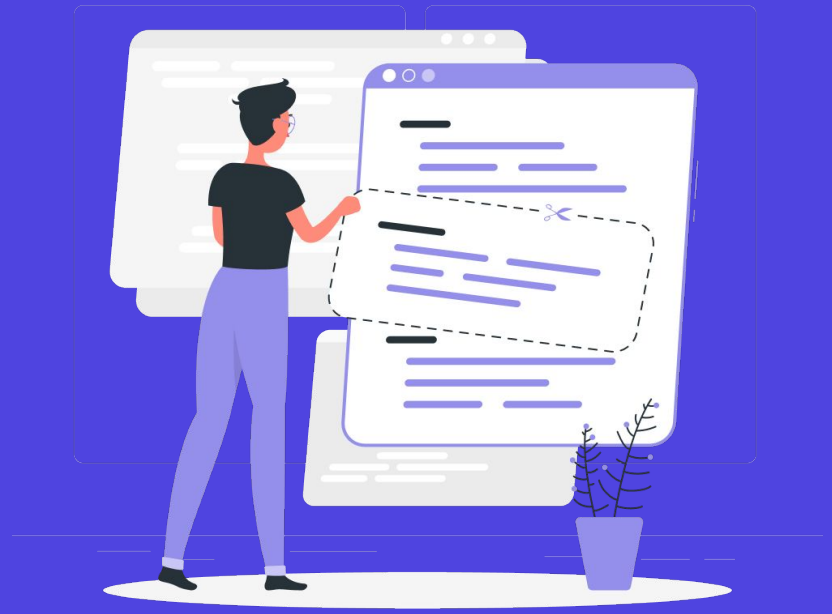
Creating Authenticated Cart APIs



Relevel
by Unacademy

Class Agenda

- We will try to understand the challenges with the current APIs from the security perspective.
- We will then process to make the category and product APIs use the JWT tokens so that only authenticated ADMIN user should be able to create/update/delete products and categories.
- We will be creating CART APIs.
- We will make sure that only authenticated User should be able to create a cart, add/remove items from the cart.

Educator Introduction

Challenges of the unauthenticated APIs

- No control on who calls the APIs
- Inadequate validations
- Lack of security
- No control on customer overusing the APIs

Currently available APIs

Category and Product API :

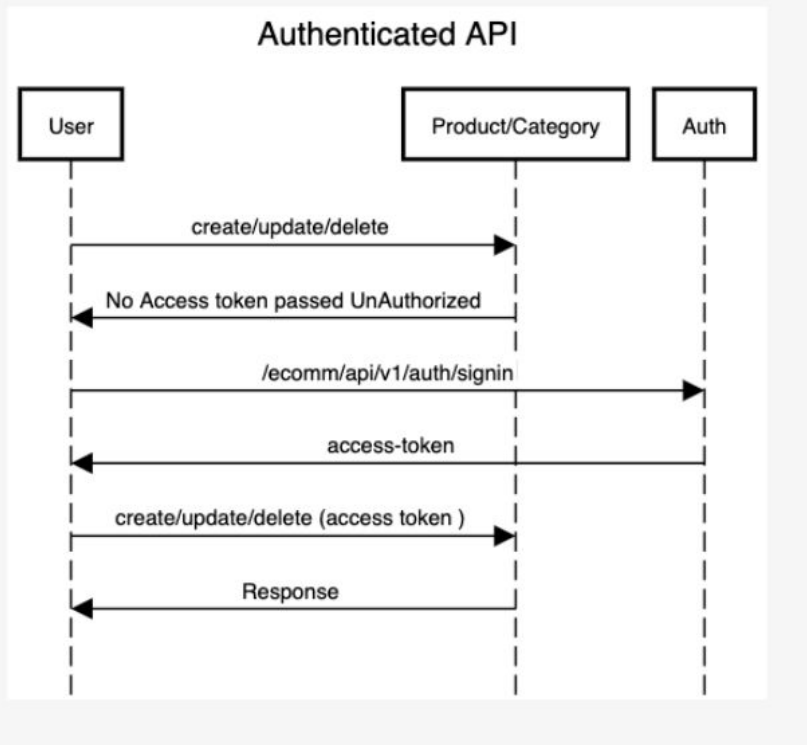
- As an eCommerce company, we would like to keep open the read APIs (GET) for Categories and Products. But we can’t allow anyone to create/update/delete these resources.
- We need to provide APIs for create/update/delete, but it should be done only by the authenticated and authorized user.
- So we need to authenticate and authorize the APIs for creating/updating/deleting the category and product resource.

Currently available APIs

Implementing authentication in these APIs

1. Any client trying to make a call to these authenticated APIs need to first access the token.
2. Access token can be achieved by making a login API call by the authenticated user
3. Based on the access token passed, the system needs to infer the type of the user. If the user has the privilege of ADMIN user, their request will be honored, else the API should

Currently available APIs



Implementation

Let us now begin to apply those authentication and authorization in our project.

Implementation

Create a middleware to check the token - authentication + authorization

https://github.com/Vishwa07dev/eCommerce/blob/session6/middlewares/authjwt.js

```
const jwt = require("jsonwebtoken");
const config = require("../configs/auth.config");
const db = require("../models");
const User = db.user;

verifyToken = (req, res, next) => {
   let token = req.headers["x-access-token"];
    if (!token) {
     return res.status(403).send({
       message: "No token provided!"
     });
   }
    jwt.verify(token, config.secret, (err, decoded) => {
     if (err) {
       return res.status(401).send({
         message: "Unauthorized!"
       });
     }
     req.userId = decoded.id;
     next();
   });
 };
```

Implementation

```
isAdmin = (req, res, next) => {
   User.findByPk(req.userId).then(user => {
     user.getRoles().then(roles => {
       for (let i = 0; i < roles.length; i++) {
         if (roles[i].name === "admin") {
           next();
           return;
         }
       }
       res.status(403).send({
         message: "Require Admin Role!"
       });
       return;
     });
   });
 };
 const authJwt = {
   verifyToken: verifyToken,
   isAdmin: isAdmin
 };
 module.exports = authJwt;
```

Implementation

Apply these middleware in the routes defined

https://github.com/Vishwa07dev/eCommerce/blob/session6/routes/product.routes.js
https://github.com/Vishwa07dev/eCommerce/blob/session6/routes/category.routes.js

Product route

```
/**
* This file will contain the routes logic for the Product resource
* and will export it.
*/

const productController = require("../controllers/product.controller")
const { authJwt, requestValidator } = require("../middlewares");

module.exports = function(app){

    //Route for the POST request to create the product

app.post("/ecomm/api/v1/products",[requestValidator.validateProductRequest,
authJwt.verifyToken,authJwt.isAdmin], productController.create);
    //Route for the GET request to fetch all the products
    app.get("/ecomm/api/v1/products", productController.findAll);
    //Route for the GET request to fetch a product based on the id
    app.get("/ecomm/api/v1/products/:id", productController.findOne);
    //Route for the PUT request to update a product based on the id

app.put("/ecomm/api/v1/products/:id",[requestValidator.validateProductReque
st, authJwt.verifyToken,authJwt.isAdmin], productController.update);
```

Implementation

```
    //Route for the DELETE request to delete a product based on the id

app.delete("/ecomm/api/v1/products/:id",[authJwt.verifyToken,authJwt.isAdmi
n], productController.delete);

    //Route for getting the list of products with cost greater than the

app.get("/ecomm/api/v1/categories/:categoryId/products",[requestValidator.v
alidateCategoryPassedInReqParam],
productController.getProductsUnderCategory);

    //Route for getting the list of products under a category

app.get("/ecomm/api/v1/categories/:categoryId/products",[requestValidator.v
alidateCategoryPassedInReqParam],
productController.getProductsUnderCategory);


}
```

Implementation

Category route

```
/**
* This file will contain the routes logic for the Category resource
* and will export it.
*/
const { authJwt, requestValidator } = require("../middlewares");

const categoryController = require("../controllers/category.controller")

module.exports = function(app){

   //Route for the POST request to create the category

app.post("/ecomm/api/v1/categories",[requestValidator.validateCategoryReque
st,authJwt.verifyToken,authJwt.isAdmin ], categoryController.create);

   //Route for the GET request to fetch all the categories
   app.get("/ecomm/api/v1/categories", categoryController.findAll);

   //Route for the GET request to fetch a category based on the id
```

Implementation

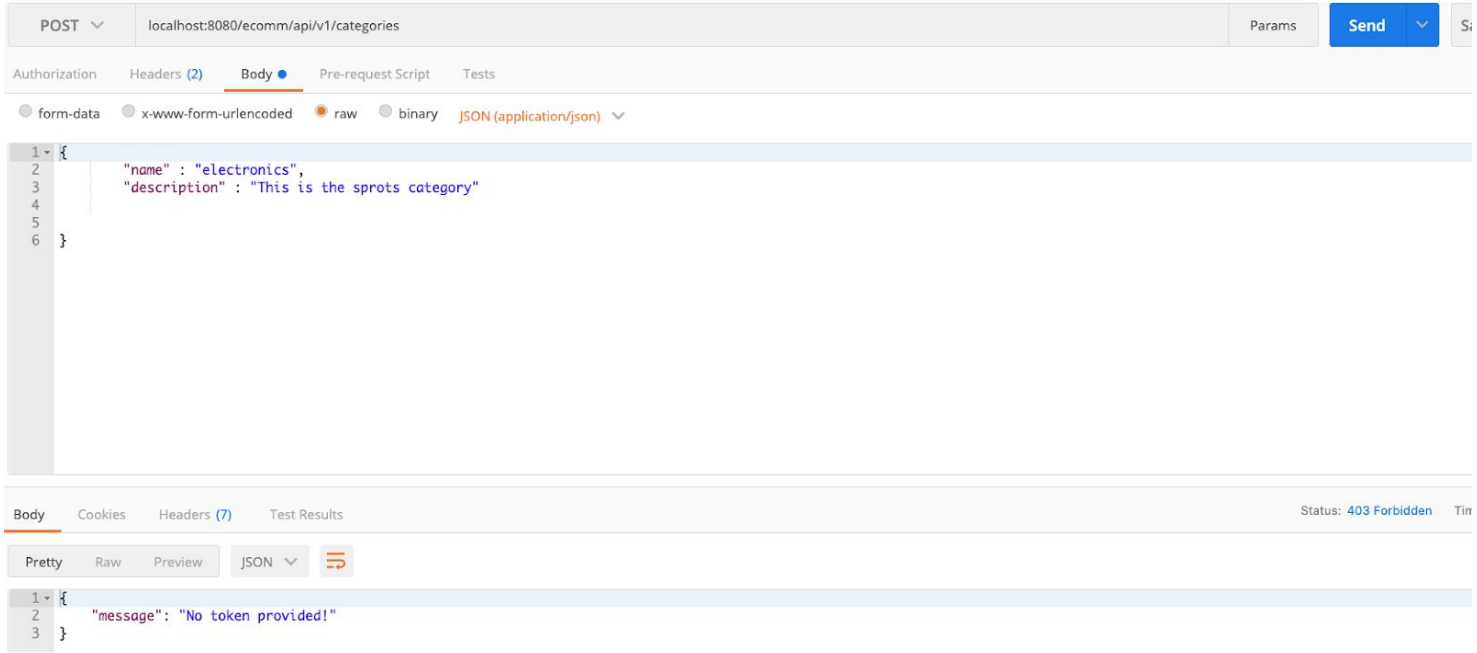
```
    app.get("/ecomm/api/v1/categories/:id", categoryController.findOne);

    //Route for the PUT request to update a category based on the id

app.put("/ecomm/api/v1/categories/:id",[requestValidator.validateCategoryRe
quest, authJwt.verifyToken,authJwt.isAdmin] ,categoryController.update);

    //Route for the DELETE request to delete a category based on the id

app.delete("/ecomm/api/v1/categories/:id",[authJwt.verifyToken,authJwt.isAd
min], categoryController.delete);
}
```

Testing APIs

Let us test the above authenticated endpoint using postman

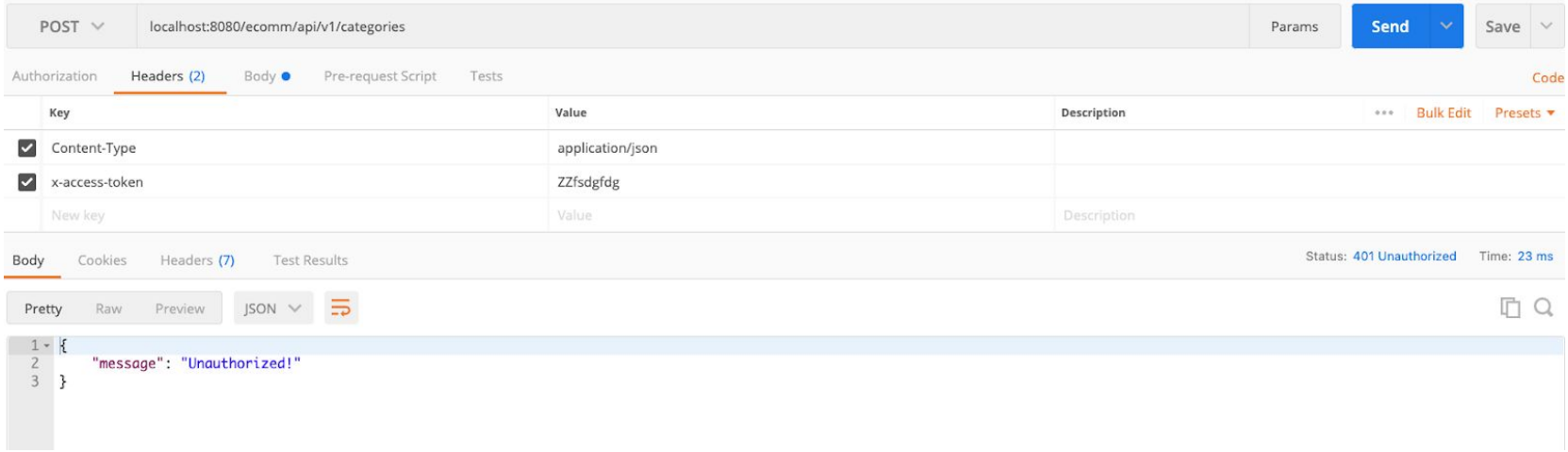
Testing APIs

Trying to create Category without passing access token



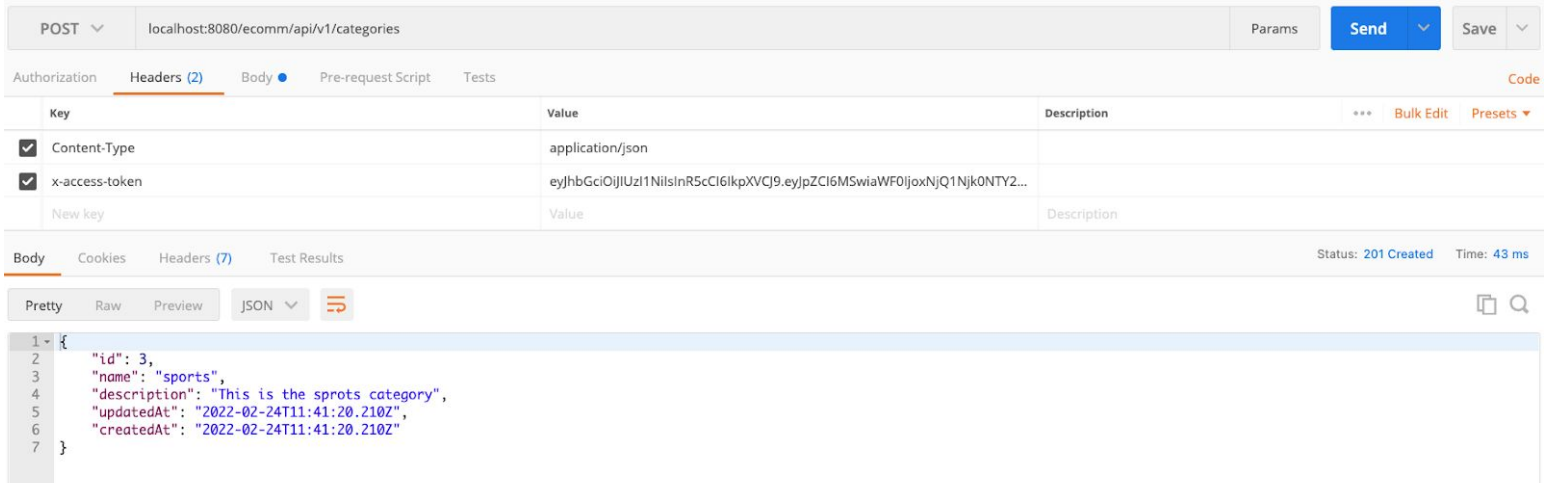
Testing APIs

Trying to create category with passing invalid access token



Testing APIs

Create Category after passing the valid token



Similarly we need to test if the Product APIs are properly authenticated and only user with ADMIN role is allowed to make changes.

Next set of features

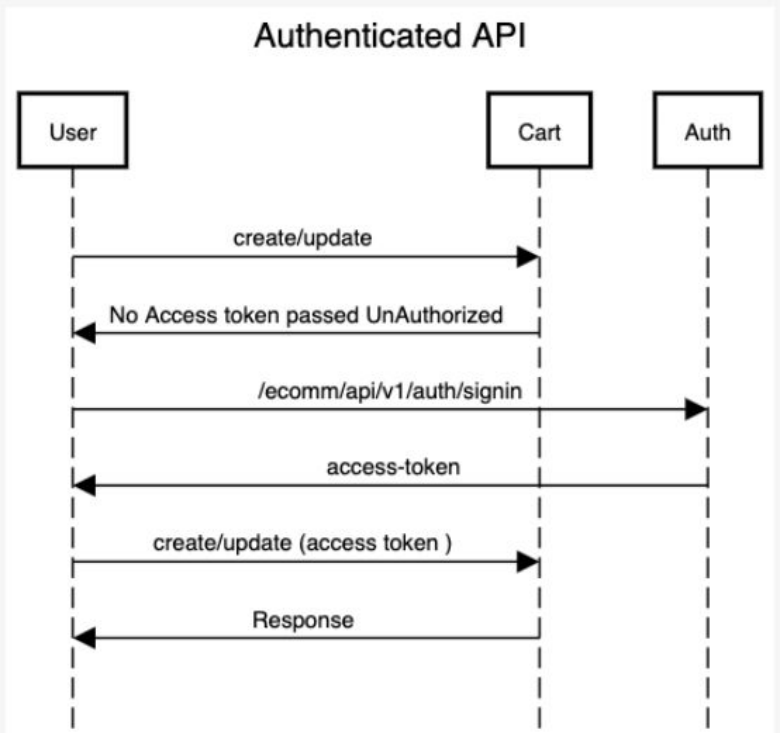
Let us move ahe.ad and add some new endpoints and implement new features.

Next set of features

Now since we have already authenticated the existing APIs with JWT token, let's add few more features to our eCommerce Application

- Authenticated user should be able to create a cart
- Authenticated user should be able to add products in the cart
- Authenticated user should be able to update products in the cart

Next set of features



Implementation :

Let us now start writing code for new feature.

Implementation :

Creation of the Cart model

https://github.com/Vishwa07dev/eCommerce/blob/session6/models/cart.model.js

```
module.exports = (sequelize, Sequelize) => {
    const Cart = sequelize.define("cart", {
        id: {
            type: Sequelize.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        cost:{
            type: Sequelize.INTEGER
        }

    });
    return Cart;
}
```

Implementation :

2. Establishing the relationship between Cart and Product

- Cart and Product has Many to Many relationship

```
/**
 * Establishing the relationship between Cart and Items : Many to Many
 */
 db.product.belongsToMany(db.cart, {
   through: "cart_products",
   foreignKey: "productId",
   otherKey: "cartId"
});
db.cart.belongsToMany(db.product, {
   through: "cart_products",
   foreignKey: "cartId",
   otherKey: "productId"
});
```

Implementation :

3. Writing the cart controller

https://github.com/Vishwa07dev/eCommerce/blob/session6/controllers/cart.controller.js

```
/**
* This file contains the controller logic for the cart resource.
* Everytime any CRUD request come for the cart, methods defined in this
* controller file will be executed.
*/

const { cart } = require("../models");
const db = require("../models");
const Product = db.product;
const Cart = db.cart;
const Op = db.Sequelize.Op;

/**
* Create and save a new Cart
*/
exports.create = (req, res) => {

   const cart = {
       userId: req.userId  // We will get this from the middleware
   };

   const itemIds = req.body.items;
   Cart.create(cart).then(cart => {
       res.status(201).send(cart);
   }).catch(err => {
       console.log(err.message);
       res.status(500).send({
           message: "Some internal server error happened"
```

Implementation :

```
    })
}

/**
* Update a given cart by adding more item to it
*/
exports.update = (req, res) => {
   const cartId = req.params.id;

   Cart.findByPk(cartId).then(cart => {
       console.log(cart);
       Product.findAll({
           where: {
               id: req.body.productIds
           }
       }).then(items => {
           if (!items) {
               res.status(400).send({
                   message: "item trying to be added doesn't exist"
               })
           }
           cart.setProducts(items).then(() => {
               console.log("Products successfully added in the cart");
               var cost = 0;
               const productsSelected = [];
               cart.getProducts().then(products => {
                   for (i = 0; i < products.length; i++) {
                       cost = cost + products[i].cost;
                       productsSelected.push({
                           id: products[i].id,
                           name: products[i].name,
                           cost: products[i].cost
                       });
                   }

                   res.status(200).send({
                       id: cart.id,
                       productsSelected: productsSelected,
                       cost: cost
```

Implementation :

```
        })
    })
  })
}

/**
* Controller to get the cart based on the cartId
*/
exports.getCart = (req, res) => {

  Cart.findByPk(req.params.cartId).then(cart => {
      var cost = 0;
      const productsSelected = [];
      cart.getProducts().then(products => {
          for (i = 0; i < products.length; i++) {
              cost = cost + products[i].cost;
              productsSelected.push({
                  id: products[i].id,
                  name: products[i].name,
                  cost: products[i].cost
              });
          }

          res.status(200).send({
              id: cart.id,
              productsSelected: productsSelected,
              cost: cost
          });
      });

  });

}
```

Implementation :

4. Writing the cart route and adding the authentication middleware
https://github.com/Vishwa07dev/eCommerce/blob/session6/routes/cart.routes.js

```
const orderController = require("../controllers/cart.controller")
const {authJwt, requestValidator } = require("../middlewares");

module.exports = function(app){

   //Route for the POST request to create the cart
   app.post("/ecomm/api/v1/carts",[authJwt.verifyToken],
orderController.create);

   //Route for the PUT request to create the product
   app.put("/ecomm/api/v1/carts/:id",[authJwt.verifyToken],
orderController.update);


   //Route for the GET request to get the product
   app.get("/ecomm/api/v1/carts/:cartId",[authJwt.verifyToken],
orderController.getCart);

}
```

Implementation :

5. Finally exposing the cart route through server.js

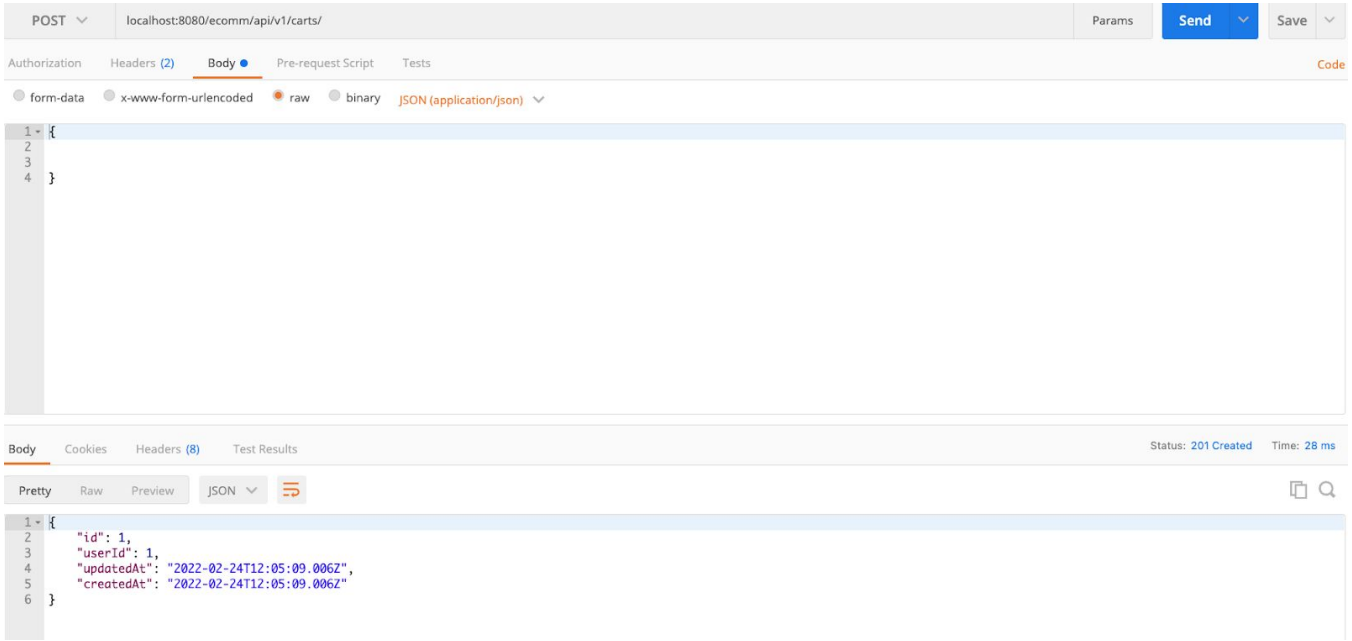
```
/**
* Importing the routes and using it
*/
require('./routes/category.routes')(app);
require('./routes/product.routes')(app);
require('./routes/auth.routes')(app);
require('./routes/cart.routes')(app);
```

Testing using POSTMAN

Let us now test our cart APIs using POSTMAN

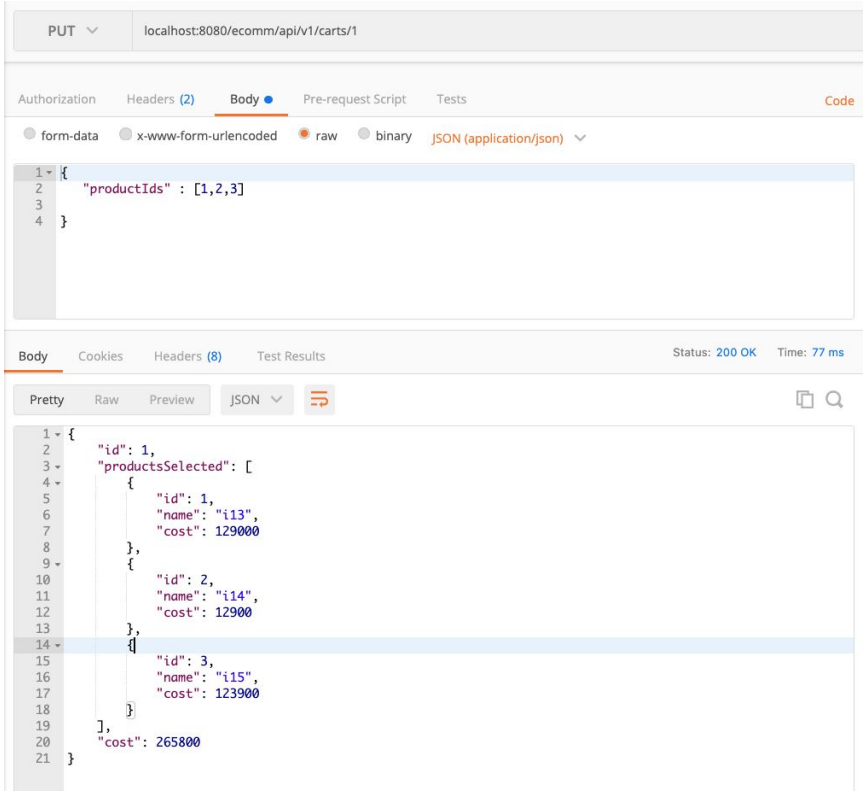
Testing using POSTMAN

1. Creating a cart



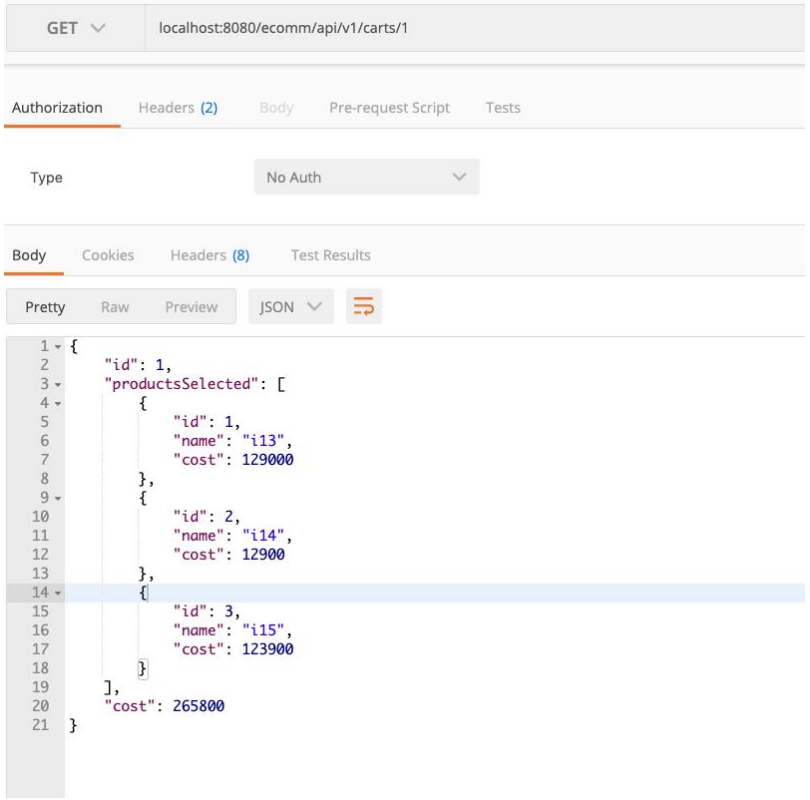
Testing using POSTMAN

2. Updating a cart/adding items to the cart



Testing using POSTMAN

3. Getting the details about a cart using cart id



With this we conclude the session. We saw a lot of authentication and authorization in the action in today's session.

MCQs

1. **How do we create a chainable route handler for any route path in ExpressJs?**
   A. Using app.route()
   B. Using app.routes()
   C. Using app.router()
   D. Using app.routing()

MCQs

1. **How do we create a chainable route handler for any route path in ExpressJs?**
   A. Using app.route()
   B. Using app.routes()
   C. Using app.router()
   D. Using app.routing()

Answer: A

MCQs

2. **Which is the correct middleware in ExpressJs?**
   A. function (req) { }
   B. method (req) { }
   C. function (req, res, next) { }
   D. method (req, res, next) { }

MCQs

2. **Which is the correct middleware in ExpressJs?**
    A. function (req) { }
    B. method (req) { }
    C. function (req, res, next) { }
    D. method (req, res, next) { }

Answer: C

MCQs

3. **Backlog arguments is defined as?**
   A. A port number on which the server should accept incoming requests.
   B. Name of the domain
   C. The maximum number of queued pending connections
   D. An asynchronous function that is called when the server starts listening for requests.

MCQs

3. **Backlog arguments is defined as?**
   A. A port number on which the server should accept incoming requests.
   B. Name of the domain
   C. The maximum number of queued pending connections
   D. An asynchronous function that is called when the server starts listening for requests.

Answer: C

MCQs

4. How do we initialize a model using sequelize using below codes

```
const sequelize = new Sequelize(my_db', 'root', '', {
host: 'localhost',
dialect: 'mysql'
});
```

A. const User = sequelize.define('user', { //model definition });
B. const User = sequelize.create('user', { //model definition });
C. const User = sequelize.add('user', { //model definition });
D. None of the Above

MCQs

4. How do we initialize a model using sequelize using below codes

```
const sequelize = new Sequelize(my_db', 'root', '', {
host: 'localhost',
dialect: 'mysql'
});
```

A. const User = sequelize.define('user', { //model definition });
B. const User = sequelize.create('user', { //model definition });
C. const User = sequelize.add('user', { //model definition });
D. None of the Above

Answer: A

MCQs

5. **What is the status code for Authorized access?**
   A. 401
   B. 400
   C. 404
   D. 402

MCQs

5. **What is the status code for Authorized access?**
   A. 401
   B. 400
   C. 404
   D. 402

Answer: A

Practice Problem

Let us continue with our mini project and add some more authentication which we learnt in today's class.

1. Modify table books by adding another column named user with value of user id to make only the owner of a book should only have access to modify it or delete it.
2. Similarly modify get all books routes to only fetch logged in user books only.

Next session

- What is testing
- What is Unit Testing
- What is Integration Testing
- What is Test Driven Development (TDD)
- Setup for writing the Unit Tests

Thank you!