E-commerce Application on IBM cloud foundry

phase - 4 [Development part 2]

Done By : Sri Nivetha . S

Building an eCommerce platform with Nodejs as the backend and for the front end, we will have 3 different technologies (Angular, React, and Vujs). We will make a shopping cart in Nodejs.

Steps for Building a Shopping Cart in Nodejs :

We will break down this article into two parts, the backend and the frontend . Our application will have basic features like adding products and adding products to the cart.

Prerequisites :

Familiarity with HTML, CSS, and Javascript (ES6+).
Vs code or any code editor installed on your development machine.
POSTMAN is installed on your development machine.
Basic knowledge of Reactjs and Expressjs.
We will start by setting up the backend for our application . Let's create a new directory for our application and initialize a new nodejs application . Open up your terminal and type the following

```
cd desktop
mkdir reactcart && cd reactcart
npm init -y
code .
```

Installing the necessary packages :

We will have to install some packages for our application

body-parser: is a piece of express middleware that reads a form's input and stores it as a javascript object accessible through req.body.

nodemon : Will watch our files for any changes and then restarts the server when any change occurs.
express will be used to build our nodejs server.

cors : is a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin .

dotenv : will store all of our environment variables. This is where we will store our email variables.

morgan: is a package that will log all our application routes.

mongoose: is an object modeling tool used to asynchronous query mongoDB.

multer: is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files.

To install this packages open your terminal an

```
npm i express mongoose morgan dotenv multer body-parser cors nodemon --save
```

Running this command will create a node_modules folder. You have to create a .gitignore file and add the node_modules file inside it.

Setting up the server :

We will continue by creating an src/index.js file and add the following lines of code:

```
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const morgan = require('morgan');
const app = express();
app.use(morgan('dev'));
app.use(cors());
app.use(bodyParser.json())
app.get('/', (req, res) => {
    res.json({
        message: 'Arise MERN developers'
    });
});
const port = process.env.PORT|| 4000;
app.listen(port, () => {
    console.log(`Application is running on ${port}`);
});
```

After adding this, We can run our application using nodemon by typing nodemon src in our terminal. Running this will output Application is running on 4000. Now that our server is running, We have to set up our mongoDB server.

To do this create a new directory src/config and create a mongoose.js file and add the following codes:

```
const mongoose = require("mongoose");
module.exports = app => {
    mongoose.connect('mongodb://localhost:27017/cart', {
        useUnifiedTopology: true,
        useNewUrlParser: true,
        useFindAndModify: false
    }).then(res => console.log("conneceted")).catch(err => console
.log(err))
    mongoose.Promise = global.Promise;
    process.on("SIGINT", cleanup);
    process.on("SIGTERM", cleanup);
    process.on("SIGHUP", cleanup);
    if (app) {
        app.set("mongoose", mongoose);
    }
};
function cleanup() {
    mongoose.connection.close(function () {
        process.exit(0);
    });
}
```

Now we need to register this config in our index.js file:

```
require("./config/mongoose.js")(app);
```

Adding this will connect to our database whenever our Nodejs server is running.

Note that you have to declare this after you have declared the instance of express.

We now have to create our mongoDB models and routes for our products and cart.

Create an src/app directory, This is where we will be creating our modules. Inside this directory, Create a product directory and add the following file:

model.js
controller.js
repository.js
route.js
It's also a good idea to take all DB communications to the repository file.

Let's define our product model by adding this to our model.js file:

```
const mongoose = require("mongoose");
const productSchema = mongoose.Schema({
  name: {
    type: String,
    required: [true, "Please include the product name"],
  },
  price: {
    type: String,
    required: [true, "Please include the product price"],
  },
 image: {
    type: String,
    required: true,
  },
});
const Product = mongoose.model("Product", productSchema);
module.exports = Product;
```

Our product model will be basic as possible as it holds the product name, price, and image.

We now need to define our DB requests in our repository.js file:

```
const Product = require("./model");
exports.products = async () => {
    const products = await Product.find();
    return products;
};
exports.productById = async id => {
    const product = await Product.findById(id);
    return product;
}
exports.createProduct = async payload => {
    const newProduct = await Product.create(payload);
    return newProduct
}
exports.removeProduct = async id => {
    const product = await Product.findByIdAndRemove(id);
    return product
}
```

We need to define our basic routes to get all products, get single product details, remove products and create products. The logic is the routes will be talking to our controllers and the controller talks to the repository and the repository talks to our model.

Before we define our routes we need to configure multer for our image upload.Create a multer.js file and add the following code:

```
const multer = require("multer");
const path = require("path");
//image upload
const storage = multer.diskStorage({
    destination: (req, res, cb) => {
        cb(null, path.join("./files/"));
    },
    filename: (req, file, cb) => {
        cb(null, new Date().toISOString() + file.originalname);
    }
});
//  checking file type
const fileFilter = (req, file, cb) => {
    if (file.mimetype.startsWith('image')) {
        cb(null, true);
    } else {
        cb(new Error('Not an image! Please upload an image.', 400),
false);
    }
};
exports.upload = multer({
    storage: storage,
    limits: {
        fileSize: 1024 * 1024 * 6
    },
    fileFilter: fileFilter
});
```

create a files directory in the root of your application. This is where all uploaded images will
be stored.

Since all images go to the files directory, We have to make that files folder. To do this head over to the
index.js file and add this:

```
const router = require("express").Router();
const productController = require("./controller");
const multerInstance = require('../../config/multer')
router.post("/", multerInstance.upload.single('image'),
productController.createProduct);
router.get("/", productController.getProducts);
router.get("/:id", productController.getProductById);
router.delete("/:id", productController.removeProduct);
module.exports = router;
```

We now have to define the methods for these routes. To do that create
add this to the controller.
js file:

```
const productRepository = require('./repository')
exports.createProduct = async (req, res) => {
    try {
        let payload = {
            name: req.body.name,
            price: req.body.price,
            image: req.file.path
        }
        let product = await productRepository.createProduct({
            ...payload
        });
        res.status(200).json({
            status: true,
            data: product,
        })
```

```
    } catch (err) {
        console.log(err)
        res.status(500).json({
            error: err,
            status: false,
        })
    }
}
exports.getProducts = async (req, res) => {
    try {
        let products = await productRepository.products();
        res.status(200).json({
            status: true,
            data: products,
        })
    } catch (err) {
        console.log(err)
        res.status(500).json({
            error: err,
            status: false,
        })
    }
}
```

Create a routerHandler.js file inside the src directory, This will be our global routes handler:

```
const productRoutes = require("./Product/routes")
module.exports = app => {
    app.use("/product", productRoutes);
}
```

Then register it in the index.js file. Make sure to register this file after the mongoose instance.

```
require('./app/routeHandler')(app)
```

Testing our routes :

We can now start working on our cart features. Create a new directory Cart inside the src/app directory. Just like we did for the products module we will define the model, routes, repository, and controller files.

Let's start by defining our cart models:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
let ItemSchema = new Schema({
    productId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Product",
    },
    quantity: {
        type: Number,
        required: true,
        min: [1, 'Quantity can not be less then 1.']
    },
```

```
    price: {
        type: Number,
        required: true
    },
    total : {
        type: Number,
        required: true,
    }
}, {
    timestamps : true
})
const CartSchema = new Schema({
    items : [ItemSchema],
    subTotal : {
        default : 0,
        type: Number
    }
}, {
    timestamps : true
})

module.exports = mongoose.model('cart', CartSchema);
```

Here we create our first schema to hold the instance of our current product and then create the second file which will hold the array of items in our cart.

Now we have to define our repository.js file:

```
const Cart = require("./model");
exports.cart = async () => {
    const carts = await Cart.find().populate({
        path: "items.productId",
        select: "name price total"
    });;
    return carts[0];
};
exports.addItem = async payload => {
    const newItem = await Cart.create(payload);
    return newItem
}
```

Basically, we write two methods that will get all cart items in our database and add an item to the cart model.

We can now create our controllers for our cart, We will have 3 controllers:

Get all cart items
Add product items to the cart
Empty cart

```
    const cartRepository = require('./repository')
    const productRepository = require('../Product/repository');

    exports.addItemToCart = async (req, res) => {
        const {
            productId
        } = req.body;
        const quantity = Number.parseInt(req.body.quantity);
        try {
            let cart = await cartRepository.cart();
            let productDetails = await productRepository.productById(
productId);
                if (!productDetails) {
                return
```

```
res.status(500).json({
                type: "Not Found",
                msg: "Invalid request"
            })
        }
        //-- If Cart Exists ----
        if (cart) {
            //---- Check if index exists ----
            const indexFound = cart.items.findIndex(item => item.productId.id ==
productId);
            //------This removes an item from the the cart if the quantity is set to
zero, We can use this method to remove an item from the list  -------
            if (indexFound !== -1 && quantity <= 0) {
                cart.items.splice(indexFound, 1);
                if (cart.items.length == 0) {
                    cart.subTotal = 0;
                } else {
                    cart.subTotal = cart.items.map(item => item.total).reduce((acc,
next) => acc + next);
                }
            }
            //---------Check if product exist, just add the previous quantity with the
new quantity and update the total price-------
            else if (indexFound !== -1) {
                cart.items[indexFound].quantity = cart.items[indexFound].quantity +
quantity;
                cart.items[indexFound].total = cart.items[indexFound].quantity *


productDetails.price;
                cart.items[indexFound].price = productDetails.price
                cart.subTotal = cart.items.map(item => item.total).reduce((acc, next)
=> acc + next);
            }//----Check if quantity is greater than 0 then add item to items array ----
            else if (quantity > 0) {
                cart.items.push({
                    productId: productId,
                    quantity: quantity,
                    price: productDetails.price,
                    total: parseInt(productDetails.price * quantity)
                })
```

```
            cart.subTotal = cart.items.map(item => item.total).reduce((acc, next)
=> acc + next);
        }
        //---If quantity of price is 0 throw the error -------
        else {
            return res.status(400).json({
                type: "Invalid",
                msg: "Invalid request"
            })
        }
        let data = await cart.save();
        res.status(200).json({
            type: "success",
            mgs: "Process successful",
            data: data
        })
    }
    //------------This creates a new cart and then adds the item to the cart that
has been created------------
    else {
        const cartData = {
            items: [{
                productId: productId,
                quantity: quantity,
                total: parseInt(productDetails.price * quantity),
                price: productDetails.price
            }],
            subTotal: parseInt(productDetails.price * quantity)
        }
        cart = await cartRepository.addItem(cartData)
        //  let data = await cart.save();
        res.json(cart);
    }
} catch (err) {
    console.log(err)
    res.status(400).json({
        type: "Invalid",
        msg: "Something went wrong",
        err: err
    })
}
```

```
}
exports.getCart = async (req, res) => {
    try {
        let cart = await cartRepository.cart()
        if (!cart) {
            return res.status(400).json({
                type: "Invalid",
                msg: "Cart not Found",
            })
        }
        res.status(200).json({
            status: true,
            data: cart
        })
    } catch (err) {
        console.log(err)
        res.status(400).json({
            type: "Invalid",
            msg: "Something went wrong",
            err: err
        })
    }
}

exports.emptyCart = async (req, res) => {
    try {
        let cart = await cartRepository.cart();
        cart.items = [];
        cart.subTotal = 0
        let data = await cart.save();
        res.status(200).json({
            type: "success",
            mgs: "Cart has been emptied",
            data: data
        })
    } catch (err) {
        console.log(err)
        res.status(400).json({
            type: "Invalid",
            msg: "Something went wrong",
            err: err
        })
    }
```

Now we have to define our repository.js file:

```
const Cart = require("./model");
exports.cart = async () => {
    const carts = await Cart.find().populate({
        path: "items.productId",
        select: "name price total"
    });;
    return carts[0];
};
exports.addItem = async payload => {
    const newItem = await Cart.create(payload);
    return newItem
}
```

Basically, we write two methods that will get all cart items in our database and add an item to the cart model.

We can now create our controllers for our cart, We will have 3 controllers:

Get all cart items
Add product items to the cart
Empty cart

```
    const cartRepository = require('./repository')
    const productRepository = require('../Product/repository');

    exports.addItemToCart = async (req, res) => {
        const {
            productId
        } = req.body;
        const quantity = Number.parseInt(req.body.quantity);
        try {
            let cart = await cartRepository.cart();
            let productDetails = await productRepository.productById(
productId);
                if (!productDetails) {
                return res.status(500).json({
                    type: "Not Found",
                    msg: "Invalid request"
                })
```

```
        if (cart) {
            //----Check if index exists ----
            const indexFound = cart.items.findIndex(item => item.productId.id ==
productId);
            //------This removes an item from the the cart if the quantity is set to
zero, We can use this method to remove an item from the list  -------
            if (indexFound !== -1
&& quantity <= 0) {
                cart.items.splice(indexFound, 1);
                if (cart.items.length == 0) {
                    cart.subTotal = 0;
                } else {
                    cart.subTotal = cart.items.map(item => item.total).reduce((acc,
next) => acc + next);
                }
            }
            //--------Check if product exist, just add the previous quantity with the
new quantity and update the total price-------
            else if (indexFound !== -1) {
                cart.items[indexFound].quantity = cart.items[indexFound].quantity +
quantity;
                cart.items[indexFound].total = cart.items[indexFound].quantity *
productDetails.price;
                cart.items[indexFound].price = productDetails.price
                cart.subTotal = cart.items.map(item => item.total).reduce((acc, next)
=> acc + next);
            }
            //----Check if quantity is greater than 0 then add item to items array ----
            else if (quantity > 0) {
                cart.items.push({
                    productId: productId,
                    quantity: quantity,
                    price: productDetails.price,
                    total: parseInt(productDetails.price * quantity)
                })
                cart.subTotal = cart.items.map(item => item.total).reduce((acc, next)
=> acc + next);
            }
            //----If quantity of price is 0 throw the error -------
            else {
                return res.status(400).jso
```

```
n({
                type: "Invalid",
                msg: "Invalid request"
            })

        }
        let data = await cart.save();
        res.status(200).json({
            type: "success",
            mgs: "Process successful",
            data: data
        })
    }
    //-----------This creates a new cart and then adds the item to the cart that has been created
------------
    else {
        const cartData = {
            items: [{
                productId: productId,
                quantity: quantity,
                total: parseInt(productDetails.price * quantity),
                price: productDetails.price
            }],
            subTotal: parseInt(productDetails.price * quantity)
        }
        cart = await cartRepository.addItem(cartData)
        //  let data = await
cart.save();
        res.json(cart);
    }
} catch (err) {
    console.log(err)
    res.status(400).json({
        type: "Invalid",
        msg: "Something went wrong",
        err: err
    })
}
}
```

```
exports.getCart = async (req, res) => {
    try {
        let cart = await cartRepository.cart()
        if (!cart) {
            return res.status(400).json({
                type: "Invalid",
                msg: "Cart not Found",
            })
        }
        res.status(200).json({
            status: true,
            data: cart
        })
    } catch (err) {
        console.log(err)
        res.status(400).json({
            type: "Invalid",
            msg: "Something went wrong",
            err: err
        })
    }
}

exports.emptyCart = async (req, res) => {
    try {
        let cart = await cartRepository.cart();
        cart.items = [];
        cart.subTotal = 0
        let data = await cart.save();
        res.status(200).json({
            type: "success",
            mgs: "Cart has been emptied",
            data: data
        })
    } catch (err) {
        console.log(err)
        res.status(400).json({
            type: "Invalid",
            msg: "Something went wrong",
            err: err
        })
    }
}
```

The code snippet has been commented on for ease and understanding.

We can now define our module routes and then define the global routes. Add this to the routes.js file:

```
const router = require("express").Router();
const cartController = require("./controller");
router.post("/", cartController.addItemToCart);
router.get("/", cartController.getCart);
router.delete("/empty-cart", cartController.emptyCart);
module.exports = router;
```

And then update the routeHandler.js file to this:

```
const productRoutes = require("./Product/routes");
const cartRoutes = require('./Cart/routes')
module.exports = app => {
    app.use("/product", productRoutes);
    app.use("/cart", cartRoutes);
```

Testing the cart features :

Adding Item to Cart

Get cart items

Empty card

For testing purposes, Create some products using POSTMAN We will be using this in our Frontend Application for testing purposes.

Thank you !