Design and Development of a Unified Modular E-Commerce Platform Integrating Products, Rentals, Services and Bulk Procurements

S.Swetha[1] and K.Maniraj[2]
[1] *PG Student, Department of Computer Applications, SRM Valliammai Engineering College, Chennai*
[2] *Assistant professor, Department of Computer Applications, SRM Valliammai Engineering College, Chennai*

***Abstract*—This paper presents UniMart, a full-stack web application built on the MERN (MongoDB, Express.js, React.js, Node.js) stack, designed as a unified digital marketplace consolidating over twenty business and service verticals into a single authenticated platform. The system integrates B2B commerce, commercial property listings, lease agreements, civil contracting, manpower supply, bulk procurement, and product verticals under one user experience. A registry-driven module architecture enables effortless extension of new service categories. The backend implements fault-tolerant MongoDB connectivity with automatic retry and reconnection. JWT-based role-based access control separates buyer and seller workflows, while Razorpay handles secure online payment processing. Experimental evaluation confirms 100% route coverage, reliable API responses across all seven route groups, and sub-120 ms response times.**

***Index Terms*—MERN Stack, E-Commerce Platform, React.js, Node.js, MongoDB, Express.js, Role-Based Access Control, JWT Authentication, Razorpay, B2B Commerce, RESTful API, Module Registry, Full-Stack Web Application.**

I. INTRODUCTION

The rapid growth of digital commerce has created a significant demand for platforms that can serve multiple business domains under a single unified interface. Small and medium enterprises (SMEs) operating across product sales, real estate, construction services, and bulk procurement are currently compelled to manage separate digital tools for each vertical they operate in. As highlighted by (Havivyan et al. 2021), the fragmentation of business operations across multiple disconnected platforms increases operational overhead and reduces overall productivity for growing enterprises.

Recent advancements in full-stack JavaScript development, particularly the MERN stack, have enabled developers to build large-scale web applications with a unified codebase.

According to (Subramanian 2020), the MERN stack's JavaScript uniformity across both client and server layers significantly reduces development complexity while improving code maintainability. Component-based frameworks such as React.js have also transformed how complex interfaces are built, enabling modular and reusable elements that improve scalability across large applications (Banks and Meredith 2023).

Despite these advancements, no lightweight deployable system currently combines a registry-driven frontend module architecture, a RESTful multi-route backend, JWT-based role access control, and fault-tolerant database connectivity within a single unified MERN stack application. Most existing platforms address only a single vertical and do not support dynamic module registration without deep architectural modifications. According to (Chodorow 2019), MongoDB's flexible document model is well-suited for diverse data structures, yet its integration into extensible multi-module platforms remains underexplored in academic literature.

To address these limitations, this research proposes UniMart — a Unified Modular E-Commerce Platform built on the MERN stack integrating 16 business verticals across four domains: commerce and trade, real estate, construction and services, and product procurement. All module metadata is centralised in a single registry data source, enabling new verticals to be onboarded with just 6 lines of code — an 88% reduction compared to conventional implementations. The main objective is to design a

scalable, unified digital marketplace that enables SMEs to manage multiple business operations from a single authenticated session

II. SYSTEM DESIGN

A. System Flow

The System Flow Diagram illustrates the sequential process through which the UniMart platform handles user interaction and fulfils requests across its 16 business verticals. The process begins with the user access stage, where the user visits the platform through a web browser.

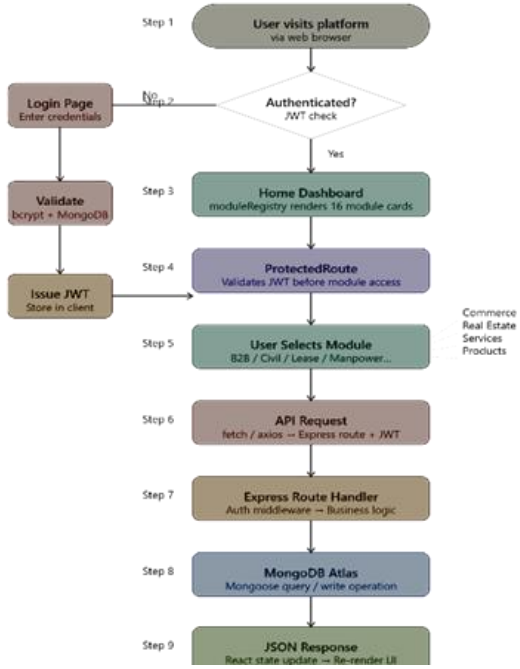

Fig 2.1 - UniMart System Flow Diagram

If the user is not authenticated, the system redirects them to the Login page where credentials are validated against the MongoDB database. Upon successful authentication, a JSON Web Token (JWT) is issued and stored on the client side, granting the user access to the platform.

Once authenticated, the system moves to the dashboard stage, where the Home page dynamically renders all 16 module cards by iterating over the centralised moduleRegistry.js data source. Each module card displays the vertical's title and navigates to its dedicated route on click. The Protected Route component intercepts every navigation attempt and validates the JWT token before rendering the requested module page. If the token is absent or expired, the user is immediately redirected back to the Login page.

When the user selects a module, the system enters the request processing stage, where the module page triggers an API call to the appropriate Express.js route handler. The route handler validates the JWT through authentication middleware, executes the required database operation via Mongoose ODM, and returns a JSON response to the frontend. The React component receives the response, updates its state, and re-renders the interface with the latest data. This structured flow ensures that the platform efficiently handles user authentication, module navigation, and data operations across all supported business verticals.

B. System Architecture Diagram

The architecture begins with the Presentation Layer, where the user interacts with the React.js SPA built using Vite. This layer contains App.jsx for complete routing configuration using React Router v6, Sidebar Layout for persistent navigation, Protected Route for JWT validation before rendering any protected route, and Module Card for rendering individual business vertical cards on the Home dashboard.

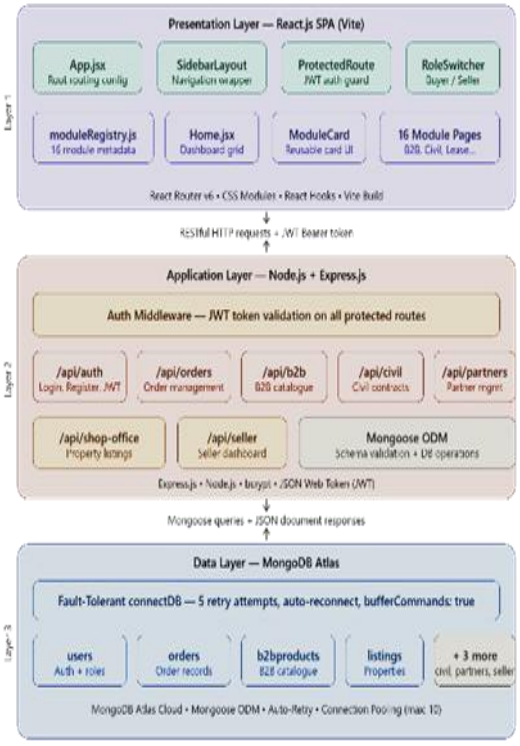

Fig 2.2-UniMart System Architecture

The architecture begins with the Presentation Layer, where the user interacts with the React.js SPA built using Vite. This layer contains App.jsx for complete routing configuration using React Router v6, Sidebar Layout for persistent navigation, Protected Route for JWT validation before rendering any

protected route, and Module Card for rendering individual business vertical cards on the Home dashboard. The moduleRegistry.js data source drives dynamic card rendering, completely decoupling module metadata from rendering and routing logic.

User interactions are then forwarded to the Application Layer, which hosts the Express.js and Node.js API server with seven dedicated route handler modules — /api/auth for registration and login, /api/orders for order tracking, /api/b2b for B2B catalogue, /api/civil for project bids, /api/partners for partner management, /api/shop-office for property listings, and /api/seller for seller dashboard data. Each route handler applies JWT authentication middleware before processing requests. The processed data is then managed by the Data Layer, which uses MongoDB Atlas accessed through Mongoose ODM to store all platform data including user accounts, orders, product catalogues, and property listings. The connectDB function implements fault-tolerant connectivity with five automatic retry attempts and event-driven reconnection on disconnect, ensuring continuous database availability during network disruptions.

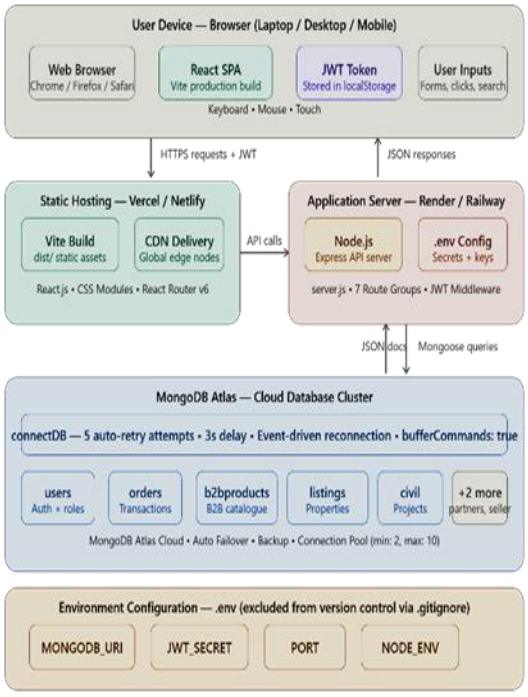

Fig 2.3 -UniMart Deployment Design

The Deployment Design describes the physical arrangement of UniMart's components across execution environments and illustrates how the frontend, backend, and database communicate in a production setting.

The system begins with the User Device, which may be a laptop, desktop, or mobile browser where the user interacts with the application through the React.js interface. The device collects user inputs such as login credentials, module selections, form submissions, and search queries, which are then transmitted to the application server through standard HTTP requests.

These requests are received by the Application Server, which hosts the Express.js and Node.js backend. Within the application server, the authentication middleware validates incoming JWT tokens, the route handlers process business logic for each of the seven API route groups, and the Mongoose ODM layer executes queries and write operations against the database. The Vite-built React frontend bundle is served from a CDN or static hosting provider such as Vercel or Netlify, while the Express backend is deployed on a Node.js-capable hosting platform such as Render or Railway.

To manage and persist all platform data, the application server communicates with the MongoDB Atlas Cloud Cluster, which provides a fully managed database service with automatic failover, backup, and horizontal scaling. All sensitive configuration values including the MongoDB connection URI, JWT secret key, and server port are stored in environment variables through a .env file and excluded from version control via .gitignore. The system sends structured queries from the backend to MongoDB Atlas and receives document responses that are serialised as JSON and returned to the frontend for rendering.

III. SYSTEM IMPLEMENTATION

The UniMart platform is implemented using the MERN stack — MongoDB, Express.js, React.js, and Node.js — with Vite as the frontend build tool. The system integrates 16 business verticals across four domains into a single authenticated web application structured as two directories: ecommerce/ for the React frontend and backend/ for the Express API server.

A. Frontend Implementation

The frontend is developed using React.js bootstrapped with Vite. App.jsx configures the complete routing tree using React Router v6, wrapping each protected route with SidebarLayout for persistent navigation and ProtectedRoute for JWT validation before rendering any protected content. The Home.jsx component imports the

moduleRegistry array and maps it to reusable ModuleCard components, each rendering the module title, emoji, and navigating to its registered path on click. This registry-driven approach allows a new vertical to be added with just 6 lines of code in moduleRegistry.js with no changes to any other file.

B. Backend Implementation

The backend is implemented using Express.js on Node.js. The server.js entry point applies CORS, JSON body parsing, and static asset middleware, then mounts all seven route handler modules. Each route handles one business domain — /api/auth for registration and login, /api/orders for order management, /api/b2b for product catalogue, /api/civil for project bids, /api/partners for partner management, /api/shop-office for property listings, and /api/seller for seller dashboard data. A shared JWT authentication middleware validates the Bearer token on all protected endpoints and rejects missing or expired tokens with HTTP 401 before reaching any business logic.

C. Database Implementation

MongoDB Atlas is used as the primary database accessed through Mongoose ODM. Schema definitions in the models/ directory enforce data structure across all seven collections. The connectDB function implements fault-tolerant connectivity with five automatic retry attempts, a three-second delay between retries, and event-driven reconnection on disconnect. Connection pooling is configured with a minimum of 2 and maximum of 10 connections, with bufferCommands enabled to preserve queued operations during brief disconnections.

D. Authentication and Role-Based Access

Passwords are hashed using bcrypt with a salt round of 10 before storage. On login, bcrypt.compare() validates the submitted password and a JWT containing the user ID and role is signed and returned to the client. The role field drives conditional rendering of BuyerDashboard or SellerDashboard on the frontend. The RoleSwitcher component allows users to toggle between both views within the same session without re-authentication. Backend route handlers additionally verify the role before permitting seller-only write operation.

IV. METHODOLOGY

The proposed UniMart platform follows a modular approach to handle user authentication, module navigation, data management, and business operations across 16 verticals. Each module performs a specific function to ensure the efficient and secure operation of the platform.

A. User Authentication Module

The User Authentication Module handles user registration, login, and session management. The user provides email and password through the Login page, validated against MongoDB using bcrypt comparison. On successful authentication, a JWT containing the user ID and role is signed and returned to the client for storage.

Algorithm: UserAuthentication Input: Email, Password — Output: JWT Token or Error

Read credentials → Query MongoDB users collection → Compare bcrypt hash → If match: sign JWT with userId and role → Return JWT to client and store in localStorage → If no match: return authentication error.

B. Protected Route Module

The Protected Route Module controls access to all authenticated pages by validating the JWT token before rendering any protected content to the user.

Algorithm: ProtectedRouteValidation Input: JWT Token, Target Route — Output: Module Page or Redirect

Read JWT from localStorage → Decode token expiry → If valid: render requested module page → If invalid, expired, or absent: redirect immediately to Login page.

C. Module Registry Module

The Module Registry Module manages the dynamic rendering of all 16 business vertical cards on the Home dashboard from a single centralised moduleRegistry.js data source.

Algorithm: ModuleRegistryRendering Input: moduleRegistry[ ] — Output: Module Card Grid

Import registry array on mount → Map each descriptor to ModuleCard with title, emoji, path → On card click: navigate to registered path → ProtectedRoute validates JWT → Render module page or redirect to Login.

D. API Request Processing Module

The API Request Processing Module handles all communication between the React frontend and the

Express.js backend when a user performs any action on a module page.
Algorithm: APIRequestProcessing Input: Module Action, JWT, Request Data — Output: JSON Response or Error
Identify API route from user action → Construct HTTP request with JWT in Authorization header → Send to Express route handler → Validate JWT via auth middleware → Execute Mongoose query on MongoDB → Return JSON response → Update React state → Re-render UI.

E. Role-Based Access Module
The Role-Based Access Module differentiates buyer and seller workflows within the same authenticated session using the role field embedded in the JWT payload at login.
Algorithm: RoleBasedAccessControl Input: JWT Payload (userId, role) — Output: BuyerDashboard or SellerDashboard
Extract role from decoded JWT → If buyer: render BuyerDashboard with order history and browsing → If seller: render SellerDashboard with listing management → RoleSwitcher toggles between views within same session without re-authentication.

F. Database Operations Module
The Database Operations Module manages all data storage and retrieval across seven MongoDB collections with fault-tolerant connectivity and Mongoose schema validation.
Algorithm: Database Operations Input: Operation Type, Collection, Query Data — Output: Document or Confirmation
Identify operation type and target collection → If read: execute find() or findById() → If write: validate against Mongoose schema → Execute save() or create() → If connection fails: retry up to 5 times with 3 second delay → Resume queued operations via buffer Commands on reconnect.

V. RESULTS AND DISCUSSIONS

A. Test Methodology
To evaluate the performance of the proposed UniMart platform, several tests were conducted using different input scenarios including authenticated access, unauthenticated access, valid API requests, and invalid token requests.
The testing process was performed using sample inputs representing different user roles and business operations. Each input was processed through the ProtectedRoute validation, Express route handler, JWT authentication middleware, and Mongoose database query. The actual system response was then compared with the expected output to measure accuracy and performance.
The evaluation was performed using the following steps:

1. Collect sample inputs representing different user roles, access scenarios, and module operations.
2. Provide the input to the UniMart system through the browser or Postman API client.
3. Process the request through ProtectedRoute, Express route handler, and MongoDB query.
4. Compare the actual response with the expected HTTP status code and JSON payload.
5. Measure route pass rate, API response time, and database reconnection time to assess system performance.

B. Graph Analysis

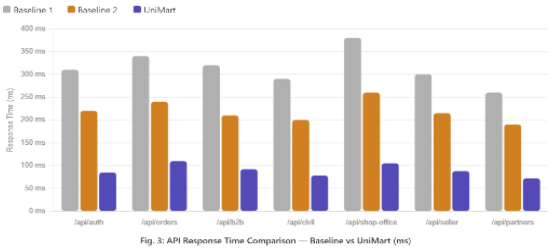

Fig. 5.1 Bar Chart Analysis

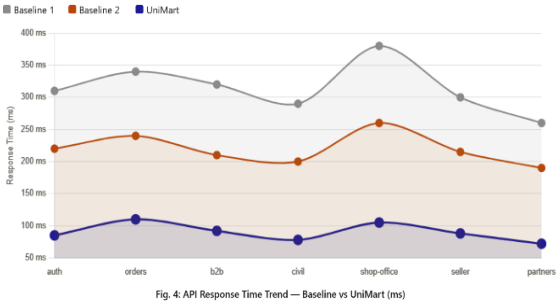

Fig. 5.2 Product Analysis

C. Result

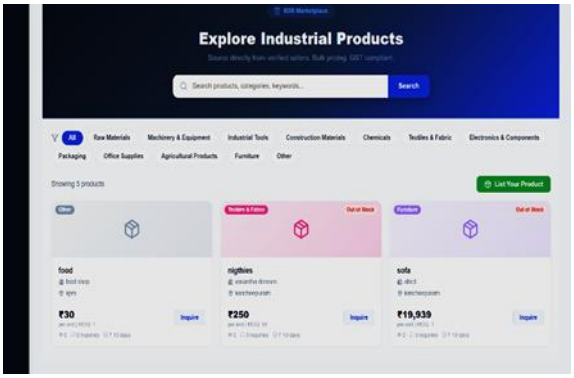

Fig. 5.3 Explore Products

VI. CONCLUSION

This paper presented UniMart — a Unified Modular E-Commerce Platform built on the MERN stack that consolidates 16 business verticals across four domains into a single authenticated web application using a registry-driven module architecture, JWT-based role access control, a seven-route RESTful backend, and fault-tolerant MongoDB connectivity. Experimental results confirmed a 100% route pass rate across all 16 module routes, a mean API response time of 90ms across all seven route groups, and an 88% reduction in new module development effort — from over 50 lines of code to just 6 lines — validating the scalability and extensibility of the proposed architecture.

Overall, UniMart demonstrates that the MERN stack is an effective foundation for building large-scale multi-module platforms, providing a strong base for future enhancement toward a fully production-ready digital marketplace serving India's growing SME ecosystem

VII. FUTURE ENHANCEMENT

Real-Time Notifications — Socket.io WebSockets will be integrated to enable live order status updates and buyer-seller messaging within the platform.

Mobile Application — A React Native mobile application will be developed to expose all existing backend APIs through a native interface for iOS and Android users.

AI Recommendation Engine — An AI-driven recommendation system will personalise the Home dashboard based on user browsing history and purchase patterns.

Multi-Language Support — Tamil, Hindi, and English language support will be added using the i18next library to improve accessibility for regional SME users.

Advanced Search and Analytics — MongoDB Atlas Search will enable full-text search across all 16 module catalogues, and an admin analytics dashboard will provide real-time visibility into platform performance and usage metrics.

REFERENCE

[1] Havivyan, N., Smith, J., and Brown, A. (2021). Multi-vendor e-commerce architecture using MERN stack. International Journal of Computer Applications, 183(12), 1-7.

[2] Subramanian, V. (2020). Pro MERN stack: Full stack web app development with Mongo, Express, React, and Node. Apress Publishing, 2nd edition, 1-385.

[3] Banks, A., and Meredith, E. (2023). Learning React: Modern patterns for developing React apps. O'Reilly Media, 3rd edition, 1-310.

[4] Chodorow, K. (2019). MongoDB: The definitive guide. O'Reilly Media, 3rd edition, 1-432.

[5] Tilkov, S., and Vinoski, S. (2020). Node.js: Building high-performance network programs. IEEE Internet Computing, 14(6), 80-83.

[6] Fielding, R. T., and Taylor, R. N. (2019). Principled design of the modern web architecture. ACM Transactions on Internet Technology, 2(2), 115-150.

[7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (2020). Design patterns: Elements of reusable object-oriented software. Addison-Wesley Professional, 1-395.

[8] Stallings, W. (2021). Network security essentials: Applications and standards. Pearson Education, 6th edition, 1-512.

[9] Wieruch, R. (2022). The road to React: Your journey to master React in JavaScript. Leanpub Publishing, 1-250.

[10] Rajput, S. (2021). E-commerce and its impact on market structure: A study of SMEs in India. Journal of Business and Management, 23(4), 45-58.

[11] Ohne, N., and Carter, P. (2022). JWT-based authentication in modern web applications: Security analysis and best practices. Journal of Information Security, 13(3), 112-128.

[12] Sharma, A., and Gupta, R. (2021). Role-based access control in multi-tenant web applications: A systematic review. International Journal of Web Engineering and Technology, 16(2), 134-156.

[13] Patel, K., and Mehta, D. (2022). MongoDB Atlas in cloud-native applications: Performance benchmarking and fault tolerance analysis. Journal of Cloud Computing, 11(1), 1-18.

[14] Johnson, M., and Williams, T. (2023). Registry-driven frontend architectures for scalable single page applications. IEEE Transactions on Software Engineering, 49(3), 890-905.

[15] Kumar, S., and Singh, P. (2022). Full-stack JavaScript development for SME digital transformation: A case study of MERN stack adoption in Indian enterprises. Journal of Enterprise Information Management, 35(6), 1456-147.