E-commerce App Structure Guide

React Frontend + Python Backend

Frontend (React)

```
src/
├── components/
│   ├── common/
│   │   ├── Header.jsx
│   │   ├── Footer.jsx
│   │   ├── Navbar.jsx
│   │   ├── Button.jsx
│   │   ├── ProductCard.jsx
│   │   ├── Rating.jsx
│   │   └── Loader.jsx
│   ├── auth/
│   │   ├── LoginForm.jsx
│   │   ├── RegisterForm.jsx
│   │   └── ForgotPassword.jsx
│   ├── products/
│   │   ├── ProductList.jsx
│   │   ├── ProductDetail.jsx
│   │   ├── ProductFilters.jsx
│   │   └── ProductSearch.jsx
│   ├── cart/
│   │   ├── Cart.jsx
│   │   ├── CartItem.jsx
│   │   └── CartSummary.jsx
│   ├── checkout/
│   │   ├── CheckoutForm.jsx
│   │   ├── ShippingForm.jsx
│   │   ├── PaymentForm.jsx
│   │   └── OrderSummary.jsx
│   └── user/
│       ├── Profile.jsx
│       ├── OrderHistory.jsx
│       └── OrderDetail.jsx
├── pages/
│   ├── HomePage.jsx
│   ├── ProductsPage.jsx
│   ├── ProductDetailPage.jsx
│   ├── CartPage.jsx
│   ├── CheckoutPage.jsx
│   ├── OrderConfirmationPage.jsx
│   ├── LoginPage.jsx
│   ├── RegisterPage.jsx
│   ├── ProfilePage.jsx
│   └── NotFoundPage.jsx
├── services/
│   ├── api.js          # API service wrapper
│   ├── auth.service.js
│   ├── product.service.js
│   ├── cart.service.js
│   └── order.service.js
│   └── orderService.js
├── hooks/
│   ├── useAuth.js
│   ├── useCart.js
│   └── useProducts.js
├── context/
│   ├── AuthContext.jsx
│   └── CartContext.jsx
├── utils/
│   ├── formatCurrency.js
│   ├── validateForm.js
│   └── localStorage.js
├── App.jsx
└── index.jsx
```

Key Frontend Components

1. **Common Components**
   - Reusable UI elements across the application
   - Standardizes the look and feel
2. **Auth Components**
   - User authentication flow
   - Registration, login, password reset
3. **Product Components**
   - Display and interaction with product catalog
   - Filtering, searching, and viewing product details
4. **Cart Components**
   - Shopping cart management
   - Add, remove, and update quantities
5. **Checkout Components**
   - Multi-step checkout process
   - Shipping, billing, and payment information collection
6. **Services**
   - API communication layer
   - Abstracts backend interactions
7. **Context & Hooks**
   - State management
   - Shared functionality across components

Backend (Python - FastAPI Example)

```
backend/
├── app/
│   ├── api/
│   │   ├── __init__.py
│   │   ├── auth.py
│   │   ├── products.py
│   │   ├── cart.py
│   │   ├── orders.py
│   │   └── users.py
│   ├── core/
│   │   ├── __init__.py
│   │   ├── config.py         # App configuration
│   │   ├── security.py       # Auth helpers, JWT
│   │   └── exceptions.py     # Custom exceptions
│   ├── db/
│   │   ├── __init__.py
│   │   ├── database.py       # DB setup and session
│   │   └── init_db.py        # DB initialization
│   ├── models/
│   │   ├── __init__.py
│   │   ├── user.py
│   │   ├── product.py
│   │   ├── category.py
│   │   ├── cart.py
│   │   └── order.py
│   ├── schemas/
│   │   ├── __init__.py
│   │   ├── user.py
│   │   ├── product.py
│   │   ├── category.py
│   │   ├── cart.py
│   │   └── order.py
│   ├── services/
│   │   ├── __init__.py
│   │   ├── auth_service.py
│   │   ├── product_service.py
│   │   ├── cart_service.py
│   │   ├── order_service.py
│   │   └── payment_service.py
│   ├── utils/
│   │   ├── __init__.py
│   │   └── helpers.py        # Utility functions
│   └── main.py               # Application entry point
├── tests/
│   ├── __init__.py
│   ├── conftest.py
│   ├── test_auth.py
│   ├── test_products.py
│   └── test_orders.py
```

```
├── alembic/                # Database migrations
│   ├── versions/
│   └── alembic.ini
├── static/                 # For uploaded files
│   └── images/
├── .env                    # Environment variables
├── requirements.txt
└── Dockerfile
```

Key Backend Components

1. **API Endpoints**
   - REST API routes organized by domain
   - Handles HTTP requests and responses
2. **Core**
   - Configuration, security, and foundation modules
   - Environment settings and application setup
3. **Database Layer**
   - Database connection and session management
   - Migration tools for schema changes
4. **Models**
   - Database table definitions using ORM
   - Relationships between entities
5. **Schemas**
   - Data validation and serialization
   - API request/response models
6. **Services**
   - Business logic implementation
   - Separates concerns from API handlers
7. **Utils**
   - Helper functions and shared utilities
   - Common functionality across the application

API Endpoints

```
# Auth
POST /api/auth/register          - Register new user
POST /api/auth/login             - User login
POST /api/auth/refresh-token     - Refresh JWT token

# Products
GET  /api/products               - List all products
GET  /api/products/{id}          - Get product details
GET  /api/products/category/{category_id} - Products by category
POST /api/products/search        - Search products

# Cart
GET    /api/cart                 - Get user's cart
POST   /api/cart/items           - Add item to cart
PUT    /api/cart/items/{id}      - Update cart item
DELETE /api/cart/items/{id}      - Remove from cart

# Orders
POST   /api/orders               - Create new order
GET    /api/orders               - Get user's orders
GET    /api/orders/{id}          - Get order details
PUT    /api/orders/{id}/status   - Update order status

# User
GET  /api/users/me               - Get user profile
PUT  /api/users/me               - Update user profile
GET  /api/users/me/orders        - Get user's order history
```

Database Schema

Main Tables

1. **Users**
   - id (PK)
   - email
   - password_hash
   - first_name
   - last_name
   - is_active
   - created_at
2. **Products**
   - id (PK)
   - name
   - description
   - price
   - inventory_count
   - category_id (FK)
   - image_url
   - created_at
3. **Categories**
   - id (PK)
   - name
   - parent_id (FK, self-referential)
4. **Cart_Items**
   - id (PK)
   - user_id (FK)
   - product_id (FK)
   - quantity
   - added_at
5. **Orders**
   - id (PK)
   - user_id (FK)
   - status
   - total_amount
   - shipping_address_id (FK)
   - billing_address_id (FK)
   - payment_method

- created_at

6. **Order_Items**
    - id (PK)
    - order_id (FK)
    - product_id (FK)
    - quantity
    - price_at_time
7. **Addresses**
    - id (PK)
    - user_id (FK)
    - address_line1
    - address_line2
    - city
    - state
    - postal_code
    - country
    - is_default

Communication Flow

1. User interacts with React frontend
2. Frontend makes HTTP requests to Python backend API
3. Backend processes requests, interacts with database
4. Backend returns JSON responses
5. Frontend updates UI based on responses

Development Steps

1. Set up project structure for both frontend and backend
2. Implement database models and migrations
3. Build API endpoints with authentication
4. Develop core frontend pages and components
5. Connect frontend to backend via API services
6. Implement cart functionality and state management
7. Build checkout flow with payment integration
8. Add user account management features
9. Test the entire application flow
10. Deploy frontend and backend to production

Technology Stack Options

Frontend

- React
- State Management: Redux, Context API, or React Query
- Styling: Tailwind CSS, Material UI, or Styled Components
- Routing: React Router
- Form Handling: Formik or React Hook Form
- API Client: Axios or Fetch API

Backend

- Python Frameworks: FastAPI, Django, or Flask
- ORM: SQLAlchemy, Django ORM, or Peewee
- Authentication: JWT, OAuth2, or Session-based
- Database: PostgreSQL, MySQL, or MongoDB
- Payment Processing: Stripe, PayPal, or Braintree
- Image Storage: AWS S3, Google Cloud Storage, or local file system

---

This structure provides a comprehensive foundation for building a full-featured e-commerce application with React and Python.