1. Backend (Node.js and Express)

Project Setup

1. Initialize a new Node.js project:

```
mkdir ecommerce-app
cd ecommerce-app
npm init -y
npm install express mongoose bcryptjs jsonwebtoken dotenv
npm install --save-dev nodemon
```

2. Add the following scripts in package.json:

```
"scripts": {
  "start": "node backend/server.js",
  "dev": "nodemon backend/server.js"
}
```

Create the Project Structure

```
mkdir backend
cd backend
mkdir models routes controllers middleware config
touch server.js
```

2. Backend Code

2.1. server.js (Main Entry Point)

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const authRoutes = require('./routes/auth');
const productRoutes = require('./routes/product');
const cartRoutes = require('./routes/cart');
const { protect } = require('./middleware/authMiddleware');

dotenv.config();
```

```
const app = express();

// Middleware
app.use(express.json());

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/products', productRoutes);
app.use('/api/cart', protect, cartRoutes); // Protected route

// MongoDB connection
mongoose
  .connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

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

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

2.2. Authentication Routes (auth.js)

Models/userModel.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  isAdmin: { type: Boolean, default: false },
}, { timestamps: true });
```

```
// Password hashing
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) {
    next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

module.exports = mongoose.model('User', userSchema);
```

Routes/auth.js

```
const express = require('express');
const { registerUser, loginUser } = require('../controllers/authController');
const router = express.Router();

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

module.exports = router;
```

Controllers/authController.js

```
const User = require('../models/userModel');
const jwt = require('jsonwebtoken');

// Generate JWT token
const generateToken = (id) => {
```

```
  return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: '30d' });
};

// Register new user
const registerUser = async (req, res) => {
  const { name, email, password } = req.body;

  const userExists = await User.findOne({ email });
  if (userExists) {
    return res.status(400).json({ message: 'User already exists' });
  }

  const user = await User.create({ name, email, password });

  if (user) {
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
      token: generateToken(user._id),
    });
  } else {
    res.status(400).json({ message: 'Invalid user data' });
  }
};

// Login user
const loginUser = async (req, res) => {
  const { email, password } = req.body;
```

```
  const user = await User.findOne({ 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).json({ message: 'Invalid email or password' });
  }
};

module.exports = { registerUser, loginUser };
```

Middleware/authMiddleware.js

```
const jwt = require('jsonwebtoken');
const User = require('../models/userModel');

const protect = async (req, res, next) => {
  let token;
  if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select('-password');
      next();
    } catch (error) {
      res.status(401).json({ message: 'Not authorized, token failed' });
    }
```

```
  }

  if (!token) {
    res.status(401).json({ message: 'Not authorized, no token' });
  }
};

module.exports = { protect };
```

2.3. Product Routes (product.js)

Models/productModel.js

```
const mongoose = require('mongoose');

const productSchema = mongoose.Schema({
  name: { type: String, required: true },
  description: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: String, required: true },
  image: { type: String, required: true },
  stock: { type: Number, required: true },
}, { timestamps: true });

module.exports = mongoose.model('Product', productSchema);
```

Routes/product.js

```
const express = require('express');
const { getProducts, getProductById, createProduct, updateProduct, deleteProduct } =
require('../controllers/productController');
const { protect } = require('../middleware/authMiddleware');
const router = express.Router();
```

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

router.route('/:id')
  .get(getProductById)
  .put(protect, updateProduct)
  .delete(protect, deleteProduct);

module.exports = router;
```

Controllers/productController.js

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

// Get all products
const getProducts = async (req, res) => {
  const products = await Product.find({});
  res.json(products);
};

// Get product by ID
const getProductById = async (req, res) => {
  const product = await Product.findById(req.params.id);
  if (product) {
    res.json(product);
  } else {
    res.status(404).json({ message: 'Product not found' });
  }
};

// Create a new product (admin only)
```

```
const createProduct = async (req, res) => {
  const { name, description, price, category, image, stock } = req.body;
  const product = new Product({ name, description, price, category, image, stock });
  const createdProduct = await product.save();
  res.status(201).json(createdProduct);
};

// Update product (admin only)
const updateProduct = async (req, res) => {
  const product = await Product.findById(req.params.id);
  if (product) {
    product.name = req.body.name || product.name;
    product.description = req.body.description || product.description;
    product.price = req.body.price || product.price;
    product.category = req.body.category || product.category;
    product.image = req.body.image || product.image;
    product.stock = req.body.stock || product.stock;

    const updatedProduct = await product.save();
    res.json(updatedProduct);
  } else {
    res.status(404).json({ message: 'Product not found' });
  }
};

// Delete product (admin only)
const deleteProduct = async (req, res) => {
  const product = await Product.findById(req.params.id);
  if (product) {
    await product.remove();
    res.json({ message: 'Product removed' });
```

```
  } else {
    res.status(404).json({ message: 'Product not found' });
  }
};

module.exports = { getProducts, getProductById, createProduct, updateProduct, deleteProduct };
```

2.4. Cart Routes (cart.js)

Models/cartModel.js

```
const mongoose = require('mongoose');

const cartSchema = mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  items: [
    {
      productId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },
      quantity: { type: Number, required: true, default: 1 },
    },
  ],
  total: { type: Number, required: true },
}, { timestamps: true });

module.exports = mongoose.model('Cart', cartSchema);
```

Routes/cart.js

```
const express = require('express');
const { getCart, addItemToCart, removeItemFromCart } = require('../controllers/cartController');
const router = express.Router();

router.route('/')
  .get(getCart)
```

```
  .post(addItemToCart);

router.route('/:id')
  .delete(removeItemFromCart);

module.exports = router;
```

Controllers/cartController.js

```
const Cart = require('../models/cartModel');
const Product = require('../models/productModel');

// Get user's cart
const getCart = async (req, res) => {
  const cart = await Cart.findOne({ userId: req.user._id });
  if (cart) {
    res.json(cart);
  } else {
    res.status(404).json({ message: 'Cart not found' });
  }
};

// Add item to cart
const addItemToCart = async (req, res) => {
  const { productId, quantity } = req.body;
  const product = await Product.findById(productId);

  if (!product) {
    return res.status(404).json({ message: 'Product not found' });
  }

  let cart = await Cart.findOne({ userId: req.user._id });
```

```
  if (!cart) {
    cart = new Cart({ userId: req.user._id, items: [], total: 0 });
  }

  const existingItemIndex = cart.items.findIndex(item => item.productId.toString() === productId);

  if (existingItemIndex !== -1) {
    cart.items[existingItemIndex].quantity += quantity;
  } else {
    cart.items.push({ productId, quantity });
  }

  cart.total += product.price * quantity;

  await cart.save();
  res.json(cart);
};

// Remove item from cart
const removeItemFromCart = async (req, res) => {
  const cart = await Cart.findOne({ userId: req.user._id });
  if (!cart) {
    return res.status(404).json({ message: 'Cart not found' });
  }

  const itemIndex = cart.items.findIndex(item => item._id.toString() === req.params.id);
  if (itemIndex !== -1) {
    const removedItem = cart.items[itemIndex];
    cart.total -= removedItem.quantity * removedItem.productId.price;
    cart.items.splice(itemIndex, 1);
```

```
  }

  await cart.save();
  res.json(cart);
};

module.exports = { getCart, addItemToCart, removeItemFromCart };
```

3. Frontend (React)

Project Setup

1. Initialize a React project:

```
npx create-react-app frontend
cd frontend
npm install axios redux react-redux redux-thunk react-router-dom
```

2. Set up the folder structure:

```
mkdir src/components src/redux
touch src/redux/store.js
```

3.1. Redux Setup

redux/store.js

```
import { createStore, combineReducers, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import { composeWithDevTools } from 'redux-devtools-extension';

const rootReducer = combineReducers({
  // reducers
});

const store = createStore(
  rootReducer,
```

```
  composeWithDevTools(applyMiddleware(thunk))
);

export default store;
```

3.2. Components

components/AuthForm.js

```
import React, { useState } from 'react';
import axios from 'axios';

const AuthForm = ({ isLogin }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [name, setName] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    const endpoint = isLogin ? '/api/auth/login' : '/api/auth/register';
    const data = isLogin ? { email, password } : { name, email, password };

    try {
      const response = await axios.post(endpoint, data);
      console.log(response.data);
      // Save JWT token in local storage
      localStorage.setItem('token', response.data.token);
    } catch (error) {
      console.error(error);
    }
  };

  return (
```

```
    <form onSubmit={handleSubmit}>
      {!isLogin && <input type="text" placeholder="Name" value={name} onChange={(e) => setName(e.target.value)} />}
      <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">{isLogin ? 'Login' : 'Register'}</button>
    </form>
  );
};

export default AuthForm;
```

components/ProductList.js

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

const ProductList = () => {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    const fetchProducts = async () => {
      const response = await axios.get('/api/products');
      setProducts(response.data);
    };
    fetchProducts();
  }, []);

  return (
    <div>
```

```
      {products.map(product => (
        <div key={product._id}>
          <h3>{product.name}</h3>
          <p>{product.description}</p>
          <p>{product.price}</p>
        </div>
      ))}
    </div>
  );
};

export default ProductList;
```

components/Cart.js

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

const Cart = () => {
  const [cart, setCart] = useState({ items: [], total: 0 });

  useEffect(() => {
    const fetchCart = async () => {
      const token = localStorage.getItem('token');
      const response = await axios.get('/api/cart', {
        headers: { Authorization: `Bearer ${token}` }
      });
      setCart(response.data);
    };
    fetchCart();
  }, []);
```

```
  return (
    <div>
      <h2>Shopping Cart</h2>
      {cart.items.map(item => (
        <div key={item.productId}>
          <p>Product ID: {item.productId}</p>
          <p>Quantity: {item.quantity}</p>
        </div>
      ))}
      <h3>Total: {cart.total}</h3>
    </div>
  );
};

export default Cart;
```

4. Run the Application

1. **Backend**:

```
cd backend
npm run dev
```

2.

3. **Frontend**:

```
cd frontend
npm start
```