclient.net)
- Authorization: JWT eyJhbGciOiJUzI1NiIsInR5

The Response tab displays the JSON response for a product creation:

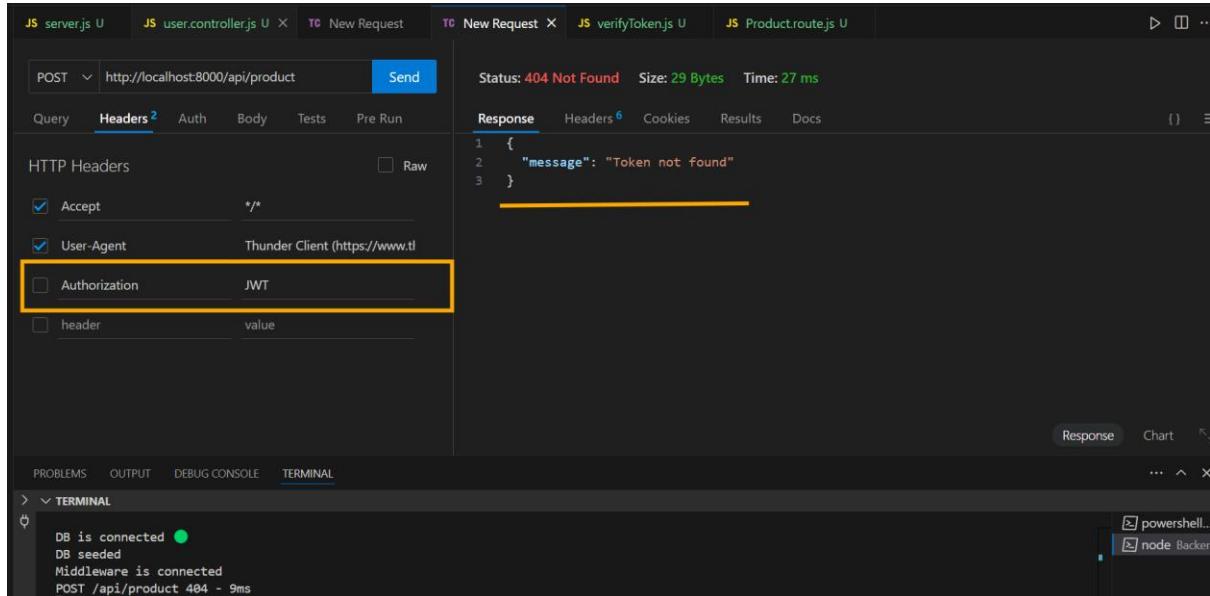
```
1 {
2   "New product is created": {
3     "product_name": "Crystal Bracelet Set",
4     "imageUrl": "https://rukminim3.flixcart.com/image/1042/971/xif0q/shopsy-bangle-bracelet-armlet/z/5/c/free-2-evil-eye-bracelets-for-women-stylish-couple-love-bracelet-original-imagt2fu6ub42fty.jpeg?q=60&crop=false",
5     "description": "Evil Eye Bracelets for Women Stylish Couple Love Bracelet for Girls /Women/Boys/Men",
6     "price": "99 INR",
7     "ratings": 2.8,
8     "category": "Jewelry"
9   }
10 }
```

The Terminal tab shows the verification token output and the POST command:

```
Middleware is connected
{ id: '685aca9d875f2eacdbc75a6d', iat: 1750858864, exp: 1750862464 } verifiedToken
{
  _id: new ObjectId("685aca9d875f2eacdbc75a6d"),
  fullName: 'Hemalatha Thangavel',
  email: 'hema2810@gmail.com',
  password: '$2b$10$F7UDkmadL31Jw.lm8uVyVOJF04gBsjREA/VTBQBIXHgzD2QXmAXzy',
  createdAt: 2025-06-24T15:56:13.979Z,
  __v: 0
} user
POST /api/product 201 - 51ms
```

ERROR handling

If I not click authorization check box in headers



POST Send

Status: 404 Not Found Size: 29 Bytes Time: 27 ms

Response Headers Cookies Results Docs

```
1 {
2   "message": "Token not found"
3 }
```

HTTP Headers Raw

Accept: */*

User-Agent: Thunder Client (https://www.thunderclient.com)

Authorization: JWT

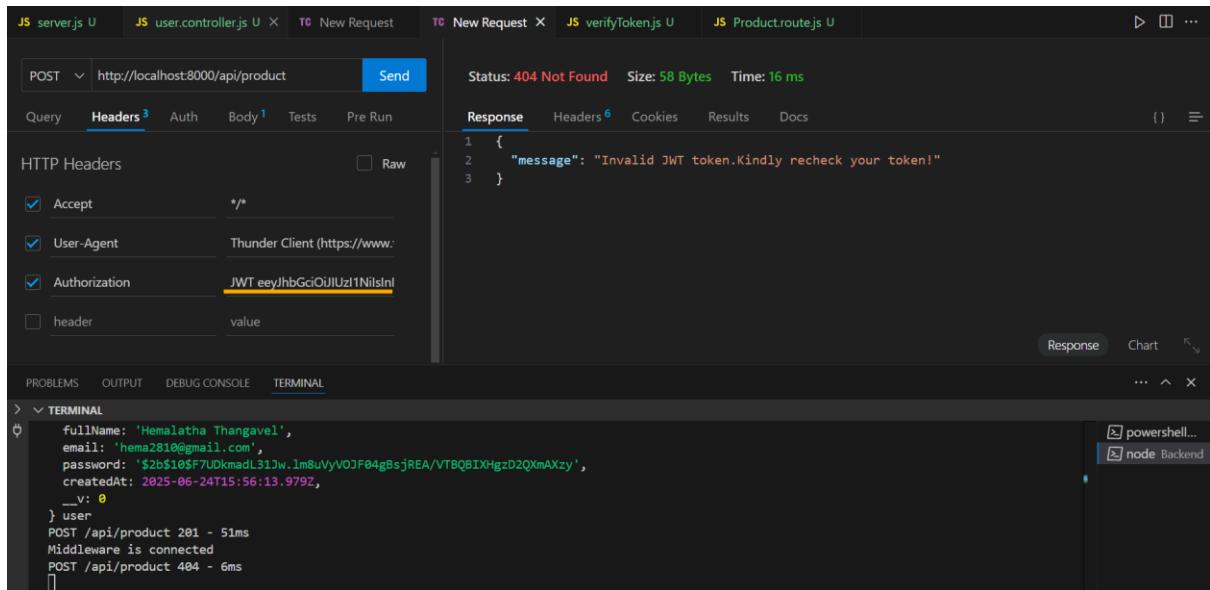
header value

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

DB is connected DB seeded Middleware is connected POST /api/product 404 - 9ms

powershell... node Backend

If I give invalid or wrong access token for authorization



POST Send

Status: 404 Not Found Size: 58 Bytes Time: 16 ms

Response Headers Cookies Results Docs

```
1 {
2   "message": "Invalid JWT token. Kindly recheck your token!"
3 }
```

HTTP Headers Raw

Accept: */*

User-Agent: Thunder Client (https://www.thunderclient.com)

Authorization: JWT eeyJhbGciOiJIUzI1NiLslnI

header value

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

full Name: 'Hemalatha Thangavel', email: 'hemaa2810@gmail.com', password: '\$2b\$10\$F7UDkmaoL31Jw.lm8uVyVOJF04gBsjREAVTBQBIXHgzD2QXmAXzy', created At: 2025-06-24T15:56:13.979Z, _V: 0 } user POST /api/product 201 - 51ms Middleware is connected POST /api/product 404 - 6ms

powershell... node Backend

Conclusion

- Node.js and Express API Setup
- connected with MongoDB compass
- defined routes for products and cart
- performed CRUD operations with proper error handling
- Used MongoDB to store product data and cart items.
- Added screenshots from MongoDB Database
- Performed Authentication & Authorization
- Tested with Thunder-Client
- Ensured all api is working
- included comments and documentations

GitHub link

<https://github.com/Hema2802/Backend-ecommerce-shoppy--app>