SWDV:Capstone 691

Professor Gradecki

Maukan Mir

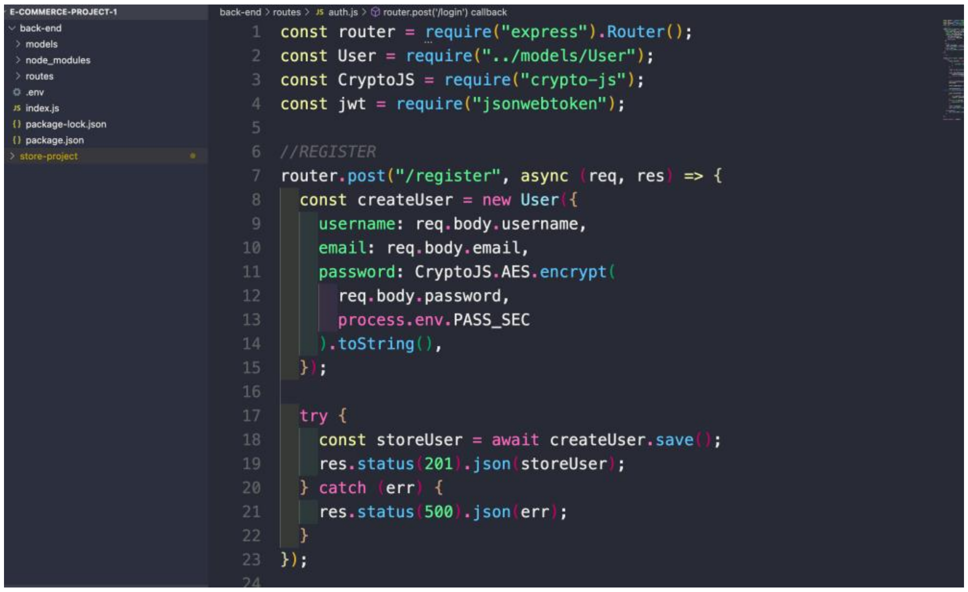
Re-Design Service Layers

I have broken up the service layers into sections. This way it becomes easier to debug each process when they have been broken down into smaller sections. This way errors become easier to mitigate, locate and address. For this process I will be using NodeJS and Express to create the applications RestAPI.

Service layer sections:

- Products
- Authentication
- Shopping Cart Information
- Order Information
- Payment Confirmation

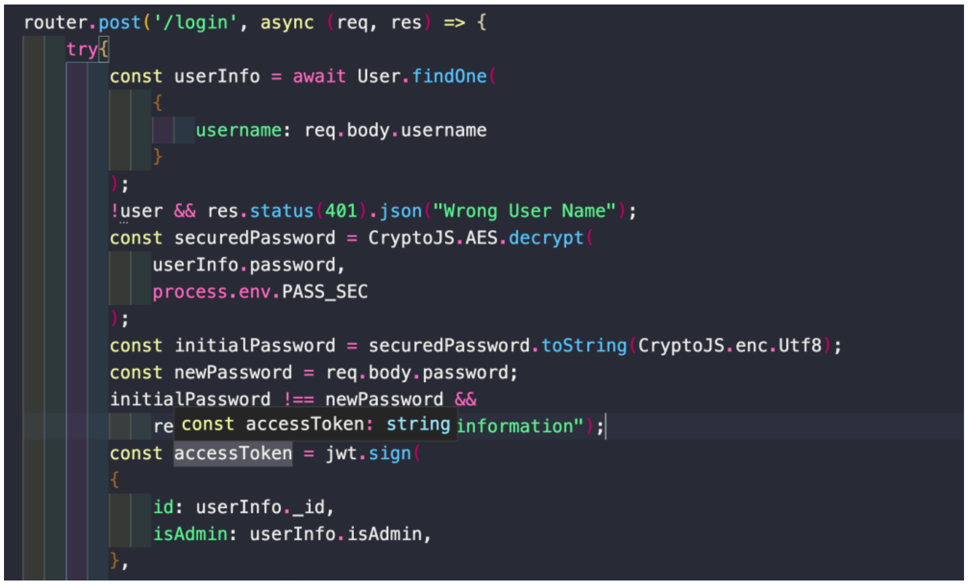
Register page:



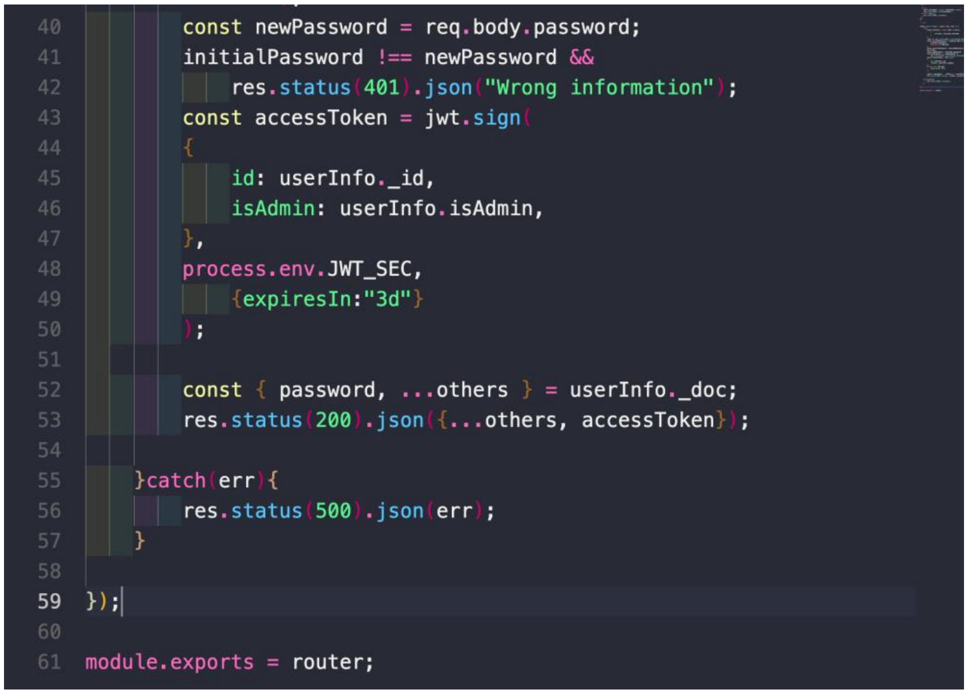
```javascript
const router = require("express").Router();
const User = require("../models/User");
const CryptoJS = require("crypto-js");
const jwt = require("jsonwebtoken");

//REGISTER
router.post("/register", async (req, res) => {
  const createUser = new User({
    username: req.body.username,
    email: req.body.email,
    password: CryptoJS.AES.encrypt(
      req.body.password,
      process.env.PASS_SEC
    ).toString(),
  });

  try {
    const storeUser = await createUser.save();
    res.status(201).json(storeUser);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

The register page will create a new user in the database, encrypt their password information and save the user into the database. This will save the username, email, password and datetime stamps.

Login page:



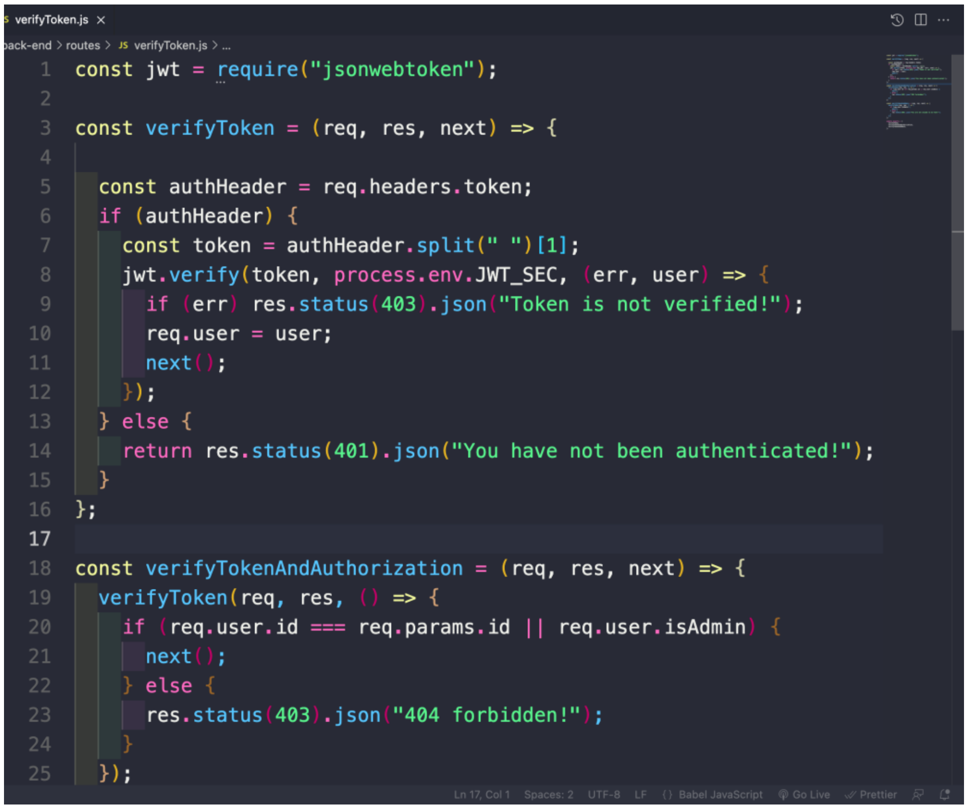
```
router.post('/login', async (req, res) => {
    try{
        const userInfo = await User.findOne(
            {
                username: req.body.username
            }
        );
        !user && res.status(401).json("Wrong User Name");
        const securedPassword = CryptoJS.AES.decrypt(
            userInfo.password,
            process.env.PASS_SEC
        );
        const initialPassword = securedPassword.toString(CryptoJS.enc.Utf8);
        const newPassword = req.body.password;
        initialPassword !== newPassword &&
            re const accessToken: string information");
        const accessToken = jwt.sign(
        {
            id: userInfo._id,
            isAdmin: userInfo.isAdmin,
        },
```



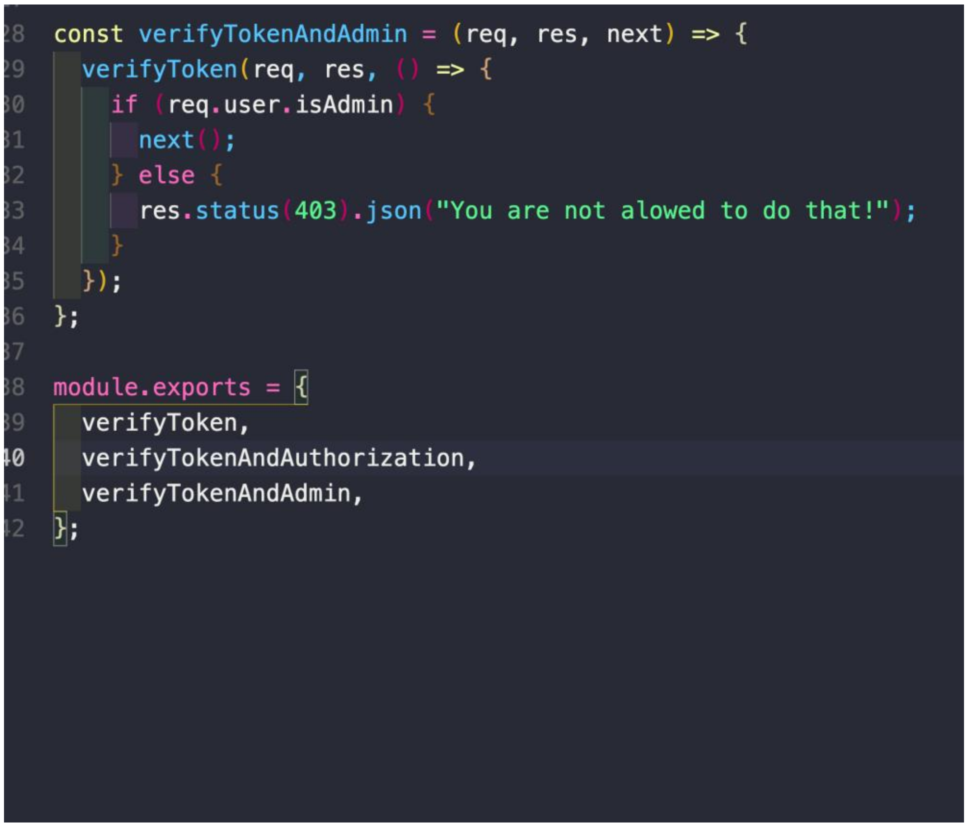
```
            const newPassword = req.body.password;
            initialPassword !== newPassword &&
                res.status(401).json("Wrong information");
            const accessToken = jwt.sign(
            {
                id: userInfo._id,
                isAdmin: userInfo.isAdmin,
            },
            process.env.JWT_SEC,
                {expiresIn:"3d"}
            );

            const { password, ...others } = userInfo._doc;
            res.status(200).json({...others, accessToken});

        }catch(err){
            res.status(500).json(err);
        }

});

module.exports = router;
```

The login page will do three things. It will create a access token that will not expire in three days. It will verify password information and it save. It will also not save password information to the database. It will create a hashed password using cryptoJS.

JWT Token Verification:


verifyToken.js
ack-end > routes > verifyToken.js > ...
Ln 17, Col 1 Spaces: 2 UTF-8 LF {} Babel JavaScript Go Live Prettier

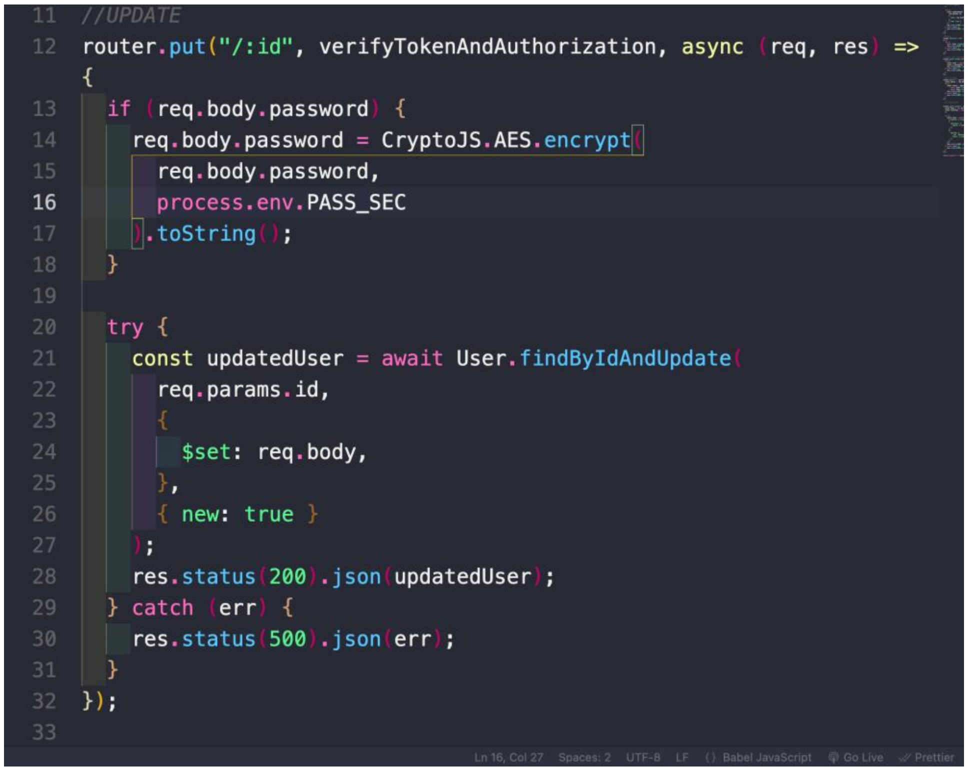
```javascript
const jwt = require("jsonwebtoken");

const verifyToken = (req, res, next) => {

  const authHeader = req.headers.token;
  if (authHeader) {
    const token = authHeader.split(" ")[1];
    jwt.verify(token, process.env.JWT_SEC, (err, user) => {
      if (err) res.status(403).json("Token is not verified!");
      req.user = user;
      next();
    });
  } else {
    return res.status(401).json("You have not been authenticated!");
  }
};

const verifyTokenAndAuthorization = (req, res, next) => {
  verifyToken(req, res, () => {
    if (req.user.id === req.params.id || req.user.isAdmin) {
      next();
    } else {
      res.status(403).json("404 forbidden!");
    }
  });
```



```
const verifyTokenAndAdmin = (req, res, next) => {
  verifyToken(req, res, () => {
    if (req.user.isAdmin) {
      next();
    } else {
      res.status(403).json("You are not alowed to do that!");
    }
  });
};

module.exports = {
  verifyToken,
  verifyTokenAndAuthorization,
  verifyTokenAndAdmin,
};
```

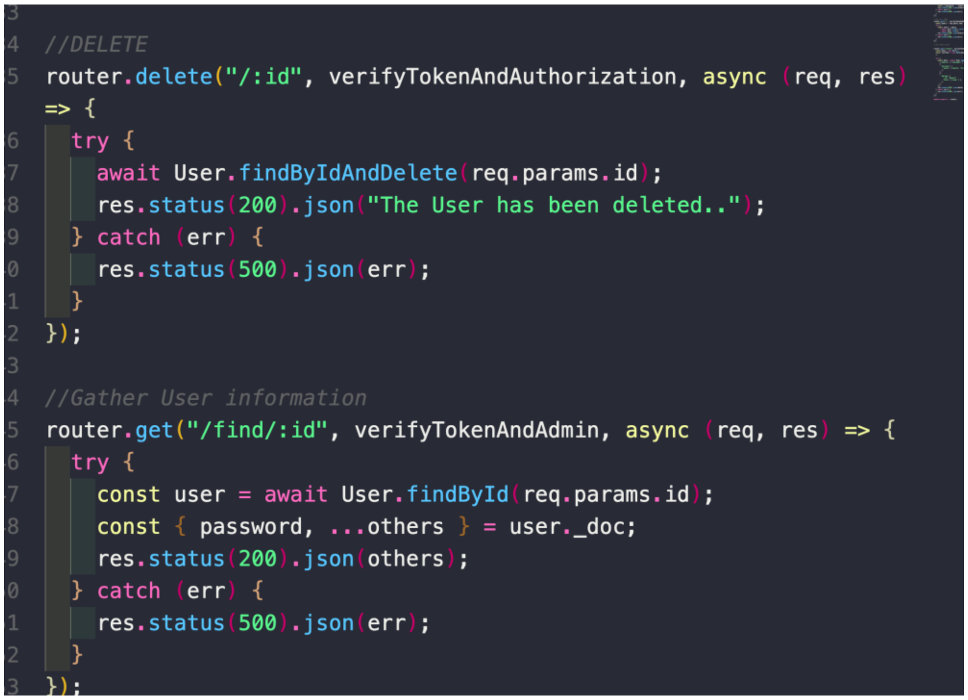
The JWT token will verify users and admin users. Allows them to change their cart and account information. It adds an extra layer of security, this way users cannot change other users account information without a verified token.

USER API:



```
//UPDATE
router.put("/:id", verifyTokenAndAuthorization, async (req, res) =>
{
  if (req.body.password) {
    req.body.password = CryptoJS.AES.encrypt(
      req.body.password,
      process.env.PASS_SEC
    ).toString();
  }

  try {
    const updatedUser = await User.findByIdAndUpdate(
      req.params.id,
      {
        $set: req.body,
      },
      { new: true }
    );
    res.status(200).json(updatedUser);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

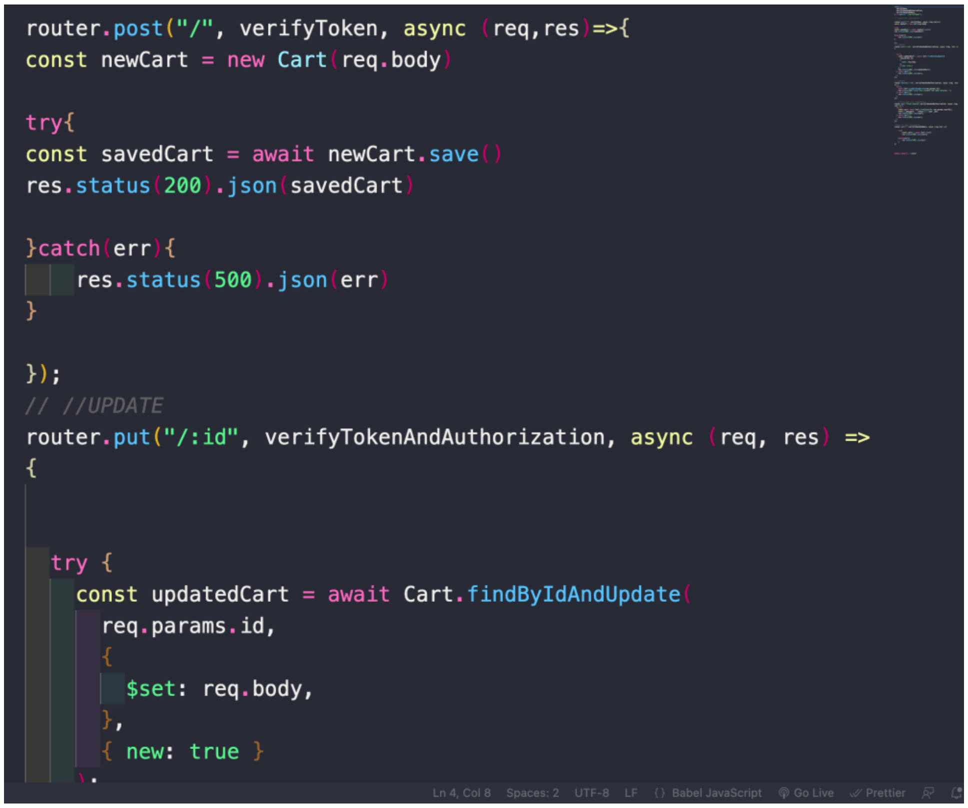
Ln 16, Col 27   Spaces: 2   UTF-8   LF   {} Babel JavaScript   Go Live   Prettier



```
//DELETE
router.delete("/:id", verifyTokenAndAuthorization, async (req, res)
=> {
  try {
    await User.findByIdAndDelete(req.params.id);
    res.status(200).json("The User has been deleted..");
  } catch (err) {
    res.status(500).json(err);
  }
});

//Gather User information
router.get("/find/:id", verifyTokenAndAdmin, async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    const { password, ...others } = user._doc;
    res.status(200).json(others);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

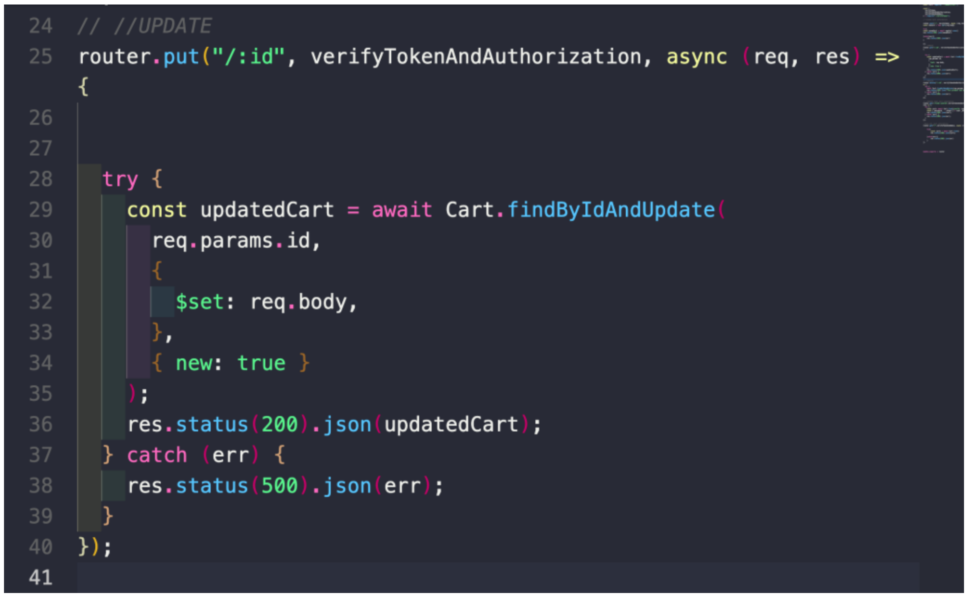
The User APi can update the users information, this way they can update their username and password information. The admin side can also find users by their ID. This makes it easier to track users information in the case of a mishap.

Cart API:

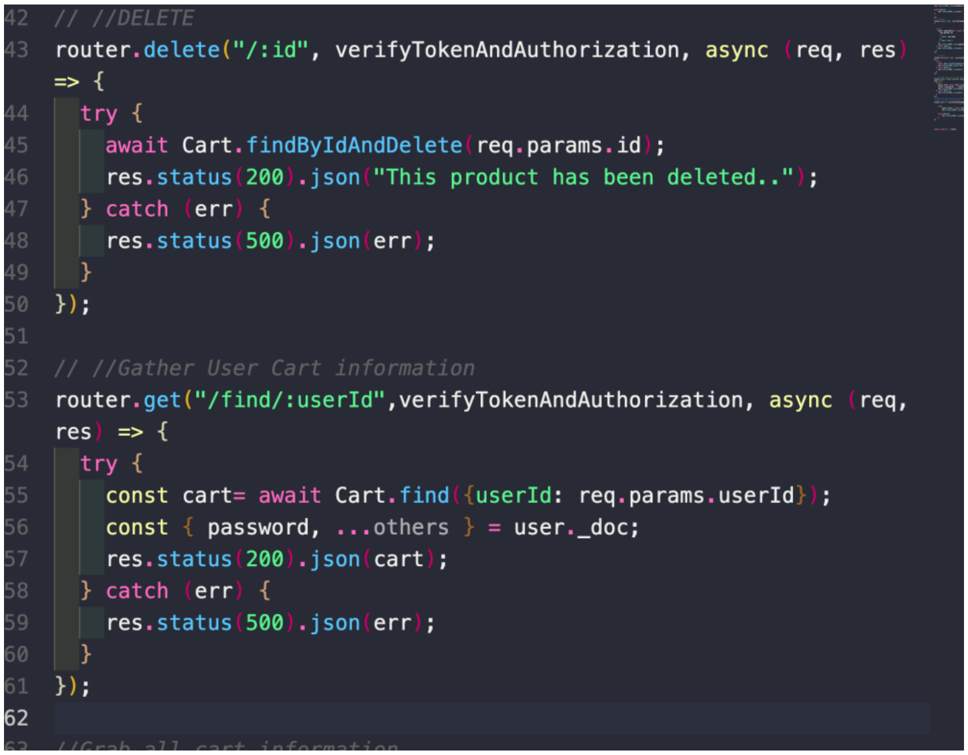


```
router.post("/", verifyToken, async (req,res)=>{
const newCart = new Cart(req.body)

try{
const savedCart = await newCart.save()
res.status(200).json(savedCart)

}catch(err){
    res.status(500).json(err)
}

});
// //UPDATE
router.put("/:id", verifyTokenAndAuthorization, async (req, res) =>
{


  try {
    const updatedCart = await Cart.findByIdAndUpdate(
      req.params.id,
      {
        $set: req.body,
      },
      { new: true }
```

```
Ln 4, Col 8   Spaces: 2   UTF-8   LF   {} Babel JavaScript   Go Live   Prettier
```



```
// //UPDATE
router.put("/:id", verifyTokenAndAuthorization, async (req, res) =>
{


  try {
    const updatedCart = await Cart.findByIdAndUpdate(
      req.params.id,
      {
        $set: req.body,
      },
      { new: true }
    );
    res.status(200).json(updatedCart);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

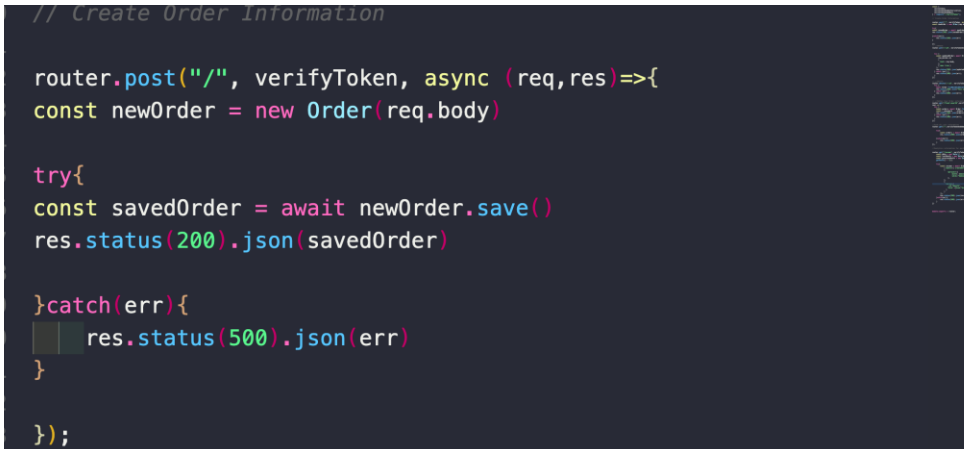


```
// //DELETE
router.delete("/:id", verifyTokenAndAuthorization, async (req, res)
=> {
  try {
    await Cart.findByIdAndDelete(req.params.id);
    res.status(200).json("This product has been deleted..");
  } catch (err) {
    res.status(500).json(err);
  }
});

// //Gather User Cart information
router.get("/find/:userId",verifyTokenAndAuthorization, async (req,
res) => {
  try {
    const cart= await Cart.find({userId: req.params.userId});
    const { password, ...others } = user._doc;
    res.status(200).json(cart);
  } catch (err) {
    res.status(500).json(err);
  }
});

```

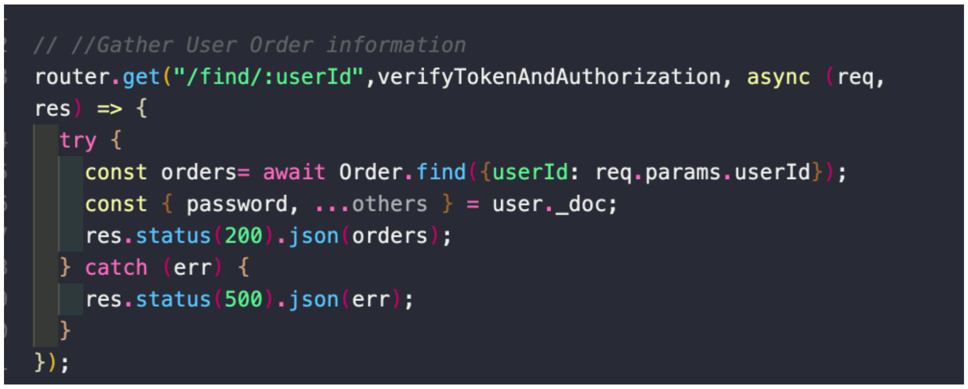
Cart Information:

The User Cart Post, get, delete and Put methods will do the following. Grab the users cart information, it can update the users cart information. It allows the user to delete their cart information. This will allow the user to add and remove items however they like.

Order API:

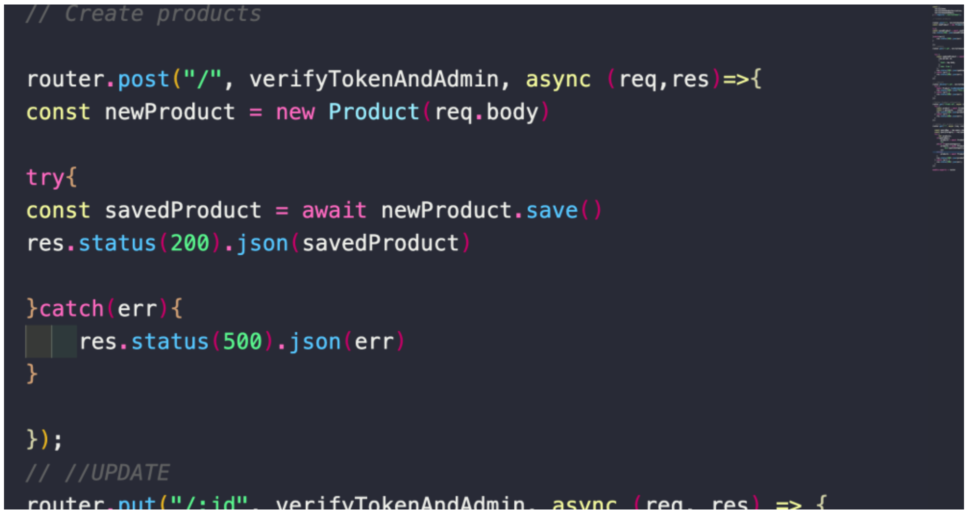


```
// Create Order Information

router.post("/", verifyToken, async (req,res)=>{
const newOrder = new Order(req.body)

try{
const savedOrder = await newOrder.save()
res.status(200).json(savedOrder)

}catch(err){
    res.status(500).json(err)
}

});
```

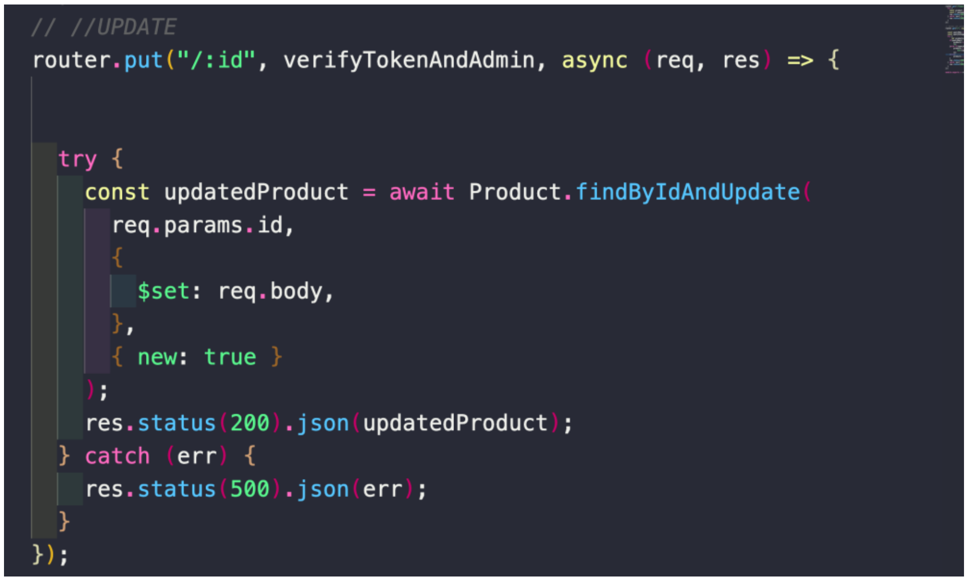


```
// //Gather User Order information
router.get("/find/:userId",verifyTokenAndAuthorization, async (req,
res) => {
  try {
    const orders= await Order.find({userId: req.params.userId});
    const { password, ...others } = user._doc;
    res.status(200).json(orders);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

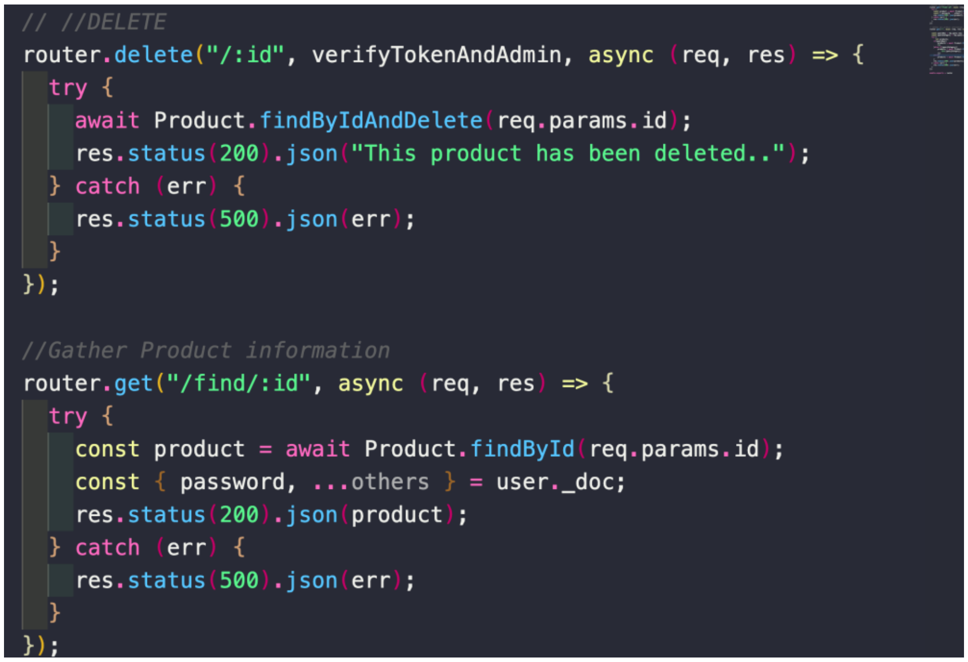
Product API:



```
// Create products

router.post("/", verifyTokenAndAdmin, async (req,res)=>{
const newProduct = new Product(req.body)

try{
const savedProduct = await newProduct.save()
res.status(200).json(savedProduct)

}catch(err){
    res.status(500).json(err)
}

});
// //UPDATE
router.put("/:id", verifyTokenAndAdmin, async (req, res) => {
```



```javascript
// //UPDATE
router.put("/:id", verifyTokenAndAdmin, async (req, res) => {


  try {
    const updatedProduct = await Product.findByIdAndUpdate(
      req.params.id,
      {
        $set: req.body,
      },
      { new: true }
    );
    res.status(200).json(updatedProduct);
  } catch (err) {
    res.status(500).json(err);
  }
});
```



```javascript
// //DELETE
router.delete("/:id", verifyTokenAndAdmin, async (req, res) => {
  try {
    await Product.findByIdAndDelete(req.params.id);
    res.status(200).json("This product has been deleted..");
  } catch (err) {
    res.status(500).json(err);
  }
});

//Gather Product information
router.get("/find/:id", async (req, res) => {
  try {
    const product = await Product.findById(req.params.id);
    const { password, ...others } = user._doc;
    res.status(200).json(product);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

For the Product API information. I can update products, I can post products, I can delete products and I can locate products by product Id. This is all stored in my database and can be retrieved quickly.