Inventory Management **System**

by Hayden & Kate

CONTENTS

Inventory Management System

What it does:

This app allows either a manager or employee of a store to sign up, log in and view an inventory of their products.

Functionality:

- *Sign up; as a manager or employee*
- *Log in; as a manager or employee*
- *Create, update or delete products*
- *Edit; products name, price, quantity and category*
- *Filter; products by name, price, quantity, category or id*
- *Backend: Managers can delete users but employees can't*
    - *this hasn't been implemented into the front end*

Example of a function:

src > functions > JS jwtFunctions.js > validateUserAuth

```
28
29  async function validateUserAuth(request, response, next){
30      console.log(request.headers);
31      const { authorization, jwt } = request.headers;
32
33      console.log(authorization, jwt)
34
35      if (!authorization && !jwt){
36          return response.status(403).json({
37              message:"No authorization token provided."
38          });
39      }
40
41      let providedToken;
42
43      if (authorization) {
44          const [,token] = authorization.split(' ');
45
46          providedToken = token;
47      } else if (jwt) {
48          providedToken = jwt;
49      }
50
51      console.log(providedToken);
52
53      let decodedData = decodeJWT(providedToken);
54      console.log(decodedData);
55      if (decodedData.userId){
56          request.authUserData = decodedData;
57          next();
58      } else {
59          return response.status(403).json({
60              message:"No authorization token provided."
61          });
62      }
```

validateUserAuth:

Checks if the user is logged in by

getting the token out of the jwt or authorization header, (t splits the authorization header and takes just the token part)

then it decodes the jwt to check the identity

If not logged in,

return message “No authorization token provided”

Example of conditional statement

src > controllers > JS ProductController.js > router.patch("/:id") callback

```
100
101  // Update Product by ID
102  router.patch("/:id", validateUserAuth, async (req, res) => {
103      // Expects updateData in the request body
104      const updateData  = req.body;
105
106      try {
107          let product;
108          const id = req.params.id;
109          // Find product by either ID or item
110          if (id) {
111              product = await updateOneProduct(id, updateData);
112          }
113          // if product is not found message
114          if (!product) {
115              return res.status(404).json({
116                  success: false,
117                  message: "Product not found to update",
118              });
119          }
120          // if product found and updated message
121          res.json({
122              success: true,
123              message: "Product updated successfully",
124              data: product,
125          });
126      // Error message
127      } catch (error) {
128          console.error(error);
129          res.status(500).json({
130              success: false,
131              message: "Error updating product",
132              error: error.message,
133          });
134      }
135  });
```

Peep the previous function mentioned being called here

If product is not found, Return error message of “Product not found to update”

Example of database operations:

Create Product function in ProductCrud.js

src > utils > crud > JS ProductCrud.js > ...

```
1   // Provide CRUD functions for the ProductModel
2   const { query } = require("express");
3   const { ProductModel } = require("../../models/ProductModel");
4
5   // Create new product
6   async function createProduct (product) {
7       //name, price, quantity, category, description = null) {
8
9       let result = await ProductModel.create(product);
10          // name: name,
11          // price: price,
12          // quantity: quantity,
13          // category: category,
14          // description: description
15      return result;
16  }
```

More examples of database operations:

Update product

```
// Update one product
async function updateOneProduct(id, updateData) {
    return result = await ProductModel.updateOne({_id: id}, updateData);
}
```

Delete product

```
// Delete one product
async function deleteOneProduct (id) {
    return result = await ProductModel.findByIdAndDelete(id);
}
```

Example of network request:

```
const handleSubmit = async (e) => {
    e.preventDefault();
    try {
        const response = await axios.post(
            "https://ims-backend-2qfp.onrender.com/products/create",
            formData
        );

        if (response.status === 201 || response.status === 200) {
            togglePopup();
            setFormData({ name: "", price: "", quantity: "", category: "" });
            fetchItems();
        }
    } catch (error) {
        console.error("Error adding item:", error);
    }
};
```

Creating Products on the front-end.

Problems encountered:

- *Price validator*
  - *tried to format the Price number into a monetary value*
  - *the validator I found online didn't work*
  - *decided to render in the front-end instead.*
- *Git merges*
  - *had conflicts merging,*
  - *put restrictions in place that meant a pull-request had to be required before merging to main*
- *the dreaded misplaced comma*
  - *forgot a comma in one of the database operations*
  - *code wouldn't work until rectified*
- *JWT/Auth issues*
  - *edit & delete wouldn't work*
  - *so made it so it accepted both JWT OR Authorization Bearer*

Time to check out the website!

https://ims-hk.netlify.app/

THANK

YOU