18. Project Structure

Directory Structure

```
shopsphere/
├── client/
│   ├── public/
│   │   ├── images/
│   │   ├── favicon.ico
│   │   └── index.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── Common/
│   │   │   ├── Customer/
│   │   │   ├── Filter/
│   │   │   ├── Layout/
│   │   │   ├── Product/
│   │   │   ├── Search/
│   │   │   └── ShopOwner/
│   │   ├── contexts/
│   │   ├── hooks/
│   │   ├── pages/
│   │   ├── services/
│   │   ├── styles/
│   │   ├── utils/
│   │   ├── App.js
│   │   ├── index.js
│   │   └── routes.js
│   ├── .env
│   ├── .gitignore
│   ├── package.json
│   └── README.md
├── server/
│   ├── config/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── tests/
│   ├── utils/
│   ├── .env
│   ├── .gitignore
│   ├── app.js
│   ├── package.json
│   └── server.js
├── .gitignore
├── docker-compose.yml
├── package.json
└── README.md
```

Backend MVC Structure

```
server/
├── controllers/  # Handle request/response logic
│   ├── authController.js
│   ├── cartController.js
│   ├── categoryController.js
│   ├── orderController.js
│   ├── paymentController.js
│   ├── productController.js
│   └── userController.js
├── models/       # Database schemas
│   ├── Cart.js
│   ├── Category.js
│   ├── Order.js
│   ├── Product.js
│   ├── Promotion.js
│   ├── Shop.js
│   └── User.js
├── routes/       # API endpoints
│   ├── authRoutes.js
│   ├── cartRoutes.js
│   ├── categoryRoutes.js
│   ├── orderRoutes.js
│   ├── paymentRoutes.js
│   ├── productRoutes.js
│   └── userRoutes.js
└── services/     # Business logic
    ├── authService.js
    ├── emailService.js
    ├── orderService.js
    ├── paymentService.js
    └── uploadService.js
```

Frontend Component Structure

```
src/
├── components/
│   ├── Common/          # Reusable UI components
│   │   ├── Button/
│   │   ├── Card/
│   │   ├── FormInput/
│   │   ├── Modal/
│   │   ├── Pagination/
│   │   └── Rating/
│   ├── Customer/        # Customer dashboard components
│   │   ├── Addresses/
│   │   ├── Orders/
│   │   ├── Profile/
│   │   └── Wishlist/
│   ├── Layout/          # Layout components
│   │   ├── Footer/
│   │   ├── Header/
│   │   ├── Navigation/
│   │   └── Sidebar/
│   ├── Product/         # Product-related components
│   │   ├── ProductCard/
│   │   ├── ProductGrid/
│   │   ├── ProductReviews/
│   │   └── RelatedProducts/
│   └── ShopOwner/       # Shop owner dashboard components
│       ├── Analytics/
│       ├── OrderManagement/
│       ├── ProductManagement/
│       └── PromotionManagement/
└── pages/              # Page components
    ├── Authentication/
    │   ├── Login.js
    │   └── Register.js
    ├── Cart.js
    ├── Checkout.js
    ├── CustomerDashboard.js
    ├── Home.js
    ├── OrderConfirmation.js
    ├── ProductDetail.js
    ├── ProductListing.js
    └── ShopOwnerDashboard.js
```

19. Implementation Timeline

Phase 1: Project Setup and Core Features (Weeks 1-2)

- Set up project structure (frontend/backend)
- Implement database schema and models
- Create basic UI components and layouts
- Implement authentication system
- Develop product listing and details

Phase 2: User Management & Shopping (Weeks 3-4)

- Implement customer dashboard
- Develop shop owner dashboard
- Create shopping cart functionality

- Implement product search and filtering
- Set up order creation and management

Phase 3: Advanced Features (Weeks 5-6)

- Integrate payment gateway
- Implement promotions and discounts
- Develop analytics for shop owners
- Create notification system
- Implement product reviews and ratings

Phase 4: Testing & Deployment (Weeks 7-8)

- Write unit tests for components and APIs
- Conduct integration testing
- Perform end-to-end testing
- Set up deployment configuration
- Deploy to production environment

20. Conclusion

This document provides a comprehensive guide for developing the SHOPSPHERE e-commerce platform using React, Node.js, and MongoDB. It covers all aspects of the project from database design to frontend implementation, security, testing, and deployment.

The platform features dual login systems for customers and shop owners, product management, shopping cart functionality, payment processing, and more. Follow the implementation timeline to systematically develop each part of the application.

For questions or clarifications, refer to the appropriate sections or consult the technical team.

Happy coding!# SHOPSPHERE E-Commerce Platform: Complete Development Guide

Table of Contents

1. System Overview

SHOPSPHERE is a multi-vendor e-commerce platform where:

- Shop owners can list and sell products
- Customers can browse, filter, and purchase products
- The system handles payments, order management, and user authentication
- Multiple product categories are supported with a filtering system
- The platform features promotional capabilities like flash sales

Key Features

- Dual login system (Shop owners & Customers)
- Social media authentication (Google, Facebook, Apple)
- Product catalog with categories and filters
- Shopping cart and checkout system
- Payment gateway integration
- User dashboards (separate for customers and sellers)
- Flash sales and promotional features
- Order tracking system

2. Technology Stack

Frontend

- **React**: For building the user interface
- **Redux/Context API**: State management
- **React Router**: Page navigation

- **Axios**: API requests
- **styled-components/SCSS**: Styling
- **Material UI/Bootstrap**: UI components (optional)
- **React Hook Form**: Form validation
- **Firebase**: Social authentication

Backend

- **Node.js**: Server runtime
- **Express.js**: Web framework
- **MongoDB**: Database
- **Mongoose**: MongoDB ODM
- **JWT**: Authentication
- **Passport.js**: Authentication strategies
- **Stripe/PayPal**: Payment processing
- **Multer/Cloudinary**: File/image uploads
- **Nodemailer**: Email notifications

DevOps

- **Git**: Version control
- **Docker**: Containerization
- **AWS/Heroku**: Deployment
- **Jest/Mocha**: Testing
- **CI/CD**: Continuous integration/deployment

3. System Architecture

Three-Tier Architecture

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│    Frontend     │     │     Backend     │     │    Database     │
│    (React)      │◄───►│   (Node.js)     │◄───►│   (MongoDB)     │
└─────────────────┘     └─────────────────┘     └─────────────────┘
```

Microservices Breakdown

1. **Authentication Service**: Handles user registration, login, and social auth
2. **Product Service**: Manages product catalog and inventory
3. **Order Service**: Processes orders and tracks status

4. **Payment Service**: Integrates with payment gateways
5. **User Service**: Manages user profiles and preferences
6. **Notification Service**: Handles emails, alerts, and notifications

4. Database Schema

Collections in MongoDB

Users Collection

```
{
  _id: ObjectId,
  userType: String, // "customer" or "shop_owner"
  email: String,
  password: String, // hashed
  firstName: String,
  lastName: String,
  profileImage: String, // URL
  phoneNumber: String,
  addresses: [{
    addressType: String, // "shipping" or "billing"
    street: String,
    city: String,
    state: String,
    zipCode: String,
    country: String,
    isDefault: Boolean
  }],
  authProvider: {
    type: String, // "local", "google", "facebook", "apple"
    providerId: String
  },
  createdAt: Date,
  updatedAt: Date
}
```

Shops Collection

```
{
  _id: ObjectId,
  ownerId: ObjectId, // reference to Users collection
  shopName: String,
  logo: String, // URL
  description: String,
  contact: {
    email: String,
    phone: String
  },
  address: {
    street: String,
    city: String,
    state: String,
    zipCode: String,
    country: String
  },
  businessInfo: {
    registrationNumber: String,
    taxId: String
  },
  isVerified: Boolean,
  rating: Number,
  createdAt: Date,
  updatedAt: Date
}
```

Products Collection

```
  {
    _id: ObjectId,
    shopId: ObjectId, // reference to Shops collection
    name: String,
    description: String,
    price: Number,
    salePrice: Number, // optional, for discounts
    category: String,
    subCategory: String,
    images: [String], // array of URLs
    attributes: {
      // dynamic fields based on product type
      color: String,
      size: String,
      weight: Number,
      // etc.
    },
    inventory: {
      quantity: Number,
      sku: String
    },
    rating: Number,
    reviews: [{
      userId: ObjectId,
      rating: Number,
      comment: String,
      createdAt: Date
    }],
    isActive: Boolean,
    isPromoted: Boolean, // for flash sales or promotions
    createdAt: Date,
    updatedAt: Date
  }
```

Categories Collection

```
{
  _id: ObjectId,
  name: String,
  description: String,
  image: String, // URL
  parentId: ObjectId, // for sub-categories
  isActive: Boolean,
  createdAt: Date,
  updatedAt: Date
}
```

Carts Collection

```
{
  _id: ObjectId,
  userId: ObjectId, // reference to Users collection
  items: [{
    productId: ObjectId,
    quantity: Number,
    price: Number,
    attributes: Object // selected product attributes
  }],
  totalAmount: Number,
  createdAt: Date,
  updatedAt: Date
}
```

Orders Collection

```
{
  _id: ObjectId,
  orderNumber: String,
  userId: ObjectId, // reference to Users collection
  items: [{
    productId: ObjectId,
    shopId: ObjectId,
    quantity: Number,
    price: Number,
    attributes: Object // selected product attributes
  }],
  billing: {
    address: Object,
    subtotal: Number,
    shipping: Number,
    discount: Number,
    total: Number
  },
  shipping: {
    address: Object,
    method: String,
    trackingNumber: String,
    estimatedDelivery: {
      from: Date,
      to: Date
    }
  },
  payment: {
    method: String,
    transactionId: String,
    status: String // "pending", "completed", "failed"
  },
  status: String, // "pending", "processing", "shipped", "delivered", "cancelled"
  createdAt: Date,
  updatedAt: Date
}
```

Promotions Collection

```
{
  _id: ObjectId,
  name: String,
  description: String,
  type: String, // "flash_sale", "discount", "coupon"
  value: Number, // discount amount or percentage
  code: String, // for coupons
  applicableProducts: [ObjectId], // or empty for store-wide
  applicableCategories: [ObjectId],
  minimumPurchase: Number,
  startDate: Date,
  endDate: Date,
  isActive: Boolean,
  createdAt: Date,
  updatedAt: Date
}
```

5. Frontend Structure

Component Hierarchy

```
App
├── Layout
```

```
│   ├── Header
│   │   ├── Logo
│   │   ├── SearchBar
│   │   ├── Navigation
│   │   ├── CartIcon
│   │   └── UserMenu
│   ├── Footer
│   └── SideNav (for filters)
├── Pages
│   ├── Home
│   │   ├── FlashSaleBanner
│   │   ├── CategoryList
│   │   └── FeaturedProducts
│   ├── ProductListing
│   │   ├── FilterSidebar
│   │   ├── ProductGrid
│   │   └── Pagination
│   ├── ProductDetail
│   │   ├── ImageGallery
│   │   ├── ProductInfo
│   │   ├── AddToCartButton
│   │   └── ReviewSection
│   ├── Cart
│   │   ├── CartItemList
│   │   ├── CartSummary
│   │   └── CheckoutButton
│   ├── Checkout
│   │   ├── ShippingForm
│   │   ├── PaymentForm
│   │   └── OrderSummary
│   ├── Authentication
│   │   ├── LoginForm
│   │   ├── RegisterForm
│   │   └── SocialLogin
│   ├── CustomerDashboard
│   │   ├── Profile
│   │   ├── Orders
│   │   ├── Wishlist
│   │   └── Addresses
│   └── ShopOwnerDashboard
│       ├── Analytics
│       ├── ProductManagement
│       ├── OrderManagement
│       └── PromotionManagement
└── Common
    ├── Button
    ├── Input
    ├── Card
    ├── Modal
    ├── Dropdown
    └── Loader
```

Key Pages and Routing

```
// src/App.js
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Layout from './components/Layout';
import Home from './pages/Home';
import ProductListing from './pages/ProductListing';
import ProductDetail from './pages/ProductDetail';
import Cart from './pages/Cart';
import Checkout from './pages/Checkout';
import Login from './pages/Authentication/Login';
import Register from './pages/Authentication/Register';
import CustomerDashboard from './pages/CustomerDashboard';
import ShopOwnerDashboard from './pages/ShopOwnerDashboard';
import ProtectedRoute from './components/ProtectedRoute';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="products" element={<ProductListing />} />
          <Route path="products/:id" element={<ProductDetail />} />
          <Route path="cart" element={<Cart />} />
          <Route path="checkout" element={<Checkout />} />
          <Route path="login" element={<Login />} />
          <Route path="register" element={<Register />} />
          <Route path="dashboard" element={
            <ProtectedRoute>
              <CustomerDashboard />
            </ProtectedRoute>
          } />
          <Route path="seller-dashboard" element={
            <ProtectedRoute requiredRole="shop_owner">
              <ShopOwnerDashboard />
            </ProtectedRoute>
          } />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

6. Authentication System

User Authentication Flow

1. **Registration Process**

```
// src/pages/Authentication/Register.js
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Register = () => {
  const [formData, setFormData] = useState({
    email: '',
    password: '',
    confirmPassword: '',
    firstName: '',
    lastName: '',
    userType: 'customer' // default to customer
  });
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (formData.password !== formData.confirmPassword) {
      setError('Passwords do not match');
      return;
    }

    try {
      const response = await axios.post('/api/auth/register', formData);
      localStorage.setItem('token', response.data.token);
      navigate(formData.userType === 'customer' ? '/dashboard' : '/seller-dashboard');
    } catch (error) {
      setError(error.response?.data?.message || 'Registration failed');
    }
  };

  return (
    <div className="register-container">
      <h2>Create an Account</h2>
      {error && <div className="error">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>First Name</label>
          <input
            type="text"
            name="firstName"
            value={formData.firstName}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Last Name</label>
          <input
            type="text"
            name="lastName"
            value={formData.lastName}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Email</label>
          <input
            type="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Password</label>
          <input
            type="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Confirm Password</label>
          <input
            type="password"
            name="confirmPassword"
            value={formData.confirmPassword}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Account Type</label>
          <select name="userType" value={formData.userType} onChange={handleChange}>
            <option value="customer">Customer</option>
            <option value="shop_owner">Shop Owner</option>
          </select>
        </div>
        <button type="submit" className="btn-primary">Register</button>
      </form>
      <div className="social-login">
        <p>Or register with:</p>
        <button className="btn-google">Google</button>
        <button className="btn-facebook">Facebook</button>
        <button className="btn-apple">Apple</button>
      </div>
      <p>
        Already have an account? <a href="/login">Login</a>
      </p>
    </div>
  );
};

export default Register;
```

2. Login Process

```
// src/pages/Authentication/Login.js
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Login = () => {
  const [formData, setFormData] = useState({
    email: '',
    password: '',
    userType: 'customer' // default to customer
  });
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('/api/auth/login', formData);
      localStorage.setItem('token', response.data.token);
      navigate(formData.userType === 'customer' ? '/dashboard' : '/seller-dashboard');
    } catch (error) {
      setError(error.response?.data?.message || 'Login failed');
    }
  };

  return (
    <div className="login-container">
      <h2>Login to Your Account</h2>
      {error && <div className="error">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Email</label>
          <input
            type="email"
            name="email"
            value={formData.email}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Password</label>
          <input
            type="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label>Account Type</label>
          <select name="userType" value={formData.userType} onChange={handleChange}>
            <option value="customer">Customer</option>
            <option value="shop_owner">Shop Owner</option>
          </select>
        </div>
        <div className="form-group">
          <a href="/forgot-password">Forgot Password?</a>
        </div>
        <button type="submit" className="btn-primary">Login</button>
      </form>
      <div className="social-login">
        <p>Or login with:</p>
        <button className="btn-google">Google</button>
        <button className="btn-facebook">Facebook</button>
        <button className="btn-apple">Apple</button>
      </div>
      <p>
        Don't have an account? <a href="/register">Register</a>
      </p>
    </div>
  );
};

export default Login;
```

3. Social Authentication

```
// src/services/authService.js
import firebase from 'firebase/app';
import 'firebase/auth';
import axios from 'axios';
```

```
// Initialize Firebase
const firebaseConfig = {
  // Your Firebase config
};

firebase.initializeApp(firebaseConfig);

export const socialLogin = async (provider) => {
  try {
    let socialProvider;

    switch (provider) {
      case 'google':
        socialProvider = new firebase.auth.GoogleAuthProvider();
        break;
      case 'facebook':
        socialProvider = new firebase.auth.FacebookAuthProvider();
        break;
      case 'apple':
        socialProvider = new firebase.auth.OAuthProvider('apple.com');
        break;
      default:
        throw new Error('Invalid provider');
    }

    const result = await firebase.auth().signInWithPopup(socialProvider);
    const user = result.user;

    // Get the token from Firebase
    const idToken = await user.getIdToken();

    // Send the token to our backend
    const response = await axios.post('/api/auth/social-login', {
      idToken,
      provider,
      email: user.email,
      name: user.displayName,
      photoURL: user.photoURL
    });

    // Save our JWT token
    localStorage.setItem('token', response.data.token);

    return response.data.user;
  } catch (error) {
    throw error;
  }
};
```

4. Protected Routes

```
// src/components/ProtectedRoute.js
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const ProtectedRoute = ({ children, requiredRole }) => {
  const { user, isLoading } = useAuth();

  if (isLoading) {
    return <div>Loading...</div>;
  }

  if (!user) {
    return <Navigate to="/login" />;
  }

  if (requiredRole && user.userType !== requiredRole) {
    return <Navigate to="/" />;
  }

  return children;
};

export default ProtectedRoute;
```

5. **Auth Context Setup**

```
// src/contexts/AuthContext.js
import React, { createContext, useContext, useState, useEffect } from 'react';
import axios from 'axios';

const AuthContext = createContext();

export const useAuth = () => useContext(AuthContext);

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const fetchUser = async () => {
      try {
        const token = localStorage.getItem('token');

        if (!token) {
          setIsLoading(false);
          return;
        }

        // Add token to axios headers
```

```
          axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;

          const response = await axios.get('/api/auth/me');
          setUser(response.data);
        } catch (error) {
          localStorage.removeItem('token');
        } finally {
          setIsLoading(false);
        }
      };

      fetchUser();
    }, []);

    const login = async (credentials) => {
      const response = await axios.post('/api/auth/login', credentials);
      localStorage.setItem('token', response.data.token);
      setUser(response.data.user);
      return response.data.user;
    };

    const register = async (userData) => {
      const response = await axios.post('/api/auth/register', userData);
      localStorage.setItem('token', response.data.token);
      setUser(response.data.user);
      return response.data.user;
    };

    const logout = () => {
      localStorage.removeItem('token');
      setUser(null);
      delete axios.defaults.headers.common['Authorization'];
    };

    const value = {
      user,
      isLoading,
      login,
      register,
      logout
    };

    return (
      <AuthContext.Provider value={value}>
        {children}
      </AuthContext.Provider>
    );
  };
```

7. Customer Features

Dashboard Components

1. Profile Management

```
// src/components/CustomerDashboard/Profile.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useAuth } from '../../contexts/AuthContext';

const Profile = () => {
  const { user, setUser } = useAuth();
  const [formData, setFormData] = useState({
    firstName: '',
    lastName: '',
    email: '',
    phoneNumber: ''
  });
  const [isEditing, setIsEditing] = useState(false);
  const [error, setError] = useState('');

  useEffect(() => {
    if (user) {
      setFormData({
        firstName: user.firstName || '',
        lastName: user.lastName || '',
        email: user.email || '',
        phoneNumber: user.phoneNumber || ''
      });
    }
  }, [user]);

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.put('/api/users/profile', formData);
      setUser({ ...user, ...response.data });
      setIsEditing(false);
      setError('');
    } catch (error) {
      setError(error.response?.data?.message || 'Update failed');
    }
  };

  return (
    <div className="profile-container">
      <h2>My Profile</h2>
      {error && <div className="error">{error}</div>}
      <form onSubmit={handleSubmit}>
```

```
          <div className="form-group">
            <label>First Name</label>
            <input
              type="text"
              name="firstName"
              value={formData.firstName}
              onChange={handleChange}
              disabled={!isEditing}
              required
            />
          </div>
          <div className="form-group">
            <label>Last Name</label>
            <input
              type="text"
              name="lastName"
              value={formData.lastName}
              onChange={handleChange}
              disabled={!isEditing}
              required
            />
          </div>
          <div className="form-group">
            <label>Email</label>
            <input
              type="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              disabled={true} // Email shouldn't be editable
              required
            />
          </div>
          <div className="form-group">
            <label>Phone Number</label>
            <input
              type="tel"
              name="phoneNumber"
              value={formData.phoneNumber}
              onChange={handleChange}
              disabled={!isEditing}
            />
          </div>
          {isEditing ? (
            <div className="button-group">
              <button type="submit" className="btn-primary">Save Changes</button>
              <button
                type="button"
                className="btn-secondary"
                onClick={() => setIsEditing(false)}
              >
                Cancel
              </button>
            </div>
          ) : (
            <button
              type="button"
              className="btn-primary"
              onClick={() => setIsEditing(true)}
            >
              Edit Profile
            </button>
          )}
        </form>
      </div>
    );
  };

  export default Profile;
```

2. Order History

```
// src/components/CustomerDashboard/Orders.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { format } from 'date-fns';

const Orders = () => {
  const [orders, setOrders] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchOrders = async () => {
      try {
        const response = await axios.get('/api/orders');
        setOrders(response.data);
        setError('');
      } catch (error) {
        setError('Failed to fetch orders');
      } finally {
        setLoading(false);
      }
    };

    fetchOrders();
  }, []);

  if (loading) return <div>Loading orders...</div>;
  if (error) return <div className="error">{error}</div>;

  return (
    <div className="orders-container">
      <h2>My Orders</h2>
      {orders.length === 0 ? (
        <p>You haven't placed any orders yet.</p>
      ) : (
        <div className="orders-list">
          {orders.map((order) => (
            <div key={order._id} className="order-card">
              <div className="order-header">
                <h3>Order #{order.orderNumber}</h3>
                <span className={`status status-${order.status.toLowerCase()}`}>
                  {order.status}
                </span>
              </div>
              <div className="order-info">
                <p>Date: {format(new Date(order.createdAt), 'MMM dd, yyyy')}</p>
                <p>Total: ${order.billing.total.toFixed(2)}</p>
                {order.shipping.estimatedDelivery && (
                  <p>
                    Estimated Delivery:
                    {format(new Date(order.shipping.estimatedDelivery.from), 'MMM dd')} -
                    {format(new Date(order.shipping.estimatedDelivery.to), 'MMM dd, yyyy')}
                  </p>
                )}
              </div>
              <div className="order-items">
                <h4>Items ({order.items.length})</h4>
                <ul>
                  {order.items.map((item, index) => (
                    <li key={index}>
                      {item.productName} x {item.quantity}
                    </li>
                  ))}
                </ul>
              </div>
              <div className="order-actions">
                <button className="btn-secondary">View Details</button>
                {order.status === 'delivered' && (
                  <button className="btn-outline">Write a Review</button>
                )}
              </div>
            </div>
          ))}
        </div>
      )}
    </div>
  );
};

export default Orders;
```

3. **Address Management**

```
// src/components/CustomerDashboard/Addresses.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const Addresses = () => {
  const [addresses, setAddresses] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [showForm, setShowForm] = useState(false);
  const [currentAddress, setCurrentAddress] = useState(null);
  const [formData, setFormData] = useState({
    addressType: 'shipping',
    street: '',
    city: '',
    state: '',
    zipCode: '',
    country: '',
    isDefault: false
  });

  useEffect(() => {
    const fetchAddresses = async () => {
      try {
        const response = await axios.get('/api/users/addresses');
        setAddresses(response.data);
        setError('');
      } catch (error) {
        setError('Failed to fetch addresses');
      } finally {
        setLoading(false);
      }
    };

    fetchAddresses();
  }, []);

  const handleChange = (e) => {
    const value = e.target.type === 'checkbox' ? e.target.checked : e.target.value;
    setFormData({ ...formData, [e.target.name]: value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      let response;

      if (currentAddress) {
        response = await axios.put(`/api/users/addresses/${currentAddress._id}`, formData);
        const updatedAddresses = addresses.map(address =>
          address._id === currentAddress._id ? response.data : address
        );
        setAddresses(updatedAddresses);
      } else {
        response = await axios.post('/api/users/addresses', formData);
        setAddresses([...addresses, response.data]);
      }

      resetForm();
      setError('');
    } catch (error) {
      setError(error.response?.data?.message || 'Failed to save address');
    }
  };

  const handleEdit = (address) => {
    setCurrentAddress(address);
    setFormData({
      addressType: address.addressType,
      street: address.street,
      city: address.city,
      state: address.state,
      zipCode: address.zipCode,
      country: address.country,
      isDefault: address.isDefault
    });
    setShowForm(true);
  };

  const handleDelete = async (id) => {
    if (window.confirm('Are you sure you want to delete this address?')) {
      try {
        await axios.delete(`/api/users/addresses/${id}`);
        setAddresses(addresses.filter(address => address._id !== id));
      } catch (error) {
        setError('Failed to delete address');
      }
    }
  };

  const resetForm = () => {
    setFormData({
      addressType: 'shipping',
      street: '',
      city: '',
      state: '',
      zipCode: '',
      country: '',
      isDefault: false
    });
    setCurrentAddress(null);
    setShowForm(false);
  };

  if (loading) return <div>Loading addresses...</div>;
```

```
    return (
      <div className="addresses-container">
        <div className="addresses-header">
          <h2>My Addresses</h2>
          <button
            className="btn-primary"
            onClick={() => setShowForm(!showForm)}
          >
            {showForm ? 'Cancel' : 'Add New Address'}
          </button>
        </div>

        {error && <div className="error">{error}</div>}

        {showForm && (
          <div className="address-form-container">
            <h3>{currentAddress ? 'Edit Address' : 'Add New Address'}</h3>
            <form onSubmit={handleSubmit}>
              <div className="form-group">
                <label>Address Type</label>
                <select
                  name="addressType"
                  value={formData.addressType}
                  onChange={handleChange}
                >
                  <option value="shipping">Shipping</option>
                  <option value="billing">Billing</option>
                </select>
              </div>
              <div className="form-group">
                <label>Street</label>
                <input
                  type="text"
                  name="street"
                  value={formData.street}
                  onChange={handleChange}
                  required
                />
              </div>
              <div className="form-row">
                <div className="form-group">
                  <label>City</label>
                  <input
                    type="text"
                    name="city"
                    value={formData.city}
                    onChange={handleChange}
                    required
                  />
                </div>
                <div className="form-group">
                  <label>State/Province</label>
                  <input
                    type="text"
                    name="state"
                    value={formData.state}
                    onChange={handleChange}
                    required
                  />
                </div>
              </div>
              <div className="form-row">
                <div className="form-group">
                  <label>Zip/Postal Code</label>
                  <input
                    type="text"
                    name="zipCode"
                    value={formData.zipCode}
                    onChange={handleChange}
                    required
                  />
                </div>
                <div className="form-group">
                  <label>Country</label>
                  <input
                    type="text"
                    name="country"
                    value={formData.country}
                    onChange={handleChange}
                    required
                  />
                </div>
              </div>
              <div className="form-group checkbox">
                <input
                  type="checkbox"
                  name="isDefault"
                  id="isDefault"
                  checked={formData.isDefault}
                  onChange={handleChange}
                />
                <label htmlFor="isDefault">Set as default address</label>
              </div>
              <div className="form-buttons">
                <button type="submit" className="btn-primary">
                  {currentAddress ? 'Update Address' : 'Add Address'}
                </button>
                <button
                  type="button"
                  className="btn-secondary"
                  onClick={resetForm}
                >
                  Cancel
                </button>
              </div>
            </form>
```

```
          </div>
        )}

        <div className="addresses-list">
          {addresses.length === 0 ? (
            <p>You don't have any saved addresses yet.</p>
          ) : (
            addresses.map((address) => (
              <div key={address._id} className="address-card">
                <div className="address-type">
                  {address.addressType.charAt(0).toUpperCase() + address.addressType.slice(1)} A
                  {address.isDefault && <span className="default-badge">Default</span>}
                </div>
                <div className="address-content">
                  <p>{address.street}</p>
                  <p>{address.city}, {address.state} {address.zipCode}</p>
                  <p>{address.country}</p>
                </div>
                <div className="address-actions">
                  <button
                    className="btn-icon"
                    onClick={() => handleEdit(address)}
                  >
                    Edit
                  </button>
                  <button
                    className="btn-icon btn-delete"
                    onClick={() => handleDelete(address._id)}
                  >
                    Delete
                  </button>
                </div>
              </div>
            ))
          )}
        </div>
      </div>
    );
  };

  export default Addresses;
```

8. Shop Owner Features

Dashboard Components

1. **Product Management**

```
// src/components/ShopOwnerDashboard/ProductManagement.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const ProductManagement = () => {
  const [products, setProducts] = useState([]);
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [showForm, setShowForm] = useState(false);
  const [currentProduct, setCurrentProduct] = useState(null);
  const [formData, setFormData] = useState({
    name: '',
    description: '',
    price: '',
    category: '',
    subCategory: '',
    quantity: '',
    isActive: true,
    isPromoted: false
  });
  const [images, setImages] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const [productsRes, categoriesRes] = await Promise.all([
          axios.get('/api/products/shop'),
          axios.get('/api/categories')
        ]);

        setProducts(productsRes.data);
        setCategories(categoriesRes.data);
        setError('');
      } catch (error) {
        setError('Failed to fetch data');
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  const handleChange = (e) => {
    const value = e.target.type === 'checkbox' ? e.target.checked : e.target.value;
    setFormData({ ...formData, [e.target.name]: value });
  };

  const handleImageChange = (e) => {
    const files = Array.from(e.target.files);
    setImages(files);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    const productData = new FormData();

    // Append form data
    Object.keys(formData).forEach(key => {
      productData.append(key, formData[key]);
    });

    // Append images
    images.forEach(image => {
      productData.append('images', image);
    });

    try {
      let response;

      if (currentProduct) {
        response = await axios.put(`/api/products/${currentProduct._id}`, productData, {
          headers: {
            'Content-Type': 'multipart/form-data'
          }
        });

        const updatedProducts = products.map(product =>
          product._id === currentProduct._id ? response.data : product
        );
        setProducts(updatedProducts);
      } else {
        response = await axios.post('/api/products', productData, {
          headers: {
            'Content-Type': 'multipart/form-data'
          }
        });

        setProducts([...products, response.data]);
      }

      resetForm();
      setError('');
    } catch (error) {
      setError(error.response?.data?.message || 'Failed to save product');
    }
  };

  const handleEdit = (product) => {
    setCurrentProduct(product);
    setFormData({
      name: product.name,
      description: product.description,
      price: product.price,
      category: product.category,
      subCategory: product.subCategory || '',
      quantity: product.inventory.quantity,
      isActive: product.isActive,
      isPromoted: product.isPromoted
    });
    setImages([]);
    setShowForm(true);
  };

  const handleDelete = async (id) => {
    if (window.confirm('Are you sure you want to delete this product?')) {
      try {
        await axios.delete(`/api/products/${id}`);
        setProducts(products.filter(product => product._id !== id));
      } catch (error) {
        setError('Failed to delete product');
      }
    }
  };

  const toggleProductStatus = async (id, isActive) => {
    try {
      const response = await axios.patch(`/api/products/${id}/status`, { isActive: !isActive }

      const updatedProducts = products.map(product =>
        product._id === id ? { ...product, isActive: response.data.isActive } : product
      );

      setProducts(updatedProducts);
    } catch (error) {
      setError('Failed to update product status');
    }
  };

  const resetForm = () => {
    setFormData({
      name: '',
      description: '',
      price: '',
      category: '',
      subCategory: '',
      quantity: '',
      isActive: true,
      isPromoted: false
    });
    setImages([]);
    setCurrentProduct(null);
    setShowForm(false);
  };

  if (loading) return <div>Loading products...</div>;

  return (
    <div className="product-management-container">
```

```
        <div className="product-management-header">
          <h2>Product Management</h2>
          <button
            className="btn-primary"
            onClick={() => setShowForm(!showForm)}
          >
            {showForm ? 'Cancel' : 'Add New Product'}
          </button>
        </div>

        {error && <div className="error">{error}</div>}

        {showForm && (
          <div className="product-form-container">
            <h3>{currentProduct ? 'Edit Product' : 'Add New Product'}</h3>
            <form onSubmit={handleSubmit}>
              <div className="form-group">
                <label>Product Name</label>
                <input
                  type="text"
                  name="name"
                  value={formData.name}
                  onChange={handleChange}
                  required
                />
              </div>
              <div className="form-group">
                <label>Description</label>
                <textarea
                  name="description"
                  value={formData.description}
                  onChange={handleChange}
                  required
                />
              </div>
              <div className="form-row">
                <div className="form-group">
                  <label>Price</label>
                  <input
                    type="number"
                    name="price"
                    min="0"
                    step="0.01"
                    value={formData.price}
                    onChange={handleChange}
                    required
                  />
                </div>
                <div className="form-group">
                  <label>Quantity</label>
                  <input
                    type="number"
                    name="quantity"
                    min="0"
                    value={formData.quantity}
                    onChange={handleChange}
                    required
                  />
                </div>
              </div>
              <div className="form-row">
                <div className="form-group">
                  <label>Category</label>
                  <select
                    name="category"
                    value={formData.category}
                    onChange={handleChange}
                    required
                  >
                    <option value="">Select a category</option>
                    {categories.map(category => (
                      <option key={category._id} value={category._id}>
                        {category.name}
                      </option>
                    ))}
                  </select>
                </div>
                <div className="form-group">
                  <label>Sub-Category</label>
                  <input
                    type="text"
                    name="subCategory"
                    value={formData.subCategory}
                    onChange={handleChange}
                  />
                </div>
              </div>
              <div className="form-group">
                <label>Product Images</label>
                <input
                  type="file"
                  multiple
                  accept="image/*"
                  onChange={handleImageChange}
                />
                <small>You can select multiple images. Maximum 5 images per product.</small>
              </div>
              <div className="form-row checkbox-row">
                <div className="form-group checkbox">
                  <input
                    type="checkbox"
                    name="isActive"
                    id="isActive"
                    checked={formData.isActive}
                    onChange={handleChange}
                  />
```

```
              <label htmlFor="isActive">Active</label>
            </div>
            <div className="form-group checkbox">
              <input
                type="checkbox"
                name="isPromoted"
                id="isPromoted"
                checked={formData.isPromoted}
                onChange={handleChange}
              />
              <label htmlFor="isPromoted">Featured/Promoted</label>
            </div>
          </div>
          <div className="form-buttons">
            <button type="submit" className="btn-primary">
              {currentProduct ? 'Update Product' : 'Add Product'}
            </button>
            <button
              type="button"
              className="btn-secondary"
              onClick={resetForm}
            >
              Cancel
            </button>
          </div>
        </form>
      </div>
    )}

    <div className="products-list">
      {products.length === 0 ? (
        <p>You don't have any products yet.</p>
      ) : (
        <table className="products-table">
          <thead>
            <tr>
              <th>Image</th>
              <th>Name</th>
              <th>Category</th>
              <th>Price</th>
              <th>Inventory</th>
              <th>Status</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {products.map((product) => (
              <tr key={product._id}>
                <td>
                  <img
                    src={product.images[0] || '/images/placeholder.png'}
                    alt={product.name}
                    className="product-thumbnail"
                  />
                </td>
                <td>{product.name}</td>
                <td>{product.categoryName}</td>
                <td>${product.price.toFixed(2)}</td>
                <td>{product.inventory.quantity}</td>
                <td>
                  <div className="status-toggle">
                    <input
                      type="checkbox"
                      id={`status-${product._id}`}
                      checked={product.isActive}
                      onChange={() => toggleProductStatus(product._id, product.isActive)}
                    />
                    <label htmlFor={`status-${product._id}`}>
                      {product.isActive ? 'Active' : 'Inactive'}
                    </label>
                  </div>
                </td>
                <td>
                  <div className="action-buttons">
                    <button
                      className="btn-icon"
                      onClick={() => handleEdit(product)}
                    >
                      Edit
                    </button>
                    <button
                      className="btn-icon btn-delete"
                      onClick={() => handleDelete(product._id)}
                    >
                      Delete
                    </button>
                  </div>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      )}
    </div>
  </div>
  );
};

export default ProductManagement;
```

2. **Shop Analytics**

```
// src/components/ShopOwnerDashboard/Analytics.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Line, Bar, Pie } from 'react-chartjs-2';
import { format, subDays } from 'date-fns';

const Analytics = () => {
  const [timeFrame, setTimeFrame] = useState('week');
  const [salesData, setSalesData] = useState({});
  const [productsData, setProductsData] = useState({});
  const [categoryData, setCategoryData] = useState({});
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchAnalytics = async () => {
      try {
        const response = await axios.get(`/api/analytics?timeFrame=${timeFrame}`);

        setSalesData(response.data.sales);
        setProductsData(response.data.products);
        setCategoryData(response.data.categories);

        setError('');
      } catch (error) {
        setError('Failed to fetch analytics data');
      } finally {
        setLoading(false);
      }
    };

    fetchAnalytics();
  }, [timeFrame]);

  const handleTimeFrameChange = (e) => {
    setTimeFrame(e.target.value);
  };

  if (loading) return <div>Loading analytics...</div>;
  if (error) return <div className="error">{error}</div>;

  // Generate labels for sales chart based on time frame
  const generateLabels = () => {
    const labels = [];
    const today = new Date();

    if (timeFrame === 'week') {
      for (let i = 6; i >= 0; i--) {
        labels.push(format(subDays(today, i), 'EEE'));
      }
    } else if (timeFrame === 'month') {
      for (let i = 29; i >= 0; i--) {
        labels.push(format(subDays(today, i), 'dd'));
      }
    } else if (timeFrame === 'year') {
      for (let i = 0; i < 12; i++) {
        labels.push(format(new Date(today.getFullYear(), i, 1), 'MMM'));
      }
    }

    return labels;
  };

  // Sales chart data
  const salesChartData = {
    labels: generateLabels(),
    datasets: [
      {
        label: 'Sales Amount ($)',
        data: salesData.amount || [],
        borderColor: '#F15A24',
        backgroundColor: 'rgba(241, 90, 36, 0.2)',
        fill: true,
        tension: 0.4
      },
      {
        label: 'Orders Count',
        data: salesData.count || [],
        borderColor: '#333',
        backgroundColor: 'rgba(51, 51, 51, 0.2)',
        fill: true,
        tension: 0.4,
        yAxisID: 'y1'
      }
    ]
  };

  const salesChartOptions = {
    responsive: true,
    scales: {
      y: {
        type: 'linear',
        position: 'left',
        ticks: {
          beginAtZero: true,
          callback: (value) => `$${value}`
        }
      },
      y1: {
        type: 'linear',
        position: 'right',
        ticks: {
          beginAtZero: true
        },
        grid: {
          drawOnChartArea: false
        }
      }
    }
  };

  // Top products chart data
  const productChartData = {
    labels: productsData.names || [],
    datasets: [
      {
        label: 'Units Sold',
        data: productsData.quantities || [],
        backgroundColor: [
          '#F15A24',
          '#FF9D5C',
          '#FFC999',
          '#FFE5CC',
          '#333333'
        ],
        borderWidth: 1
      }
    ]
  };

  // Category distribution chart data
  const categoryChartData = {
    labels: categoryData.names || [],
    datasets: [
      {
        data: categoryData.percentages || [],
        backgroundColor: [
          '#F15A24',
          '#FF7F32',
          '#FF9D5C',
          '#FFB380',
          '#FFC999',
          '#FFD6B3',
          '#FFE5CC'
        ],
        borderWidth: 1
      }
    ]
  };

  return (
    <div className="analytics-container">
      <div className="analytics-header">
        <h2>Shop Analytics</h2>
        <div className="time-frame-selector">
          <label>Time Frame:</label>
          <select value={timeFrame} onChange={handleTimeFrameChange}>
            <option value="week">Last 7 Days</option>
            <option value="month">Last 30 Days</option>
            <option value="year">This Year</option>
          </select>
        </div>
      </div>

      <div className="stats-summary">
        <div className="stat-card">
          <h3>Total Sales</h3>
          <p className="stat-value">${salesData.totalAmount?.toFixed(2) || 0}</p>
          <p className="stat-comparison">
            {salesData.percentChange >= 0 ? '+' : ''}
            {salesData.percentChange?.toFixed(2) || 0}% vs previous period
          </p>
        </div>
        <div className="stat-card">
          <h3>Orders</h3>
          <p className="stat-value">{salesData.totalOrders || 0}</p>
          <p className="stat-comparison">
            Avg. ${salesData.averageOrderValue?.toFixed(2) || 0} per order
          </p>
        </div>
        <div className="stat-card">
          <h3>Conversion Rate</h3>
          <p className="stat-value">{salesData.conversionRate?.toFixed(2) || 0}%</p>
          <p className="stat-comparison">
            From {salesData.totalVisits || 0} shop visits
          </p>
        </div>
      </div>

      <div className="chart-container">
        <h3>Sales Trend</h3>
        <Line data={salesChartData} options={salesChartOptions} />
      </div>

      <div className="charts-row">
        <div className="chart-container">
          <h3>Top Selling Products</h3>
          <Bar data={productChartData} options={{ responsive: true }} />
        </div>
        <div className="chart-container">
          <h3>Sales by Category</h3>
          <Pie data={categoryChartData} options={{ responsive: true }} />
        </div>
      </div>
    </div>
  );
};

export default Analytics;
```

3. **Promotion Management**

```
// src/components/ShopOwnerDashboard/PromotionManagement.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { format } from 'date-fns';

const PromotionManagement = () => {
  const [promotions, setPromotions] = useState([]);
  const [products, setProducts] = useState([]);
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [showForm, setShowForm] = useState(false);
  const [currentPromotion, setCurrentPromotion] = useState(null);
  const [formData, setFormData] = useState({
    name: '',
    description: '',
    type: 'discount',
    value: '',
    code: '',
    minimumPurchase: '',
    startDate: format(new Date(), 'yyyy-MM-dd'),
    endDate: format(new Date(new Date().setDate(new Date().getDate() + 7)), 'yyyy-MM-dd'),
    applicableProducts: [],
    applicableCategories: [],
    isActive: true
  });

  useEffect(() => {
    const fetchData = async () => {
      try {
        const [promotionsRes, productsRes, categoriesRes] = await Promise.all([
          axios.get('/api/promotions'),
          axios.get('/api/products/shop'),
          axios.get('/api/categories')
        ]);

        setPromotions(promotionsRes.data);
        setProducts(productsRes.data);
        setCategories(categoriesRes.data);
        setError('');
      } catch (error) {
        setError('Failed to fetch data');
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    if (type === 'checkbox') {
      setFormData({ ...formData, [name]: checked });
    } else if (name === 'applicableProducts' || name === 'applicableCategories') {
      const options = Array.from(e.target.selectedOptions, option => option.value);
      setFormData({ ...formData, [name]: options });
    } else {
      setFormData({ ...formData, [name]: value });
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      let response;

      if (currentPromotion) {
        response = await axios.put(`/api/promotions/${currentPromotion._id}`, formData);

        const updatedPromotions = promotions.map(promotion =>
          promotion._id === currentPromotion._id ? response.data : promotion
        );
        setPromotions(updatedPromotions);
      } else {
        response = await axios.post('/api/promotions', formData);
        setPromotions([...promotions, response.data]);
      }

      resetForm();
      setError('');
    } catch (error) {
      setError(error.response?.data?.message || 'Failed to save promotion');
    }
  };

  const handleEdit = (promotion) => {
    setCurrentPromotion(promotion);
    setFormData({
      name: promotion.name,
      description: promotion.description,
      type: promotion.type,
      value: promotion.value,
      code: promotion.code || '',
      minimumPurchase: promotion.minimumPurchase || '',
      startDate: format(new Date(promotion.startDate), 'yyyy-MM-dd'),
      endDate: format(new Date(promotion.endDate), 'yyyy-MM-dd'),
      applicableProducts: promotion.applicableProducts || [],
      applicableCategories: promotion.applicableCategories || [],
      isActive: promotion.isActive
    });
    setShowForm(true);
  };
  const handleDelete = async (id) => {
    if (window.confirm('Are you sure you want to delete this promotion?')) {
      try {
        await axios.delete(`/api/promotions/${id}`);
        setPromotions(promotions.filter(promotion => promotion._id !== id));
      } catch (error) {
        setError('Failed to delete promotion');
      }
    }
  };

  const togglePromotionStatus = async (id, isActive) => {
    try {
      const response = await axios.patch(`/api/promotions/${id}/status`, { isActive: !isActive

      const updatedPromotions = promotions.map(promotion =>
        promotion._id === id ? { ...promotion, isActive: response.data.isActive } : promotion
      );

      setPromotions(updatedPromotions);
    } catch (error) {
      setError('Failed to update promotion status');
    }
  };

  const resetForm = () => {
    setFormData({
      name: '',
      description: '',
      type: 'discount',
      value: '',
      code: '',
      minimumPurchase: '',
      startDate: format(new Date(), 'yyyy-MM-dd'),
      endDate: format(new Date(new Date().setDate(new Date().getDate() + 7)), 'yyyy-MM-dd'),
      applicableProducts: [],
      applicableCategories: [],
      isActive: true
    });
    setCurrentPromotion(null);
    setShowForm(false);
  };

  if (loading) return <div>Loading promotions...</div>;

  return (
    <div className="promotion-management-container">
      <div className="promotion-management-header">
        <h2>Promotion Management</h2>
        <button
          className="btn-primary"
          onClick={() => setShowForm(!showForm)}
```

```
            >
              {showForm ? 'Cancel' : 'Create Promotion'}
            </button>
          </div>

          {error && <div className="error">{error}</div>}

          {showForm && (
            <div className="promotion-form-container">
              <h3>{currentPromotion ? 'Edit Promotion' : 'Create New Promotion'}</h3>
              <form onSubmit={handleSubmit}>
                <div className="form-group">
                  <label>Promotion Name</label>
                  <input
                    type="text"
                    name="name"
                    value={formData.name}
                    onChange={handleChange}
                    required
                  />
                </div>
                <div className="form-group">
                  <label>Description</label>
                  <textarea
                    name="description"
                    value={formData.description}
                    onChange={handleChange}
                    required
                  />
                </div>
                <div className="form-row">
                  <div className="form-group">
                    <label>Promotion Type</label>
                    <select
                      name="type"
                      value={formData.type}
                      onChange={handleChange}
                      required
                    >
                      <option value="discount">Fixed Discount</option>
                      <option value="percentage">Percentage Discount</option>
                      <option value="flash_sale">Flash Sale</option>
                      <option value="coupon">Coupon Code</option>
                    </select>
                  </div>
                  <div className="form-group">
                    <label>Value {formData.type === 'percentage' ? '(%)' : '($)'}</label>
                    <input
                      type="number"
                      name="value"
                      min="0"
                      step={formData.type === 'percentage' ? '1' : '0.01'}
                      value={formData.value}
                      onChange={handleChange}
                      required
                    />
                  </div>
                </div>
                {formData.type === 'coupon' && (
                  <div className="form-group">
                    <label>Coupon Code</label>
                    <input
                      type="text"
                      name="code"
                      value={formData.code}
                      onChange={handleChange}
                      required={formData.type === 'coupon'}
                    />
                  </div>
                )}
                <div className="form-group">
                  <label>Minimum Purchase (optional)</label>
                  <input
                    type="number"
                    name="minimumPurchase"
                    min="0"
                    step="0.01"
                    value={formData.minimumPurchase}
                    onChange={handleChange}
                    placeholder="0.00"
                  />
                </div>
                <div className="form-row">
                  <div className="form-group">
                    <label>Start Date</label>
                    <input
                      type="date"
                      name="startDate"
                      value={formData.startDate}
                      onChange={handleChange}
                      required
                    />
                  </div>
                  <div className="form-group">
                    <label>End Date</label>
                    <input
                      type="date"
                      name="endDate"
                      value={formData.endDate}
                      onChange={handleChange}
                      required
                    />
                  </div>
                </div>
                <div className="form-group">
                  <label>Applicable Products (optional)</label>
```

```
            <select
              name="applicableProducts"
              multiple
              value={formData.applicableProducts}
              onChange={handleChange}
            >
              {products.map(product => (
                <option key={product._id} value={product._id}>
                  {product.name}
                </option>
              ))}
            </select>
            <small>Hold Ctrl/Cmd key to select multiple products. Leave empty for all produc
          </div>
          <div className="form-group">
            <label>Applicable Categories (optional)</label>
            <select
              name="applicableCategories"
              multiple
              value={formData.applicableCategories}
              onChange={handleChange}
            >
              {categories.map(category => (
                <option key={category._id} value={category._id}>
                  {category.name}
                </option>
              ))}
            </select>
            <small>Hold Ctrl/Cmd key to select multiple categories. Leave empty for all cate
          </div>
          <div className="form-group checkbox">
            <input
              type="checkbox"
              name="isActive"
              id="isActive"
              checked={formData.isActive}
              onChange={handleChange}
            />
            <label htmlFor="isActive">Active</label>
          </div>
          <div className="form-buttons">
            <button type="submit" className="btn-primary">
              {currentPromotion ? 'Update Promotion' : 'Create Promotion'}
            </button>
            <button
              type="button"
              className="btn-secondary"
              onClick={resetForm}
            >
              Cancel
            </button>
          </div>
        </form>
```

```
          </div>
        )}

        <div className="promotions-list">
          {promotions.length === 0 ? (
            <p>You don't have any promotions yet.</p>
          ) : (
            <table className="promotions-table">
              <thead>
                <tr>
                  <th>Name</th>
                  <th>Type</th>
                  <th>Value</th>
                  <th>Date Range</th>
                  <th>Status</th>
                  <th>Actions</th>
                </tr>
              </thead>
              <tbody>
                {promotions.map((promotion) => (
                  <tr key={promotion._id}>
                    <td>{promotion.name}</td>
                    <td>
                      {promotion.type === 'discount' && 'Fixed Discount'}
                      {promotion.type === 'percentage' && 'Percentage'}
                      {promotion.type === 'flash_sale' && 'Flash Sale'}
                      {promotion.type === 'coupon' && `Coupon (${promotion.code})`}
                    </td>
                    <td>
                      {promotion.type === 'percentage'
                        ? `${promotion.value}%`
                        : `$${promotion.value.toFixed(2)}`
                      }
                    </td>
                    <td>
                      {format(new Date(promotion.startDate), 'MMM dd, yyyy')} -
                      {format(new Date(promotion.endDate), 'MMM dd, yyyy')}
                    </td>
                    <td>
                      <div className="status-toggle">
                        <input
                          type="checkbox"
                          id={`status-${promotion._id}`}
                          checked={promotion.isActive}
                          onChange={() => togglePromotionStatus(promotion._id, promotion.isActiv
                        />
                        <label htmlFor={`status-${promotion._id}`}>
                          {promotion.isActive ? 'Active' : 'Inactive'}
                        </label>
                      </div>
                    </td>
                    <td>
                      <div className="action-buttons">
                        <button
                          className="btn-icon"
                          onClick={() => handleEdit(promotion)}
                        >
                          Edit
                        </button>
                        <button
                          className="btn-icon btn-delete"
                          onClick={() => handleDelete(promotion._id)}
                        >
                          Delete
                        </button>
                      </div>
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          )}
        </div>
      </div>
    );
  };

  export default PromotionManagement;
```

9. Product Management

Product Component

```
// src/components/Product/ProductCard.js
import React from 'react';
import { Link } from 'react-router-dom';
import { useCart } from '../../contexts/CartContext';

const ProductCard = ({ product }) => {
  const { addToCart } = useCart();

  const handleAddToCart = (e) => {
    e.preventDefault();
    e.stopPropagation();
    addToCart(product, 1);
  };

  return (
    <div className="product-card">
      <Link to={`/products/${product._id}`} className="product-link">
        <div className="product-image">
          <img
            src={product.images[0] || '/images/placeholder.png'}
            alt={product.name}
          />
          {product.isPromoted && <span className="product-badge">Sale</span>}
        </div>
        <div className="product-info">
          <h3 className="product-name">{product.name}</h3>
          <div className="product-price">
            {product.salePrice ? (
              <>
                <span className="original-price">${product.price.toFixed(2)}</span>
                <span className="sale-price">${product.salePrice.toFixed(2)}</span>
              </>
            ) : (
              <span>${product.price.toFixed(2)}</span>
            )}
          </div>
          <div className="product-rating">
            {/* Display star rating here */}
            <span className="rating-value">{product.rating}</span>
          </div>
        </div>
      </Link>
      <button
        className="add-to-cart-btn"
        onClick={handleAddToCart}
      >
        Add to Cart
      </button>
    </div>
  );
};

export default ProductCard;
```

Product Detail Component

```
// src/pages/ProductDetail.js
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import axios from 'axios';
import { useCart } from '../contexts/CartContext';

const ProductDetail = () => {
  const { id } = useParams();
  const { addToCart } = useCart();
  const [product, setProduct] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [selectedImage, setSelectedImage] = useState(0);
  const [quantity, setQuantity] = useState(1);
  const [selectedAttributes, setSelectedAttributes] = useState({});
```

```
useEffect(() => {
  const fetchProduct = async () => {
    try {
      const response = await axios.get(`/api/products/${id}`);
      setProduct(response.data);

      // Initialize selected attributes
      if (response.data.attributes) {
        const initialAttributes = {};
        Object.keys(response.data.attributes).forEach(key => {
          if (Array.isArray(response.data.attributes[key]) && response.data.attributes[key].
            initialAttributes[key] = response.data.attributes[key][0];
          }
        });
        setSelectedAttributes(initialAttributes);
      }

      setError('');
    } catch (error) {
      setError('Failed to load product details');
    } finally {
      setLoading(false);
    }
  };

  fetchProduct();
}, [id]);

const handleQuantityChange = (e) => {
  const value = parseInt(e.target.value);
  if (value > 0 && value <= (product?.inventory?.quantity || 10)) {
    setQuantity(value);
  }
};

const handleAttributeChange = (attribute, value) => {
  setSelectedAttributes(prev => ({
    ...prev,
    [attribute]: value
  }));
};

const handleAddToCart = () => {
  addToCart(product, quantity, selectedAttributes);
};

if (loading) return <div className="loading">Loading product details...</div>;
if (error) return <div className="error">{error}</div>;
if (!product) return <div className="error">Product not found</div>;

return (
  <div className="product-detail-container">
```

```
        <div className="product-detail-content">
          <div className="product-images">
            <div className="main-image">
              <img
                src={product.images[selectedImage] || '/images/placeholder.png'}
                alt={product.name}
              />
            </div>
            {product.images.length > 1 && (
              <div className="thumbnail-images">
                {product.images.map((image, index) => (
                  <div
                    key={index}
                    className={`thumbnail ${index === selectedImage ? 'active' : ''}`}
                    onClick={() => setSelectedImage(index)}
                  >
                    <img src={image} alt={`${product.name} thumbnail ${index + 1}`} />
                  </div>
                ))}
              </div>
            )}
          </div>

          <div className="product-info">
            <h1 className="product-name">{product.name}</h1>

            <div className="product-meta">
              <div className="product-rating">
                {/* Stars here */}
                <span>{product.rating} ★</span>
                <span className="review-count">({product.reviews?.length || 0} reviews)</span>
              </div>
              <div className="product-category">
                Category: <span>{product.categoryName}</span>
              </div>
            </div>

            <div className="product-price">
              {product.salePrice ? (
                <>
                  <span className="original-price">${product.price.toFixed(2)}</span>
                  <span className="sale-price">${product.salePrice.toFixed(2)}</span>
                  <span className="discount-badge">
                    {Math.round((1 - product.salePrice / product.price) * 100)}% OFF
                  </span>
                </>
              ) : (
                <span>${product.price.toFixed(2)}</span>
              )}
            </div>

            <div className="product-description">
              <p>{product.description}</p>
            </div>

            {/* Product attributes (if any) */}
            {product.attributes && Object.keys(product.attributes).length > 0 && (
              <div className="product-attributes">
                {Object.keys(product.attributes).map(attribute => (
                  <div key={attribute} className="attribute-group">
                    <h3>{attribute.charAt(0).toUpperCase() + attribute.slice(1)}</h3>
                    <div className="attribute-options">
                      {Array.isArray(product.attributes[attribute]) ? (
                        product.attributes[attribute].map(value => (
                          <button
                            key={value}
                            className={`attribute-btn ${selectedAttributes[attribute] === value
                            onClick={() => handleAttributeChange(attribute, value)}
                          >
                            {value}
                          </button>
                        ))
                      ) : (
                        <span>{product.attributes[attribute]}</span>
                      )}
                    </div>
                  </div>
                ))}
              </div>
            )}

            <div className="product-actions">
              <div className="quantity-selector">
                <button
                  onClick={() => quantity > 1 && setQuantity(quantity - 1)}
                  disabled={quantity <= 1}
                >
                  -
                </button>
                <input
                  type="number"
                  value={quantity}
                  onChange={handleQuantityChange}
                  min="1"
                  max={product.inventory?.quantity || 10}
                />
                <button
                  onClick={() => quantity < (product.inventory?.quantity || 10) && setQuantity(q
                  disabled={quantity >= (product.inventory?.quantity || 10)}
                >
                  +
                </button>
              </div>

              <button
                className="add-to-cart-btn"
                onClick={handleAddToCart}
                disabled={product.inventory?.quantity <= 0}
              >
                {product.inventory?.quantity <= 0 ? 'Out of Stock' : 'Add to Cart'}
              </button>
            </div>

            <div className="product-meta-info">
              <div className="availability">
                Availability:
                <span className={product.inventory?.quantity > 0 ? 'in-stock' : 'out-of-stock'}>
                  {product.inventory?.quantity > 0 ? 'In Stock' : 'Out of Stock'}
                </span>
              </div>
              <div className="sku">
                SKU: <span>{product.inventory?.sku || 'N/A'}</span>
              </div>
            </div>
          </div>
        </div>

        <div className="product-tabs">
          <div className="tabs-header">
            <button className="tab-btn active">Description</button>
            <button className="tab-btn">Reviews ({product.reviews?.length || 0})</button>
            <button className="tab-btn">Shipping & Returns</button>
          </div>

          <div className="tab-content">
            <div className="tab-pane active">
              <div className="product-description-full">
                <p>{product.description}</p>
                {/* Additional product details here */}
              </div>
            </div>
          </div>
        </div>

        <div className="related-products">
          <h2>You May Also Like</h2>
          {/* Related products would be rendered here */}
        </div>
      </div>
    );
  };

  export default ProductDetail;
```

10. Shopping Cart & Checkout

Cart Context

```
// src/contexts/CartContext.js
import React, { createContext, useContext, useState, useEffect } from 'react';
import axios from 'axios';
import { useAuth } from './AuthContext';

const CartContext = createContext();

export const useCart = () => useContext(CartContext);

export const CartProvider = ({ children }) => {
  const { user } = useAuth();
  const [cart, setCart] = useState({
    items: [],
    totalItems: 0,
    totalAmount: 0
  });
  const [loading, setLoading] = useState(true);

  // Fetch cart from API if user is logged in, otherwise from localStorage
  useEffect(() => {
    const fetchCart = async () => {
      try {
        if (user) {
          // Fetch from API
          const response = await axios.get('/api/cart');
          setCart({
            items: response.data.items || [],
            totalItems: response.data.items.reduce((total, item) => total + item.quantity, 0),
            totalAmount: response.data.totalAmount || 0
          });
        } else {
          // Fetch from localStorage
          const savedCart = JSON.parse(localStorage.getItem('cart')) || { items: [], totalAmou
          setCart({
            items: savedCart.items || [],
            totalItems: savedCart.items.reduce((total, item) => total + item.quantity, 0),
            totalAmount: savedCart.totalAmount || 0
          });
        }
      } catch (error) {
        console.error('Error fetching cart:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchCart();
  }, [user]);

  // Save cart to localStorage when it changes (for non-logged in users)
```

```
useEffect(() => {
  if (!loading && !user) {
    localStorage.setItem('cart', JSON.stringify({
      items: cart.items,
      totalAmount: cart.totalAmount
    }));
  }
}, [cart, loading, user]);

// Add item to cart
const addToCart = async (product, quantity, attributes = {}) => {
  try {
    const item = {
      productId: product._id,
      name: product.name,
      price: product.salePrice || product.price,
      image: product.images[0],
      attributes,
      quantity
    };

    if (user) {
      // Save to API
      await axios.post('/api/cart/items', item);
      const response = await axios.get('/api/cart');

      setCart({
        items: response.data.items || [],
        totalItems: response.data.items.reduce((total, item) => total + item.quantity, 0),
        totalAmount: response.data.totalAmount || 0
      });
    } else {
      // Save to local state
      // Check if item already exists
      const existingItemIndex = cart.items.findIndex(cartItem =>
        cartItem.productId === product._id &&
        JSON.stringify(cartItem.attributes) === JSON.stringify(attributes)
      );

      let newItems;

      if (existingItemIndex >= 0) {
        // Update quantity of existing item
        newItems = [...cart.items];
        newItems[existingItemIndex].quantity += quantity;
      } else {
        // Add new item
        newItems = [...cart.items, item];
      }

      const newTotalAmount = newItems.reduce((total, item) => total + (item.price * item.qua

      setCart({
        items: newItems,
        totalItems: newItems.reduce((total, item) => total + item.quantity, 0),
        totalAmount: newTotalAmount
      });
    }
  } catch (error) {
    console.error('Error adding to cart:', error);
  }
};

// Update item quantity
const updateQuantity = async (productId, quantity, attributes = {}) => {
  try {
    if (user) {
      // Update via API
      await axios.put(`/api/cart/items/${productId}`, { quantity, attributes });
      const response = await axios.get('/api/cart');

      setCart({
        items: response.data.items || [],
        totalItems: response.data.items.reduce((total, item) => total + item.quantity, 0),
        totalAmount: response.data.totalAmount || 0
      });
    } else {
      // Update local state
      const itemIndex = cart.items.findIndex(item =>
        item.productId === productId &&
        JSON.stringify(item.attributes) === JSON.stringify(attributes)
      );

      if (itemIndex >= 0) {
        const newItems = [...cart.items];
        newItems[itemIndex].quantity = quantity;

        const newTotalAmount = newItems.reduce((total, item) => total + (item.price * item.q

        setCart({
          items: newItems,
          totalItems: newItems.reduce((total, item) => total + item.quantity, 0),
          totalAmount: newTotalAmount
        });
      }
    }
  } catch (error) {
    console.error('Error updating cart:', error);
  }
};

// Remove item from cart
const removeFromCart = async (productId, attributes = {}) => {
  try {
    if (user) {
      // Remove via API
      await axios.delete(`/api/cart/items/${productId}`, { data: { attributes } });
      const response = await axios.get('/api/cart');

      setCart({
        items: response.data.items || [],
        totalItems: response.data.items.reduce((total, item) => total + item.quantity, 0),
        totalAmount: response.data.totalAmount || 0
      });
    } else {
      // Remove from local state
      const newItems = cart.items.filter(item =>
        !(item.productId === productId &&
          JSON.stringify(item.attributes) === JSON.stringify(attributes))
      );

      const newTotalAmount = newItems.reduce((total, item) => total + (item.price * item.qua

      setCart({
        items: newItems,
        totalItems: newItems.reduce((total, item) => total + item.quantity, 0),
        totalAmount: newTotalAmount
      });
    }
  } catch (error) {
    console.error('Error removing from cart:', error);
  }
};

// Clear cart
const clearCart = async () => {
  try {
    if (user) {
      // Clear via API
      await axios.delete('/api/cart');
    }

    // Reset local state
    setCart({
      items: [],
      totalItems: 0,
      totalAmount: 0
    });

    // Clear localStorage if user not logged in
    if (!user) {
      localStorage.removeItem('cart');
    }
  } catch (error) {
    console.error('Error clearing cart:', error);
  }
};

// Apply discount code
const applyDiscount = async (code) => {
  try {
    if (user) {
      // Apply via API
      const response = await axios.post('/api/cart/discount', { code });

      setCart({
        items: response.data.items || [],
        totalItems: response.data.items.reduce((total, item) => total + item.quantity, 0),
        totalAmount: response.data.totalAmount || 0,
        discount: response.data.discount || 0
      });

      return { success: true, message: 'Discount applied successfully' };
    } else {
      // TODO: Implement local discount logic
      return { success: false, message: 'Please login to apply discount codes' };
    }
  } catch (error) {
    console.error('Error applying discount:', error);
    return {
      success: false,
      message: error.response?.data?.message || 'Failed to apply discount code'
    };
  }
};

const value = {
  cart,
  loading,
  addToCart,
  updateQuantity,
  removeFromCart,
  clearCart,
  applyDiscount
};

return (
  <CartContext.Provider value={value}>
    {children}
  </CartContext.Provider>
);
};
```

Cart Page

```
// src/pages/Cart.js
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useCart } from '../contexts/CartContext';
import { useAuth } from '../contexts/AuthContext';
```

```
const Cart = () => {
  const { cart, loading, updateQuantity, removeFromCart, applyDiscount } = useCart();
  const { user } = useAuth();
  const [promoCode, setPromoCode] = useState('');
  const [promoMessage, setPromoMessage] = useState('');
  const [promoError, setPromoError] = useState(false);
  const navigate = useNavigate();

  const handleQuantityChange = (productId, quantity, attributes) => {
    if (quantity > 0) {
      updateQuantity(productId, quantity, attributes);
    }
  };

  const handleRemoveItem = (productId, attributes) => {
    removeFromCart(productId, attributes);
  };

  const handleApplyPromoCode = async () => {
    if (!promoCode.trim()) {
      setPromoError(true);
      setPromoMessage('Please enter a valid promo code');
      return;
    }

    const result = await applyDiscount(promoCode);

    if (result.success) {
      setPromoError(false);
      setPromoMessage(result.message);
    } else {
      setPromoError(true);
      setPromoMessage(result.message);
    }
  };

  const handleCheckout = () => {
    if (user) {
      navigate('/checkout');
    } else {
      navigate('/login?redirect=checkout');
    }
  };

  if (loading) return <div className="loading">Loading cart...</div>;

  return (
    <div className="cart-page">
      <div className="cart-container">
        <h1>Your Shopping Cart</h1>

        {cart.items.length === 0 ? (
          <div className="empty-cart">
            <i className="cart-icon">🛒</i>
            <p>Your cart is empty</p>
            <Link to="/products" className="btn-primary">Continue Shopping</Link>
          </div>
        ) : (
          <div className="cart-content">
            <div className="cart-items">
              <div className="cart-header">
                <div className="cart-col product-col">Product</div>
                <div className="cart-col price-col">Price</div>
                <div className="cart-col quantity-col">Quantity</div>
                <div className="cart-col subtotal-col">Subtotal</div>
                <div className="cart-col action-col"></div>
              </div>

              {cart.items.map((item, index) => (
                <div key={index} className="cart-item">
                  <div className="cart-col product-col">
                    <div className="product-info">
                      <img
                        src={item.image || '/images/placeholder.png'}
                        alt={item.name}
                        className="product-thumbnail"
                      />
                      <div className="product-details">
                        <h3>{item.name}</h3>
                        {Object.keys(item.attributes).length > 0 && (
                          <div className="product-attributes">
                            {Object.entries(item.attributes).map(([key, value]) => (
                              <span key={key}>
                                {key.charAt(0).toUpperCase() + key.slice(1)}: {value}
                              </span>
                            ))}
                          </div>
                        )}
                      </div>
                    </div>
                  </div>

                  <div className="cart-col price-col">
                    ${item.price.toFixed(2)}
                  </div>

                  <div className="cart-col quantity-col">
                    <div className="quantity-selector">
                      <button
                        onClick={() => handleQuantityChange(item.productId, item.quantity - 1,
                        disabled={item.quantity <= 1}
                      >
                        -
                      </button>
                      <input
                        type="number"
                        value={item.quantity}
                        onChange={(e) => handleQuantityChange(item.productId, parseInt(e.targe
                        min="1"
                      />
                      <button
                        onClick={() => handleQuantityChange(item.productId, item.quantity + 1,
                      >
                        +
                      </button>
                    </div>
                  </div>

                  <div className="cart-col subtotal-col">
                    ${(item.price * item.quantity).toFixed(2)}
                  </div>

                  <div className="cart-col action-col">
                    <button
                      className="btn-remove"
                      onClick={() => handleRemoveItem(item.productId, item.attributes)}
                    >
                      Remove
                    </button>
                  </div>
                </div>
              ))}
            </div>

            <div className="cart-summary">
              <h2>Order Summary</h2>

              <div className="summary-row">
                <span>Subtotal</span>
                <span>${cart.totalAmount.toFixed(2)}</span>
              </div>

              {cart.discount > 0 && (
                <div className="summary-row discount">
                  <span>Discount</span>
                  <span>-${cart.discount.toFixed(2)}</span>
                </div>
              )}

              <div className="promo-code">
                <input
                  type="text"
                  placeholder="Enter promo code"
                  value={promoCode}
                  onChange={(e) => setPromoCode(e.target.value)}
                />
                <button
                  className="btn-apply"
```

~~~
                onClick={handleApplyPromoCode}
              >
                Apply
              </button>
            </div>

            {promoMessage && (
              <div className={`promo-message ${promoError ? 'error' : 'success'}`}>
                {promoMessage}
              </div>
            )}

            <div className="summary-row total">
              <span>Total</span>
              <span>${(cart.totalAmount - (cart.discount || 0)).toFixed(2)}</span>
            </div>

            <button
              className="btn-checkout"
              onClick={handleCheckout}
            >
              Proceed to Checkout
            </button>

            <div className="continue-shopping">
              <Link to="/products">Continue Shopping</Link>
            </div>
          </div>
        </div>
      )}
    </div>
  </div>
 );
};

export default Cart;

### Checkout Page
```javascript
// src/pages/Checkout.js
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import { useCart } from '../contexts/CartContext';
import { useAuth } from '../contexts/AuthContext';

const Checkout = () => {
  const { cart, loading: cartLoading, clearCart } = useCart();
  const { user } = useAuth();
  const navigate = useNavigate();

  const [addresses, setAddresses] = useState([]);
  const [selectedAddress, setSelectedAddress] = useState('');
~~~

```
  const [paymentMethod, setPaymentMethod] = useState('card');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [orderProcessing, setOrderProcessing] = useState(false);

  const [cardInfo, setCardInfo] = useState({
    cardNumber: '',
    cardName: '',
    expiry: '',
    cvv: ''
  });

  useEffect(() => {
    // Redirect if cart is empty
    if (!cartLoading && cart.items.length === 0) {
      navigate('/cart');
    }

    // Fetch user addresses
    const fetchAddresses = async () => {
      try {
        const response = await axios.get('/api/users/addresses');
        setAddresses(response.data);

        // Set default address if available
        const defaultAddress = response.data.find(addr => addr.isDefault);
        if (defaultAddress) {
          setSelectedAddress(defaultAddress._id);
        } else if (response.data.length > 0) {
          setSelectedAddress(response.data[0]._id);
        }

        setError('');
      } catch (error) {
        setError('Failed to fetch addresses');
      } finally {
        setLoading(false);
      }
    };

    if (user) {
      fetchAddresses();
    } else {
      navigate('/login?redirect=checkout');
    }
  }, [user, cartLoading, cart.items.length, navigate]);

  const handleCardInfoChange = (e) => {
    const { name, value } = e.target;
    setCardInfo({ ...cardInfo, [name]: value });
  };

  const validateForm = () => {
    if (!selectedAddress) {
      setError('Please select a shipping address');
      return false;
    }

    if (paymentMethod === 'card') {
      if (!cardInfo.cardNumber || !cardInfo.cardName || !cardInfo.expiry || !cardInfo.cvv) {
        setError('Please complete all payment details');
        return false;
      }

      // Basic card validation
      if (!/^\d{16}$/.test(cardInfo.cardNumber.replace(/\s/g, ''))) {
        setError('Invalid card number');
        return false;
      }

      if (!/^\d{3,4}$/.test(cardInfo.cvv)) {
        setError('Invalid CVV');
        return false;
      }

      const [month, year] = cardInfo.expiry.split('/');
      const expiryDate = new Date(20 + year.trim(), month.trim() - 1);
      if (expiryDate < new Date()) {
        setError('Card has expired');
        return false;
      }
    }

    return true;
  };

  const handlePlaceOrder = async () => {
    if (!validateForm()) return;

    setOrderProcessing(true);

    try {
      // Create order
      const orderData = {
        addressId: selectedAddress,
        items: cart.items.map(item => ({
          productId: item.productId,
          quantity: item.quantity,
          attributes: item.attributes
        })),
        payment: {
          method: paymentMethod,
          // For real implementation, you'd handle payment tokenization here
          details: paymentMethod === 'card' ? {
            last4: cardInfo.cardNumber.slice(-4)
          } : {}
        }
      };

      const response = await axios.post('/api/orders', orderData);

      // Clear cart after successful order
      await clearCart();

      // Redirect to order confirmation
      navigate(`/order-confirmation/${response.data._id}`);
    } catch (error) {
      setError(error.response?.data?.message || 'Failed to place order');
      setOrderProcessing(false);
    }
  };

  if (loading || cartLoading) return <div className="loading">Loading checkout...</div>;

  return (
    <div className="checkout-page">
      <div className="checkout-container">
        <h1>Checkout</h1>

        {error && <div className="error-message">{error}</div>}

        <div className="checkout-content">
          <div className="checkout-main">
            {/* Shipping Section */}
            <div className="checkout-section">
              <h2>Shipping Information</h2>

              {addresses.length === 0 ? (
                <div className="no-addresses">
                  <p>You don't have any saved addresses.</p>
                  <button
                    className="btn-primary"
                    onClick={() => navigate('/dashboard/addresses?redirect=checkout')}
                  >
                    Add New Address
                  </button>
                </div>
              ) : (
                <div className="address-selection">
                  {addresses.map(address => (
                    <div key={address._id} className="address-option">
                      <input
                        type="radio"
                        id={`address-${address._id}`}
                        name="shipping-address"
                        value={address._id}
                        checked={selectedAddress === address._id}
                        onChange={() => setSelectedAddress(address._id)}
                      />
                      <label htmlFor={`address-${address._id}`}>
                        <div className="address-details">
                          <p>
                            <strong>{user.firstName} {user.lastName}</strong>
                          </p>
                          <p>{address.street}</p>
                          <p>{address.city}, {address.state} {address.zipCode}</p>
                          <p>{address.country}</p>
                        </div>
                      </label>
                    </div>
                  ))}

                  <button
                    className="btn-text"
                    onClick={() => navigate('/dashboard/addresses?redirect=checkout')}
                  >
                    Add New Address
                  </button>
                </div>
              )}
            </div>

            {/* Payment Section */}
            <div className="checkout-section">
              <h2>Payment Method</h2>

              <div className="payment-methods">
                <div className="payment-option">
                  <input
                    type="radio"
                    id="payment-card"
                    name="payment-method"
                    value="card"
                    checked={paymentMethod === 'card'}
                    onChange={() => setPaymentMethod('card')}
                  />
                  <label htmlFor="payment-card">Credit/Debit Card</label>
                </div>

                <div className="payment-option">
                  <input
                    type="radio"
                    id="payment-paypal"
                    name="payment-method"
                    value="paypal"
                    checked={paymentMethod === 'paypal'}
                    onChange={() => setPaymentMethod('paypal')}
                  />
                  <label htmlFor="payment-paypal">PayPal</label>
                </div>
              </div>
```

```
          {paymentMethod === 'card' && (
            <div className="card-payment-form">
              <div className="form-group">
                <label>Card Number</label>
                <input
                  type="text"
                  name="cardNumber"
                  value={cardInfo.cardNumber}
                  onChange={handleCardInfoChange}
                  placeholder="1234 5678 9012 3456"
                  maxLength="19"
                />
              </div>

              <div className="form-group">
                <label>Name on Card</label>
                <input
                  type="text"
                  name="cardName"
                  value={cardInfo.cardName}
                  onChange={handleCardInfoChange}
                  placeholder="John Doe"
                />
              </div>

              <div className="form-row">
                <div className="form-group">
                  <label>Expiry Date</label>
                  <input
                    type="text"
                    name="expiry"
                    value={cardInfo.expiry}
                    onChange={handleCardInfoChange}
                    placeholder="MM/YY"
                    maxLength="5"
                  />
                </div>

                <div className="form-group">
                  <label>CVV</label>
                  <input
                    type="text"
                    name="cvv"
                    value={cardInfo.cvv}
                    onChange={handleCardInfoChange}
                    placeholder="123"
                    maxLength="4"
                  />
                </div>
              </div>
            </div>
          )}
```

```
            {paymentMethod === 'paypal' && (
              <div className="paypal-info">
                <p>You will be redirected to PayPal to complete your payment securely.</p>
              </div>
            )}
          </div>
        </div>

        {/* Order Summary */}
        <div className="checkout-sidebar">
          <div className="order-summary">
            <h2>Order Summary</h2>

            <div className="order-items">
              {cart.items.map((item, index) => (
                <div key={index} className="order-item">
                  <div className="item-info">
                    <img
                      src={item.image || '/images/placeholder.png'}
                      alt={item.name}
                      className="item-thumbnail"
                    />
                    <div className="item-details">
                      <h4>{item.name}</h4>
                      <p className="item-attributes">
                        {Object.entries(item.attributes).map(([key, value]) => (
                          <span key={key}>{value}</span>
                        ))}
                      </p>
                      <p className="item-quantity">Qty: {item.quantity}</p>
                    </div>
                  </div>
                  <div className="item-price">
                    ${(item.price * item.quantity).toFixed(2)}
                  </div>
                </div>
              ))}
            </div>

            <div className="summary-details">
              <div className="summary-row">
                <span>Subtotal</span>
                <span>${cart.totalAmount.toFixed(2)}</span>
              </div>

              <div className="summary-row">
                <span>Shipping</span>
                <span>$2.25</span>
              </div>

              {cart.discount > 0 && (
                <div className="summary-row discount">
                  <span>Discount</span>
                  <span>-${cart.discount.toFixed(2)}</span>
                </div>
              )}

              <div className="summary-row total">
                <span>Total</span>
                <span>${(cart.totalAmount + 2.25 - (cart.discount || 0)).toFixed(2)}</span>
              </div>
            </div>

            <button
              className="btn-place-order"
              onClick={handlePlaceOrder}
              disabled={orderProcessing || addresses.length === 0}
            >
              {orderProcessing ? 'Processing...' : 'Place Order'}
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>
);
  };

  export default Checkout;
```

11. Payment Gateway Integration

Stripe Integration Service

```
// src/services/paymentService.js
import axios from 'axios';

export const createPaymentIntent = async (amount, currency = 'usd') => {
  try {
    const response = await axios.post('/api/payments/create-intent', {
      amount,
      currency
    });

    return response.data;
  } catch (error) {
    throw error;
  }
};

export const confirmCardPayment = async (clientSecret, paymentMethod) => {
  try {
    const response = await axios.post('/api/payments/confirm-payment', {
      clientSecret,
      paymentMethod
    });

    return response.data;
  } catch (error) {
    throw error;
  }
};

export const createPaymentMethod = async (cardDetails) => {
  try {
    const response = await axios.post('/api/payments/create-payment-method', cardDetails);

    return response.data;
  } catch (error) {
    throw error;
  }
};
```

Backend Payment Controller

```
// server/controllers/paymentController.js
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);
const Order = require('../models/Order');
const User = require('../models/User');

// Create payment intent
exports.createPaymentIntent = async (req, res) => {
  try {
    const { amount, currency = 'usd' } = req.body;

    const paymentIntent = await stripe.paymentIntents.create({
      amount: Math.round(amount * 100), // Convert to cents
      currency,
      metadata: {
        userId: req.user._id.toString()
      }
    });

    res.status(200).json({
      clientSecret: paymentIntent.client_secret
    });
  } catch (error) {
    console.error('Error creating payment intent:', error);
    res.status(500).json({ message: 'Payment processing failed' });
  }
};

// Confirm payment
exports.confirmPayment = async (req, res) => {
  try {
    const { orderId, paymentIntentId } = req.body;

    // Retrieve the payment intent
    const paymentIntent = await stripe.paymentIntents.retrieve(paymentIntentId);

    if (paymentIntent.status !== 'succeeded') {
      return res.status(400).json({ message: 'Payment not successful' });
    }

    // Update order with payment info
    const order = await Order.findByIdAndUpdate(
      orderId,
      {
        'payment.status': 'completed',
        'payment.transactionId': paymentIntentId,
        status: 'processing'
      },
      { new: true }
    );

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

    res.status(200).json({
      success: true,
      order
    });
  } catch (error) {
    console.error('Error confirming payment:', error);
    res.status(500).json({ message: 'Failed to confirm payment' });
  }
};

// Create payment method
exports.createPaymentMethod = async (req, res) => {
  try {
    const { cardNumber, expMonth, expYear, cvc } = req.body;

    const paymentMethod = await stripe.paymentMethods.create({
      type: 'card',
      card: {
        number: cardNumber,
        exp_month: expMonth,
        exp_year: expYear,
        cvc
      }
    });

    res.status(200).json({
      paymentMethodId: paymentMethod.id
    });
  } catch (error) {
    console.error('Error creating payment method:', error);
    res.status(500).json({ message: 'Failed to process payment details' });
  }
};

// Webhook handler for Stripe events
exports.handleWebhook = async (req, res) => {
  const sig = req.headers['stripe-signature'];
  let event;

  try {
    event = stripe.webhooks.constructEvent(
      req.rawBody,
      sig,
      process.env.STRIPE_WEBHOOK_SECRET
    );
  } catch (error) {
    console.error('Webhook signature verification failed:', error);
    return res.status(400).send(`Webhook Error: ${error.message}`);
  }

  // Handle the event
  switch (event.type) {
    case 'payment_intent.succeeded':
      const paymentIntent = event.data.object;
      await handleSuccessfulPayment(paymentIntent);
      break;

    case 'payment_intent.payment_failed':
      const failedPayment = event.data.object;
      await handleFailedPayment(failedPayment);
      break;

    default:
      console.log(`Unhandled event type ${event.type}`);
  }

  res.status(200).json({ received: true });
};

// Helper functions
async function handleSuccessfulPayment(paymentIntent) {
  try {
    // Find order by transaction ID and update
    await Order.findOneAndUpdate(
      { 'payment.transactionId': paymentIntent.id },
      {
        'payment.status': 'completed',
        status: 'processing'
      }
    );
  } catch (error) {
    console.error('Error handling successful payment:', error);
  }
}

async function handleFailedPayment(paymentIntent) {
  try {
    // Find order by transaction ID and update
    await Order.findOneAndUpdate(
      { 'payment.transactionId': paymentIntent.id },
      {
        'payment.status': 'failed',
        status: 'payment_failed'
      }
    );
  } catch (error) {
    console.error('Error handling failed payment:', error);
  }
}
```

Stripe Payment Component

```
// src/components/Checkout/StripePayment.js
import React, { useState, useEffect } from 'react';
import { CardElement, useStripe, useElements } from '@stripe/react-stripe-js';
import { createPaymentIntent } from '../../services/paymentService';

const StripePayment = ({ amount, onPaymentSuccess, onPaymentError }) => {
  const stripe = useStripe();
  const elements = useElements();
  const [clientSecret, setClientSecret] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchPaymentIntent = async () => {
      try {
        const { clientSecret } = await createPaymentIntent(amount);
        setClientSecret(clientSecret);
      } catch (error) {
        setError('Could not initiate payment. Please try again.');
        onPaymentError('Payment initialization failed');
      }
    };

    if (amount > 0) {
      fetchPaymentIntent();
    }
  }, [amount, onPaymentError]);

  const handleSubmit = async (event) => {
    event.preventDefault();
    if (!stripe || !elements) {
      return;
    }

    setLoading(true);
    setError('');

    const cardElement = elements.getElement(CardElement);

    const { error, paymentIntent } = await stripe.confirmCardPayment(clientSecret, {
      payment_method: {
        card: cardElement,
        billing_details: {
          name: 'Customer Name' // Ideally get this from form
        }
      }
    });

    if (error) {
      setError(error.message);
      onPaymentError(error.message);
    } else if (paymentIntent.status === 'succeeded') {
      onPaymentSuccess(paymentIntent.id);
    }

    setLoading(false);
  };

  return (
    <div className="stripe-payment">
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Card Details</label>
          <div className="card-element-container">
            <CardElement
              options={{
                style: {
                  base: {
                    fontSize: '16px',
                    color: '#424770',
                    '::placeholder': {
                      color: '#aab7c4',
                    },
                  },
                  invalid: {
                    color: '#9e2146',
                  },
                },
              }}
            />
          </div>
        </div>
        {error && <div className="error-message">{error}</div>}

        <button
          className="btn-primary btn-pay"
          disabled={!stripe || loading}
          type="submit"
        >
          {loading ? 'Processing...' : `Pay ${amount.toFixed(2)}`}
        </button>
      </form>
    </div>
  );
};

export default StripePayment;
```

PayPal Integration Component

```
// src/components/Checkout/PayPalPayment.js
import React, { useEffect } from 'react';

const PayPalPayment = ({ amount, onPaymentSuccess, onPaymentError }) => {
  useEffect(() => {
    const addPayPalScript = () => {
      const script = document.createElement('script');
      script.type = 'text/javascript';
      script.src = `https://www.paypal.com/sdk/js?client-id=${process.env.REACT_APP_PAYPAL_CLI
      script.async = true;
      script.onload = () => {
        initializePayPal();
      };
      document.body.appendChild(script);
    };

    const initializePayPal = () => {
      window.paypal
        .Buttons({
          createOrder: (data, actions) => {
            return actions.order.create({
              purchase_units: [
                {
                  amount: {
                    currency_code: 'USD',
                    value: amount.toFixed(2)
                  }
                }
              ]
            });
          },
          onApprove: async (data, actions) => {
            const order = await actions.order.capture();
            onPaymentSuccess(order.id);
          },
          onError: (err) => {
            onPaymentError('PayPal payment failed');
            console.error('PayPal Error:', err);
          }
        })
        .render('#paypal-button-container');
    };

    if (window.paypal) {
      initializePayPal();
    } else {
      addPayPalScript();
    }

    return () => {
      // Cleanup if needed
    };
  }, [amount, onPaymentSuccess, onPaymentError]);

  return (
    <div className="paypal-payment">
      <div id="paypal-button-container"></div>
    </div>
  );
};

export default PayPalPayment;
```

12. Search & Filter System

Search Component

```
// src/components/Search/SearchBar.js
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

const SearchBar = () => {
  const [query, setQuery] = useState('');
  const navigate = useNavigate();

  const handleSearch = (e) => {
    e.preventDefault();
    if (query.trim()) {
      navigate(`/products?search=${encodeURIComponent(query.trim())}`);
    }
  };
  return (
    <form className="search-form" onSubmit={handleSearch}>
      <input
        type="text"
        placeholder="Search for anything..."
        value={query}
        onChange={(e) => setQuery(e.target.value)}
        className="search-input"
      />
      <button type="submit" className="search-button">
        <i className="search-icon">🔍</i>
      </button>
    </form>
  );
};

export default SearchBar;
```

Filter Component

```
// src/components/Filter/FilterSidebar.js
import React, { useState, useEffect } from 'react';
import { useNavigate, useLocation } from 'react-router-dom';
import axios from 'axios';

const FilterSidebar = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const queryParams = new URLSearchParams(location.search);

  const [categories, setCategories] = useState([]);
  const [priceRange, setPriceRange] = useState({
    min: queryParams.get('minPrice') || '',
    max: queryParams.get('maxPrice') || ''
  });
  const [selectedCategories, setSelectedCategories] = useState(
    queryParams.get('categories') ? queryParams.get('categories').split(',') : []
  );
  const [rating, setRating] = useState(queryParams.get('rating') || '');
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchCategories = async () => {
      try {
        const response = await axios.get('/api/categories');
        setCategories(response.data);
      } catch (error) {
        console.error('Error fetching categories:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchCategories();
  }, []);

  useEffect(() => {
    // Update state when URL params change
    setSelectedCategories(
      queryParams.get('categories') ? queryParams.get('categories').split(',') : []
    );
    setPriceRange({
      min: queryParams.get('minPrice') || '',
      max: queryParams.get('maxPrice') || ''
    });
    setRating(queryParams.get('rating') || '');
  }, [location.search, queryParams]);

  const handlePriceChange = (e) => {
    const { name, value } = e.target;
    setPriceRange(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleCategoryChange = (categoryId) => {
    setSelectedCategories(prev => {
      if (prev.includes(categoryId)) {
        return prev.filter(id => id !== categoryId);
      } else {
        return [...prev, categoryId];
      }
    });
  };

  const handleRatingChange = (value) => {
    setRating(value === rating ? '' : value);
  };

  const applyFilters = () => {
    const newParams = new URLSearchParams(location.search);

    // Update category filter
    if (selectedCategories.length > 0) {
      newParams.set('categories', selectedCategories.join(','));
    } else {
      newParams.delete('categories');
    }

    // Update price filter
    if (priceRange.min) {
      newParams.set('minPrice', priceRange.min);
    } else {
      newParams.delete('minPrice');
    }

    if (priceRange.max) {
      newParams.set('maxPrice', priceRange.max);
    } else {
      newParams.delete('maxPrice');
    }

    // Update rating filter
    if (rating) {
      newParams.set('rating', rating);
    } else {
      newParams.delete('rating');
    }

    // Navigate with new filters
    navigate(`${location.pathname}?${newParams.toString()}`);
  };

  const clearFilters = () => {
    const newParams = new URLSearchParams();

    // Preserve search query if it exists
    const searchQuery = queryParams.get('search');
    if (searchQuery) {
      newParams.set('search', searchQuery);
    }

    // Reset states
    setSelectedCategories([]);
    setPriceRange({ min: '', max: '' });
    setRating('');

    // Navigate with cleared filters
    navigate(`${location.pathname}?${newParams.toString()}`);
  };

  if (loading) return <div>Loading filters...</div>;

  return (
    <div className="filter-sidebar">
      <div className="filter-header">
        <h3>Filters</h3>
        <button className="btn-clear-filters" onClick={clearFilters}>
          Clear All
        </button>
      </div>

      <div className="filter-section">
        <h4>Categories</h4>
        <div className="category-filters">
          {categories.map(category => (
            <div key={category._id} className="filter-checkbox">
              <input
                type="checkbox"
                id={`category-${category._id}`}
                checked={selectedCategories.includes(category._id)}
                onChange={() => handleCategoryChange(category._id)}
              />
              <label htmlFor={`category-${category._id}`}>{category.name}</label>
            </div>
          ))}
        </div>
      </div>

      <div className="filter-section">
        <h4>Price Range</h4>
        <div className="price-range-inputs">
          <div className="form-group">
            <input
              type="number"
              name="min"
              placeholder="Min"
              value={priceRange.min}
              onChange={handlePriceChange}
              min="0"
            />
          </div>
          <span className="price-range-separator">-</span>
          <div className="form-group">
            <input
              type="number"
              name="max"
              placeholder="Max"
              value={priceRange.max}
              onChange={handlePriceChange}
              min="0"
            />
          </div>
        </div>
      </div>

      <div className="filter-section">
        <h4>Customer Rating</h4>
        <div className="rating-filters">
          {[5, 4, 3, 2, 1].map(value => (
            <div key={value} className="filter-radio">
              <input
                type="radio"
                id={`rating-${value}`}
                name="rating"
                checked={rating === value.toString()}
                onChange={() => handleRatingChange(value.toString())}
              />
              <label htmlFor={`rating-${value}`}>
```

```
                {[...Array(value)].map((_, i) => (
                  <span key={i} className="star">★</span>
                ))}
                {[...Array(5 - value)].map((_, i) => (
                  <span key={i} className="star-empty">☆</span>
                ))}
                <span className="rating-text">& Up</span>
              </label>
            </div>
          ))}
        </div>
      </div>

      <button className="btn-apply-filters" onClick={applyFilters}>
        Apply Filters
      </button>
    </div>
  );
};

export default FilterSidebar;
```

Product Listing with Search & Filter

```
// src/pages/ProductListing.js
import React, { useState, useEffect } from 'react';
import { useLocation } from 'react-router-dom';
import axios from 'axios';
import ProductCard from '../components/Product/ProductCard';
import FilterSidebar from '../components/Filter/FilterSidebar';
import Pagination from '../components/Common/Pagination';

const ProductListing = () => {
  const location = useLocation();
  const queryParams = new URLSearchParams(location.search);

  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [totalProducts, setTotalProducts] = useState(0);
  const [currentPage, setCurrentPage] = useState(parseInt(queryParams.get('page')) || 1);
  const [sortBy, setSortBy] = useState(queryParams.get('sort') || 'newest');

  const productsPerPage = 12;

  useEffect(() => {
    const fetchProducts = async () => {
      setLoading(true);

      try {
        // Build query params
        const params = new URLSearchParams();

        // Pagination
        params.set('page', currentPage.toString());
        params.set('limit', productsPerPage.toString());

        // Sorting
        params.set('sort', sortBy);

        // Search term
        const searchQuery = queryParams.get('search');
        if (searchQuery) {
          params.set('search', searchQuery);
        }

        // Categories
        const categories = queryParams.get('categories');
        if (categories) {
          params.set('categories', categories);
        }

        // Price range
        const minPrice = queryParams.get('minPrice');
        const maxPrice = queryParams.get('maxPrice');
        if (minPrice) params.set('minPrice', minPrice);
        if (maxPrice) params.set('maxPrice', maxPrice);

        // Rating
        const rating = queryParams.get('rating');
        if (rating) params.set('rating', rating);

        const response = await axios.get(`/api/products?${params.toString()}`);

        setProducts(response.data.products);
        setTotalProducts(response.data.total);
        setError('');
      } catch (error) {
        setError('Failed to fetch products');
        console.error('Error fetching products:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchProducts();
  }, [location.search, currentPage, sortBy, queryParams]);

  const handlePageChange = (pageNumber) => {
    setCurrentPage(pageNumber);

    // Update URL with new page number
    const newParams = new URLSearchParams(location.search);
    newParams.set('page', pageNumber.toString());
    window.history.pushState({}, '', `${location.pathname}?${newParams.toString()}`);
  };

  const handleSortChange = (e) => {
    const newSortBy = e.target.value;
    setSortBy(newSortBy);

    // Update URL with new sort parameter
    const newParams = new URLSearchParams(location.search);
    newParams.set('sort', newSortBy);

    if (currentPage !== 1) {
      newParams.set('page', '1');
      setCurrentPage(1);
    }

    window.history.pushState({}, '', `${location.pathname}?${newParams.toString()}`);
  };

  const searchTerm = queryParams.get('search');

  return (
    <div className="product-listing-page">
      <div className="product-listing-container">
        <FilterSidebar />

        <div className="product-content">
          <div className="product-header">
            {searchTerm && (
              <h1 className="search-results-title">
                Search Results for "{searchTerm}"
              </h1>
            )}

            <div className="product-sorting">
              <div className="results-count">
                {totalProducts} {totalProducts === 1 ? 'Product' : 'Products'}
              </div>

              <div className="sort-control">
                <label>Sort by:</label>
                <select value={sortBy} onChange={handleSortChange}>
                  <option value="newest">Newest Arrivals</option>
                  <option value="price_asc">Price: Low to High</option>
                  <option value="price_desc">Price: High to Low</option>
                  <option value="rating_desc">Highest Rated</option>
                  <option value="popular">Most Popular</option>
                </select>
              </div>
            </div>
          </div>
          {loading ? (
            <div className="loading-products">Loading products...</div>
          ) : error ? (
            <div className="error-message">{error}</div>
          ) : products.length === 0 ? (
            <div className="no-products-found">
              <p>No products found matching your criteria.</p>
              <p>Try adjusting your filters or search terms.</p>
            </div>
          ) : (
            <>
              <div className="product-grid">
                {products.map(product => (
                  <ProductCard key={product._id} product={product} />
                ))}
              </div>

              <Pagination
                currentPage={currentPage}
                totalPages={Math.ceil(totalProducts / productsPerPage)}
                onPageChange={handlePageChange}
              />
            </>
          )}
        </div>
      </div>
    </div>
  );
};

export default ProductListing;
```

13. UI/UX Design Guidelines

Color Palette

```
/* src/styles/variables.scss */
:root {
  /* Primary Colors */
  --primary: #F15A24; /* Main Orange */
  --primary-light: #FF9D5C;
  --primary-dark: #D13800;

  /* Secondary Colors */
  --secondary: #333333; /* Dark Gray */
  --secondary-light: #666666;
  --secondary-dark: #111111;

  /* Neutral Colors */
  --white: #FFFFFF;
  --light-gray: #F5F5F5;
  --medium-gray: #E0E0E0;
  --dark-gray: #888888;

  /* Status Colors */
  --success: #4CAF50;
  --warning: #FFC107;
  --error: #F44336;
  --info: #2196F3;

  /* Text Colors */
  --text-primary: #212121;
  --text-secondary: #757575;
  --text-disabled: #9E9E9E;

  /* Border Colors */
  --border-light: #E0E0E0;
  --border-medium: #BDBDBD;

  /* Background Colors */
  --bg-main: #FFFFFF;
  --bg-light: #F9F9F9;
  --bg-dark: #333333;

  /* Box Shadow */
  --shadow-light: 0 2px 4px rgba(0, 0, 0, 0.1);
  --shadow-medium: 0 4px 8px rgba(0, 0, 0, 0.1);
  --shadow-dark: 0 8px 16px rgba(0, 0, 0, 0.1);
}
```

Typography

```
/* src/styles/typography.scss */
:root {
  /* Font Families */
  --font-primary: 'Open Sans', sans-serif;
  --font-heading: 'Montserrat', sans-serif;

  /* Font Sizes */
  --font-size-xs: 0.75rem;   /* 12px */
  --font-size-sm: 0.875rem;  /* 14px */
  --font-size-md: 1rem;      /* 16px */
  --font-size-lg: 1.125rem;  /* 18px */
  --font-size-xl: 1.25rem;   /* 20px */
  --font-size-2xl: 1.5rem;   /* 24px */
  --font-size-3xl: 1.875rem; /* 30px */
  --font-size-4xl: 2.25rem;  /* 36px */

  /* Font Weights */
  --font-weight-light: 300;
  --font-weight-regular: 400;
```

```
    --font-weight-medium: 500;
    --font-weight-semibold: 600;
    --font-weight-bold: 700;

    /* Line Heights */
    --line-height-tight: 1.25;
    --line-height-normal: 1.5;
    --line-height-relaxed: 1.75;
  }

  /* Base Typography Styles */
  body {
    font-family: var(--font-primary);
    font-size: var(--font-size-md);
    font-weight: var(--font-weight-regular);
    line-height: var(--line-height-normal);
    color: var(--text-primary);
  }

  h1, h2, h3, h4, h5, h6 {
    font-family: var(--font-heading);
    font-weight: var(--font-weight-bold);
    line-height: var(--line-height-tight);
    margin-top: 0;
  }

  h1 {
    font-size: var(--font-size-4xl);
  }

  h2 {
    font-size: var(--font-size-3xl);
  }

  h3 {
    font-size: var(--font-size-2xl);
  }

  h4 {
    font-size: var(--font-size-xl);
  }

  h5 {
    font-size: var(--font-size-lg);
  }

  h6 {
    font-size: var(--font-size-md);
  }

  p {
    margin-top: 0;
    margin-bottom: 1rem;
```

```
}

a {
  color: var(--primary);
  text-decoration: none;
  transition: color 0.2s ease-in-out;
}

a:hover {
  color: var(--primary-dark);
  text-decoration: underline;
}

.text-small {
  font-size: var(--font-size-sm);
}

.text-large {
  font-size: var(--font-size-lg);
}

.text-center {
  text-align: center;
}

.text-bold {
  font-weight: var(--font-weight-bold);
}

.text-medium {
  font-weight: var(--font-weight-medium);
}

.text-light {
  font-weight: var(--font-weight-light);
}

.text-primary {
  color: var(--primary);
}

.text-secondary {
  color: var(--text-secondary);
}

.text-error {
  color: var(--error);
}

.text-success {
  color: var(--success);
}
```

Button Component

```
// src/components/Common/Button.js
import React from 'react';
import PropTypes from 'prop-types';
import './Button.scss';

const Button = ({
  children,
  type = 'button',
  variant = 'primary',
  size = 'medium',
  disabled = false,
  fullWidth = false,
  className = '',
  onClick,
  ...props
}) => {
  return (
    <button
      type={type}
      className={`
        btn
        btn-${variant}
        btn-${size}
        ${fullWidth ? 'btn-full-width' : ''}
        ${className}
      `}
      disabled={disabled}
      onClick={onClick}
      {...props}
    >
      {children}
    </button>
  );
};

Button.propTypes = {
  children: PropTypes.node.isRequired,
  type: PropTypes.oneOf(['button', 'submit', 'reset']),
  variant: PropTypes.oneOf(['primary', 'secondary', 'outline', 'text', 'danger']),
  size: PropTypes.oneOf(['small', 'medium', 'large']),
  disabled: PropTypes.bool,
  fullWidth: PropTypes.bool,
  className: PropTypes.string,
  onClick: PropTypes.func
};

export default Button;
```

```
/* src/components/Common/Button.scss */
.btn {
  display: inline-flex;
  align-items: center;
  justify-content: center;
  font-family: var(--font-primary);
  font-weight: var(--font-weight-medium);
  border-radius: 4px;
  border: none;
  cursor: pointer;
  transition: all 0.2s ease-in-out;

  &:focus {
    outline: none;
    box-shadow: 0 0 0 3px rgba(241, 90, 36, 0.25);
  }

  &:disabled {
    opacity: 0.6;
    cursor: not-allowed;
  }

  /* Variants */
  &.btn-primary {
    background-color: var(--primary);
    color: white;

    &:hover:not(:disabled) {
      background-color: var(--primary-dark);
    }
  }

  &.btn-secondary {
    background-color: var(--secondary);
    color: white;

    &:hover:not(:disabled) {
      background-color: var(--secondary-dark);
    }
  }

  &.btn-outline {
    background-color: transparent;
    border: 1px solid var(--primary);
    color: var(--primary);

    &:hover:not(:disabled) {
      background-color: var(--primary);
      color: white;
    }
  }
```

```
    &.btn-text {
      background-color: transparent;
      color: var(--primary);
      padding: 0;

      &:hover:not(:disabled) {
        color: var(--primary-dark);
        text-decoration: underline;
      }
    }

    &.btn-danger {
      background-color: var(--error);
      color: white;

      &:hover:not(:disabled) {
        background-color: darken(var(--error), 10%);
      }
    }

    /* Sizes */
    &.btn-small {
      font-size: var(--font-size-sm);
      padding: 0.375rem 0.75rem;
    }

    &.btn-medium {
      font-size: var(--font-size-md);
      padding: 0.5rem 1rem;
    }

    &.btn-large {
      font-size: var(--font-size-lg);
      padding: 0.75rem 1.5rem;
    }

    /* Full width */
    &.btn-full-width {
      width: 100%;
    }
  }
```

Responsive Layout

```
/* src/styles/layout.scss */
.container {
  width: 100%;
  padding-right: 1rem;
  padding-left: 1rem;
  margin-right: auto;
  margin-left: auto;
```

```
}

/* Breakpoints */
@media (min-width: 576px) {
  .container {
    max-width: 540px;
  }
}

@media (min-width: 768px) {
  .container {
    max-width: 720px;
  }
}

@media (min-width: 992px) {
  .container {
    max-width: 960px;
  }
}

@media (min-width: 1200px) {
  .container {
    max-width: 1140px;
  }
}

/* Grid System */
.row {
  display: flex;
  flex-wrap: wrap;
  margin-right: -1rem;
  margin-left: -1rem;
}

.col {
  flex-basis: 0;
  flex-grow: 1;
  max-width: 100%;
  padding-right: 1rem;
  padding-left: 1rem;
}

.col-auto {
  flex: 0 0 auto;
  width: auto;
  max-width: 100%;
  padding-right: 1rem;
  padding-left: 1rem;
}

@for $i from 1 through 12 {
  .col-#{$i} {
      flex: 0 0 calc(#{$i} / 12 * 100%);
      max-width: calc(#{$i} / 12 * 100%);
      padding-right: 1rem;
      padding-left: 1rem;
    }
  }

  /* Responsive columns */
  @media (min-width: 576px) {
    @for $i from 1 through 12 {
      .col-sm-#{$i} {
        flex: 0 0 calc(#{$i} / 12 * 100%);
        max-width: calc(#{$i} / 12 * 100%);
      }
    }
  }

  @media (min-width: 768px) {
    @for $i from 1 through 12 {
      .col-md-#{$i} {
        flex: 0 0 calc(#{$i} / 12 * 100%);
        max-width: calc(#{$i} / 12 * 100%);
      }
    }
  }

  @media (min-width: 992px) {
    @for $i from 1 through 12 {
      .col-lg-#{$i} {
        flex: 0 0 calc(#{$i} / 12 * 100%);
        max-width: calc(#{$i} / 12 * 100%);
      }
    }
  }

  @media (min-width: 1200px) {
    @for $i from 1 through 12 {
      .col-xl-#{$i} {
        flex: 0 0 calc(#{$i} / 12 * 100%);
        max-width: calc(#{$i} / 12 * 100%);
      }
    }
  }
```

14. API Endpoints

User API

```
// server/routes/userRoutes.js
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');
const authMiddleware = require('../middleware/authMiddleware');

// Public routes
router.post('/register', userController.register);
router.post('/login', userController.login);
router.post('/social-login', userController.socialLogin);
router.post('/forgot-password', userController.forgotPassword);
router.post('/reset-password', userController.resetPassword);

// Protected routes (require authentication)
router.use(authMiddleware.authenticate);

router.get('/me', userController.getCurrentUser);
router.put('/profile', userController.updateProfile);
router.put('/password', userController.changePassword);

// Address management
router.get('/addresses', userController.getUserAddresses);
router.post('/addresses', userController.addAddress);
router.put('/addresses/:id', userController.updateAddress);
router.delete('/addresses/:id', userController.deleteAddress);

module.exports = router;
```

Product API

```
// server/routes/productRoutes.js
const express = require('express');
const router = express.Router();
const productController = require('../controllers/productController');
const authMiddleware = require('../middleware/authMiddleware');

// Public routes
router.get('/', productController.getProducts);
router.get('/:id', productController.getProductById);
router.get('/category/:categoryId', productController.getProductsByCategory);
router.get('/search', productController.searchProducts);

// Protected routes (require authentication)
router.use(authMiddleware.authenticate);

// Shop owner routes
router.use('/shop', authMiddleware.isShopOwner);
router.get('/shop', productController.getShopProducts);
router.post('/', productController.createProduct);
router.put('/:id', productController.updateProduct);
router.delete('/:id', productController.deleteProduct);
router.patch('/:id/status', productController.updateProductStatus);
```

```
module.exports = router;
```

Order API

```
// server/routes/orderRoutes.js
const express = require('express');
const router = express.Router();
const orderController = require('../controllers/orderController');
const authMiddleware = require('../middleware/authMiddleware');

// All routes require authentication
router.use(authMiddleware.authenticate);

// Customer routes
router.get('/', orderController.getUserOrders);
router.get('/:id', orderController.getOrderById);
router.post('/', orderController.createOrder);
router.post('/:id/cancel', orderController.cancelOrder);

// Shop owner routes
router.get('/shop', authMiddleware.isShopOwner, orderController.getShopOrders);
router.patch('/:id/status', authMiddleware.isShopOwner, orderController.updateOrderStatus);

module.exports = router;
```

Cart API

```
// server/routes/cartRoutes.js
const express = require('express');
const router = express.Router();
const cartController = require('../controllers/cartController');
const authMiddleware = require('../middleware/authMiddleware');

// All routes require authentication
router.use(authMiddleware.authenticate);

router.get('/', cartController.getCart);
router.post('/items', cartController.addItem);
router.put('/items/:productId', cartController.updateItem);
router.delete('/items/:productId', cartController.removeItem);
router.delete('/', cartController.clearCart);
router.post('/discount', cartController.applyDiscount);

module.exports = router;
```

Category API

```
// server/routes/categoryRoutes.js
const express = require('express');
const router = express.Router();
const categoryController = require('../controllers/categoryController');
const authMiddleware = require('../middleware/authMiddleware');

// Public routes
router.get('/', categoryController.getCategories);
router.get('/:id', categoryController.getCategoryById);

// Admin routes
router.use(authMiddleware.authenticate);
router.use(authMiddleware.isAdmin);

router.post('/', categoryController.createCategory);
router.put('/:id', categoryController.updateCategory);
router.delete('/:id', categoryController.deleteCategory);

module.exports = router;
```

Payment API

```
// server/routes/paymentRoutes.js
const express = require('express');
const router = express.Router();
const paymentController = require('../controllers/paymentController');
const authMiddleware = require('../middleware/authMiddleware');

// Webhook route (no auth required)
router.post('/webhook', express.raw({ type: 'application/json' }), paymentController.handleWeb

// Protected routes
router.use(authMiddleware.authenticate);

router.post('/create-intent', paymentController.createPaymentIntent);
router.post('/confirm-payment', paymentController.confirmPayment);
router.post('/create-payment-method', paymentController.createPaymentMethod);

module.exports = router;
```

15. Security Implementation

Authentication Middleware

```
// server/middleware/authMiddleware.js
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const Shop = require('../models/Shop');

// Authentication middleware
exports.authenticate = async (req, res, next) => {
  try {
    // Get token from Authorization header
    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      return res.status(401).json({ message: 'Authentication required' });
    }

    const token = authHeader.split(' ')[1];

    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Find user
    const user = await User.findById(decoded.userId).select('-password');

    if (!user) {
      return res.status(401).json({ message: 'User not found' });
    }

    // Add user to request
    req.user = user;

    next();
  } catch (error) {
    if (error.name === 'JsonWebTokenError') {
      return res.status(401).json({ message: 'Invalid token' });
    }

    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Token expired' });
    }

    console.error('Auth middleware error:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

// Shop owner middleware
exports.isShopOwner = async (req, res, next) => {
  try {
    if (req.user.userType !== 'shop_owner') {
      return res.status(403).json({ message: 'Access denied' });
    }

    // Verify shop ownership
    const shop = await Shop.findOne({ ownerId: req.user._id });
    if (!shop) {
      return res.status(403).json({ message: 'No shop associated with this account' });
    }

    // Add shop to request
    req.shop = shop;

    next();
  } catch (error) {
    console.error('Shop owner middleware error:', error);
    res.status(500).json({ message: 'Server error' });
  }
};

// Admin middleware
exports.isAdmin = (req, res, next) => {
  if (req.user.userType !== 'admin') {
    return res.status(403).json({ message: 'Admin access required' });
  }

  next();
};
```

Password Hashing

```
// server/utils/passwordUtils.js
const bcrypt = require('bcryptjs');

// Hash password
exports.hashPassword = async (password) => {
  const salt = await bcrypt.genSalt(10);
  return await bcrypt.hash(password, salt);
};

// Compare password
exports.comparePassword = async (password, hashedPassword) => {
  return await bcrypt.compare(password, hashedPassword);
};
```

Input Validation

```
// server/middleware/validationMiddleware.js
const { validationResult, check } = require('express-validator');

// Validation error handler
exports.validationHandler = (req, res, next) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({
      message: 'Validation error',
      errors: errors.array().map(err => ({
        field: err.param,
        message: err.msg
      }))
    });
  }

  next();
};

// User registration validation
exports.validateRegistration = [
  check('email')
    .isEmail()
    .withMessage('Please enter a valid email address')
    .normalizeEmail(),

  check('password')
    .isLength({ min: 8 })
    .withMessage('Password must be at least 8 characters long')
    .matches(/\d/)
    .withMessage('Password must contain at least one number')
    .matches(/[A-Z]/)
    .withMessage('Password must contain at least one uppercase letter'),

  check('firstName')
    .trim()
    .notEmpty()
    .withMessage('First name is required'),

  check('lastName')
    .trim()
    .notEmpty()
    .withMessage('Last name is required'),

  check('userType')
    .isIn(['customer', 'shop_owner'])
    .withMessage('Invalid user type')
];

// Product validation
exports.validateProduct = [
  check('name')
    .trim()
    .notEmpty()
    .withMessage('Product name is required'),

  check('description')
    .trim()
    .notEmpty()
    .withMessage('Product description is required'),
  check('price')
    .isFloat({ min: 0.01 })
    .withMessage('Price must be a positive number'),

  check('category')
    .isMongoId()
    .withMessage('Invalid category ID'),

  check('inventory.quantity')
    .isInt({ min: 0 })
    .withMessage('Quantity must be a non-negative integer')
];

// Order validation
exports.validateOrder = [
  check('addressId')
    .isMongoId()
    .withMessage('Invalid address ID'),

  check('items')
    .isArray({ min: 1 })
    .withMessage('Order must contain at least one item'),

  check('items.*.productId')
    .isMongoId()
    .withMessage('Invalid product ID'),

  check('items.*.quantity')
    .isInt({ min: 1 })
    .withMessage('Quantity must be a positive integer')
];
```

CORS Configuration

```
// server/app.js
const express = require('express');
const cors = require('cors');
const app = express();

// CORS configuration
const corsOptions = {
  origin: process.env.CORS_ORIGIN || 'http://localhost:3000',
  methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
  credentials: true,
  optionsSuccessStatus: 204
};

app.use(cors(corsOptions));
```

Rate Limiting

```
// server/middleware/rateLimitMiddleware.js
const rateLimit = require('express-rate-limit');

// API rate limiter
exports.apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: 'Too many requests, please try again later',
  standardHeaders: true, // Return rate limit info in the `RateLimit-*` headers
  legacyHeaders: false, // Disable the `X-RateLimit-*` headers
});

// More strict limiter for auth routes
exports.authLimiter = rateLimit({
  windowMs: 60 * 60 * 1000, // 1 hour
  max: 10, // limit each IP to 10 login attempts per hour
  message: 'Too many login attempts, please try again later',
  standardHeaders: true,
  legacyHeaders: false,
});
```

Helmet for HTTP Headers

```
// server/app.js
const express = require('express');
const helmet = require('helmet');
const app = express();

// Set security HTTP headers
app.use(helmet());
```

16. Deployment Guidelines

Environment Configuration

```
// .env.example
# Server Configuration
PORT=5000
NODE_ENV=development

# Database Configuration
MONGODB_URI=mongodb://localhost:27017/shopsphere

# JWT Configuration
JWT_SECRET=your_jwt_secret
```

```
JWT_EXPIRES_IN=7d

# CORS Configuration
CORS_ORIGIN=http://localhost:3000

# Cloudinary Configuration (for image uploads)
CLOUDINARY_CLOUD_NAME=your_cloud_name
CLOUDINARY_API_KEY=your_api_key
CLOUDINARY_API_SECRET=your_api_secret

# Stripe Configuration (for payments)
STRIPE_SECRET_KEY=your_stripe_secret_key
STRIPE_WEBHOOK_SECRET=your_stripe_webhook_secret

# Email Configuration (for notifications)
EMAIL_SERVICE=gmail
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_email_password

# Social Auth Configuration
GOOGLE_CLIENT_ID=your_google_client_id
GOOGLE_CLIENT_SECRET=your_google_client_secret
FACEBOOK_APP_ID=your_facebook_app_id
FACEBOOK_APP_SECRET=your_facebook_app_secret
APPLE_CLIENT_ID=your_apple_client_id
APPLE_TEAM_ID=your_apple_team_id
APPLE_KEY_ID=your_apple_key_id
APPLE_PRIVATE_KEY=your_apple_private_key
```

Docker Configuration

```
# Dockerfile for Node.js server
FROM node:14-alpine

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5000

CMD ["npm", "start"]
```

```
# docker-compose.yml
version: '3'
```

```
services:
  # MongoDB service
  mongo:
    image: mongo
    restart: always
    volumes:
      - mongo_data:/data/db
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: rootpassword

  # Backend API service
  backend:
    build: ./server
    restart: always
    depends_on:
      - mongo
    env_file:
      - ./server/.env
    ports:
      - "5000:5000"

  # Frontend service
  frontend:
    build: ./client
    restart: always
    depends_on:
      - backend
    ports:
      - "3000:80"

volumes:
  mongo_data:
```

Nginx Configuration

```
# nginx.conf
server {
    listen 80;
    server_name example.com;

    # Redirect HTTP to HTTPS
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name example.com;
    # SSL Configuration
    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    # SSL Settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-GCM-SHA
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:10m;

    # Frontend
    location / {
        root /var/www/html;
        try_files $uri $uri/ /index.html;

        # Caching static assets
        expires 1y;
        add_header Cache-Control "public, max-age=31536000, immutable";

        # Remove for assets with hash in filename
        location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
            expires 1y;
            add_header Cache-Control "public, max-age=31536000, immutable";
        }
    }

    # API
    location /api {
        proxy_pass http://backend:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }

    # Security headers
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";
    add_header X-Frame-Options "SAMEORIGIN";
    add_header Referrer-Policy "no-referrer-when-downgrade";
}
```

17. Testing Strategy

Unit Testing (Jest)

```
// src/components/Common/Button.test.js
import React from 'react';
import { render, screen, fireEvent } from '@testing-library/react';
import Button from './Button';

describe('Button component', () => {
  test('renders button with correct text', () => {
    render(<Button>Click me</Button>);
    const button = screen.getByText('Click me');
    expect(button).toBeInTheDocument();
  });

  test('applies primary variant class by default', () => {
    render(<Button>Button</Button>);
    const button = screen.getByText('Button');
    expect(button).toHaveClass('btn-primary');
  });

  test('applies specified variant class', () => {
    render(<Button variant="outline">Button</Button>);
    const button = screen.getByText('Button');
    expect(button).toHaveClass('btn-outline');
  });

  test('applies specified size class', () => {
    render(<Button size="large">Button</Button>);
    const button = screen.getByText('Button');
    expect(button).toHaveClass('btn-large');
  });

  test('applies full width class when fullWidth is true', () => {
    render(<Button fullWidth>Button</Button>);
    const button = screen.getByText('Button');
    expect(button).toHaveClass('btn-full-width');
  });

  test('calls onClick handler when clicked', () => {
    const handleClick = jest.fn();
    render(<Button onClick={handleClick}>Button</Button>);
    const button = screen.getByText('Button');
    fireEvent.click(button);
    expect(handleClick).toHaveBeenCalledTimes(1);
  });

  test('is disabled when disabled prop is true', () => {
    render(<Button disabled>Button</Button>);
    const button = screen.getByText('Button');
    expect(button).toBeDisabled();
  });
});
```

API Testing (Supertest)

```
// server/tests/product.test.js
const request = require('supertest');
const app = require('../app');
const mongoose = require('mongoose');
const Product = require('../models/Product');
const User = require('../models/User');
const { generateToken } = require('../utils/tokenUtils');

describe('Product API', () => {
  let token;
  let shopOwnerToken;
  let productId;

  beforeAll(async () => {
    // Connect to test database
    await mongoose.connect(process.env.MONGODB_URI_TEST);

    // Create test users
    const customer = await User.create({
      email: 'customer@test.com',
      password: 'password123',
      firstName: 'Test',
      lastName: 'Customer',
      userType: 'customer'
    });

    const shopOwner = await User.create({
      email: 'shop@test.com',
      password: 'password123',
      firstName: 'Test',
      lastName: 'Shop',
      userType: 'shop_owner'
    });

    // Generate tokens
    token = generateToken(customer._id);
    shopOwnerToken = generateToken(shopOwner._id);

    // Create test product
    const product = await Product.create({
      name: 'Test Product',
      description: 'This is a test product',
      price: 19.99,
      category: mongoose.Types.ObjectId(),
      inventory: {
        quantity: 10,
        sku: 'TEST-123'
      },
      shopId: shopOwner._id
    });
    productId = product._id;
  });

  afterAll(async () => {
    // Clean up test database
    await User.deleteMany({});
    await Product.deleteMany({});

    // Disconnect from test database
    await mongoose.connection.close();
  });

  describe('GET /api/products', () => {
    test('should get all products', async () => {
      const res = await request(app).get('/api/products');

      expect(res.statusCode).toBe(200);
      expect(res.body.products).toBeInstanceOf(Array);
      expect(res.body.products.length).toBeGreaterThan(0);
    });

    test('should filter products by category', async () => {
      const product = await Product.findById(productId);

      const res = await request(app)
        .get(`/api/products?categories=${product.category}`);

      expect(res.statusCode).toBe(200);
      expect(res.body.products).toBeInstanceOf(Array);
      expect(res.body.products.length).toBeGreaterThan(0);
      expect(res.body.products[0].category.toString()).toBe(product.category.toString());
    });
  });

  describe('GET /api/products/:id', () => {
    test('should get product by ID', async () => {
      const res = await request(app)
        .get(`/api/products/${productId}`);

      expect(res.statusCode).toBe(200);
      expect(res.body._id.toString()).toBe(productId.toString());
      expect(res.body.name).toBe('Test Product');
    });

    test('should return 404 if product not found', async () => {
      const nonExistentId = mongoose.Types.ObjectId();

      const res = await request(app)
        .get(`/api/products/${nonExistentId}`);

      expect(res.statusCode).toBe(404);
      expect(res.body.message).toBe('Product not found');
    });
  });

  describe('POST /api/products', () => {
    test('should require authentication', async () => {
      const res = await request(app)
        .post('/api/products')
        .send({
          name: 'New Product',
          description: 'New product description',
          price: 29.99,
          category: mongoose.Types.ObjectId(),
          inventory: {
            quantity: 5
          }
        });

      expect(res.statusCode).toBe(401);
    });

    test('should require shop owner role', async () => {
      const res = await request(app)
        .post('/api/products')
        .set('Authorization', `Bearer ${token}`)
        .send({
          name: 'New Product',
          description: 'New product description',
          price: 29.99,
          category: mongoose.Types.ObjectId(),
          inventory: {
            quantity: 5
          }
        });

      expect(res.statusCode).toBe(403);
    });

    test('should create a new product', async () => {
      const newProduct = {
        name: 'New Product',
        description: 'New product description',
        price: 29.99,
        category: mongoose.Types.ObjectId(),
        inventory: {
          quantity: 5
        }
      };

      const res = await request(app)
        .post('/api/products')
        .set('Authorization', `Bearer ${shopOwnerToken}`)
        .send(newProduct);
      expect(res.statusCode).toBe(201);
      expect(res.body.name).toBe(newProduct.name);
      expect(res.body.price).toBe(newProduct.price);
    });
  });

  describe('PUT /api/products/:id', () => {
    test('should update a product', async () => {
      const updateData = {
        name: 'Updated Product',
        price: 24.99
      };

      const res = await request(app)
        .put(`/api/products/${productId}`)
        .set('Authorization', `Bearer ${shopOwnerToken}`)
        .send(updateData);

      expect(res.statusCode).toBe(200);
      expect(res.body.name).toBe(updateData.name);
      expect(res.body.price).toBe(updateData.price);
    });
  });

  describe('DELETE /api/products/:id', () => {
    test('should delete a product', async () => {
      const res = await request(app)
        .delete(`/api/products/${productId}`)
        .set('Authorization', `Bearer ${shopOwnerToken}`);

      expect(res.statusCode).toBe(200);
      expect(res.body.message).toBe('Product deleted successfully');

      // Verify product is deleted
      const checkProduct = await Product.findById(productId);
      expect(checkProduct).toBeNull();
    });
  });
});
```

E2E Testing (Cypress)

```
// cypress/integration/checkout.spec.js
describe('Checkout Flow', () => {
  beforeEach(() => {
    // Login before each test
    cy.visit('/login');
    cy.get('input[name=email]').type('test@example.com');
    cy.get('input[name=password]').type('password123');
    cy.get('button[type=submit]').click();
    // Wait for login to complete
    cy.url().should('include', '/dashboard');
  });

  it('should add a product to cart and complete checkout', () => {
    // Navigate to products page
    cy.visit('/products');

    // Select a product
    cy.get('.product-card').first().click();

    // Add to cart
    cy.get('.add-to-cart-btn').click();

    // Navigate to cart
    cy.get('.cart-icon').click();

    // Verify product is in cart
    cy.get('.cart-item').should('have.length', 1);

    // Proceed to checkout
    cy.get('.btn-checkout').click();

    // Verify on checkout page
    cy.url().should('include', '/checkout');

    // Select first address
    cy.get('input[name="shipping-address"]').first().check();

    // Select card payment
    cy.get('#payment-card').check();

    // Fill card details
    cy.get('input[name="cardNumber"]').type('4242424242424242');
    cy.get('input[name="cardName"]').type('Test User');
    cy.get('input[name="expiry"]').type('12/25');
    cy.get('input[name="cvv"]').type('123');

    // Place order
    cy.get('.btn-place-order').click();

    // Verify order confirmation
    cy.url().should('include', '/order-confirmation');
    cy.contains('Order Confirmed').should('exist');
  });

  it('should show error for invalid card details', () => {
    // Add product to cart (reusing previous test steps)
    cy.visit('/products');
    cy.get('.product-card').first().click();
    cy.get('.add-to-cart-btn').click();
    cy.get('.cart-icon').click();
    cy.get('.btn-checkout').click();
    // Select address and payment method
    cy.get('input[name="shipping-address"]').first().check();
    cy.get('#payment-card').check();

    // Fill invalid card details
    cy.get('input[name="cardNumber"]').type('1111111111111111');
    cy.get('input[name="cardName"]').type('Test User');
    cy.get('input[name="expiry"]').type('12/25');
    cy.get('input[name="cvv"]').type('123');

    // Place order
    cy.get('.btn-place-order').click();

    // Verify error message
    cy.get('.error-message').should('exist');
    cy.contains('Invalid card number').should('exist');
  });
});
```