# Project Design Phase Documentation

**Project:** MERN Ecommerce

**Date:** 7 August 2025

**Team ID:** [PNT2025TMID10691]

## Introduction

The Project Design Phase transforms the gathered requirements into a comprehensive blueprint that guides the development process. This phase plays a pivotal role in ensuring the system's functionality, usability, maintainability, security, and scalability objectives are met. The design for the MERN Ecommerce application emphasizes modularity, a seamless user experience, and robust backend architecture to support high performance and future scalability.

## System Architecture Overview

The MERN stack forms the foundation: MongoDB as the NoSQL database for flexible data modeling, Express.js as the backend web framework, React.js to build a dynamic and responsive frontend, and Node.js as the server environment running JavaScript across the stack.

### Architecture Components and Interaction:

| Component | Description | Key Interaction Points |
|---|---|---|
| Frontend (React) | Provides user interfaces, handles user inputs, and presents real-time product data to users. | Communicates with backend APIs over HTTP/HTTPS |
| Backend (Express + Node.js) | Processes business logic, handles authentication, API routing, and interacts with the database. | Receives requests from frontend, sends responses, manages user sessions |

| | | |
|---|---|---|
| Database (MongoDB) | Stores collections such as Users, Products, Orders, Reviews with flexible schema design. | Provides data via queries; supports indexing for speed |

The modular separation ensures that UI, business logic, and data handling remain loosely coupled, simplifying maintenance and testing. RESTful APIs standardize communication, allowing extensibility and potential mobile app integration in the future.

A typical data flow starts when a user interacts on the frontend (e.g., adds products to cart). The frontend sends API requests to the backend, which applies business logic and reads/writes data in MongoDB, then returns the updated state to display.

## User Roles and Access Control

The system supports multiple user roles, each with tailored access rights:

| User Role | Access Rights & Permissions | Description of Capabilities |
|---|---|---|
| Admin | Full access to all modules including product management, user management, order monitoring, and system settings. | Controls site-wide content, user roles, and order statuses. |
| Seller | Can add, update, and delete their product listings; view order statuses for their products. | Manages own catalogue and interacts with order reports. |
| Buyer | Can browse products, manage cart, place and track orders, and update personal profiles. | Performs shopping-related activities with limited access. |

Role-based access control (RBAC) is enforced through middleware ensuring users can only perform authorized actions, critical for securing sensitive operations like product deletions or order modifications.

## UI/UX Design

The UI design aims to provide a clean, intuitive, and fully responsive experience that adapts across desktops, tablets, and smartphones.

**Design Principles Implemented:**

- Use of a consistent color scheme and typography for brand identity
- Intuitive navigation with breadcrumb trails, hover effects for actionable items
- Accessibility considerations: ARIA roles, keyboard navigability, color contrast

## Key Page Wireframes and Features

| Page | Purpose | Key Features & UI Elements |
|---|---|---|
| Homepage | Welcomes users and displays featured products | Search bar, carousel banner, featured categories, user login/signup prompts |
| Product Listing | Shows products with filters and sorting options | Category filters, price range slider, pagination |
| Product Details | Detailed info, reviews, add to cart | Images gallery, product specs, customer ratings, wishlist button |
| Cart & Checkout | Shopping cart overview and payment processing | Cart editing, promo code entry, multi-step checkout with payment gateway integration like Razorpay |
| Admin Dashboard | Administrative controls and analytics | User management, product inventory, order tracking, sales reports |

Wireframes or mockups created in design tools (e.g., Figma) guide frontend implementation, ensuring cohesive user flow and actionable feedback loops.

## Functional Design Specifications

Functional modules are designed to address core ecommerce needs:

| Feature | Description | Key Design Considerations |
|---|---|---|
| User Registration/Login | Multi-method authentication (email/password, OAuth via Google/Facebook) | Data validation, email confirmation, password hashing |
| Product Catalog | CRUD operations for products including images, descriptions, prices | Efficient search and filter mechanisms for UX |
| Shopping Cart & Checkout | Persistent carts, secure checkout flow with payment gateway integration | Transaction integrity, error handling, order confirmation |
| Order Management | Tracking order statuses, history, and user notifications | Real-time status updates, email alerts |
| Reviews & Ratings | User-generated feedback on products | Moderation and aggregation |

Authentication tokens (JWTs) secure user sessions, with role-based middleware protecting sensitive routes.

## Database Design

MongoDB collections are modeled for flexibility and performance. Key entities are:

| Entity | Description | Key Attributes | Relationships |
|---|---|---|---|
| Users | Stores user profiles and credentials | userId, name, email, hashedPassword, role, createdAt | One-to-many: Orders, Reviews |
| Products | Holds product details and inventory info | productId, sellerId, name, description, price, stockQty, category, images, createdAt | Related to Sellers and Reviews |
| Orders | Captures transaction details for purchases | orderId, buyerId, products[], totalAmount, status, createdAt | Linked to Users and Products |

| | | | |
|---|---|---|---|
| Reviews | Contains product feedback from buyers | reviewId, productId, userId, rating, comment, createdAt | Connected to Users and Products |

Indexes on product name, category, and user email improve query speed. Mongoose schema validation enforces constraints like required fields and data types.

## API Design

Backend APIs follow RESTful conventions to support frontend operations:

| Endpoint | HTTP Method | Request Body / Params | Response | Authentication | Notes |
|---|---|---|---|---|---|
| /api/users/register | POST | { name, email, password } | User object | No | Validation, email confirmation |
| /api/users/login | POST | { email, password } | JWT token | No | Token-based authentication |
| /api/products | GET | Query params: category, price range, search term | Products list | No | Supports pagination |
| /api/products/:id | GET | Product ID | Product details | No | |

| | | | | | |
|---|---|---|---|---|---|
| /api/orders | POST | Order details | Order object | Yes | Protected; buyer only |
| /api/admin/products | POST/PUT/DELETE | Product data | Updated product | Yes (Admin role) | CRUD operations for admins |

Error responses return meaningful HTTP status codes (e.g., 400 for validation errors, 401 unauthorized, 404 resource not found). Input payloads are sanitized to prevent injection attacks.

## Non-Functional Requirements Addressed in Design

Security is paramount due to the sensitive nature of user data and financial transactions. Measures include:

- Passwords hashed using bcrypt with salts
- HTTPS enforced for all client-server communications
- Input validation and sanitization on both frontend and backend
- CORS policies and rate limiting on APIs

Performance optimizations involve caching frequent queries, lazy loading images, and minimizing frontend bundle sizes. The design envisions scalability through MongoDB's sharding capability and stateless backend servers to enable horizontal scaling.

Reliability is enhanced with proper error logging, fallback mechanisms, and database backups.

## Tools and Technologies Used

| Tool/Technology | Purpose |
|---|---|
| Visual Studio Code | Code editor |
| Git & GitHub | Version control and code repository |
| MongoDB & Mongoose | Database and ODM for schema validation |
| Express.js & Node.js | Backend web framework and runtime |
| React.js | Frontend UI library |
| Redux Toolkit | State management |
| Razorpay API | Payment gateway integration |
| Postman | API testing |
| Figma/Adobe XD | UI/UX design and wireframing |

## Risk Analysis and Mitigation

Potential risks include delays in payment gateway integration, security vulnerabilities, and performance bottlenecks as user base grows. The team mitigates these risks by:

- Modular and incremental development enabling early testing
- Security best practices like penetration testing and code reviews
- Performance profiling and load testing during staging
- Continuous integration and delivery pipelines for rapid feedback