Project Report: E-Commerce Shopping Application (React + FakeStore API)

1. Introduction

This project is a fully functional e-commerce shopping application built with React and powered by the FakeStore API. The application demonstrates key concepts of modern front-end development such as API integration, state management, routing, authentication, persistent storage, and responsive design. The project not only replicates a real-world e-commerce platform but also emphasizes software engineering practices including loading/error states, form validation, reusable components, conditional rendering, and context-based state sharing.

2. Objectives

The primary objectives of this project are:

- To design and develop a user-friendly shopping experience with seamless navigation.
- To implement dynamic product handling through API fetching.
- To build a shopping cart system with persistence using localStorage.
- To integrate authentication and profile management for personalized experiences.
- To ensure a responsive and accessible design that works across devices.

3. Technologies Used

- **Frontend Framework**: React (TypeScript + Vite)
- **UI Library**: Chakra UI (responsive components, accessibility features)
- **API**: FakeStore API (20 products, 4 categories)
- **State Management**: React Context API (CartContext, AuthContext)
- **Data Persistence**: localStorage for cart and authentication state
- **Routing**: React Router for multi-page navigation
- **Utilities**: async/await API calls, debounce search, conditional rendering

4. Key Concepts Covered

The project covers fundamental front-end development concepts, including:

- API fetching with fetch and async/await
- State and props management in React
- Shopping cart functionality with localStorage persistence
- Product search with debounce (500ms delay)
- Category filtering and sorting
- Multi-page navigation with React Router
- Form handling (checkout and authentication forms)

- Conditional rendering for loading, error, and empty states
- Authentication system with login, signup, and profile management
- Responsive UI design (mobile-first approach)

5. Functional Modules

5.1 Home Page (/)

The home page showcases featured products (the first 8 items from the API) in a hero section with a banner and call-to-action. Product cards display the image, title, price, rating, and an "Add to Cart" button. It features loading skeletons while fetching data and a retry button on error.

5.2 Products Page (/products)

This page lists all products with **pagination** (8 products per page). A sidebar provides advanced **filtering** by category, price range, and rating. Users can also **sort** products and use a **debounced search** bar to filter by title in real time. The page supports a toggle between Grid and List view.

5.3 Category Page (/category/:categoryName)

Dedicated pages display products filtered by a specific category (e.g., electronics, jewelry). These pages are accessible via a dropdown in the navbar, show the product count, and support the same sorting and search functionalities as the main products page.

5.4 Product Details Page (/product/:id)

This page provides a detailed view of a single product. It features a large image with a **zoom on hover** effect, full product information, and a simulated **10% discount**. A quantity selector and an "Add to Cart" button are also present. The page includes a section for related products from the same category.

5.5 Shopping Cart Page (/cart)

The cart page lists all items with product images, titles, prices, and quantity controls. An order summary calculates the subtotal, a simulated **8% tax**, and a flat **$10 shipping fee** (free for orders over $100). The cart state is persistent via `localStorage`, and an empty state message is displayed when there are no items.

5.6 Checkout Page (/checkout)

This is a **multi-step form** that collects shipping and payment information. It features **real-time form validation** with error messages. An order summary is displayed before the user places the order. This is a **protected route**, requiring a user to be logged in.

5.7 Order Success Page (`/order-success`)

Upon a successful order, the user is redirected to a confirmation page with an order number and an estimated delivery date. The cart is cleared at this point to prepare for a new shopping session.

5.8 User Authentication & Profile

A login/signup modal allows for user authentication. The **`AuthContext`** manages the global state and persistence. When logged in, a user menu appears in the navbar with options for a profile page and order history (simulated). The profile page allows users to view and edit their information.

6. Core Components Developed

- **`ProductCard`**: A reusable component that accepts a product object and integrates with the `CartContext` for real-time updates.
- **`Products`**: The main component for displaying products, featuring enhanced loading/error states and a responsive grid layout.
- **`Navbar`**: An enhanced navigation bar with a dynamic cart counter, user authentication menu, and a sticky design.
- **`Cart`**: Manages the full cart functionality, including adding, removing, and updating items, as well as real-time totals.

7. Advanced Features

The application incorporates advanced features to improve performance and user experience:

- **Debounced search**: A 500ms delay on product search prevents excessive API calls.
- **Persistent authentication**: Login state is saved in `localStorage`.
- **Protected routes**: Checkout is only accessible to logged-in users.
- **Price formatting**: Prices are consistently displayed with a dollar symbol and two decimals.
- **Image fallback**: A default image is shown if a product image fails to load.
- **Loading states**: Skeletons, spinners, and disabled buttons provide clear visual feedback.
- **Error boundaries**: Ensures graceful handling of API and other errors.
- **Responsive design**: Optimized for a mobile-first approach.

8. Conclusion

This project successfully demonstrates the development of a modern, responsive, and feature-rich e-commerce application using React. It highlights best practices in state

management, API integration, authentication, and user experience design. By leveraging the FakeStore API, the application simulates real-world e-commerce flows including browsing, filtering, searching, adding to cart, managing orders, and user authentication. The modular design ensures scalability, while the focus on loading/error handling and responsive design enhances usability across devices.