**E-commerce Website**

* **Concept:** A functional online store with product listings, a shopping cart, user accounts, and a checkout process.
* **Backend:**
  + Models for Product, User, Order, OrderItem, Category, Review.
  + Implement user authentication, product management (add, edit, delete), order processing.
  + Consider integrating a payment gateway (e.g., Stripe, PayPal) on the backend.
* **Frontend:**
  + Components for product display, search and filtering, product details, shopping cart, checkout forms.
  + Redux Toolkit for managing product state, cart items, user session, and order information.
  + Implement search, pagination, and sorting for products.

**Building a complete E-commerce project is a significant undertaking, involving hundreds or thousands of lines of code, database setup, environment configuration, and debugging.** It's a multi-day or even multi-week effort for a single developer.

**What I *can* do is provide you with a detailed, step-by-step guide, including:**

1. **Project Setup Instructions:** How to initialize your backend and frontend.
2. **Core Code Snippets and Logic:** Examples for models, controllers, routes, Redux slices, and React components.
3. **Detailed Explanations:** For each part of the stack, explaining *why* we're doing things a certain way.
4. **Best Practices:** Tips for security, scalability, and maintainability.
5. **Troubleshooting Tips:** Common issues and how to resolve them.
6. **Recommended Libraries and Tools:** Beyond the core stack.

**Let's start by outlining the phased approach we'll take to build this E-commerce application.** This will make the process manageable.

**Project Name:** ecom-fullstack-app (You can change this)

**Overall Architecture:**

ecom-fullstack-app/

├── backend/

│ ├── src/

│ │ ├── config/ (db config, .env handling)

│ │ ├── models/ (Sequelize models)

│ │ ├── controllers/ (API logic)

│ │ ├── routes/ (Express routes)

│ │ ├── middleware/ (Auth, error handling)

│ │ ├── utils/ (JWT, password hashing)

│ │ └── app.js (main Express app)

│ ├── .env

│ ├── package.json

│ └── server.js (entry point)

└── frontend/

├── public/

├── src/

│ ├── components/ (reusable UI)

│ ├── pages/ (main views)

│ ├── redux/ (slices, store)

│ ├── services/ (API calls)

│ ├── utils/ (helpers)

│ ├── App.js

│ ├── index.js

│ └── index.css

├── package.json

└── .env

**Phase 1: Backend Setup & Core Functionality (Users & Products)**

This phase will focus on getting your Node.js/Express backend running, setting up MySQL with Sequelize, and implementing basic user authentication and product management.

**Step 1: Project Initialization & Directory Structure**

1. **Create Project Folder:**

Bash

mkdir ecom-fullstack-app

cd ecom-fullstack-app

1. **Initialize Backend:**

Bash

mkdir backend

cd backend

npm init -y

1. **Install Backend Dependencies:**

Bash

npm install express mysql2 sequelize dotenv bcryptjs jsonwebtoken multer nodemon

* + express: Web framework.
  + mysql2: MySQL client library.
  + sequelize: ORM for MySQL.
  + dotenv: To load environment variables from .env file.
  + bcryptjs: For password hashing.
  + jsonwebtoken: For creating and verifying JWTs.
  + multer: For handling multipart/form-data (file uploads).
  + nodemon: (Dev dependency) Automatically restarts the server on file changes.

1. **Create Backend Directory Structure:**

Bash

mkdir src

cd src

mkdir config controllers middleware models routes utils

touch app.js

cd ..

touch server.js .env

**Step 2: Database Setup (MySQL & Sequelize Configuration)**

1. **Start MySQL Server:** Ensure your MySQL server is running.
2. **Create Database:** Log in to your MySQL client (e.g., MySQL Workbench, command line) and create a new database.

SQL

CREATE DATABASE ecom\_db;

1. **.env file (backend/.env):** This file will store your sensitive information. **NEVER push this file to Git.**

Code snippet

PORT=5000

NODE\_ENV=development

DB\_HOST=localhost

DB\_USER=root

DB\_PASSWORD=your\_mysql\_password

DB\_NAME=ecom\_db

JWT\_SECRET=aVeryStrongSecretKeyThatYouShouldChangeInProduction!

JWT\_EXPIRE=30d # Token expires in 30 days

* + **Important:** Replace your\_mysql\_password with your actual MySQL root password.
  + **Best Practice:** For production, use stronger, randomly generated secrets for JWT\_SECRET.

1. **backend/src/config/db.js (Sequelize Configuration):** This file will handle the database connection.

JavaScript

const { Sequelize } = require('sequelize');

require('dotenv').config(); // Load environment variables

const sequelize = new Sequelize(

process.env.DB\_NAME,

process.env.DB\_USER,

process.env.DB\_PASSWORD,

{

host: process.env.DB\_HOST,

dialect: 'mysql',

logging: false, // Set to true to see SQL queries in console

}

);

const connectDB = async () => {

try {

await sequelize.authenticate();

console.log('MySQL connected successfully with Sequelize.');

} catch (error) {

console.error('Unable to connect to the database:', error);

process.exit(1); // Exit process with failure

}

};

module.exports = { sequelize, connectDB };

**Step 3: Sequelize Models**

Create your Sequelize models in backend/src/models/. These will represent your database tables.

1. **backend/src/models/User.js:**

JavaScript

const { DataTypes } = require('sequelize');

const { sequelize } = require('../config/db');

const bcrypt = require('bcryptjs');

const User = sequelize.define('User', {

id: {

type: DataTypes.UUID,

defaultValue: DataTypes.UUIDV4,

primaryKey: true,

},

name: {

type: DataTypes.STRING,

allowNull: false,

},

email: {

type: DataTypes.STRING,

allowNull: false,

unique: true,

validate: {

isEmail: true,

},

},

password: {

type: DataTypes.STRING,

allowNull: false,

},

isAdmin: {

type: DataTypes.BOOLEAN,

allowNull: false,

defaultValue: false,

},

}, {

tableName: 'users', // Explicitly define table name

timestamps: true, // Adds createdAt and updatedAt fields

});

// Hash password before saving

User.beforeCreate(async (user) => {

const salt = await bcrypt.genSalt(10);

user.password = await bcrypt.hash(user.password, salt);

});

// Compare entered password with hashed password

User.prototype.matchPassword = async function (enteredPassword) {

return await bcrypt.compare(enteredPassword, this.password);

};

module.exports = User;

1. **backend/src/models/Product.js:**

JavaScript

const { DataTypes } = require('sequelize');

const { sequelize } = require('../config/db');

const Product = sequelize.define('Product', {

id: {

type: DataTypes.UUID,

defaultValue: DataTypes.UUIDV4,

primaryKey: true,

},

user\_id: { // User who created the product (for admin panel)

type: DataTypes.UUID,

allowNull: false,

references: {

model: 'users', // Refers to table name

key: 'id',

}

},

name: {

type: DataTypes.STRING,

allowNull: false,

},

image: { // Storing image path/URL

type: DataTypes.STRING,

allowNull: false,

},

brand: {

type: DataTypes.STRING,

allowNull: false,

},

category: {

type: DataTypes.STRING, // For simplicity now, later we can link to Category model

allowNull: false,

},

description: {

type: DataTypes.TEXT,

allowNull: false,

},

rating: {

type: DataTypes.DECIMAL(2, 1), // e.g., 4.5

allowNull: false,

defaultValue: 0.0,

},

numReviews: {

type: DataTypes.INTEGER,

allowNull: false,

defaultValue: 0,

},

price: {

type: DataTypes.DECIMAL(10, 2), // e.g., 99.99

allowNull: false,

defaultValue: 0.0,

},

countInStock: {

type: DataTypes.INTEGER,

allowNull: false,

defaultValue: 0,

},

}, {

tableName: 'products',

timestamps: true,

});

module.exports = Product;

1. **backend/src/models/index.js (Associations & Sync):** This file will define associations between your models and sync them with the database.

JavaScript

const { sequelize } = require('../config/db');

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

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

// const Order = require('./Order'); // Will add later

// const OrderItem = require('./OrderItem'); // Will add later

// const Review = require('./Review'); // Will add later

// const Category = require('./Category'); // Will add later

// Define Associations

// User and Product (User who created the product)

User.hasMany(Product, { foreignKey: 'user\_id' });

Product.belongsTo(User, { foreignKey: 'user\_id' });

// (Add other associations as you create models for Order, OrderItem, Review, Category)

// user\_id

const syncDatabase = async () => {

try {

// `alter: true` will update the table schema without dropping it if changes are detected.

// Use `force: true` ONLY IN DEVELOPMENT to drop and recreate tables on every restart.

await sequelize.sync({ alter: true });

console.log('Database synced successfully!');

} catch (error) {

console.error('Error syncing database:', error);

}

};

module.exports = {

sequelize,

User,

Product,

// Export other models here

syncDatabase,

};

**Step 4: Utility Functions (JWT Generation)**

Create backend/src/utils/generateToken.js to create JWTs.

1. **backend/src/utils/generateToken.js:**

JavaScript

const jwt = require('jsonwebtoken');

const generateToken = (id) => {

return jwt.sign({ id }, process.env.JWT\_SECRET, {

expiresIn: process.env.JWT\_EXPIRE,

});

};

module.exports = generateToken;

**Step 5: Controllers (API Logic)**

Create backend/src/controllers/ to handle the logic for your API routes.

1. **backend/src/controllers/userController.js:**

JavaScript

const asyncHandler = require('express-async-handler'); // npm install express-async-handler

const User = require('../models/User');

const generateToken = require('../utils/generateToken');

// @desc Auth user & get token

// @route POST /api/users/login

// @access Public

const authUser = asyncHandler(async (req, res) => {

const { email, password } = req.body;

const user = await User.findOne({ where: { email } });

if (user && (await user.matchPassword(password))) {

res.json({

id: user.id,

name: user.name,

email: user.email,

isAdmin: user.isAdmin,

token: generateToken(user.id),

});

} else {

res.status(401); // Unauthorized

throw new Error('Invalid email or password');

}

});

// @desc Register a new user

// @route POST /api/users/register

// @access Public

const registerUser = asyncHandler(async (req, res) => {

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

const userExists = await User.findOne({ where: { email } });

if (userExists) {

res.status(400); // Bad Request

throw new Error('User already exists');

}

const user = await User.create({

name,

email,

password, // Password will be hashed by the User.beforeCreate hook

});

if (user) {

res.status(201).json({ // Created

id: user.id,

name: user.name,

email: user.email,

isAdmin: user.isAdmin,

token: generateToken(user.id),

});

} else {

res.status(400);

throw new Error('Invalid user data');

}

});

// @desc Get user profile

// @route GET /api/users/profile

// @access Private

const getUserProfile = asyncHandler(async (req, res) => {

// req.user is set by the protect middleware

const user = await User.findByPk(req.user.id);

if (user) {

res.json({

id: user.id,

name: user.name,

email: user.email,

isAdmin: user.isAdmin,

});

} else {

res.status(404);

throw new Error('User not found');

}

});

// @desc Update user profile

// @route PUT /api/users/profile

// @access Private

const updateUserProfile = asyncHandler(async (req, res) => {

const user = await User.findByPk(req.user.id);

if (user) {

user.name = req.body.name || user.name;

user.email = req.body.email || user.email;

if (req.body.password) {

// The User.beforeUpdate hook (if you add one) or manually hash here

const salt = await bcrypt.genSalt(10);

user.password = await bcrypt.hash(req.body.password, salt);

}

const updatedUser = await user.save();

res.json({

id: updatedUser.id,

name: updatedUser.name,

email: updatedUser.email,

isAdmin: updatedUser.isAdmin,

token: generateToken(updatedUser.id),

});

} else {

res.status(404);

throw new Error('User not found');

}

});

module.exports = { authUser, registerUser, getUserProfile, updateUserProfile };

* + **Install express-async-handler:** npm install express-async-handler (This simplifies error handling in async Express routes).

1. **backend/src/controllers/productController.js:**

JavaScript

const asyncHandler = require('express-async-handler');

const Product = require('../models/Product');

const User = require('../models/User'); // Import User model for association include

// @desc Fetch all products

// @route GET /api/products

// @access Public

const getProducts = asyncHandler(async (req, res) => {

const products = await Product.findAll();

res.json(products);

});

// @desc Fetch single product

// @route GET /api/products/:id

// @access Public

const getProductById = asyncHandler(async (req, res) => {

const product = await Product.findByPk(req.params.id);

if (product) {

res.json(product);

} else {

res.status(404);

throw new Error('Product not found');

}

});

// @desc Create a product

// @route POST /api/products

// @access Private/Admin

const createProduct = asyncHandler(async (req, res) => {

// For simplicity, we'll assign a default product created by an admin (req.user.id)

// In a real app, form data for new product would come from frontend

const { name, price, description, image, brand, category, countInStock } = req.body;

const product = await Product.create({

name: name || 'Sample name',

price: price || 0,

user\_id: req.user.id, // ID of the admin creating the product

image: image || '/images/sample.jpg',

brand: brand || 'Sample Brand',

category: category || 'Sample Category',

countInStock: countInStock || 0,

description: description || 'Sample description',

});

res.status(201).json(product);

});

// @desc Update a product

// @route PUT /api/products/:id

// @access Private/Admin

const updateProduct = asyncHandler(async (req, res) => {

const { name, price, description, image, brand, category, countInStock } = req.body;

const product = await Product.findByPk(req.params.id);

if (product) {

product.name = name || product.name;

product.price = price || product.price;

product.description = description || product.description;

product.image = image || product.image;

product.brand = brand || product.brand;

product.category = category || product.category;

product.countInStock = countInStock !== undefined ? countInStock : product.countInStock;

const updatedProduct = await product.save();

res.json(updatedProduct);

} else {

res.status(404);

throw new Error('Product not found');

}

});

// @desc Delete a product

// @route DELETE /api/products/:id

// @access Private/Admin

const deleteProduct = asyncHandler(async (req, res) => {

const product = await Product.findByPk(req.params.id);

if (product) {

await product.destroy();

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

} else {

res.status(404);

throw new Error('Product not found');

}

});

module.exports = {

getProducts,

getProductById,

createProduct,

updateProduct,

deleteProduct,

};

**Step 6: Middleware (Authentication & Error Handling)**

1. **backend/src/middleware/authMiddleware.js:** This middleware will protect routes by verifying JWTs.

JavaScript

const jwt = require('jsonwebtoken');

const asyncHandler = require('express-async-handler');

const User = require('../models/User');

const protect = asyncHandler(async (req, res, next) => {

let token;

if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {

try {

// Get token from header

token = req.headers.authorization.split(' ')[1];

// Verify token

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

// Get user from the token and attach it to the request object

req.user = await User.findByPk(decoded.id, {

attributes: { exclude: ['password'] }, // Exclude password from the returned user object

});

if (!req.user) {

res.status(401);

throw new Error('Not authorized, user not found');

}

next();

} catch (error) {

console.error(error);

res.status(401);

throw new Error('Not authorized, token failed');

}

}

if (!token) {

res.status(401);

throw new Error('Not authorized, no token');

}

});

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

if (req.user && req.user.isAdmin) {

next();

} else {

res.status(403); // Forbidden

throw new Error('Not authorized as an admin');

}

};

module.exports = { protect, admin };

1. **backend/src/middleware/errorMiddleware.js:** For centralized error handling.

JavaScript

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

const error = new Error(`Not Found - ${req.originalUrl}`);

res.status(404);

next(error);

};

const errorHandler = (err, req, res, next) => {

const statusCode = res.statusCode === 200 ? 500 : res.statusCode;

res.status(statusCode);

res.json({

message: err.message,

stack: process.env.NODE\_ENV === 'production' ? null : err.stack,

});

};

module.exports = { notFound, errorHandler };

**Step 7: Express Routes**

Create backend/src/routes/ for organizing your API endpoints.

1. **backend/src/routes/userRoutes.js:**

JavaScript

const express = require('express');

const router = express.Router();

const {

authUser,

registerUser,

getUserProfile,

updateUserProfile,

} = require('../controllers/userController');

const { protect } = require('../middleware/authMiddleware');

router.post('/login', authUser);

router.post('/register', registerUser);

router.route('/profile').get(protect, getUserProfile).put(protect, updateUserProfile);

module.exports = router;

1. **backend/src/routes/productRoutes.js:**

JavaScript

const express = require('express');

const router = express.Router();

const {

getProducts,

getProductById,

createProduct,

updateProduct,

deleteProduct,

} = require('../controllers/productController');

const { protect, admin } = require('../middleware/authMiddleware');

router.route('/').get(getProducts).post(protect, admin, createProduct);

router

.route('/:id')

.get(getProductById)

.put(protect, admin, updateProduct)

.delete(protect, admin, deleteProduct);

module.exports = router;

**Step 8: Main Express App & Server Entry Point**

1. **backend/src/app.js (Main Express Application):**

JavaScript

const express = require('express');

const morgan = require('morgan'); // npm install morgan (for logging HTTP requests)

const cors = require('cors'); // npm install cors (for cross-origin requests from frontend)

const userRoutes = require('./routes/userRoutes');

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

const { notFound, errorHandler } = require('./middleware/errorMiddleware');

const app = express();

if (process.env.NODE\_ENV === 'development') {

app.use(morgan('dev')); // Log HTTP requests in development

}

app.use(express.json()); // Body parser for JSON

app.use(cors()); // Enable CORS for all origins (for development)

// API Routes

app.use('/api/users', userRoutes);

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

// Default route for testing

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

res.send('API is running...');

});

// Error Handling Middleware

app.use(notFound);

app.use(errorHandler);

module.exports = app;

* + **Install morgan and cors:** npm install morgan cors

1. **backend/server.js (Entry point):**

JavaScript

const app = require('./src/app');

const { connectDB } = require('./src/config/db');

const { syncDatabase } = require('./src/models'); // Import syncDatabase function

require('dotenv').config();

const PORT = process.env.PORT || 5000;

// Connect to database

connectDB();

// Sync database models (creates tables if they don't exist)

// IMPORTANT: In production, use Sequelize migrations instead of sync({ alter: true }) or sync({ force: true })

syncDatabase();

app.listen(PORT, () => {

console.log(`Server running in ${process.env.NODE\_ENV} mode on port ${PORT}`);

});

**Step 9: Run the Backend**

1. **Add start script to backend/package.json:**

JSON

{

"name": "backend",

"version": "1.0.0",

"description": "",

"main": "server.js",

"scripts": {

"start": "node server.js",

"dev": "nodemon server.js"

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

"bcryptjs": "^2.4.3",

"cors": "^2.8.5",

"dotenv": "^16.4.5",

"express": "^4.19.2",

"express-async-handler": "^1.2.0",

"jsonwebtoken": "^9.0.2",

"morgan": "^1.10.0",

"multer": "^1.4.5-lts.1",

"mysql2": "^3.9.7",

"sequelize": "^6.37.3"

},

"devDependencies": {

"nodemon": "^3.1.0"

}

}

1. **Start the server:**

Bash

cd backend

npm run dev

You should see: MySQL connected successfully with Sequelize. Database synced successfully! Server running in development mode on port 5000

1. **Test with Postman/Insomnia:**
   * GET http://localhost:5000/api/products (Should return an empty array initially)
   * POST http://localhost:5000/api/users/register (Register a new user)
     + Body (JSON): { "name": "Test User", "email": "test@example.com", "password": "password123" }
   * POST http://localhost:5000/api/users/login
     + Body (JSON): { "email": "test@example.com", "password": "password123" }
     + Copy the token from the response.
   * GET http://localhost:5000/api/users/profile
     + Headers: Authorization: Bearer <your\_token\_here>
   * To test product creation by an admin:
     + In your MySQL database, manually update isAdmin to 1 for your test@example.com user in the users table.
     + POST http://localhost:5000/api/products (Use the admin user's token)
       - Headers: Authorization: Bearer <admin\_token>
       - Body (JSON): { "name": "Gaming Mouse", "price": 49.99, "description": "High precision gaming mouse", "image": "/images/mouse.jpg", "brand": "Logitech", "category": "Electronics", "countInStock": 10 }

**This completes Phase 1: Backend Setup & Core Functionality.**

**Next, we will move to Phase 2: Frontend Setup (React & Redux Toolkit) and integrate it with this backend.**

Please set up your backend first, ensure it's running, and test the endpoints using Postman or Insomnia. Let me know when you're ready to proceed to the frontend!

Excellent! That's great progress. Now that your backend is up and running and you've tested the core endpoints, let's dive into **Phase 2: Frontend Setup with React and Redux Toolkit**.

This phase will focus on building the user interface, managing state with Redux Toolkit, and making API calls to your backend.

**Phase 2: Frontend Setup (React & Redux Toolkit)**

**Step 1: Frontend Project Initialization & Directory Structure**

1. **Navigate back to the root of your project:**

Bash

cd .. # This should take you from 'backend' to 'ecom-fullstack-app'

1. **Initialize React App (using Vite for speed and modern tooling - highly recommended):**

Bash

npm create vite@latest frontend -- --template react

* + When prompted, choose React for the framework and JavaScript for the variant.
  + If you prefer Create React App, you can use npx create-react-app frontend, but Vite is generally faster for development.

1. **Navigate into the new frontend directory:**

Bash

cd frontend

1. **Install Frontend Dependencies:**

Bash

npm install react-router-dom @reduxjs/toolkit react-redux axios

* + react-router-dom: For client-side routing.
  + @reduxjs/toolkit: The official, opinionated, batteries-included toolset for efficient Redux development.
  + react-redux: Official React bindings for Redux.
  + axios: Promise-based HTTP client for making API requests.

1. **Clean up Boilerplate (Optional but Recommended):** Inside your frontend/src directory, you can delete App.css, index.css (we'll create our own later), assets/, logo.svg, and modify App.jsx and main.jsx to be cleaner.
2. **Create Frontend Directory Structure (inside frontend/src):**

Bash

cd src

mkdir components pages redux services utils

# You might want to remove existing App.css, index.css, assets directory

touch index.css App.css # We'll use these for basic styling

Your frontend/src directory should now look something like this:

frontend/src/

├── components/

├── pages/

├── redux/

│ ├── slices/ # For Redux Toolkit slices

│ └── store.js # Redux store configuration

├── services/ # For making API calls

├── utils/ # General utility functions

├── App.jsx # Main application component

├── index.css # Global styles

├── main.jsx # Entry point for React

**Step 2: Basic Styling (frontend/src/index.css & App.css)**

Let's add some minimal global styles to make things readable.

1. **frontend/src/index.css:**

CSS

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');

:root {

font-family: 'Roboto', sans-serif;

line-height: 1.5;

font-weight: 400;

color-scheme: light dark;

color: rgba(255, 255, 255, 0.87);

background-color: #242424;

font-synthesis: none;

text-rendering: optimizeLegibility;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

body {

margin: 0;

display: flex;

place-items: center;

min-width: 320px;

min-height: 100vh;

background-color: #f0f2f5; /\* Light grey background \*/

color: #333; /\* Dark text for readability \*/

}

#root {

max-width: 1280px;

margin: 0 auto;

padding: 2rem;

text-align: center;

width: 100%; /\* Ensure root takes full width \*/

}

a {

font-weight: 500;

color: #646cff;

text-decoration: inherit;

}

a:hover {

color: #535bf2;

}

h1 {

font-size: 3.2em;

line-height: 1.1;

color: #333;

}

button {

border-radius: 8px;

border: 1px solid transparent;

padding: 0.6em 1.2em;

font-size: 1em;

font-weight: 500;

font-family: inherit;

background-color: #1a1a1a;

cursor: pointer;

transition: border-color 0.25s;

}

button:hover {

border-color: #646cff;

}

button:focus,

button:focus-visible {

outline: 4px auto -webkit-focus-ring-color;

}

1. **frontend/src/App.css (for basic layout):**

CSS

.container {

padding: 1rem;

}

.header {

background-color: #333;

color: #fff;

padding: 1rem 0;

display: flex;

justify-content: space-between;

align-items: center;

padding: 1rem 2rem;

border-radius: 8px;

margin-bottom: 2rem;

}

.header .logo a {

color: #fff;

font-size: 1.5rem;

font-weight: bold;

}

.header nav ul {

list-style: none;

margin: 0;

padding: 0;

display: flex;

gap: 1.5rem;

}

.header nav ul li a {

color: #fff;

text-decoration: none;

font-size: 1rem;

}

.header nav ul li a:hover {

color: #ddd;

}

.main-content {

min-height: 70vh; /\* Adjust as needed \*/

padding: 1rem 0;

}

.footer {

margin-top: 2rem;

padding: 1rem;

background-color: #333;

color: #fff;

text-align: center;

border-radius: 8px;

}

/\* Product Grid \*/

.product-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

gap: 2rem;

padding: 2rem 0;

}

.product-card {

background-color: #fff;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

padding: 1.5rem;

text-align: left;

display: flex;

flex-direction: column;

justify-content: space-between;

height: 100%;

}

.product-card img {

max-width: 100%;

height: 200px; /\* Fixed height for consistency \*/

object-fit: contain; /\* Ensures entire image is visible \*/

border-radius: 4px;

margin-bottom: 1rem;

}

.product-card h3 {

font-size: 1.2rem;

margin-top: 0.5rem;

margin-bottom: 0.5rem;

color: #333;

}

.product-card p {

font-size: 1rem;

color: #666;

margin-bottom: 1rem;

}

.product-card .price {

font-size: 1.4rem;

font-weight: bold;

color: #007bff; /\* Blue for price \*/

margin-bottom: 1rem;

}

.product-card button {

background-color: #007bff;

color: white;

border: none;

padding: 0.8rem 1.2rem;

border-radius: 5px;

cursor: pointer;

font-size: 1rem;

align-self: flex-start; /\* Align button to start \*/

}

.product-card button:hover {

background-color: #0056b3;

}

/\* Forms \*/

.form-container {

background-color: #fff;

padding: 2rem;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

max-width: 400px;

margin: 2rem auto;

text-align: left;

}

.form-container h2 {

text-align: center;

margin-bottom: 1.5rem;

color: #333;

}

.form-group {

margin-bottom: 1rem;

}

.form-group label {

display: block;

margin-bottom: 0.5rem;

font-weight: bold;

color: #555;

}

.form-group input {

width: 100%;

padding: 0.8rem;

border: 1px solid #ddd;

border-radius: 5px;

box-sizing: border-box; /\* Include padding in width \*/

font-size: 1rem;

}

.form-container button[type="submit"] {

width: 100%;

padding: 0.8rem;

background-color: #28a745; /\* Green for submit \*/

color: white;

border: none;

border-radius: 5px;

font-size: 1.1rem;

cursor: pointer;

}

.form-container button[type="submit"]:hover {

background-color: #218838;

}

.form-container .text-center {

text-align: center;

margin-top: 1rem;

color: #666;

}

.form-container .text-center a {

color: #007bff;

text-decoration: none;

}

.form-container .text-center a:hover {

text-decoration: underline;

}

/\* Messages \*/

.message {

padding: 0.75rem 1.25rem;

margin-bottom: 1rem;

border: 1px solid transparent;

border-radius: 0.25rem;

text-align: center;

}

.message.error {

color: #721c24;

background-color: #f8d7da;

border-color: #f5c6cb;

}

.message.success {

color: #155724;

background-color: #d4edda;

border-color: #c3e6cb;

}

.message.info {

color: #0c5460;

background-color: #d1ecf1;

border-color: #bee5eb;

}

**Step 3: Redux Toolkit Setup (frontend/src/redux)**

Redux Toolkit makes Redux development much simpler.

1. **frontend/src/redux/store.js:**

JavaScript

import { configureStore } from '@reduxjs/toolkit';

import productReducer from './slices/productSlice';

import userReducer from './slices/userSlice';

// Import other reducers as you create them (e.g., cartReducer, orderReducer)

const store = configureStore({

reducer: {

products: productReducer,

user: userReducer,

// Add other reducers here

},

// Middleware is included by default with configureStore (redux-thunk, etc.)

// You can add custom middleware if needed

});

export default store;

1. **frontend/src/redux/slices/productSlice.js:** This slice will handle fetching and managing product data.

JavaScript

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import axios from 'axios';

// Define base URL for your backend API

const API\_URL = 'http://localhost:5000/api/products'; // Ensure this matches your backend port

// Async Thunk for fetching all products

export const fetchProducts = createAsyncThunk(

'products/fetchProducts',

async (\_, { rejectWithValue }) => {

try {

const response = await axios.get(API\_URL);

return response.data;

} catch (error) {

return rejectWithValue(error.response.data.message || error.message);

}

}

);

// Async Thunk for fetching a single product

export const fetchProductDetails = createAsyncThunk(

'products/fetchProductDetails',

async (id, { rejectWithValue }) => {

try {

const response = await axios.get(`<span class="math-inline">\{API\\_URL\}/</span>{id}`);

return response.data;

} catch (error) {

return rejectWithValue(error.response.data.message || error.message);

}

}

);

const productSlice = createSlice({

name: 'products',

initialState: {

allProducts: [],

productDetails: {},

loading: false,

error: null,

},

reducers: {

// You can add sync reducers here if needed

},

extraReducers: (builder) => {

builder

// Handle fetchProducts

.addCase(fetchProducts.pending, (state) => {

state.loading = true;

state.error = null;

})

.addCase(fetchProducts.fulfilled, (state, action) => {

state.loading = false;

state.allProducts = action.payload;

})

.addCase(fetchProducts.rejected, (state, action) => {

state.loading = false;

state.error = action.payload;

})

// Handle fetchProductDetails

.addCase(fetchProductDetails.pending, (state) => {

state.loading = true;

state.error = null;

state.productDetails = {}; // Clear previous details

})

.addCase(fetchProductDetails.fulfilled, (state, action) => {

state.loading = false;

state.productDetails = action.payload;

})

.addCase(fetchProductDetails.rejected, (state, action) => {

state.loading = false;

state.error = action.payload;

state.productDetails = {};

});

},

});

export default productSlice.reducer;

1. **frontend/src/redux/slices/userSlice.js:** This slice will handle user authentication state (login, register, profile).

JavaScript

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import axios from 'axios';

const API\_URL = 'http://localhost:5000/api/users'; // Ensure this matches your backend port

// Get user info from localStorage if it exists

const userInfoFromStorage = localStorage.getItem('userInfo')

? JSON.parse(localStorage.getItem('userInfo'))

: null;

const initialState = {

userInfo: userInfoFromStorage,

loading: false,

error: null,

success: false, // For registration success message

};

// Async Thunk for user login

export const login = createAsyncThunk(

'user/login',

async ({ email, password }, { rejectWithValue }) => {

try {

const config = {

headers: {

'Content-Type': 'application/json',

},

};

const response = await axios.post(`${API\_URL}/login`, { email, password }, config);

localStorage.setItem('userInfo', JSON.stringify(response.data));

return response.data;

} catch (error) {

return rejectWithValue(error.response.data.message || error.message);

}

}

);

// Async Thunk for user registration

export const register = createAsyncThunk(

'user/register',

async ({ name, email, password }, { rejectWithValue }) => {

try {

const config = {

headers: {

'Content-Type': 'application/json',

},

};

const response = await axios.post(`${API\_URL}/register`, { name, email, password }, config);

localStorage.setItem('userInfo', JSON.stringify(response.data)); // Auto-login after register

return response.data;

} catch (error) {

return rejectWithValue(error.response.data.message || error.message);

}

}

);

// Async Thunk for updating user profile

export const updateUserProfile = createAsyncThunk(

'user/updateProfile',

async (user, { getState, rejectWithValue }) => {

try {

const { user: { userInfo } } = getState(); // Get token from current state

const config = {

headers: {

'Content-Type': 'application/json',

Authorization: `Bearer ${userInfo.token}`,

},

};

const response = await axios.put(`${API\_URL}/profile`, user, config);

localStorage.setItem('userInfo', JSON.stringify(response.data));

return response.data;

} catch (error) {

return rejectWithValue(error.response.data.message || error.message);

}

}

);

const userSlice = createSlice({

name: 'user',

initialState,

reducers: {

logout: (state) => {

localStorage.removeItem('userInfo'); // Clear localStorage

state.userInfo = null;

state.loading = false;

state.error = null;

state.success = false;

},

clearUserError: (state) => {

state.error = null;

},

clearUserSuccess: (state) => {

state.success = false;

}

},

extraReducers: (builder) => {

builder

// Handle login

.addCase(login.pending, (state) => {

state.loading = true;

state.error = null;

})

.addCase(login.fulfilled, (state, action) => {

state.loading = false;

state.userInfo = action.payload;

state.error = null; // Clear any previous errors

})

.addCase(login.rejected, (state, action) => {

state.loading = false;

state.error = action.payload;

state.userInfo = null; // Clear user info on login failure

})

// Handle register

.addCase(register.pending, (state) => {

state.loading = true;

state.error = null;

state.success = false;

})

.addCase(register.fulfilled, (state, action) => {

state.loading = false;

state.userInfo = action.payload;

state.success = true; // Indicate successful registration/login

state.error = null;

})

.addCase(register.rejected, (state, action) => {

state.loading = false;

state.error = action.payload;

state.userInfo = null;

state.success = false;

})

// Handle update profile

.addCase(updateUserProfile.pending, (state) => {

state.loading = true;

state.error = null;

state.success = false;

})

.addCase(updateUserProfile.fulfilled, (state, action) => {

state.loading = false;

state.userInfo = action.payload;

state.success = true; // Indicate successful update

state.error = null;

})

.addCase(updateUserProfile.rejected, (state, action) => {

state.loading = false;

state.error = action.payload;

state.success = false;

});

},

});

export const { logout, clearUserError, clearUserSuccess } = userSlice.actions; // Export sync actions

export default userSlice.reducer;

**Step 4: Integrate Redux with React (frontend/src/main.jsx)**

Wrap your App component with Provider to make the Redux store available.

1. **frontend/src/main.jsx:**

JavaScript

import React from 'react';

import ReactDOM from 'react-dom/client';

import { BrowserRouter as Router } from 'react-router-dom';

import { Provider } from 'react-redux';

import store from './redux/store';

import App from './App.jsx';

import './index.css'; // Global styles

ReactDOM.createRoot(document.getElementById('root')).render(

<React.StrictMode>

<Provider store={store}>

<Router>

<App />

</Router>

</Provider>

</React.StrictMode>,

);

**Step 5: Basic React Components (frontend/src/components)**

1. **frontend/src/components/Header.jsx:**

JavaScript

import React from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useSelector, useDispatch } from 'react-redux';

import { logout } from '../redux/slices/userSlice';

const Header = () => {

const dispatch = useDispatch();

const navigate = useNavigate();

const { userInfo } = useSelector((state) => state.user);

const logoutHandler = () => {

dispatch(logout());

navigate('/login'); // Redirect to login after logout

};

return (

<header className="header">

<div className="logo">

<Link to="/">EcomShop</Link>

</div>

<nav>

<ul>

<li>

<Link to="/cart">Cart</Link>

</li>

{userInfo ? (

// Dropdown or simple links for logged-in user

<>

<li>

<Link to="/profile">{userInfo.name}</Link>

</li>

<li>

<a href="#" onClick={logoutHandler}>

Logout

</a>

</li>

</>

) : (

<li>

<Link to="/login">Sign In</Link>

</li>

)}

{/\* Admin Links (will add isAdmin check later) \*/}

{userInfo && userInfo.isAdmin && (

<li>

<Link to="/admin/products">Admin Products</Link>

</li>

)}

</ul>

</nav>

</header>

);

};

export default Header;

1. **frontend/src/components/Footer.jsx:**

JavaScript

import React from 'react';

const Footer = () => {

return (

<footer className="footer">

<p>&copy; {new Date().getFullYear()} EcomShop. All rights reserved.</p>

</footer>

);

};

export default Footer;

1. **frontend/src/components/Loader.jsx:**

JavaScript

import React from 'react';

const Loader = () => {

return (

<div style={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100px' }}>

<div

style={{

border: '4px solid #f3f3f3',

borderTop: '4px solid #3498db',

borderRadius: '50%',

width: '40px',

height: '40px',

animation: 'spin 1s linear infinite',

}}

></div>

<style>

{`

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

`}

</style>

</div>

);

};

export default Loader;

1. **frontend/src/components/Message.jsx:**

JavaScript

import React from 'react';

const Message = ({ variant, children }) => {

// variant can be 'info', 'success', 'error'

return (

<div className={`message ${variant}`}>

{children}

</div>

);

};

Message.defaultProps = {

variant: 'info', };

export default Message;

```

1. **frontend/src/components/ProductCard.jsx:**

JavaScript

import React from 'react';

import { Link } from 'react-router-dom';

const ProductCard = ({ product }) => {

return (

<div className="product-card">

<Link to={`/product/${product.id}`}>

<img src={product.image} alt={product.name} />

</Link>

<div className="product-info">

<Link to={`/product/${product.id}`}>

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

</Link>

<p className="price">${product.price.toFixed(2)}</p>

{/\* You could add rating stars here later \*/}

<button>Add to Cart</button>

</div>

</div>

);

};

export default ProductCard;

**Step 6: React Pages (frontend/src/pages)**

1. **frontend/src/pages/HomePage.jsx:**

JavaScript

import React, { useEffect } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { fetchProducts } from '../redux/slices/productSlice';

import ProductCard from '../components/ProductCard';

import Loader from '../components/Loader';

import Message from '../components/Message';

const HomePage = () => {

const dispatch = useDispatch();

const { allProducts, loading, error } = useSelector((state) => state.products);

useEffect(() => {

dispatch(fetchProducts());

}, [dispatch]);

return (

<div className="container">

<h1>Latest Products</h1>

{loading ? (

<Loader />

) : error ? (

<Message variant="error">{error}</Message>

) : (

<div className="product-grid">

{allProducts.map((product) => (

<ProductCard key={product.id} product={product} />

))}

</div>

)}

</div>

);

};

export default HomePage;

1. **frontend/src/pages/ProductDetailsPage.jsx:**

JavaScript

import React, { useEffect } from 'react';

import { useDispatch, useSelector } from 'react-redux';

import { useParams, Link } from 'react-router-dom';

import { fetchProductDetails } from '../redux/slices/productSlice';

import Loader from '../components/Loader';

import Message from '../components/Message';

const ProductDetailsPage = () => {

const { id } = useParams(); // Get product ID from URL

const dispatch = useDispatch();

const { productDetails, loading, error } = useSelector((state) => state.products);

useEffect(() => {

dispatch(fetchProductDetails(id));

}, [dispatch, id]);

return (

<div className="container">

<Link className="btn btn-light" to="/" style={{display: 'inline-block', marginTop: '1rem', marginBottom: '1rem', padding: '0.5rem 1rem', backgroundColor: '#f4f4f4', border: '1px solid #ddd', borderRadius: '5px', textDecoration: 'none', color: '#333'}}>

Go Back

</Link>

{loading ? (

<Loader />

) : error ? (

<Message variant="error">{error}</Message>

) : productDetails.id ? ( // Check if productDetails is populated

<div style={{ display: 'flex', gap: '2rem', flexWrap: 'wrap', backgroundColor: '#fff', padding: '2rem', borderRadius: '8px', boxShadow: '0 4px 8px rgba(0,0,0,0.1)' }}>

<div style={{ flex: 1, minWidth: '300px' }}>

<img src={productDetails.image} alt={productDetails.name} style={{ maxWidth: '100%', height: 'auto', borderRadius: '8px' }} />

</div>

<div style={{ flex: 2, minWidth: '400px', textAlign: 'left' }}>

<h2>{productDetails.name}</h2>

<p><strong>Brand:</strong> {productDetails.brand}</p>

<p><strong>Category:</strong> {productDetails.category}</p>

<p><strong>Description:</strong> {productDetails.description}</p>

<p style={{ fontSize: '1.8rem', fontWeight: 'bold', color: '#007bff' }}>Price: ${productDetails.price.toFixed(2)}</p>

<p><strong>Status:</strong> {productDetails.countInStock > 0 ? 'In Stock' : 'Out of Stock'}</p>

<button

style={{

backgroundColor: '#007bff',

color: 'white',

border: 'none',

padding: '0.8rem 1.2rem',

borderRadius: '5px',

cursor: 'pointer',

fontSize: '1rem',

}}

disabled={productDetails.countInStock === 0}

>

Add to Cart

</button>

</div>

{/\* Product Reviews Section can go here later \*/}

</div>

) : (

<Message variant="info">Product not found.</Message> // Fallback if productDetails is empty but no error

)}

</div>

);

};

export default ProductDetailsPage;

1. **frontend/src/pages/LoginPage.jsx:**

JavaScript

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

import { Link, useNavigate, useLocation } from 'react-router-dom';

import { useDispatch, useSelector } from 'react-redux';

import { login, clearUserError } from '../redux/slices/userSlice';

import Loader from '../components/Loader';

import Message from '../components/Message';

const LoginPage = () => {

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

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

const dispatch = useDispatch();

const navigate = useNavigate();

const location = useLocation();

const { loading, error, userInfo } = useSelector((state) => state.user);

// Redirect if already logged in

const redirect = location.search ? location.search.split('=')[1] : '/';

useEffect(() => {

if (userInfo) {

navigate(redirect);

}

// Clear error when component mounts or unmounts to prevent stale messages

return () => {

dispatch(clearUserError());

};

}, [navigate, userInfo, redirect, dispatch]);

const submitHandler = (e) => {

e.preventDefault();

dispatch(login({ email, password }));

};

return (

<div className="form-container">

<h2>Sign In</h2>

{error && <Message variant="error">{error}</Message>}

{loading && <Loader />}

<form onSubmit={submitHandler}>

<div className="form-group">

<label htmlFor="email">Email Address</label>

<input

type="email"

id="email"

placeholder="Enter email"

value={email}

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

required

/>

</div>

<div className="form-group">

<label htmlFor="password">Password</label>

<input

type="password"

id="password"

placeholder="Enter password"

value={password}

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

required

/>

</div>

<button type="submit">Sign In</button>

</form>

<div className="text-center">

New Customer?{' '}

<Link to={redirect ? `/register?redirect=${redirect}` : '/register'}>

Register

</Link>

</div>

</div>

);

};

export default LoginPage;

1. **frontend/src/pages/RegisterPage.jsx:**

JavaScript

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

import { Link, useNavigate, useLocation } from 'react-router-dom';

import { useDispatch, useSelector } from 'react-redux';

import { register, clearUserError } from '../redux/slices/userSlice';

import Loader from '../components/Loader';

import Message from '../components/Message';

const RegisterPage = () => {

const [name, setName] = useState('');

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

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

const [confirmPassword, setConfirmPassword] = useState('');

const [message, setMessage] = useState(null); // For local messages like password mismatch

const dispatch = useDispatch();

const navigate = useNavigate();

const location = useLocation();

const { loading, error, userInfo } = useSelector((state) => state.user);

const redirect = location.search ? location.search.split('=')[1] : '/';

useEffect(() => {

if (userInfo) {

navigate(redirect);

}

// Clear error when component mounts or unmounts to prevent stale messages

return () => {

dispatch(clearUserError());

setMessage(null); // Clear local messages too

};

}, [navigate, userInfo, redirect, dispatch]);

const submitHandler = (e) => {

e.preventDefault();

setMessage(null); // Clear previous messages

if (password !== confirmPassword) {

setMessage('Passwords do not match');

} else {

dispatch(register({ name, email, password }));

}

};

return (

<div className="form-container">

<h2>Sign Up</h2>

{message && <Message variant="error">{message}</Message>}

{error && <Message variant="error">{error}</Message>}

{loading && <Loader />}

<form onSubmit={submitHandler}>

<div className="form-group">

<label htmlFor="name">Name</label>

<input

type="text"

id="name"

placeholder="Enter name"

value={name}

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

required

/>

</div>

<div className="form-group">

<label htmlFor="email">Email Address</label>

<input

type="email"

id="email"

placeholder="Enter email"

value={email}

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

required

/>

</div>

<div className="form-group">

<label htmlFor="password">Password</label>

<input

type="password"

id="password-register"

placeholder="Enter password"

value={password}

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

required

/>

</div>

<div className="form-group">

<label htmlFor="confirmPassword">Confirm Password</label>

<input

type="password"

id="confirmPassword"

placeholder="Confirm password"

value={confirmPassword}

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

required

/>

</div>

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

</form>

<div className="text-center">

Have an Account? <Link to={redirect ? `/login?redirect=${redirect}` : '/login'}>Login</Link>

</div>

</div>

);

};

export default RegisterPage;

**Step 7: Main App Component & Routing (frontend/src/App.jsx)**

This component will set up the main layout and define routes.

1. **frontend/src/App.jsx:**

JavaScript

import React from 'react';

import { Routes, Route } from 'react-router-dom';

import Header from './components/Header';

import Footer from './components/Footer';

import HomePage from './pages/HomePage';

import ProductDetailsPage from './pages/ProductDetailsPage';

import LoginPage from './pages/LoginPage';

import RegisterPage from './pages/RegisterPage';

import './App.css'; // Main App styles

function App() {

return (

<>

<Header />

<main className="main-content">

<div className="container"> {/\* Added a container for consistent padding \*/}

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/product/:id" element={<ProductDetailsPage />} />

<Route path="/login" element={<LoginPage />} />

<Route path="/register" element={<RegisterPage />} />

{/\* Add more routes here as you build them \*/}

</Routes>

</div>

</main>

<Footer />

</>

);

}

export default App;

**Step 8: Run the Frontend**

1. **Ensure your backend server is still running** (from Phase 1: cd backend then npm run dev).
2. **Navigate to the frontend directory:**

Bash

cd frontend

1. **Start the React development server:**

Bash

npm run dev

Vite will usually tell you which URL to open (e.g., http://localhost:5173/).

**Verification and Testing:**

1. **Open your browser** to the frontend URL (e.g., http://localhost:5173/).
2. You should see the header, an empty "Latest Products" section (as you haven't added product images yet), and the footer.
3. **Register a New User:**
   * Click "Sign In" then "Register".
   * Fill in details and register. You should be automatically logged in and redirected to the home page.
   * Check your browser's developer tools -> Application -> Local Storage. You should see userInfo stored there.
4. **Login with Existing User:**
   * Click "Logout" from the header (if logged in).
   * Click "Sign In" and use the credentials of the user you just registered.
5. **Product Display:**
   * Since you've already created a sample product via Postman/Insomnia in the backend, the HomePage should now fetch and display that product.
   * The ProductCard will show product name and price.
   * Click on a product card to go to the ProductDetailsPage. It should display the product details fetched from the backend.
6. **Check Console:** Watch your browser's console for any errors.

**Congratulations! You've completed Phase 2: Frontend Setup.**

You now have a functional frontend communicating with your backend, displaying products, and handling user authentication (register/login/logout).

**Next Steps (Future Phases):**

* **Phase 3: Shopping Cart & Order Flow:** Implement adding to cart, cart page, shipping address, payment methods, and placing orders.
* **Phase 4: Admin Functionality:** Build out the admin panel for managing products, users, and orders.
* **Phase 5: Reviews, Search, Pagination & Deployment:** Add reviews, refine search/filter/pagination, and prepare for deployment.

Let me know if you encounter any issues during this phase, and we can troubleshoot them!