Application Overview:

- Node Express web application with portal and shopping cart functionality.
- MongoDB is used as the database.
- Key features: login, logoff, registration, product display (10 products), about, contact, cart, and thank you web content.

User Authentication:

- Users must log in with valid credentials to access any web content.
- Registration allowed with a user ID and password stored in the "login" collection.

WebCart Functionality:

- Shopping cart allows users to add or delete items.
- If the user checks out, the cart for that user is deleted.
- Saved web carts automatically loaded upon user login.
- Grand total displayed, and a link to the checkout page with a "Thank you for shopping with us" message.

Database Specifications:

- Database Name: cis485

Collections:

- login: {user id: String, password: String}

- cart: {user id: String, item id: String, item qty: Number, item price: Number}
- catalog: {item id: String, item name: String, item qty: Number, item price: Number}

Project File Structure:

Public folder: Contains static assets (images, CSS, JavaScript files).

Views folder: Contains main page content as EJS files.

Functional Layout Template: Ensure the use of a functional layout template in the project.

Directory structures & Explanation

STEP 1:

Frontend Sandbox

Ie the environment where you design and test the front end application before converting them to ejs format for backend application. They include static files like HTML, CSS and javascript. Where CSS has multiple subdirectory that connects to a mian.css for better debugging and organizing of codes.

STEP 2

- Set Up Node.js Project and Install all the required dependencies
- NPM Install:

- npm init -y
- express: Web application framework for Node.js.
- mongoose: MongoDB object modeling tool designed to work in an asynchronous environment.
- body-parser: Middleware to handle HTTP POST requests.
- express-session: Session middleware for Express.
- Bcrypt: Library to hash passwords securely.
- ejs: Templating engine for rendering views.

Step 3 (Project Structures)

- /public Folder

- /CSS

- /Products Directory

- /main.js

- /views Folder

- index.ejs

- login.ejs

- register.ejs

- about.ejs

- message.ejs

- contact.ejs

- products.ejs

- cart.ejs

- checkout.ejs

- contactreply.ejs

- layout.ejs

App.js (contains server side code and route handling for backend)

Catalog.json (is a json file that contains product credentials which is imported into the mongodb catalog collection that was created. It also follows a schema that has been defined in the server. Ie File description/ format

```
  {
  code: String,// code is the name of the img files rendered
  name: String,
  price: Number,
  quantity: Number,
}
```

Step 4: Set up Database Connection

Mongodb Essentials

- Atlas cluster for cloud Access or you could use localhost

- Connection string to input into node.js
- Authentication username(required for atlas)
- Authentication Password(required for atlas)

The next step is created automatically when the server is run with the necessary code and mongo db essentials however this was created during the testing procedure

Create Database Name: cis485

Collections:

- login: {userid: String, password: String}
- cart: {title: String, price: String, productImg: String, quantity: Number, userid: String,}

Note cartschema is defined to accommodate the frontend attribute that was established in the frontend file

- catalog: {itemid: String, itemname: String, itemqty: Number, itemprice: Number}

Step 5:
Import the catalog.json file into the mongodb catalog collection. There is a code defined in the server that uses this route to display all the pages in the catalog collection to the products.ejs file for client side viewing.

Directory Conclusion

- Public folder contains static files that apply to the front end of the application such as CSS, products Images and the client side javascript for pulling cart items and storing it in an object.

A important note: To apply backend functionality to this client side javascript an api route had to be defined in the static file so that when the user clicks buy now, that object is sent to the server side code to then be appended to the user id that was logged into the session and saved in the database

Ie Code Example from main.js

```
   function saveCartToServer(){
const payload = {
    cartItems: cartItems,
  };

  fetch('/api/cart', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(payload),
  })
    .then(response => response.json())
    .then(data => {
      console.log(data);
    })
    .catch(error => {
      console.error('Error:', error);
    });
}

// Call this function when you want to save the cart to the server (e.g., after the buy button is clicked)
function saveCartToServerAndClear() {
  saveCartToServer();
  cartItems = [];
  saveCartItemsToLocalStorage();
  updateCart();
}
```

<u>Views Folder</u>

Contains multiple ejs files that are rendered based on what the client side selects or the server api route has been programmed to display back to the client. One notable one is the layout.ejs
It's the building block of all other ejs files. It contains the header and the footer for all major navigation. The rest of the ejs files are body to be dynamically inserted and displayed with the

layout.ejs files. One key aspect is the login and or logoff that is dynamically updated on the ejs layout to correctly display when user is logged on the header

Finally Server side

App.js is super comprehensive and the section and definition is all there for you to explore

ENJOY MY WORK!!!