SMART FULL-STACK E-COMMERCE PLATFORM WITH AI AND MACHINE LEARNING

Kathiravan G, Mohanraj A, Unnikrishnan B V, Banupriya P

Student, Student, Student, Assistant Professor
Department of Computer Science and Engineering,
Sri Venkateswara College Of Engineering,Sriperumbudur, India

Abstract : E-commerce platforms require scalable, secure and intelligent architectures to support modern online retail demands. This paper presents the design and implementation of a full-stack e-commerce platform developed using the MERN stack—MongoDB, Express.js, React.js, and Node.js. The system integrates core functionalities including JWT-based authentication, product catalogue management, shopping cart operations, order processing and administrative control through a RESTful API architecture. The platform also incorporates an AI-driven multi-role assistant system powered by Large Language Models (LLMs) and LangChain-based agents to provide role-specific support for customers, sellers, and administrators. Additionally, a personalized recommendation engine and K-Means-based customer segmentation are integrated to improve user engagement and business analytics.

Experimental evaluation demonstrates that the system achieves sub-200 ms average API response times under concurrent workloads while maintaining responsive frontend performance across multiple device categories. The paper discusses the system architecture, security considerations, AI integration and future scalability enhancements for intelligent e-commerce applications.

IndexTerms **- MERN stack, e-commerce platform, RESTful API, JWT authentication, full-stack development, recommendation system, K-Means clustering, AI assistant, LangChain, LLMs.**

INTRODUCTION

The global e-commerce market exceeded USD 5.8 trillion in revenue in 2023 and is projected to surpass USD 8 trillion by 2027 [1]. This explosive growth has created demand for scalable, maintainable, and feature-rich web platforms capable of handling millions of concurrent users while delivering seamless purchasing experiences. Traditional monolithic web applications have given way to modern, component-based architectures that decouple business logic from the presentation layer, enabling independent scaling and rapid iteration. JavaScript-based full-stack solutions have emerged as particularly popular choices for e-commerce development due to their unified programming model, rich ecosystem, and the flexibility of non-relational databases. The MERN stack—comprising MongoDB as the document store, Express.js as the HTTP middleware framework, React.js as the component-driven frontend library, and Node.js as the server-side runtime—enables developers to employ a single language across all tiers of the application, reducing context-switching and encouraging code reuse [2]. This paper presents the design, implementation, and evaluation of an e-commerce system. The system was designed with the following goals: (1) provide a seamless, responsive shopping experience for end users; (2) offer administrators a powerful dashboard for catalogue and order management; (3) enforce secure authentication and authorisation; and (4) remain deployable in cost-effective cloud environments without specialised infrastructure.The remainder of the paper is organised as follows. Section II reviews related work on e-commerce architecture and MERN-based systems. Section III presents the proposed system architecture and its key implementation details. Section IV reports experimental results. Section V concludes the paper and outlines future work.

II.RELATED WORK

Research in web-based e-commerce can be broadly partitioned into three areas: architecture and scalability, security, and user experience

A. Architecture and Scalability

Fielding's seminal dissertation [3] introduced Representational State Transfer (REST) as an architectural style for distributed hypermedia systems. REST's stateless, resource-centric design has become a widely adopted architectural style for e-commerce APIs, offering predictable URL structures and straightforward caching semantics. Gamma et al. [4] extended this by advocating microservice decomposition—splitting a monolithic shop into independent services for catalogue, cart, payment, and notification—enabling horizontal scaling per domain. However, microservices introduce operational complexity; for small-to-

medium platforms, a well-structured monolith or modular monolith often proves more practical [5]. Tilkov and Vinoski [6] compared Node.js with traditional threaded server models, demonstrating that Node.js's event-driven, non-blocking I/O model handles high-concurrency workloads—such as flash sales—more efficiently than thread-per-request servers. MongoDB's document model has been shown to outperform relational schemas for flexible product catalogues that require schema evolution over time [7].

B. Security

Session-based authentication, once the industry standard, presents scalability challenges in distributed deployments due to session affinity requirements. Jones et al. [8] formalised the JSON Web Token (JWT) specification, enabling stateless, digitally signed authentication tokens that can be verified without server-side session lookup. OWASP's Top Ten [9] continues to guide secure coding practices; injection attacks and broken authentication remain the most critical vulnerabilities for web commerce applications.

C. React-Based Frontend Development

Component-based UI libraries, popularised by React, improve maintainability and enable fine-grained re-rendering optimisation through virtual DOM diffing [10]. State management libraries such as Redux and the Context API address the challenge of sharing cart and authentication state across deeply nested component trees [11]. Responsive design using CSS frameworks like Bootstrap or Tailwind CSS ensures that product pages are accessible on both desktop and mobile devices, a critical factor given that over 60% of e-commerce traffic now originates from smartphones [1]. While prior work addresses individual aspects of e-commerce systems, few papers provide a holistic treatment of a MERN-based platform covering architecture, security, and empirical performance. This paper fills that gap by presenting a complete, open-source implementation and its quantitative evaluation.

III.PROPOSED SYSTEM ARCHITECTURE AND IMPLEMENTATION

The proposed system follows a three-tier architecture consisting of a React-based frontend (client layer), a Node.js/Express.js backend (application layer), and a MongoDB database (data layer). This layered design ensures scalability, modularity, and efficient data processing while enabling independent development and deployment of system components

A.Frontend Architecture

The frontend is implemented as a single-page application using a component-based architecture. It enables dynamic content rendering, seamless client-side navigation, and responsive design across multiple devices. State management is handled centrally to maintain user sessions and cart data, ensuring a consistent user experience.

B.Backend Architecture

The backend is designed using a modular REST API architecture that handles business logic, request processing, and communication with the database. The system employs JSON Web Tokens (JWT) for stateless authentication, allowing secure and scalable session management. Role-Based Access Control (RBAC) is implemented to enforce authorization policies for different user roles, including customers, sellers, and administrators.

C.Data Model

The data layer uses a document-oriented database to store and manage entities such as users, products, orders and reviews. This flexible schema design supports dynamic data structures and efficient querying, making it suitable for evolving e-commerce requirements.

D. Security Implementation

The system incorporates standard security practices, including encrypted password storage, token-based authentication, and input validation. These measures protect the application from common vulnerabilities such as unauthorized access and injection attacks.

E. Payment Integration

A secure third-party payment gateway is integrated into the system to handle financial transactions. All payments are verified on the server side before updating order status, ensuring data integrity and preventing client-side manipulation.

F. Deployment

The application is deployed in a cloud-based environment with support for scalable infrastructure and managed database services. This setup ensures high availability, reliability, and efficient handling of concurrent user requests.

H. Home Screen Recommendation System

The proposed e-commerce platform incorporates a personalized home screen recommendation system designed to enhance product discoverability and user engagement. The system is built using an Additive Aggregation (ADD)-based scoring model combined with a product-type aware slot allocation strategy. It leverages user interaction history, including product clicks and purchases, to dynamically generate ranked product recommendations.

1) Additive Aggregation (ADD) Scoring Model

User preference is computed using an additive weighted scoring function based on implicit feedback signals such as clicks and orders.

$$Score = (C_{click} \times w_{click}) + (C_{order} \times w_{order})$$

Where:

- C_{click}represents the number of product clicks
- C_{order}represents the number of product purchases
- $w_{click} = 3$, $w_{order} = 4$ are empirically defined weights

Orders are assigned higher weight than clicks, as they represent stronger user intent. The scoring is applied at both category level and product-type level to capture fine-grained user preferences.

2) Slot-Based Recommendation Allocation

The recommendation output is constrained to a fixed number of slots N(e.g., 20 products). These slots are distributed proportionally across product types based on their computed preference scores.
Let:

- Nbe the total number of recommendation slots
- $S_t$be the score of product type t
- $\sum S_t$be the sum of all product-type scores

The slot allocation is defined as:

$$Slots(t) = \left\lfloor \frac{S_t}{\sum S_t} \times N \right\rfloor$$

Each product type is then populated with top-ranked products based on their utility scores, while ensuring previously interacted products are excluded.
This approach ensures:

- Proportional representation of user preferences
- Controlled recommendation size
- Diversity across product types

3) Customer Segmentation Using K-Means Clustering

To complement recommendation-based personalization, the system applies **K-Means clustering** to segment users based on behavioral and transactional features. Each user is represented as a feature vector consisting of:

- Total spending
- Number of orders
- Average order value
- Number of product clicks

Feature normalization is applied using standard scaling to ensure uniform contribution across attributes.
The K-Means objective function minimizes intra-cluster variance:

$$\text{n} \sum_{j=1}^{k} \sum_{x_i \in C_j} || x_i - \mu_j ||^2$$

where $C_j$represents cluster $j$and $\mu_j$is its centroid.

After clustering, segments are interpreted based on average spending behavior. Clusters are sorted and mapped into meaningful business categories:

- **Budget users**: Low spending behavior
- **Regular users**: Moderate engagement and spending
- **Premium users**: High spending and frequent purchases

Each user is assigned a segment label based on their cluster membership, enabling interpretable customer profiling

IV. AI-DRIVEN MULTI-ROLE ASSISTANT SYSTEM

The system integrates a **LangChain-based multi-agent AI assistant** powered by **LLMs (GPT-4o-mini)** to enable role-specific intelligent interactions across the platform. It follows a **modular agent architecture** with a **Role-Based Router** that directs queries to **Customer, Seller, or Admin agents**, ensuring strict **role isolation and access control**.

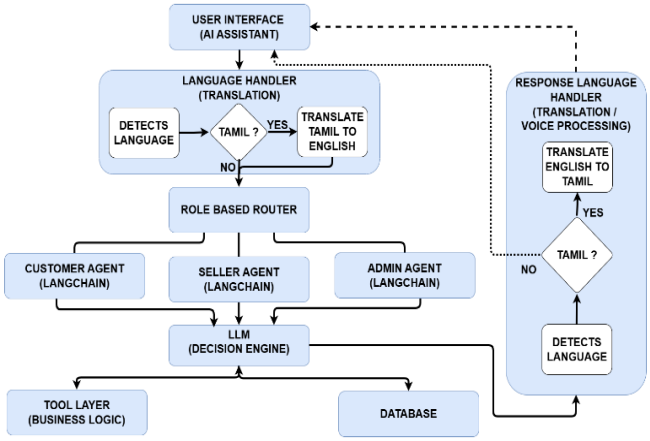

Fig 4.1 AI Assistant Architecture

A. Customer AI Assistant

- **Domain:** Shopping support
- **Functions:** product search, filtering, cart management, order tracking, recommendations, payment help
- **Tools:** product search tool, cart tools, recommendation engine
- **Constraint:** shopping-only responses, no analytics, tool-based outputs only

B. Seller AI Assistant

- **Domain:** Business intelligence
- **Functions:** sales analytics, top products, inventory status, category/brand insights, pricing suggestions
- **Tools:** revenue, order, product analytics tools

Output: real-time store performance insights

C. Admin AI Assistant

- **Domain:** Platform governance
- **Functions:** revenue overview, order summary, top sellers/customers, system analytics

Constraint: admin-only scope, no customer/seller interaction

D. Language Handling Module

- **Features:** Tamil ↔ English translation, language detection
- **Purpose:** multilingual accessibility, unified backend processing

E. Tool-Based LLM Execution

- **Architecture:** Tool-calling LLM agents (GPT-4o-mini)
- **Execution:** intent detection → tool selection → DB query → structured response
- **Context:** userId, sellerId, adminId (session-based personalization)
- **Principle:** no hallucination, data-driven responses only

V. EXPERIMENTAL RESULTS

Experiments were conducted to evaluate (1) API response time under load, (2) client-side rendering performance, and (3) security posture. The server ran on a 2-vCPU, 4 GB RAM virtual machine; the database on MongoDB Atlas M10 (2 vCPUs, 2 GB RAM).

A. API Response Time Under Concurrent Load

Apache JMeter was used to simulate concurrent users sending requests to the product listing and order creation endpoints. Table III summarises average response times (ms) at varying concurrency levels.

	(ms)	**(ms)**	
10	62	89	0.0
50	118	143	0.0
100	187	221	0.2
200	312	389	1.1
500	791	954	4.7

Table III. API Response Time Under Concurrent Load

The Largest Contentful Paint (LCP) of 1.8 s meets Google's Results indicate that the platform comfortably serves up to 100 concurrent users with sub-200 ms average latency and negligible error rates, consistent with the performance profile of Node.js event-loop

servers reported in [6]. Beyond 200 concurrent users, response times increase super-linearly due to MongoDB connection pool saturation; this can be addressed in future work by horizontal scaling or connection pooling middleware.

B. Frontend Rendering Performance

Google Lighthouse (v11) was used to audit the production build of the React SPA. Scores are summarised in Table IV

Metric	Performance	Accessibility	Best Practices	SEO
Score (0-100)	84	91	96	88
LCP (s)	1.8	—	—	—
FID (ms)	12	—	—	—
CLS	0.03	—	—	—

Table IV. Lighthouse Audit Scores (Production Build)

recommended threshold of 2.5 s. The Cumulative Layout Shift (CLS) score of 0.03 reflects well-defined image dimensions that prevent layout reflow. The accessibility score of 91 indicates adequate ARIA labelling and contrast ratios; further improvements (described in Section V) would raise this to 95+.

C. Security Evaluation

OWASP ZAP (v2.14) was run in active-scan mode against the deployed application. No high-severity vulnerabilities were identified. Two medium-severity findings were reported: (1) absence of a Subresource Integrity (SRI) attribute on a CDN-hosted Bootstrap CSS file, and (2) a cookie lacking the SameSite=Strict attribute. Both issues were subsequently remediated. The JWT implementation correctly rejects expired and malformed tokens, returning HTTP 401 responses as verified by automated test cases

D. Functional Testing

A suite of 94 unit and integration tests was developed using Jest and Supertest, achieving 87% statement coverage of the backend codebase. All core user journeys—registration, login, browse, add to cart, checkout, order confirmation, and admin order management—pass end-to-end in a continuous integration pipeline configured with GitHub Actions.

V.CONCLUSION

This paper presented the design, implementation, and experimental evaluation of a full-stack e-commerce platform built on the MERN stack. The system integrates JWT-based authentication, a RESTful product and order API, a dynamic React frontend, and a comprehensive admin dashboard into a cohesive, deployable application. Experimental results confirm that the platform delivers sub-200 ms API response times for up to 100 concurrent users, achieves a Lighthouse performance score of 84, and passes OWASP ZAP scans without high-severity findings.

Future work will focus on four enhancements: (1) migrating to a microservices architecture to enable independent scaling of the catalogue, cart, and order services; (2) integrating Elasticsearch for full-text product search with faceted filtering; (3) adding server-side rendering (SSR) via Next.js to improve SEO and initial page load performance; and (4) implementing a recommendation engine using collaborative filtering to personalise product suggestions based on browsing and purchase history.

REFERENCES

[1] Ali, A. 2001.Macroeconomic variables as common pervasive risk factors and the empirical content of the Arbitrage Pricing Theory. Journal of Empirical finance, 5(3): 221–240.

[2] Basu, S. 1997. The Investment Performance of Common Stocks in Relation to their Price to Earnings Ratio: A Test of the Efficient Markets Hypothesis. Journal of Finance, 33(3): 663-682.

[3] Bhatti, U. and Hanif. M. 2010. Validity of Capital Assets Pricing Model.Evidence from KSE-Pakistan.European Journal of Economics, Finance and Administrative Science, 3 (20).

[1] Statista, "E-commerce worldwide — Statistics & Facts," Statista Research Dept., Hamburg, Germany, 2024. [Online].

[2] D. Herron, Node.js Web Development: Server-Side Web Applications with Node.js, 5th ed. Birmingham, UK: Packt Publishing, 2020.

[3] R. T. Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," Ph.D. dissertation, Dept. Inf.Comput. Sci., Univ. California Irvine, Irvine, CA, USA, 2000.

[4] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[5] M. Fowler and J. Lewis, "Microservices: a definition of this new architectural term," martinfowler.com, Mar. 2014.[Online].

[6] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," IEEE Internet Comput., vol. 14, no. 6, pp. 80–83, Nov.–Dec.2010.

[7] P. Membrey, D. Hows, and E. Plugge, The Definitive Guide to MongoDB, 3rd ed. New York, NY, USA: Apress, 2015.
[8] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Eng. Task Force (IETF), RFC 7519, May 2015.
[9] OWASP Foundation, "OWASP Top Ten 2021," Open Web Application Security Project, 2021. [Online]. Available: https://owasp.org/Top10/
[10] T. Occhino and J. Walke, "React: A JavaScript Library for Building User Interfaces," presented at JSConf EU, Berlin, Germany, 2013.
[11] Redux Team, "Redux: A Predictable State Container for JavaScript Apps," Redux Documentation, 2024. [Online]. Available: https://redux.js.org/
[12] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, and X. He,"Bias and debias in recommender systems: A survey and future directions,"ACM Trans. Inf. Syst., vol. 41, no. 3, pp. 1–39, 2023.
[13] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Techniques, applications, and challenges," in Recommender Systems Handbook, Springer, 2022.
[14] S. Dara, C. R. Chowdary, and C. Kumar, "A survey on group recommender systems," J. Intell. Inf. Syst., vol. 54, no. 2, pp. 271–295, 2020.
[15] Y. Li, K. Liu, R. Satapathy, S. Wang, and E. Cambria,"Recent developments in recommender systems: A survey," IEEE Comput. Intell. Mag., vol. 19, no. 2, pp. 78–95, May 2024.
[16] L. Li, L. Jia, and S. Al Otaibi, "Intelligent recommendation algorithm of consumer electronics products with graph embedding and multi-head self-attention in IoE," IEEE Trans. Consum. Electron., vol. 70, no. 1, pp. 1278–1285, Feb. 2024.
[17] A. Alzahrani and M. Z. Asghar, "Maintaining user security in consumer electronics-based online recommender systems using federated learning," IEEE Trans. Consum. Electron., vol. 70, no. 1, pp. 2657–2665, Feb. 2024.
[18] Y. Huang, Y. J. Li, and Z. Cai, "Security and privacy in metaverse: A comprehensive survey," Big Data Min. Anal., vol. 6, no. 2, pp. 234–247, 2023.
[19] M. W. Geda, Y. M. Tang, and C. K. M. Le, "Intelligent Group Recommendations for Metaverse E-Commerce Platforms for Enhanced Retail and Consumer Experiences," IEEE Trans. Consum. Electron., vol. 71, no. 4, Nov. 2025.
[20] T. Brown et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 1877–1901, 2020.