Full Stack Development with MERN

Phase 4 : Project Design

Project Title – ShopEZ : One-Stop shop for online purchases

Team Members :

1. **Diviz Pandey (Team Lead) – Backend Developer**
2. **Bhavya Jain – Frontend Developer**
3. **Karmanya Batra – Database and Deployment**
4. **Anaswar L – Tester**

Team ID - SWTID1743669870

4.1 Problem–Solution Fit

Before writing code, it was critical to validate that our proposed solution truly addressed the identified user and business problems. This process of **problem-solution fit** ensured ShopEZ wouldn't just "work," but *matter*.

Identified Problems Recap:

- Users face confusing navigation and slow sites.
- Small sellers lack affordable, scalable e-commerce solutions.
- Admin dashboards in many platforms are cluttered and non-intuitive.
- Lack of real-time product and inventory control.

ShopEZ Solution Alignment:

- **User-Centric Design:** Intuitive, minimal UI with mobile responsiveness using Tailwind CSS.
- **Modular Architecture:** Separated frontend/backend with clearly scoped routes and logic.
- **Admin Empowerment:** Full control over inventory, orders, and users.
- **Scalability:** MERN stack allows database growth, route expansion, and performance optimization.

This phase focused on transforming these solution directions into visual and system-level designs.

4.2 Proposed Solution

ShopEZ is designed as a full-stack e-commerce platform with the following characteristics:

Core Functional Components

Module	**Description**
User Authentication	Secure registration/login using JWT; hashed passwords
Product Catalog	Display of items with filters and categories
Cart & Checkout	Real-time cart update, quantity changes, order placement
Admin Panel	CRUD for products, categories, users; view orders

Module	Description
User Dashboard	View past orders and manage account
REST API Layer	Routes for auth, products, orders, categories, users
State Management	Redux Toolkit with AsyncThunk for API integration
Responsive Design	Built using Tailwind CSS for all devices

4.3 Solution Architecture

The architecture of ShopEZ is componentized into three primary segments — **Frontend**, **Backend**, and **Database** — all of which communicate over defined interfaces.

A. Frontend Architecture (React.js)

- **SPA Structure:** Single Page Application for fast transitions and client-side routing.
- **Components:** Reusable blocks like ProductCard, CartItem, AdminTable, etc.
- **Pages:** Separate views for Home, Login, Register, Checkout, Admin Panel.
- **Routing:** react-router-dom manages transitions and route protection.
- **Redux Toolkit:** Global state for cart, auth, and product lists using AsyncThunk.
- **Auth Management:** JWT stored in memory/localStorage and attached via axios interceptors.
- **UI/UX:** Styled using Tailwind CSS with minimal design and focus on usability.

Frontend Folder Highlights:

- /src/pages – Pages for routing: Home, Login, ProductDetails, AdminDashboard
- /src/features – Redux slices: AuthSlice, ProductSlice, OrderSlice
- /src/utils – Helpers: price formatter, auth guard
- /src/assets – Icons, logos, images

B. Backend Architecture (Node.js + Express)

- **RESTful API Design:** Separate endpoints for users, products, orders, cart, admin.
- **Authentication Flow:**
    - User registers/logins → Server generates JWT access + refresh tokens
    - Tokens stored in cookies/localStorage

    - Middleware verifies and protects routes (verifyJWT, verifyAdminJWT)
- **Middleware Components:**
    - Auth middleware
    - Error handling
    - Input validation
    - Pagination for large datasets
- **Controllers:** Handle business logic, input sanitation, and DB interactions.

Key Routes:

- POST /api/users/register – Create account
- POST /api/users/login – Auth login
- GET /api/products – List products
- POST /api/orders – Checkout endpoint
- GET /api/admin/users – Admin: view all users

Backend Folder Highlights:

- /controllers – Logic handlers: productController.js, authController.js
- /router – Route files: productRoutes.js, authRoutes.js
- /models – Mongoose schemas for User, Product, Order, Category
- /middleware – Auth + pagination logic

C. Database Architecture (MongoDB)

- **Collections:**
    - Users: name, email, hashed password, role, address
    - Products: title, description, price, category, image, stock
    - Orders: userId, product list, amount, shipping address, status
    - Categories: name, description
    - Wishlist (planned): userId, productIds[]
- **Schema Definition:**
    - Enforced using Mongoose
    - Indexed on frequent queries like email, product name

- **Relationships:**
    - Orders reference userId
    - Wishlist references userId and productIds
    - Embedded product snapshots for order consistency

4.4 UI Design & Wireframing

Using **Figma**, we built interactive wireframes for all key pages:

User Interface:

- **Home Page:** Hero banner, featured products, category tiles.
- **Product Page:** Image carousel, specs, reviews, "Add to Cart."
- **Cart:** Quantity controls, price summary, "Proceed to Checkout."
- **Checkout:** Address entry, payment method (planned), place order.
- **Order Summary:** View all past orders with status indicators.

Admin Interface:

- **Dashboard Overview:** Sales snapshot (planned), user stats.
- **Product Management:** Add/edit/delete with form validation.
- **Order Management:** View orders with update status dropdown.
- **User Management:** View all users with role change/delete options.

Every component was designed for **mobile-first responsiveness**, ensuring fluid resizing and layout shifts across screen sizes.

4.5 Component Interaction & Flow

Here's how key components interact in real-time:

[User]

↓ Login/Register

[React Frontend]

↓ API Request

[Express Backend]

↓ Query

[MongoDB]

↑ Data Fetch

[Backend]

↑ Response

[Redux State Updated]

All communications are asynchronous, and Redux handles dispatching state changes upon successful or failed API calls. For example:

- Adding a product to cart dispatches an async thunk → Backend → Redux state updates → UI re-renders.

4.6 Summary

The design phase of ShopEZ served as the **blueprint** for actual implementation. With the problem-solution fit validated, we developed a scalable, modular, and secure architecture. Every major component — from data flow to user interface — was planned to ensure minimal friction during development and maximum value to the end user.

From reusable React components and Redux logic to secure JWT-based backend APIs and a well-structured NoSQL database, the project was designed with modern best practices in mind. This ensures not only smooth development but future readiness for features like payment integration, AI recommendations, and multi-seller capabilities.