

ApniMart

Premier E-commerce Application

Full-Stack MERN Architecture

v1.0.0 | Development

Created By
Mausam Kar
2025 Edition

Abstract

Executive Summary

The rapid digitalization of retail commerce has necessitated robust, scalable, and user-centric e-commerce solutions. **ApniMart** is a full-stack web application designed to bridge the gap between local vendors and global consumers. Built upon the MERN (MongoDB, Express.js, React, Node.js) stack, the platform prioritizes performance through a decoupled architecture. This report details the development lifecycle of ApniMart, highlighting its secure authentication protocols (JWT), real-time inventory management, and seamless payment integration via Stripe. The project demonstrates a modern approach to handling complex state management using Redux Toolkit and optimizes frontend performance using the Vite build tool.

Contents

1	Introduction	3
1.1	Project Scope	3
1.2	Problem Statement	3
1.3	Proposed Solution	3
2	System Architecture & Technology	4
2.1	The MERN Stack	4
2.2	Architecture Diagram	4
2.3	State Management Strategy	4
3	Module Specifications	5
3.1	User Authentication	5
3.2	Shopping Cart & Checkout	5
3.3	Admin Dashboard	5
4	Database Design	6
5	Implementation & DevOps	7
5.1	DevOps Pipeline	7
5.2	Project Folder Structure	7
5.3	Installation Commands	7
6	Conclusion & Future Scope	8
6.1	Future Enhancements	8
6.2	Conclusion	8

Chapter 1

Introduction

1.1 Project Scope

ApniMart is engineered to serve as a comprehensive B2C (Business-to-Consumer) platform. The primary objective is to provide a seamless shopping experience for users while offering granular control to administrators. Unlike static catalog sites, ApniMart features dynamic inventory tracking, secure role-based access control, and automated payment processing.

1.2 Problem Statement

Many existing open-source e-commerce solutions suffer from distinct disadvantages:

- **Bloated Codebases:** Difficulty in customization due to legacy code.
- **Performance Bottlenecks:** Slow load times caused by older bundlers (like Webpack without optimization).
- **Security Vulnerabilities:** Weak authentication flows that leave user data susceptible to XSS and CSRF attacks.
- **Monolithic Limitations:** Tightly coupled front-end and back-end logic making scaling difficult.

1.3 Proposed Solution

ApniMart addresses these issues through a modern engineering approach:

1. **Decoupling:** Separating the Client (Frontend) and Server (Backend) allows for independent scaling and maintenance.
2. **Security First:** Implementing a dual-token system (Access & Refresh tokens) ensures robust session management.
3. **Performance:** Utilizing **Vite** for lightning-fast HMR (Hot Module Replacement) and efficient bundling.
4. **Maintainability:** Adopting a "Service-Controller" pattern in the backend to keep business logic separate from API routing.

Chapter 2

System Architecture & Technology

2.1 The MERN Stack

The project utilizes the MERN stack to ensure a unified language (JavaScript) across the entire application lifecycle.

MongoDB (Database): Chosen for its flexible schema (NoSQL), allowing for nested product attributes without complex migrations.

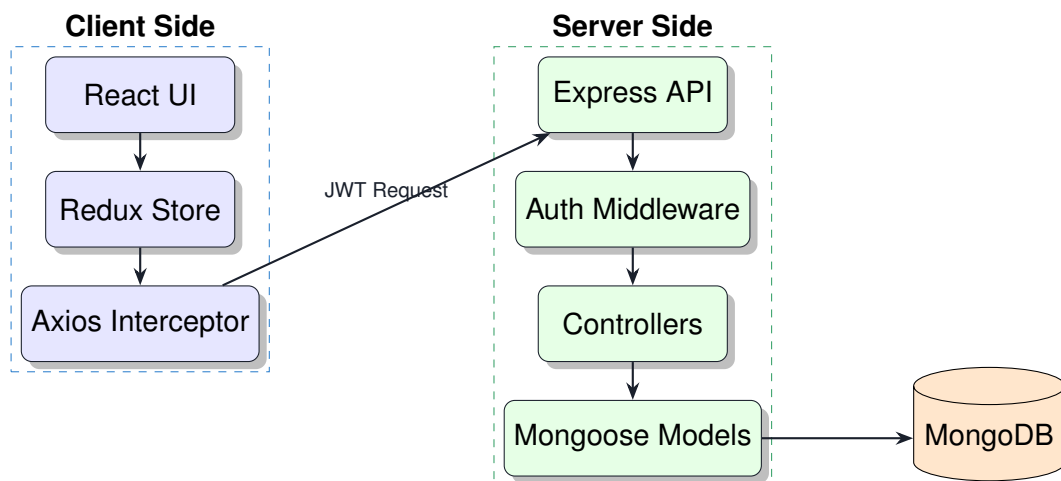
Express.js (Backend): A minimal framework handling routing and middleware integration.

React.js (Frontend): Used to build a Single Page Application (SPA) where content updates dynamically.

Node.js (Runtime): Handles non-blocking I/O operations, critical for concurrent user requests.

2.2 Architecture Diagram

Below is the visual representation of the decoupled data flow.



2.3 State Management Strategy

ApniMart moves beyond simple `useState`. It implements **Redux Toolkit** for global state management. This is crucial for features like the **Shopping Cart**, where data needs to be accessible across the entire application (e.g., updating the cart icon number in the Navbar when a user adds an item on the Product Details page).

Chapter 3

Module Specifications

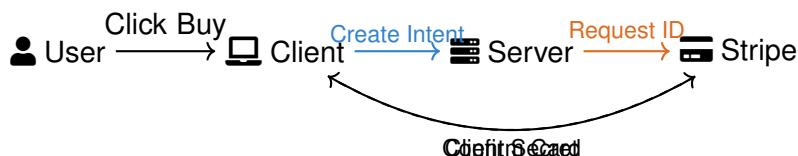
3.1 User Authentication

Security is the backbone of ApniMart. The system uses **Bcrypt** for hashing passwords before storage.

- **Registration:** Validates email format and password strength.
- **Dual-Token Flow:**
 1. **Access Token:** Short lifespan (15 mins), used for API authorization.
 2. **Refresh Token:** Long lifespan (7 days), stored in a secure, HTTP-Only cookie.

3.2 Shopping Cart & Checkout

The cart logic checks stock availability in real-time. If User A has an item in their cart, but User B buys the last unit, User A is notified upon checkout.



3.3 Admin Dashboard

The administrative panel allows for complete store management without touching code.

- **CRUD Operations:** Admins can Create, Read, Update, and Delete products.
- **Bulk Uploads:** The system includes a seeding script that parses a local directory structure to automatically upload images to Cloudinary and populate MongoDB.

Chapter 4

Database Design

Although MongoDB is non-relational, the data models are referenced to maintain integrity.

Schema Definitions

User Schema: Stores name, email, password_hash, role (Admin/User), and order_history.

Product Schema: Contains title, description, price, stock_quantity, images (array of URLs), and references to Category models.

Order Schema: A complex object containing userId, an array of product_items (snapshot of price at time of purchase), payment_status, and delivery_status.

Chapter 5

Implementation & DevOps

5.1 DevOps Pipeline

The application is structured for cloud environments.

- **Frontend:** Deployed via Vercel, leveraging edge caching.
- **Backend:** Hosted on Node.js compatible environments (e.g., Render).
- **Environment Variables:** Sensitive keys (Stripe Secrets, MongoDB URI) are abstracted into `.env` files.

5.2 Project Folder Structure

Directory Tree

```
d: client/ Frontend React Application  src/  components/ Reusable UI components  redux/
Global state slices  server/ Backend Node.js API  controllers/ Request handlers  models/
Database Schemas  route/ API Route definitions  README.md
```

5.3 Installation Commands

Terminal Setup

1. Clone Repository `git clone https://github.com/mausamkar/apnimart.git`
2. Server Setup `cd server npm install npm run dev`
3. Client Setup `cd client npm install npm run dev`

Chapter 6

Conclusion & Future Scope

6.1 Future Enhancements

- **AI Recommendations:** Integrating a Python-based microservice to recommend products based on user browsing history.
- **Wishlist Functionality:** Allowing users to save items for later.
- **Multi-Vendor Support:** Upgrading the User schema to allow "Vendor" roles who can manage their own subset of products.

6.2 Conclusion

ApniMart successfully demonstrates the capability of the MERN stack to build a production-ready e-commerce application. By adhering to modern security standards (JWT) and performance best practices (indexing, CDN usage), the platform provides a robust foundation for digital retail. The modular code structure ensures that the application remains maintainable and scalable for future development.