**Folder Structure**

This structure will help keep the project organized:

plaintext

Copy code

project-folder/

│

├── backend/

│ ├── db.js # MySQL database connection

│ ├── index.js # Main API server

│ ├── routes/ # Folder for route files

│ │ ├── auth.js

│ │ ├── products.js

│ │ ├── cart.js

│ │ └── orders.js

│ └── .env # Environment variables

│

└── frontend/

├── src/

│ ├── components/

│ ├── App.js # Main React App component

│ ├── index.js # React DOM entry point

│ └── App.css # Styling

└── package.json

**Step 1: MySQL Database Setup**

**Create the Database and Tables**

1. **Mysql command line. create databases and insert values if needed**
2. **Database Schema**:
   * **Database**: ecommerce\_db
3. **Create Tables**:

-- Users Table

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL

);

-- Products Table

CREATE TABLE products (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL,

inventory INT DEFAULT 0,

category\_id INT,

image\_url VARCHAR(255),

FOREIGN KEY (category\_id) REFERENCES categories(id)

);

-- Categories Table

CREATE TABLE categories (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL

);

-- Reviews Table

CREATE TABLE reviews (

id INT AUTO\_INCREMENT PRIMARY KEY,

product\_id INT,

user\_id INT,

rating INT CHECK(rating BETWEEN 1 AND 5),

review\_text TEXT,

FOREIGN KEY (product\_id) REFERENCES products(id),

FOREIGN KEY (user\_id) REFERENCES users(id)

);

-- Cart Table

CREATE TABLE cart\_items (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

product\_id INT,

quantity INT DEFAULT 1,

FOREIGN KEY (user\_id) REFERENCES users(id),

FOREIGN KEY (product\_id) REFERENCES products(id)

);

-- Orders Table

CREATE TABLE orders (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

total DECIMAL(10, 2),

status ENUM('Pending', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id)

);

-- Order Items Table

CREATE TABLE order\_items (

id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

product\_id INT,

quantity INT,

price DECIMAL(10, 2),

FOREIGN KEY (order\_id) REFERENCES orders(id),

FOREIGN KEY (product\_id) REFERENCES products(id)

);

**Step 2: Backend Setup (Node.js + Express)**

**vs code, create folder in file explorer,**

**open the folder in vscode, under that create a folder backend, open that in terminal using cd backend or open with integrated terminal directly**

**1. Initialize and Install Packages**

* Inside backend, initialize the project and install required packages:

npm init -y

npm install express mysql2 dotenv bcryptjs jsonwebtoken cors

**2. Configure MySQL Connection (db.js)**

* Create backend/db.js:

const mysql = require('mysql2');

require('dotenv').config();

const db = mysql.createConnection({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

});

db.connect((err) => {

if (err) throw err;

console.log('Connected to MySQL');

});

module.exports = db;

* **Add .env file** for MySQL credentials:

DB\_HOST=localhost

DB\_USER=root

DB\_PASSWORD=yourPassword

DB\_NAME=ecommerce\_db

**3. Add Authentication and API Routes**

* **User Authentication (auth.js), Product Management (products.js), Cart (cart.js), and Order Management (orders.js) will go into backend/routes.**

**You can create each route file, such as:**

**1. User Authentication (auth.js)**

Create backend/routes/auth.js:

// backend/routes/auth.js

const express = require('express');

const router = express.Router();

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const db = require('../db');

require('dotenv').config();

const JWT\_SECRET = process.env.JWT\_SECRET || 'your\_jwt\_secret';

// User registration route (for future use)

router.post('/register', async (req, res) => {

const { username, email, password } = req.body;

const hashedPassword = await bcrypt.hash(password, 10);

db.query('INSERT INTO users (username, email, password) VALUES (?, ?, ?)',

[username, email, hashedPassword],

(err, results) => {

if (err) return res.status(500).json({ error: 'User registration failed' });

res.status(201).json({ message: 'User registered successfully' });

});

});

// Login route

router.post('/login', (req, res) => {

const { email, password } = req.body;

// Check if the email exists

db.query('SELECT \* FROM users WHERE email = ?', [email], async (err, results) => {

if (err || results.length === 0) {

return res.status(401).json({ error: 'Invalid email or password' });

}

const user = results[0];

// Compare passwords using bcrypt

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) {

return res.status(401).json({ error: 'Invalid email or password' });

}

// Generate JWT token

const token = jwt.sign({ id: user.id }, JWT\_SECRET, { expiresIn: '1h' });

res.json({ message: 'Login successful', token });

});

});

module.exports = router;

* **Explanation**:
  + After retrieving the user by email, bcrypt.compare() checks if the provided password matches the hashed password stored in the database.
  + If they match, a JWT token is generated; if not, an error is returned.
  + The register route hashes the password and inserts the user into the database.
  + The login route checks the email and password, and if valid, generates a JWT token.

**2. Product Management (products.js)**

Create backend/routes/products.js:

// backend/routes/products.js

const express = require('express');

const router = express.Router();

const db = require('../db');

// Get all products

router.get('/', (req, res) => {

db.query('SELECT \* FROM products', (err, results) => {

if (err) return res.status(500).json({ error: 'Failed to retrieve products' });

res.json(results);

});

});

// Get a single product by ID

router.get('/:id', (req, res) => {

const { id } = req.params;

db.query('SELECT \* FROM products WHERE id = ?', [id], (err, results) => {

if (err || results.length === 0) {

return res.status(404).json({ error: 'Product not found' });

}

res.json(results[0]);

});

});

// Add a new product (for simplicity, we assume this is done by an admin)

router.post('/', (req, res) => {

const { name, description, price, inventory, category\_id, image\_url } = req.body;

db.query('INSERT INTO products (name, description, price, inventory, category\_id, image\_url) VALUES (?, ?, ?, ?, ?, ?)',

[name, description, price, inventory, category\_id, image\_url],

(err, results) => {

if (err) return res.status(500).json({ error: 'Failed to add product' });

res.status(201).json({ message: 'Product added successfully', productId: results.insertId });

});

});

module.exports = router;

* **Explanation**:
  + The GET routes retrieve all products or a specific product by ID.
  + The POST route allows adding new products (for admin or authorized users).

**3. Shopping Cart (cart.js)**

Create backend/routes/cart.js:

// backend/routes/cart.js

const express = require('express');

const router = express.Router();

const db = require('../db');

// Add item to cart

router.post('/', (req, res) => {

const { userId, productId, quantity } = req.body;

db.query('INSERT INTO cart\_items (user\_id, product\_id, quantity) VALUES (?, ?, ?)',

[userId, productId, quantity],

(err, results) => {

if (err) return res.status(500).json({ error: 'Failed to add item to cart' });

res.status(201).json({ message: 'Item added to cart' });

});

});

// Get items in user's cart

router.get('/:userId', (req, res) => {

const { userId } = req.params;

db.query('SELECT \* FROM cart\_items WHERE user\_id = ?', [userId], (err, results) => {

if (err) return res.status(500).json({ error: 'Failed to retrieve cart items' });

res.json(results);

});

});

// Update quantity of a cart item

router.put('/:id', (req, res) => {

const { id } = req.params;

const { quantity } = req.body;

db.query('UPDATE cart\_items SET quantity = ? WHERE id = ?', [quantity, id], (err) => {

if (err) return res.status(500).json({ error: 'Failed to update cart item quantity' });

res.json({ message: 'Cart item updated successfully' });

});

});

// Remove an item from the cart

router.delete('/:id', (req, res) => {

const { id } = req.params;

db.query('DELETE FROM cart\_items WHERE id = ?', [id], (err) => {

if (err) return res.status(500).json({ error: 'Failed to remove item from cart' });

res.json({ message: 'Item removed from cart' });

});

});

module.exports = router;

* **Explanation**:
  + The POST route adds an item to the cart.
  + The GET route retrieves items in a user’s cart.
  + The PUT route updates item quantity, and the DELETE route removes an item from the cart.

**4. Order Management (orders.js)**

Create backend/routes/orders.js:

// backend/routes/orders.js

const express = require('express');

const router = express.Router();

const db = require('../db');

// Place an order

router.post('/', (req, res) => {

const { userId, total, items } = req.body;

// Start transaction for placing an order

db.beginTransaction((err) => {

if (err) return res.status(500).json({ error: 'Transaction failed' });

db.query('INSERT INTO orders (user\_id, total, status) VALUES (?, ?, "Pending")',

[userId, total],

(err, results) => {

if (err) return db.rollback(() => res.status(500).json({ error: 'Failed to create order' }));

const orderId = results.insertId;

const orderItems = items.map(item => [orderId, item.productId, item.quantity, item.price]);

db.query('INSERT INTO order\_items (order\_id, product\_id, quantity, price) VALUES ?',

[orderItems],

(err) => {

if (err) return db.rollback(() => res.status(500).json({ error: 'Failed to add order items' }));

db.commit((err) => {

if (err) return db.rollback(() => res.status(500).json({ error: 'Failed to complete transaction' }));

res.json({ message: 'Order placed successfully', orderId });

});

});

});

});

});

// Get user orders

router.get('/:userId', (req, res) => {

const { userId } = req.params;

db.query('SELECT \* FROM orders WHERE user\_id = ?', [userId], (err, results) => {

if (err) return res.status(500).json({ error: 'Failed to retrieve orders' });

res.json(results);

});

});

module.exports = router;

* **Explanation**:
  + The POST route allows placing an order with multiple items, using a transaction to ensure data consistency.
  + The GET route retrieves all orders for a specific user.

**5. Wishlist**

Create backend/routes/wishlist.js:

// backend/routes/wishlist.js

const express = require('express');

const router = express.Router();

const db = require('../db');

// Add product to wishlist

router.post('/', (req, res) => {

const { userId, productId } = req.body;

db.query('INSERT INTO wishlist (user\_id, product\_id) VALUES (?, ?)', [userId, productId], (err) => {

if (err) return res.status(500).json({ error: 'Failed to add to wishlist' });

res.json({ message: 'Product added to wishlist' });

});

});

// Get user's wishlist

router.get('/:userId', (req, res) => {

const { userId } = req.params;

db.query(

'SELECT p.\* FROM products p JOIN wishlist w ON p.id = w.product\_id WHERE w.user\_id = ?',

[userId],

(err, results) => {

if (err) return res.status(500).json({ error: 'Failed to retrieve wishlist' });

res.json(results);

}

);

});

// Remove product from wishlist

router.delete('/:userId/:productId', (req, res) => {

const { userId, productId } = req.params;

db.query('DELETE FROM wishlist WHERE user\_id = ? AND product\_id = ?', [userId, productId], (err) => {

if (err) return res.status(500).json({ error: 'Failed to remove from wishlist' });

res.json({ message: 'Product removed from wishlist' });

});

});

module.exports = router;

**5. Reviews**

1. **Add a Review**: Allows a user to add a review and rating for a product.
2. **Get Reviews for a Product**: Retrieves all reviews for a specific product.

Create backend/routes/reviews.js:

// backend/routes/reviews.js

const express = require('express');

const router = express.Router();

const db = require('../db');

// Add a review and rating for a product

router.post('/', (req, res) => {

const { userId, productId, rating, reviewText } = req.body;

db.query(

'INSERT INTO reviews (user\_id, product\_id, rating, review\_text) VALUES (?, ?, ?, ?)',

[userId, productId, rating, reviewText],

(err) => {

if (err) return res.status(500).json({ error: 'Failed to add review' });

res.json({ message: 'Review added successfully' });

}

);

});

// Get all reviews for a specific product

router.get('/:productId', (req, res) => {

const { productId } = req.params;

db.query(

'SELECT r.\*, u.username FROM reviews r JOIN users u ON r.user\_id = u.id WHERE r.product\_id = ?',

[productId],

(err, results) => {

if (err) return res.status(500).json({ error: 'Failed to retrieve reviews' });

res.json(results);

}

);

});

module.exports = router;

**4. index.js to Use Routes**

Update backend/index.js to use these routes:

const express = require('express');

const cors = require('cors');

const db = require('./db');

const authRoutes = require('./routes/auth');

const productRoutes = require('./routes/products');

const cartRoutes = require('./routes/cart');

const orderRoutes = require('./routes/orders');

const wishlistRoutes = require('./routes/wishlist');

const reviewRoutes = require('./routes/reviews');

const app = express();

app.use(cors());

app.use(express.json());

app.use('/auth', authRoutes);

app.use('/products', productRoutes);

app.use('/cart', cartRoutes);

app.use('/orders', orderRoutes);

app.use('/wishlist', wishlistRoutes);

app.use('/reviews', reviewRoutes);

app.listen(5000, () => {

console.log('Server is running on http://localhost:5000');

});

**Step 3: Frontend Setup (React)**

1. **Create Frontend with Create React App**

npx create-react-app frontend

1. **Install Axios** for HTTP requests:

cd frontend

npm install axios

1. **Set up Components** for user login, product listing, and shopping cart in frontend/src/components.

**(a) Login Component**

Create frontend/src/components/Login.js to handle user authentication.

// frontend/src/components/Login.js

import React, { useState } from 'react';

import axios from 'axios';

function Login({ setToken }) {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await axios.post('http://localhost:5000/auth/login', { email, password });

setToken(response.data.token); // Store token in parent or state

alert('Login successful!');

} catch (error) {

console.error('Error logging in:', error);

alert('Invalid email or password');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input

type="email"

placeholder="Email"

value={email}

onChange={(e) => setEmail(e.target.value)}

/>

<input

type="password"

placeholder="Password"

value={password}

onChange={(e) => setPassword(e.target.value)}

/>

<button type="submit">Login</button>

</form>

</div>

);

}

export default Login;

* **Explanation**:
  + This component captures user email and password inputs.
  + It sends a POST request to /auth/login and stores the JWT token (could be saved in parent state, local storage, or context for later use).

**(b) Product Listing Component**

Create frontend/src/components/ProductList.js to display available products.

// frontend/src/components/ProductList.js

import React, { useEffect, useState } from 'react';

import axios from 'axios';

function ProductList({ addToCart }) {

const [products, setProducts] = useState([]);

useEffect(() => {

axios.get('http://localhost:5000/products')

.then(response => setProducts(response.data))

.catch(error => console.error('Error fetching products:', error));

}, []);

return (

<div>

<h2>Product Listing</h2>

<ul>

{products.map((product) => (

<li key={product.id}>

<h3>{product.name}</h3>

<p>{product.description}</p>

<p>Price: ${product.price}</p>

<button onClick={() => addToCart(product)}>Add to Cart</button>

</li>

))}

</ul>

</div>

);

}

export default ProductList;

* **Explanation**:
  + ProductList fetches products from the backend using axios.get on component mount (useEffect).
  + Products are displayed with a button to "Add to Cart," which triggers the addToCart function, handled in the parent component.

**(c) Shopping Cart Component**

Create frontend/src/components/Cart.js to display items in the user’s cart and manage item quantities.

// frontend/src/components/Cart.js

import React from 'react';

function Cart({ cartItems, updateQuantity, removeFromCart }) {

const getTotal = () => {

return cartItems.reduce((total, item) => total + item.price \* item.quantity, 0).toFixed(2);

};

return (

<div>

<h2>Shopping Cart</h2>

{cartItems.length === 0 ? (

<p>Your cart is empty</p>

) : (

<ul>

{cartItems.map((item) => (

<li key={item.id}>

<h3>{item.name}</h3>

<p>Price: ${item.price}</p>

<p>Quantity: {item.quantity}</p>

<button onClick={() => updateQuantity(item.id, item.quantity + 1)}>+</button>

<button onClick={() => updateQuantity(item.id, item.quantity - 1)}>-</button>

<button onClick={() => removeFromCart(item.id)}>Remove</button>

</li>

))}

</ul>

)}

<h3>Total: ${getTotal()}</h3>

</div>

);

}

export default Cart;

* **Explanation**:
  + Cart receives cartItems, updateQuantity, and removeFromCart as props.
  + Displays each cart item with buttons to increase or decrease quantity and remove the item.

Create new components in frontend/src/components for Wishlist and Reviews.

**(d) Wishlist Component**

Create frontend/src/components/Wishlist.js:

import React, { useEffect, useState } from 'react';

import axios from 'axios';

function Wishlist({ userId }) {

const [wishlist, setWishlist] = useState([]);

useEffect(() => {

axios.get(`http://localhost:5000/wishlist/${userId}`)

.then(response => setWishlist(response.data))

.catch(error => console.error('Error fetching wishlist:', error));

}, [userId]);

const removeFromWishlist = (productId) => {

axios.delete(`http://localhost:5000/wishlist/${userId}/${productId}`)

.then(() => setWishlist(wishlist.filter(item => item.id !== productId)))

.catch(error => console.error('Error removing from wishlist:', error));

};

return (

<div>

<h2>Wishlist</h2>

{wishlist.length === 0 ? (

<p>Your wishlist is empty</p>

) : (

<ul>

{wishlist.map(item => (

<li key={item.id}>

{item.name}

<button onClick={() => removeFromWishlist(item.id)}>Remove</button>

</li>

))}

</ul>

)}

</div>

);

}

export default Wishlist;

**(e) Reviews Component**

Create frontend/src/components/Reviews.js:

import React, { useEffect, useState } from 'react';

import axios from 'axios';

function Reviews({ productId, userId }) {

const [reviews, setReviews] = useState([]);

const [rating, setRating] = useState(5);

const [reviewText, setReviewText] = useState('');

useEffect(() => {

axios.get(`http://localhost:5000/reviews/${productId}`)

.then(response => setReviews(response.data))

.catch(error => console.error('Error fetching reviews:', error));

}, [productId]);

const submitReview = () => {

axios.post('http://localhost:5000/reviews', { userId, productId, rating, reviewText })

.then(response => {

setReviews([...reviews, response.data]);

setRating(5);

setReviewText('');

})

.catch(error => console.error('Error submitting review:', error));

};

return (

<div>

<h2>Reviews</h2>

<ul>

{reviews.map((review) => (

<li key={review.id}>

<strong>{review.username}</strong>: {review.review\_text} - {review.rating}/5

</li>

))}

</ul>

<h3>Leave a Review</h3>

<select value={rating} onChange={(e) => setRating(e.target.value)}>

{[1, 2, 3, 4, 5].map((num) => (

<option key={num} value={num}>{num}</option>

))}

</select>

<textarea

value={reviewText}

onChange={(e) => setReviewText(e.target.value)}

placeholder="Write your review here"

></textarea>

<button onClick={submitReview}>Submit Review</button>

</div>

);

}

export default Reviews;

**3. Main Application Component**

Use these components in App.js and manage shared state (e.g., cart and token) for the application.

Create frontend/src/App.js:

// frontend/src/App.js

import React, { useState } from 'react';

import Login from './components/Login';

import ProductList from './components/ProductList';

import Cart from './components/Cart';

import Wishlist from './components/Wishlist';

import Reviews from './components/Reviews';

import './App.css';

function App() {

const [token, setToken] = useState(null); // Store JWT token after login

const [cartItems, setCartItems] = useState([]);

const userId = 1; // Set to 1 for the test user, or use dynamic user ID after login

const addToCart = (product) => {

const existingItem = cartItems.find(item => item.id === product.id);

if (existingItem) {

setCartItems(

cartItems.map(item =>

item.id === product.id

? { ...item, quantity: item.quantity + 1 }

: item

)

);

} else {

setCartItems([...cartItems, { ...product, quantity: 1 }]);

}

};

const updateQuantity = (productId, newQuantity) => {

if (newQuantity <= 0) {

removeFromCart(productId);

} else {

setCartItems(

cartItems.map(item =>

item.id === productId ? { ...item, quantity: newQuantity } : item

)

);

}

};

const removeFromCart = (productId) => {

setCartItems(cartItems.filter(item => item.id !== productId));

};

return (

<div className="App">

{!token ? (

<Login setToken={setToken} />

) : (

<>

<ProductList addToCart={addToCart} />

<Cart

cartItems={cartItems}

updateQuantity={updateQuantity}

removeFromCart={removeFromCart}

/>

{/\* Add Wishlist and Reviews components \*/}

<Wishlist userId={userId} />

{/\* For demonstration, assume productId = 1 for reviews \*/}

<Reviews productId={1} userId={userId} />

</>

)}

</div>

);

}

export default App;

**Step 4: Run and Test the Project**

1. **Start the Backend**:
   * Open a terminal, navigate to backend, and run:

node index.js

1. **Start the Frontend**:
   * Open another terminal, navigate to frontend, and run:

npm start

**inserting dummy values to table**

-- Dummy User

INSERT INTO users (username, email, password) VALUES

('testuser', 'testuser@example.com', '$2a$10$OZHmZXhsIp2hC6OTwTHazeARuthYoVnaeeom5VTY7JJouBxxfZ5o2');

-- Password is "password123" (hashed using bcrypt)

### “””**Step 1: Hash and Insert Password Directly in the Database**

If you didn’t already hash password123 before inserting it into the database, follow these steps:

1. Open a new Node.js file called hashPassword.js (in the same folder as your backend files).
2. Use the following code to generate the hash:

// backend\hashPassword.js

const bcrypt = require('bcryptjs');

bcrypt.hash('password123', 10, (err, hash) => {

if (err) throw err;

console.log("Hashed password:", hash);

});

1. Run this file in your terminal to generate a hash:

node hashPassword.js

1. Copy the generated hash from the terminal output and update your database with it.

UPDATE users SET password = '$2a$10$....' WHERE email = 'testuser@example.com';

Replace '$2a$10$....' with the actual hash generated by bcrypt.”””

-- Categories

INSERT INTO categories (name) VALUES

('Electronics'),

('Books'),

('Clothing');

-- Products

INSERT INTO products (name, description, price, inventory, category\_id, image\_url) VALUES

('Smartphone', 'Latest model smartphone with high-quality camera', 699.99, 10, 1, 'https://example.com/smartphone.jpg'),

('Laptop', 'Lightweight laptop with powerful performance', 999.99, 5, 1, 'https://example.com/laptop.jpg'),

('Novel Book', 'A bestselling novel by a famous author', 14.99, 20, 2, 'https://example.com/novel.jpg'),

('T-Shirt', 'Comfortable cotton T-shirt available in various sizes', 19.99, 50, 3, 'https://example.com/tshirt.jpg');

-- Reviews

INSERT INTO reviews (product\_id, user\_id, rating, review\_text) VALUES

(1, 1, 5, 'Excellent smartphone, highly recommend!');