

ZING

A MEAN STACK
ECOMMERCE STORE

DONE BY :

AAKAR MUTHA : BFDL

INTRODUCTION

Due to the pandemic, many of us have started to buy things online rather than visiting a physical store whenever possible. Using an online ecommerce platform has made buying the products with better deals a reality. With an online store, a seller can widen their audience which in turn results in more revenue. The seller can manage the products that he has on sale, give special discounts on products, sell products from different sellers on their platform and many more. Due to these advantages, having and maintaining an online store has become an essential task for any merchant.

In this Case Study I aimed at making a fully functional Ecommerce website. A customer can add products to cart and purchase them while a seller can see the stats in real time, add products, update products as well as any normal user add products to cart and purchase them.

TECH STACK



1) MONGODB - 4.4.0

MongoDB is an open source NoSQL database management program. NoSQL is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information. MongoDB supports various forms of data. It is one of the many nonrelational database technologies that arose in the mid-2000s under the NoSQL banner - normally, for use in big data applications and other processing jobs involving data that doesn't fit well in a rigid relational model. Instead of using tables and rows as in relational databases, the MongoDB architecture is made up of collections and documents.

Organizations can use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

2) EXPRESS JS - 4.17.2

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework –

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.

Since it is a framework of Node.js , most of the code is already written for programmers to work with. You can build a single page, multi-page, or hybrid web applications using Express.js. Express.js is lightweight and helps to organize web applications on the server-side into a more organized MVC architecture.

3) ANGULAR - 13.2.1

Angular is a development platform, built on TypeScript. As a platform, Angular includes:

- A component-based framework for building scalable web applications
- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
- A suite of developer tools to help you develop, build, test, and update your code

With Angular, you're taking advantage of a platform that can scale from single-developer projects to enterprise-level applications.

Angular is designed to make updating as straightforward as possible, so take advantage of the latest developments with a minimum of effort. Best of all, the Angular ecosystem consists of a diverse group of over 1.7 million developers, library authors, and content creators.

4) NODE JS 16.13.2 (LTS)

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

This is in contrast to today's more common concurrency model, in which OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. Almost no function in Node.js directly performs I/O, so the process never blocks except when the I/O is performed using synchronous methods of Node.js standard library. Because nothing blocks, scalable systems are very reasonable to develop in Node.js.

FEATURES

USER DASHBOARD

- 1) View all products from all sellers at a glance.
- 2) Add Products to cart once the user is logged in.
- 3) View product details.
- 4) View Cart.
- 5) Delete items from cart.
- 6) Make Payment.

SELLER DASHBOARD

- 1) View statistics like Total Orders, Total orders shipped, Total products shipped.
- 2) Add New Products.
- 3) View All the products sold by them.
- 4) Edit the product details like title, price and description
- 5) All the functionalities of the user.

ANGULAR CONCEPTS USED

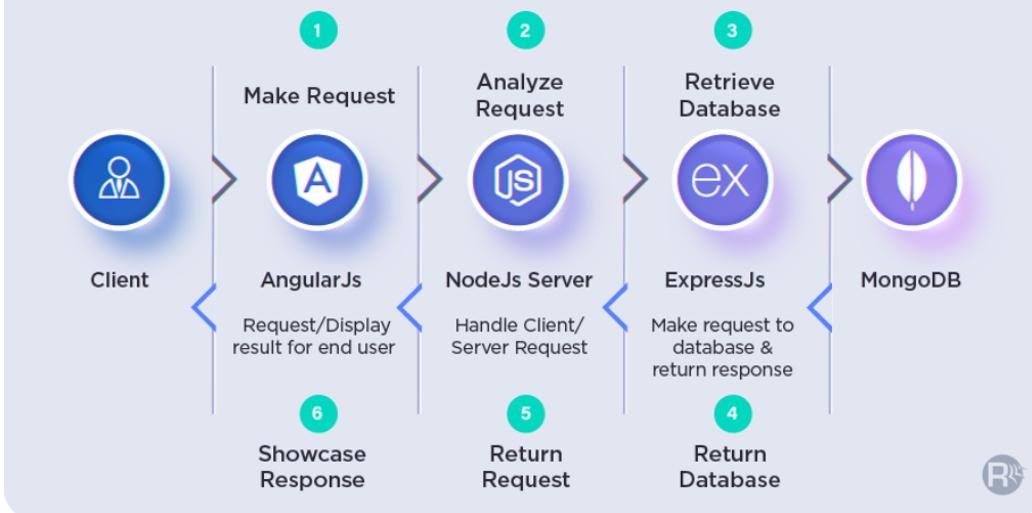
- Each Dashboard is in a different module
- Lazy loading for the different modules
- Reactive Forms to handle form data with validation
- Template Forms.
- Role Guards.
- Routing

LIBRARIES USED

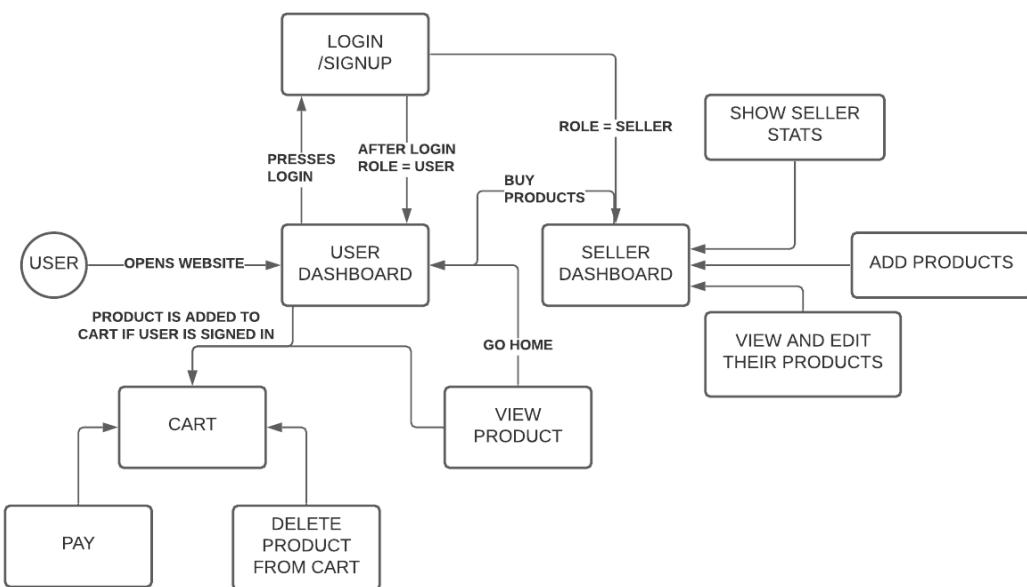
- Ng-bootstrap - 11.0.0
- Ng-fontawesome - 5.15.4
- Mongoose - 6.2.0
- Multer - 1.4.4
- Express - 4.17.2
- Cors - 2.8.5
- Angular Material - 13.2.1

ARCHITECTURE DIAGRAM

The MEAN Stack Architecture



COMPONENT DIAGRAM



API TESTS

1) POST - /signup

Used to sign up a new user.

The screenshot shows the Hoppscotch API testing tool interface. The request method is set to POST, and the URL is http://localhost:3000/signup. The Body tab is selected, showing a Raw Request Body with the following JSON payload:

```
1  {
2   "fname": "test-name",
3   "lname": "test-lname",
4   "email": "test-email@email.com",
5   "mnum": "1234567890",
6   "pass": "abcd1234",
7   "role": "user"
8 }
```

The response status is 200 OK, time taken is 174 ms, and size is 58 B. The Response Body shows a JSON object with a message key:

```
1  {
2   "message": "You Have Successfully Registered with ESTORE"
3 }
```

2) GET - /login

Used to login a user into the app.

The screenshot shows the Hoppscotch API testing tool interface. The request method is set to GET, and the URL is http://localhost:3000/login?email=test-email@email.com&pass=abcd1234. The Parameters tab is selected, showing Query Parameters:

```
1 email:test-email@email.com
2 pass:abcd1234
```

The response status is 201 Created, time taken is 170 ms, and size is 145 B. The Response Body shows a JSON object with user details:

```
1  {
2   "userId": "ed1c1360228dd",
3   "fname": "test-name",
4   "lname": "test-lname",
5   "email": "test-email@email.com",
6   "mobile": 1234567890,
7   "role": "user",
8   "status": 200
9 }
```

3) POST - /products

Used to add products to cart

The screenshot shows the Hoppscotch API testing tool interface. The top bar displays the title "HOPPSOTCH" and a star icon with the number "36,185". The main area is set up for a "POST" request to "http://localhost:3000/products". The "Body" tab is selected, showing the raw request body content:

```
1 {  
2   "userId": "6fa84c314d841",  
3   "title": "Mens Casual Premium Slim Fit T-Shirts ",  
4   "price": 22.3,  
5   "description": "Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.",  
6   "image": "https://fakestoreapi.com/img/71-3HjGNDUL._AC_SY879._SX._UX._SY_.jpg"  
7 }
```

Below the request body, the response status is shown as "Status: 200 • OK" with a time of "15 ms" and a size of "40 B". The "JSON" tab is selected, displaying the response body:

```
1 {  
2   "message": "Product Added Successfully"  
3 }
```

4) GET - /products

Used to get all products of a particular seller with userId.

The screenshot shows the Hoppscotch API testing tool interface. The top bar displays the title "HOPPSOTCH" and a star icon with the number "36,188". The main area is set up for a "GET" request to "http://localhost:3000/products". The "Parameters" tab is selected, showing the query parameters:

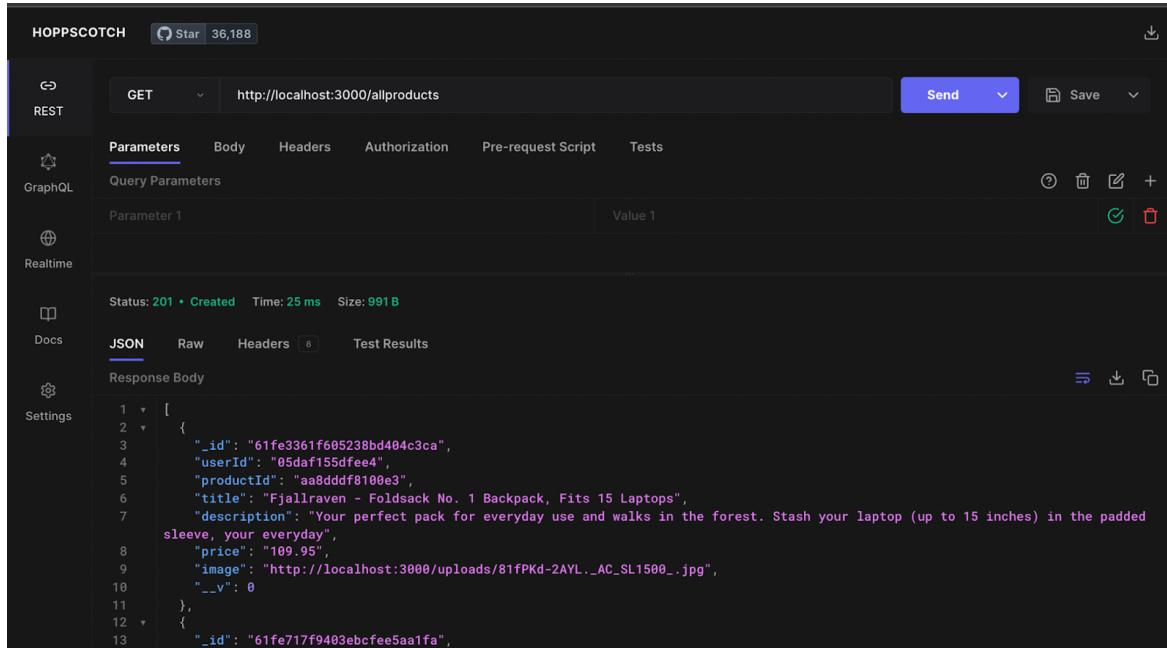
Parameter	Value
userId	6fa84c314d841
Parameter 2	Value 2

Below the parameters, the response status is shown as "Status: 201 • Created" with a time of "1176 ms" and a size of "598 B". The "JSON" tab is selected, displaying the response body:

```
1 [  
2   [  
3     {  
4       "_id": "61fe717f9403ebcf05aa1fa",  
5       "userId": "6fa84c314d841",  
6       "productId": "cbe167845d3f3",  
7       "title": "Mens Casual Premium Slim Fit T-Shirts ",  
8       "description": "Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.",  
9       "price": "22.3",  
10      "image": "https://fakestoreapi.com/img/71-3HjGNDUL._AC_SY879._SX._UX._SY_.jpg",  
11      "__v": 0  
12    }  
13  ]
```

5) GET - /allproducts

Used to get all the products to show to the user.



Request URL: `http://localhost:3000/allproducts`

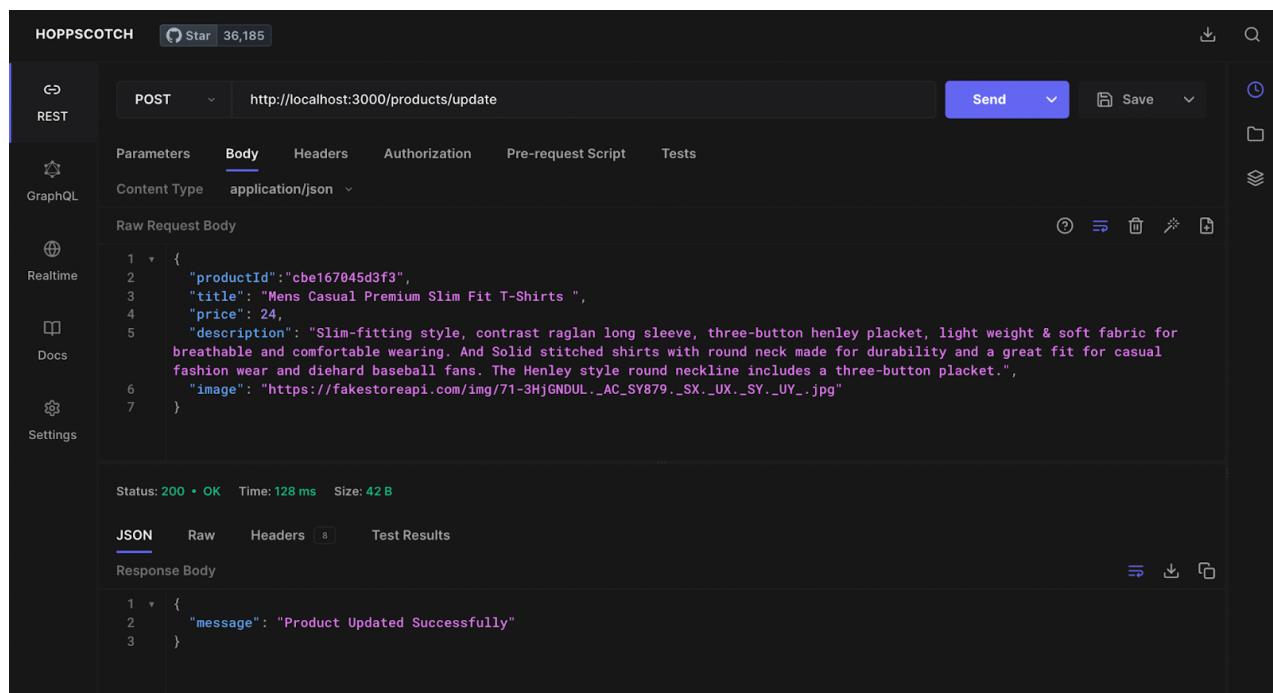
Response Status: 201 • Created Time: 25 ms Size: 991 B

Response Body:

```
1  [
2    {
3      "_id": "61fe3361f605238bd404c3ca",
4      "userId": "05daf155dfee4",
5      "productId": "aa80ddf8100e3",
6      "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
7      "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday",
8      "price": "109.95",
9      "image": "http://localhost:3000/uploads/81fPKd-2AYL._AC_SL1500_.jpg",
10     "__v": 0
11   },
12   {
13     "_id": "61fe717f9403ebcf5aa1fa",
```

6) POST - /products/update

Used to update the product details by the seller.



Request URL: `http://localhost:3000/products/update`

Content Type: application/json

Raw Request Body:

```
1  {
2    "productId": "cbe167045d3f3",
3    "title": "Mens Casual Premium Slim Fit T-Shirts",
4    "price": 24,
5    "description": "Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.",
6    "image": "https://fakestoreapi.com/img/71-3HjGNDUL.AC.SY879._SX._UY_.jpg"
7  }
```

Status: 200 • OK Time: 128 ms Size: 42 B

Response Body:

```
1  {
2    "message": "Product Updated Successfully"
3  }
```

7) POST - /products/delete

Used to delete a product by the seller

The screenshot shows the Hoppscotch API testing tool interface. The request URL is `http://localhost:3000/products/delete`. The request body is a JSON object with a single key-value pair: `{"productId": "cbe167045d3f3"}`. The response status is `200 OK`, time taken is `118 ms`, and size is `42 B`. The response body contains the message `"message": "Product Deleted Successfully"`.

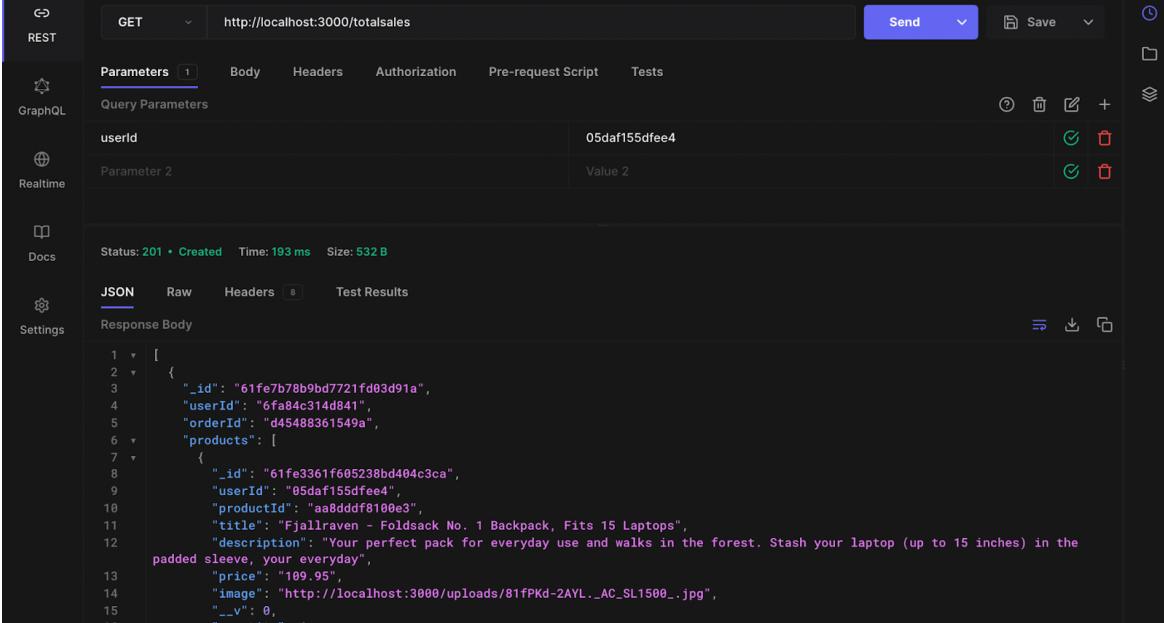
8) POST - /products/image-upload

Used to add the image of the product by the seller.

The screenshot shows the Postman API testing tool interface. The request URL is `http://localhost:3000/product/image-upload`. The request method is `POST`. The Body tab is selected, showing a form-data key `image` with a value of `hp-laptop.jpg`. The response status is `200 OK`, time taken is `29 ms`, and size is `353 B`. The response body is a JSON object with the key `path` pointing to the uploaded file URL: `http://localhost:3000/uploads/hp-laptop.jpg`.

9) GET - /totalsales

Returns the products that are of the seller to display in the seller dashboard.

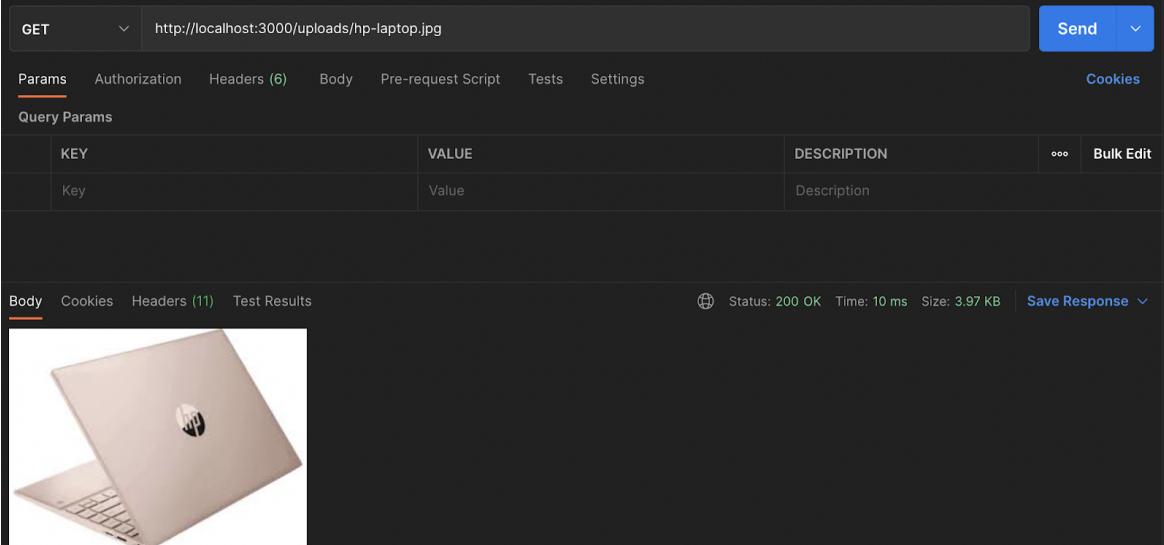


The screenshot shows a Postman request for `http://localhost:3000/totalsales` using the `REST` tab. The `Parameters` section contains a `userId` parameter set to `05daf155dfee4`. The `JSON` tab displays the response body:

```
1  [
2    {
3      "_id": "61fe7b78b9bd7721fd83d91a",
4      "userId": "6fa84c314d841",
5      "orderId": "d45488361549a",
6      "products": [
7        {
8          "_id": "61fe3361f605238bd404c3ca",
9          "userId": "05daf155dfee4",
10         "productId": "aa8ddf8f810e03",
11         "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
12         "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday",
13         "price": "109.95",
14         "image": "http://localhost:3000/uploads/81fPKd-2AYL.AC_SL1500..jpg",
15         "__v": 0,
16         "quantity": 1
17     }
18   ]
19 }
```

10) GET - /uploads

Used to get the image of a product to display in the website



The screenshot shows a Postman request for `http://localhost:3000/uploads/hp-laptop.jpg` using the `REST` tab. The `Params` section contains a `Key` parameter set to `Value`. The `Body` tab displays the response body as a binary image of a silver laptop.

11) GET - /carts

Used to get the cart of a particular user

The screenshot shows the Hoppscotch API testing tool interface. The top navigation bar includes 'HOPPSCOTCH' and a 'Star' button with the number '36,188'. Below the navigation, there's a 'REST' tab selected, followed by 'GraphQL' and 'Realtime' tabs. A 'Send' button is on the right. The main area shows a 'GET' request to 'http://localhost:3000/carts'. Under 'Parameters', there is a 'Query Parameters' section with 'userId' set to '05daf155dfee4'. Below the request, the status is shown as 'Status: 201 • Created' with a time of '25 ms' and a size of '168 B'. The 'JSON' tab is selected, showing the response body as a JSON array:

```
1  [
2    {
3      "_id": "61fe7c2bb9bd7721fd03d92b",
4      "userId": "05daf155dfee4",
5      "products": [
6        {
7          "productId": "aa8dddf8100e3",
8          "quantity": 1
9        },
10       {
11         "productId": "d18d2aee15082",
12         "quantity": 1
13       }
14     ]
15   ]
```

12) POST - /carts/delete

Used to delete item from cart

13) POST - /carts

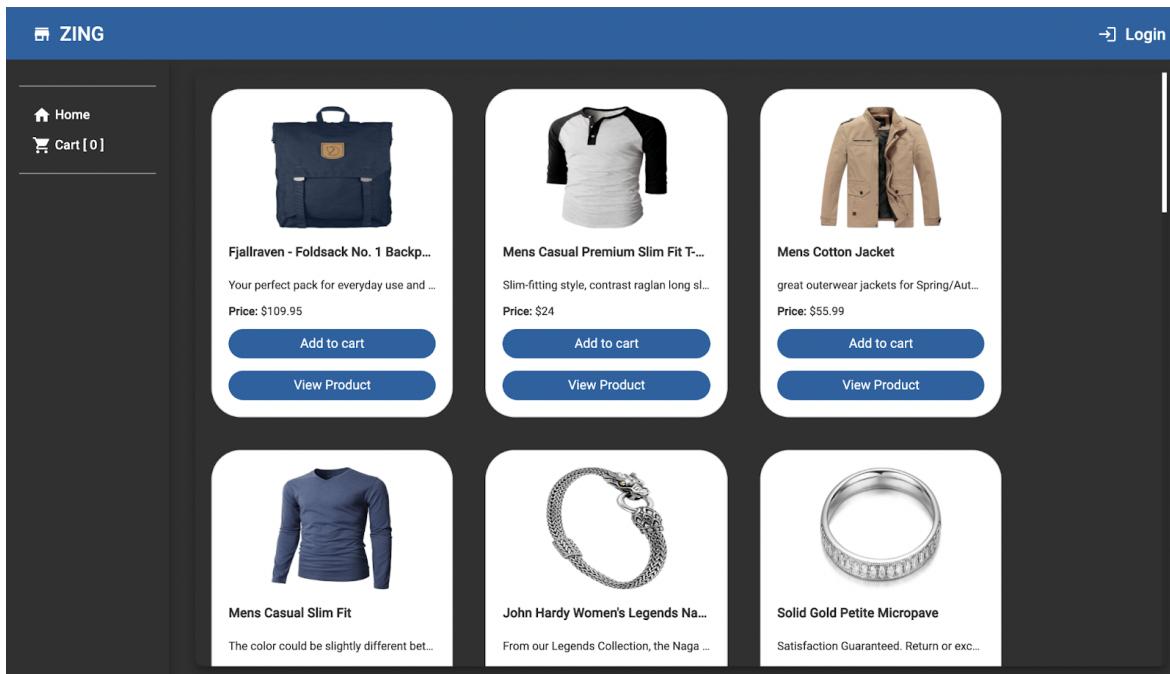
Used to add items to cart for a particular user

14) POST - /carts/pay

Used when the user makes payment and the payment is successful.

APPLICATION SCREENS

Home Screen



Login Screen

The Login screen has a dark background with a central white rectangular form. At the top center, it says 'LOGIN TO ZING'. Below that are two input fields: one for 'EMAIL' and one for 'PASSWORD', both with placeholder text and blue underline lines. At the bottom are two blue rectangular buttons labeled 'LOGIN' and 'SIGN UP'.

Signup Screen

SIGN UP FOR ZING

FIRST NAME

LAST NAME

MOBILE NUMBER

EMAIL

PASSWORD

[SIGN UP](#) [LOGIN](#)

When the User is Logged In

 **ZING** [Logout](#)



Aakar Mutha

[Home](#) [Cart \[0 \]](#)



Fjallraven - Foldsack No. 1 Backpack
Your perfect pack for everyday use and ...
Price: \$109.95

[Add to cart](#) [View Product](#)



Mens Casual Premium Slim Fit T-shirt
Slim-fitting style, contrast raglan long sl...
Price: \$24

[Add to cart](#) [View Product](#)



Mens Cotton Jacket
great outerwear jackets for Spring/Aut...
Price: \$55.99

[Add to cart](#) [View Product](#)



Mens Casual Slim Fit
The color could be slightly different bet...

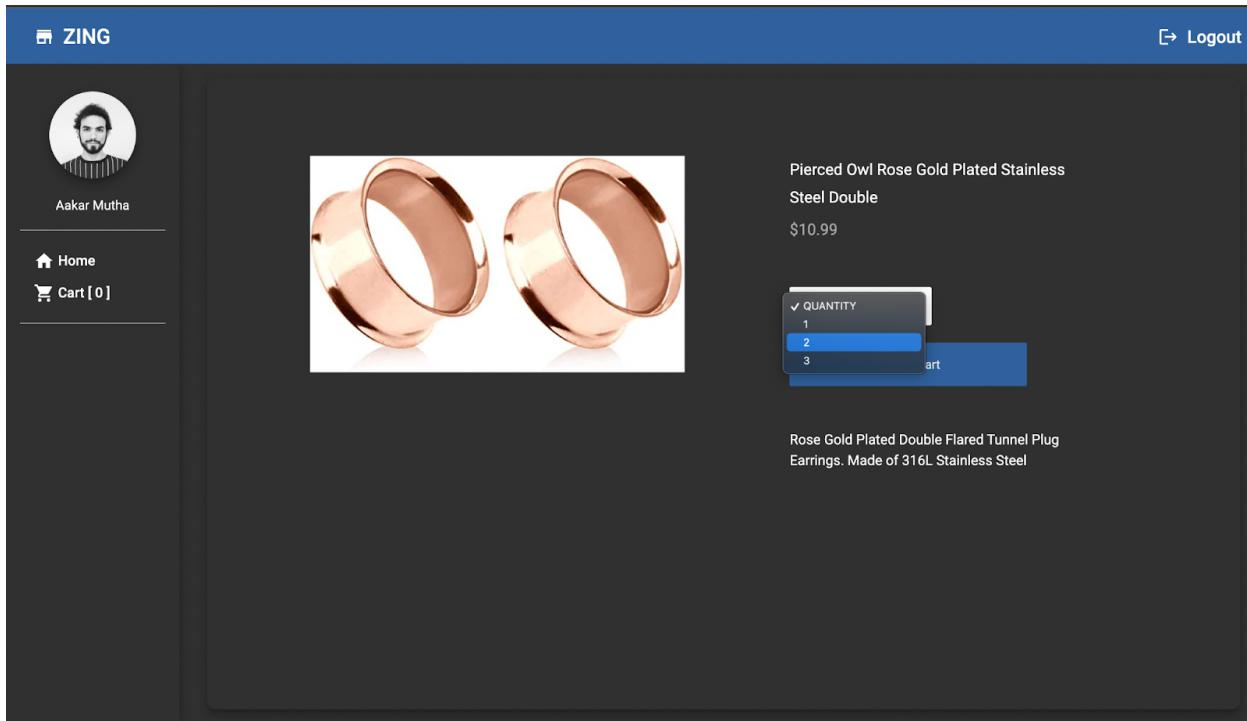


John Hardy Women's Legends Naga
From our Legends Collection, the Naga ...



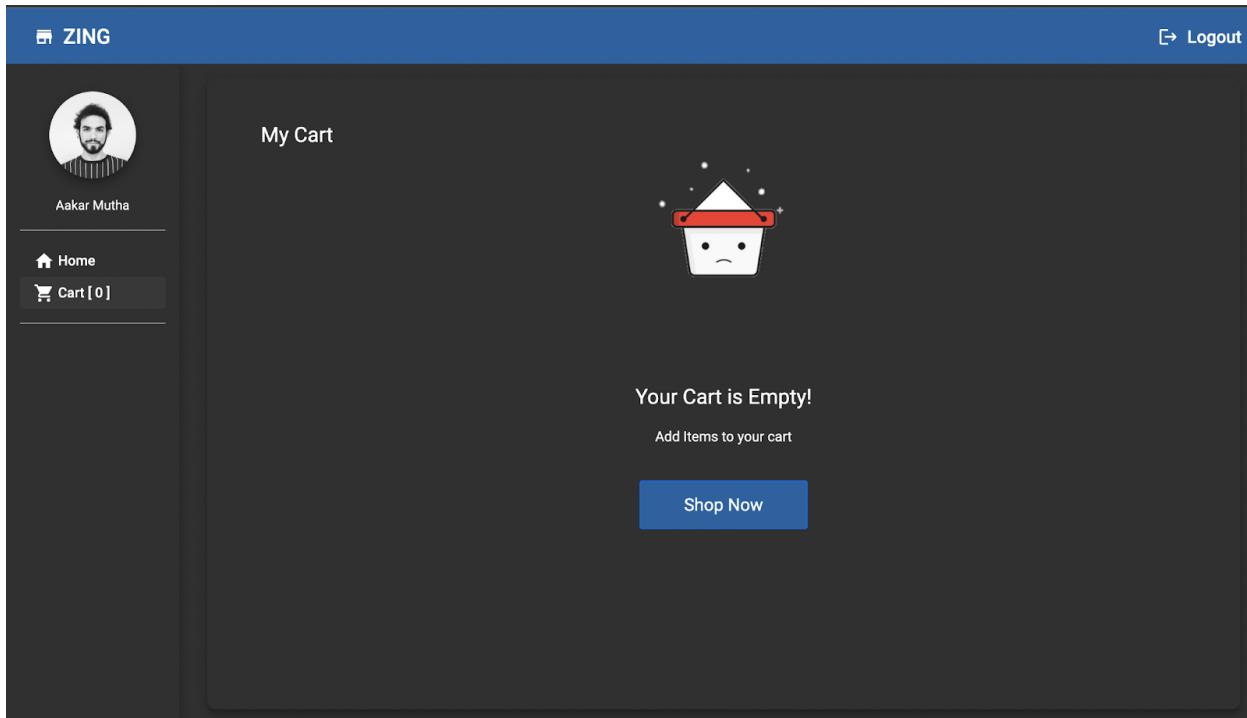
Solid Gold Petite Micropave
Satisfaction Guaranteed. Return or exc...

View Product Details



The screenshot shows a product detail page for 'Pierced Owl Rose Gold Plated Stainless Steel Double'. The product image displays two rose gold double-flared tunnel plugs. The price is listed as \$10.99. A quantity selector dropdown is open, showing options 1, 2, and 3, with 2 selected. Below the product details, a descriptive text reads: 'Rose Gold Plated Double Flared Tunnel Plug Earrings. Made of 316L Stainless Steel'.

Empty Cart



The screenshot shows an 'Empty Cart' page. The title 'My Cart' is displayed above a central illustration of a sad shopping cart character. The text 'Your Cart is Empty!' is shown, followed by a link 'Add Items to your cart'. A large blue 'Shop Now' button is at the bottom.

Home Page When there are items in the Cart

The screenshot shows the ZING home page with a dark header bar. On the left, a sidebar displays a user profile picture of Aakar Mutha, navigation links for 'Home' and 'Cart [2]', and a 'Logout' button. The main content area features four product cards arranged in a grid:

- Fjallraven - Foldsack No. 1 Backpack**
Your perfect pack for everyday use and ...
Price: \$109.95
[Add to cart](#) [View Product](#)
- Mens Casual Premium Slim Fit T...**
Slim-fitting style, contrast raglan long sl...
Price: \$24
[Add to cart](#) [View Product](#)
- Mens Cotton Jacket**
great outerwear jackets for Spring/Aut...
Price: \$55.99
[Add to cart](#) [View Product](#)
- Mens Casual Slim Fit**
The color could be slightly different bet...
[View Product](#)

Cart when there are item in it

The screenshot shows the ZING cart page with a dark header bar. The sidebar on the left is identical to the home page. The main content area displays a table of items in the cart:

Product Name	Product Image	Price	Quantity	Total	Action
Solid Gold Petite Micropave		168	3	504	Delete
Acer SB220Q bi 21.5 inches Full HD (1920 x 1080) IPS Ultra-Thin		599	1	599	Delete

Total Amount \$1,103 [Pay](#)

Seller Dashboard

The Seller Dashboard interface for ZING. It features a dark header bar with the ZING logo and a Logout button. On the left, there's a user profile section for Aakar Mutha, a Seller, with a circular profile picture and a menu bar below it containing Home, Add Products, My Products, and Buy Products.

The main area displays six summary cards:

- Total Orders: 4
- Total Orders Shipped: 4
- Total Orders Pending: 0
- Total Products Shipped: 5
- Total Earnings: \$758.455
- Returns Till Date: 0

Add Products

The Add Product form interface for ZING. It follows the same dark theme as the dashboard. The left sidebar includes the user profile for Aakar Mutha and the same menu bar as the dashboard.

The main form is titled "Add Product" and contains the following fields:

- Product name: An input field.
- Product Description: An input field.
- Product price: An input field.
- Product Image: A file input field with placeholder text "Choose file" and "No file chosen".

A green "Add Product" button is located at the bottom of the form.

My Products

The screenshot shows the ZING app interface. On the left, there's a sidebar with a user profile picture of Aakar Mutha, the name "Aakar Mutha", and the title "Seller". Below the profile are navigation links: Home, Add Products, My Products (which is selected and highlighted in grey), and Buy Products. The main area displays six product cards in a grid. The first card is for a "Mens Casual Premium Slim Fit T-shirt" with a price of \$24. The second card is for a "WD 4TB Gaming Drive Works with..." with a price of \$114. The third card is for an "Acer SB220Q bi 21.5 inches Full ..." with a price of \$599. The fourth card is for a "Samsung 49-Inch CHG90 144Hz ...". The fifth card is for a "BIYLACLESEN Women's 3-in-1 Sn...". The sixth card is for a "Lock and Love Women's Removab...". Each card has "Edit" and "Delete" buttons at the bottom.

Update Product Modal

This screenshot shows the "Edit Product" modal window overlaid on the main product list. The modal has a dark background and contains fields for updating a specific product. The product being edited is a "Samsung 49-Inch CHG90 144Hz Curved Gaming Monitor (LC49H90DMNXZA) – Super". The "Product name" field is filled with the monitor's name. The "Product Description" field contains a detailed description of the monitor's features, including its 49 INCH SUPER ULTRAWIDE 32:9 CURVED GAMING MONITOR with dual 27 inch screen side by side QUANTUM DOT (QLED) TECHNOLOGY, HDR support and factory calibration provides stunningly realistic and. The "Product price" field is set to \$999.99. At the bottom of the modal are "Update Product" and "Close" buttons. The background shows the same product cards as the previous screenshot, with the "Edit" button for the Samsung monitor being highlighted.

MONGODB ATLAS VIEW

Carts Collection

The screenshot shows the MongoDB Atlas interface for the 'bajaj-task' database. The left sidebar shows the project navigation and various service tabs like Deployment, Databases, Data Services, Security, and Data API. The 'Collections' tab is selected. The main panel displays the 'bajaj-task.carts' collection with 4 documents. A search bar at the top has the query '{ field: "value" }'. Below it, there are tabs for Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. The 'Find' tab is active, showing two results:

```
_id: ObjectId("61ff6757680aa5e6c4d53ee9")
userId: "54a6f3ab8e28ba"
> products: Array
__v: 0

_id: ObjectId("61ff69d90e15b378bbbf9bd7")
userId: "61ff69d90e15b378bbbf9bd7"
> products: Array
__v: 0
```

Orders Collection

The screenshot shows the MongoDB Atlas interface for the 'bajaj-task' database, similar to the previous one but for the 'orders' collection. The left sidebar and top navigation are identical. The 'Collections' tab is selected. The main panel displays the 'bajaj-task.orders' collection with 8 documents. The 'Find' tab is active, showing two results:

```
_id: ObjectId("61fe139a21de8fb9299a91c7")
userId: "05da155dfcc4"
orderId: "17661fffdcc76a"
> products: Array
amount: "600"
__v: 0

_id: ObjectId("61fe1dd41010dd15aa0f8fee")
userId: "54a6f3ab8e28ba"
orderId: "9779f12c28ec23"
> products: Array
amount: "1700"
__v: 0
```

At the bottom of the interface, a URL is displayed: <https://cloud.mongodb.com/v2/61fb4ee89bdfc44ac367a4a4#metrics/replicaSet/61fb7afc7a77d25e345a3320/explorer/bajaj-task/orders/find>

Products Collection

The screenshot shows the MongoDB Atlas interface for the 'bajaj-task.products' collection. The left sidebar includes sections for Deployment, Databases (selected), Data Services, and Security. The main panel displays document results with a query filter of '{ field: 'value' }'. One document is shown in detail:

```
_id: ObjectId("61fe3361f605238bd404c3ca")
userId: "6fa0f155dfe4"
productId: "aaadddfb10e3"
title: "FallRaven - Foldack No. 1 Backpack, Fits 15 Laptops"
description: "Your perfect pack for everyday use and walks in the forest. Stash your..."
price: "109.95"
image: "http://localhost:3000/uploads/81fPKd-2AYL.AC_SL1500_.jpg"
__v: 0
```

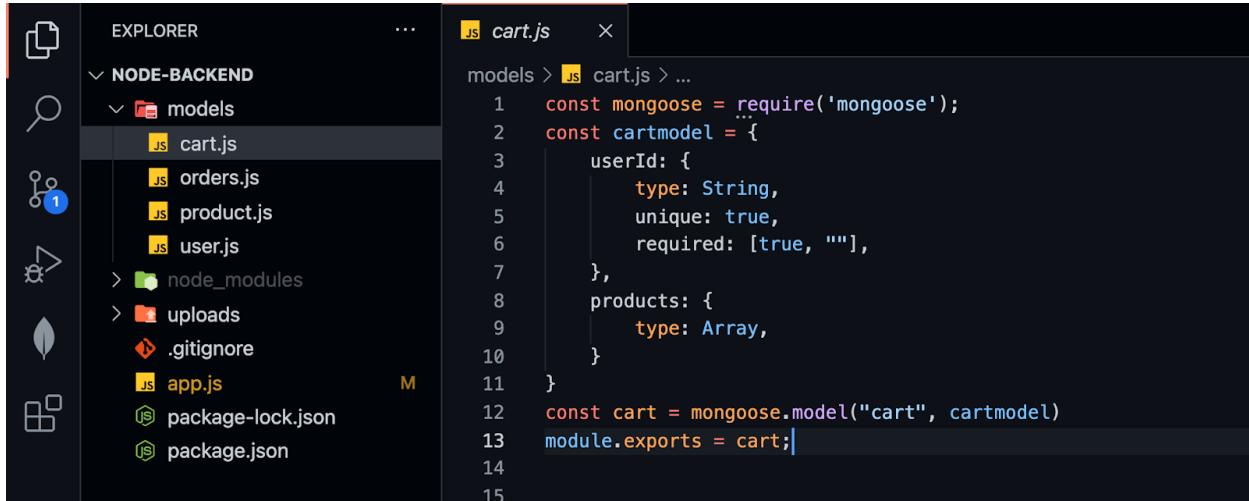
Users Collection

The screenshot shows the MongoDB Atlas interface for the 'bajaj-task.users' collection. The left sidebar includes sections for Deployment, Databases (selected), Data Services, and Security. The main panel displays document results with a query filter of '{ field: 'value' }'. One document is shown in detail:

```
_id: ObjectId("61fbc51d859ffdb1ef25f1")
userId: "5a6f12ab82e2ba"
fname: "Akash"
lname: "Muthe"
email: "akar@gmail.com"
mobile: 9657539700
password: "abc1234"
role: "user"
__v: 0
```

Models

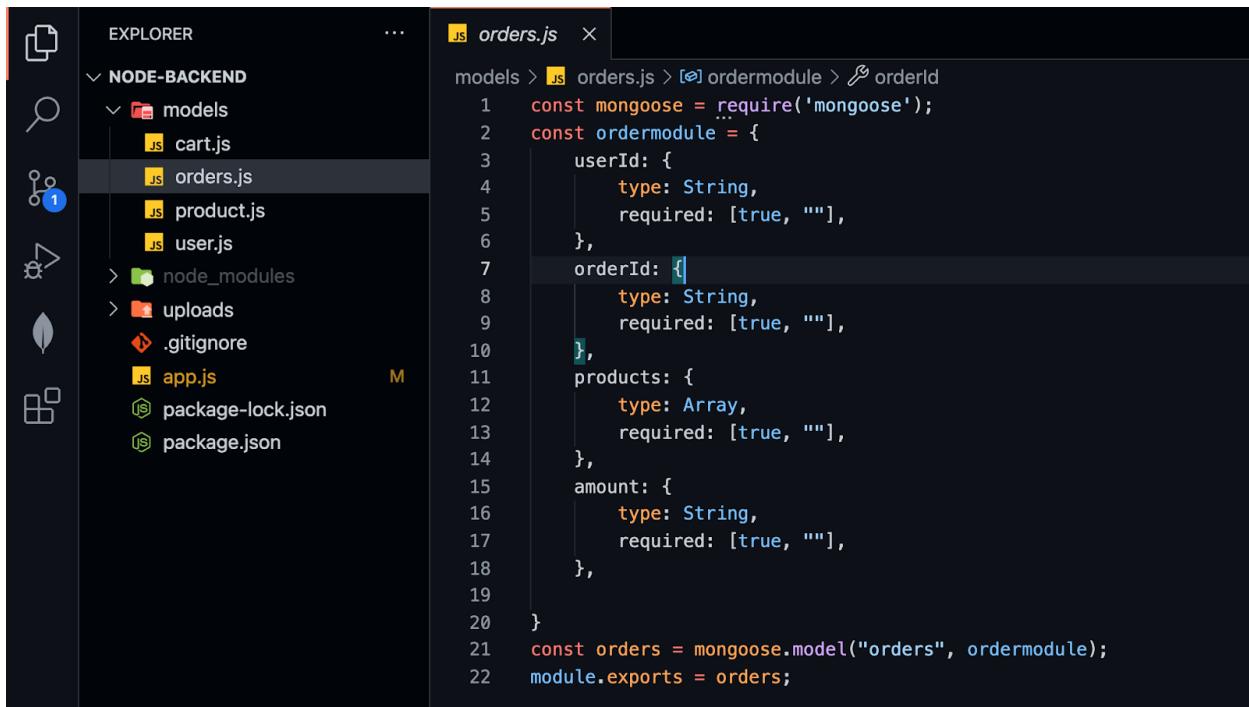
Cart Model



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'NODE-BACKEND' folder is expanded, and the 'models' folder contains four files: 'cart.js', 'orders.js', 'product.js', and 'user.js'. The 'cart.js' file is selected and open in the main editor area. The code defines a Mongoose model for a 'cart' document:

```
const mongoose = require('mongoose');
const cartmodel = {
  userId: {
    type: String,
    unique: true,
    required: [true, ""],
  },
  products: {
    type: Array,
  }
}
const cart = mongoose.model("cart", cartmodel)
module.exports = cart;
```

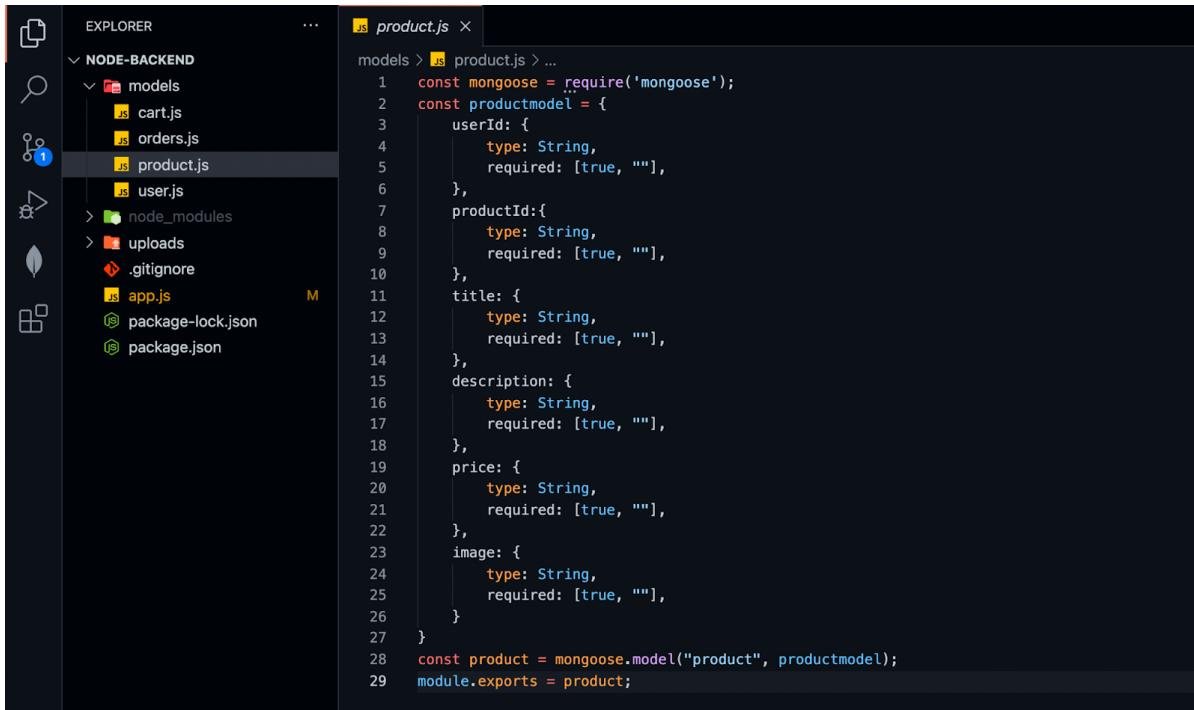
Orders Model



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'NODE-BACKEND' folder is expanded, and the 'models' folder contains four files: 'cart.js', 'orders.js', 'product.js', and 'user.js'. The 'orders.js' file is selected and open in the main editor area. The code defines a Mongoose model for an 'orders' document:

```
const mongoose = require('mongoose');
const ordermodule = {
  userId: {
    type: String,
    required: [true, ""],
  },
  orderId: {
    type: String,
    required: [true, ""],
  },
  products: {
    type: Array,
    required: [true, ""],
  },
  amount: {
    type: String,
    required: [true, ""],
  },
}
const orders = mongoose.model("orders", ordermodule);
module.exports = orders;
```

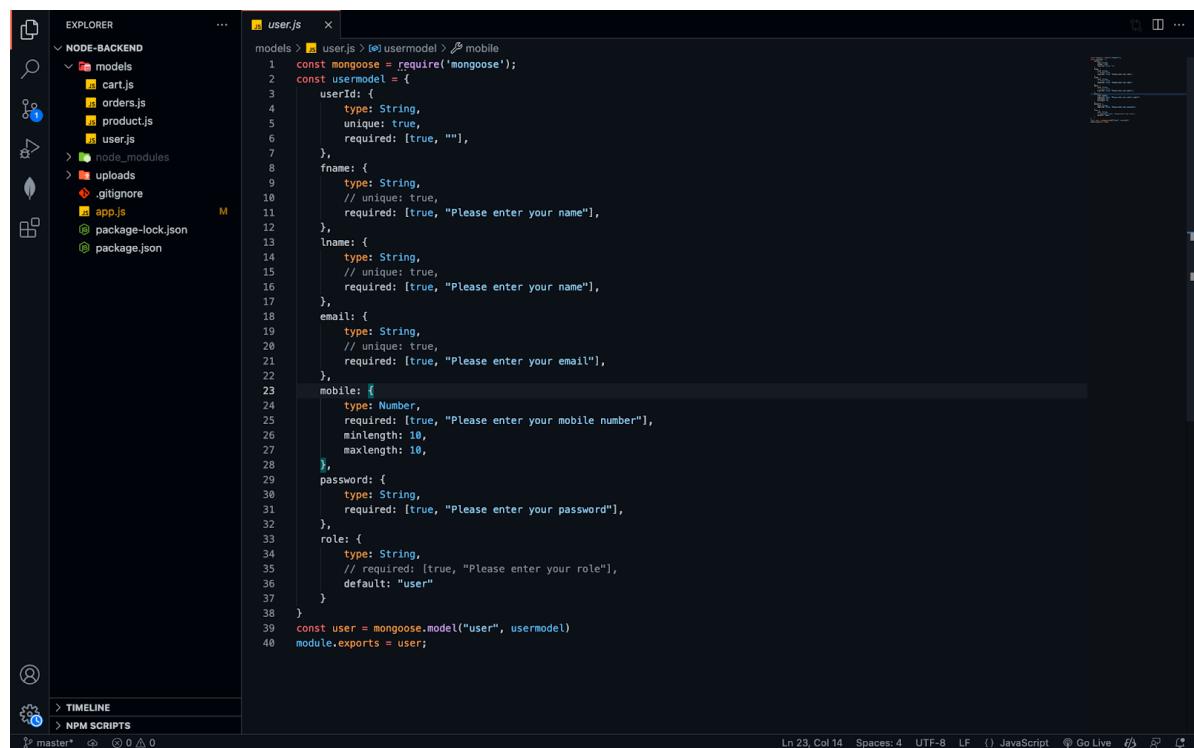
Products Model



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'NODE-BACKEND' folder is expanded, showing subfolders like 'models' which contains 'cart.js', 'orders.js', and 'product.js'. The 'product.js' file is selected and open in the main editor area. The code defines a Mongoose model for 'product' with fields: userId, productId, title, description, price, and image.

```
models > product.js > ...
1 const mongoose = require('mongoose');
2 const productmodel = {
3   userId: {
4     type: String,
5     required: [true, ""],
6   },
7   productId: {
8     type: String,
9     required: [true, ""],
10 },
11   title: {
12     type: String,
13     required: [true, ""],
14   },
15   description: {
16     type: String,
17     required: [true, ""],
18   },
19   price: {
20     type: String,
21     required: [true, ""],
22   },
23   image: {
24     type: String,
25     required: [true, ""],
26   }
27 }
28 const product = mongoose.model("product", productmodel);
29 module.exports = product;
```

User Model



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'NODE-BACKEND' folder is expanded, showing subfolders like 'models' which contains 'cart.js', 'orders.js', 'product.js', and 'user.js'. The 'user.js' file is selected and open in the main editor area. The code defines a Mongoose model for 'user' with fields: userId, name, lname, email, mobile, password, and role.

```
models > user.js > usermodel > mobile
1 const mongoose = require('mongoose');
2 const usermodel = {
3   userId: {
4     type: String,
5     unique: true,
6     required: [true, ""],
7   },
8   name: {
9     type: String,
10    // unique: true,
11    required: [true, "Please enter your name"],
12   },
13   lname: {
14     type: String,
15     // unique: true,
16     required: [true, "Please enter your name"],
17   },
18   email: {
19     type: String,
20     // unique: true,
21     required: [true, "Please enter your email"],
22   },
23   mobile: {
24     type: Number,
25     required: [true, "Please enter your mobile number"],
26     minlength: 10,
27     maxlength: 10,
28   },
29   password: {
30     type: String,
31     required: [true, "Please enter your password"],
32   },
33   role: {
34     type: String,
35     // required: [true, "Please enter your role"],
36     default: "user"
37   }
38 }
39 const user = mongoose.model("user", usermodel)
40 module.exports = user;
```

RESULTS & CONCLUSIONS

- Successfully completed implementation of a MongoDB + Express JS + Angular + Node JS (MEAN Stack) Application.
- Role based authentication is handled with Role Guards.
- Implemented of a modal window to edit products
- Gained in-depth knowledge about the working of an ecommerce store.

CODE

GITHUB LINK : <https://github.com/aakar-mutha/Zing-Ecommerce>