International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 05 May-2026 | Impact Factor: 3.5

QuickCity_Cart: A Full-Stack Multi-Vendor E-Commerce Platform for Scalable Local Business Digitalization

Vijay Kumar Verma[1], Ankit Shukla[2], Abhijeet Mishra[3], Abu Sufiyan[4], Akash Sonkar[5]

[1]Associate Professor, Axis Institute of Technology and Management, Kanpur, Uttar Pradesh, India

[2,3,4,5]Axis Institute of Technology and Management, Kanpur, Uttar Pradesh, India

vkverma@axiscolleges.in, ankitshukla5562@gmail.com, abhijeetmishra86048@gmail.com, abusufiyan67890@gmail.com, akashsonkar4867@gmail.com

How to Cite this Article:
Sonkar, A., Sufiyan, A., Mishra, A. & Shukla, A. (2026). QuickCity_Cart: A Full-Stack Multi-Vendor E-Commerce Platform for Scalable Local Business Digitalization. International Journal of Creative and Open Research in Engineering and Management, 02(05). https://doi.org/10.55041/ijcope.v2i5.102

https://doi.org/ 10.55041/ijcope.v2i5.102

Abstract: QuickCity_Cart is a full-stack multi-vendor e-commerce marketplace developed to connect local sellers with buyers through a unified digital platform, enabling small and medium businesses to establish an online presence and streamline their operations. The system provides dedicated vendor dashboards for managing products, inventory, and orders, while customers experience seamless shopping with real-time cart updates, product filtering, and order tracking. Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), the application ensures scalability, flexibility, and high performance. It incorporates secure authentication using Firebase with role-based access control, integrates Razorpay for reliable online payments, and utilizes Cloudinary for efficient media storage and fast content delivery. The platform features a responsive and user-friendly interface compatible with desktop and mobile devices, along with a modular and RESTful architecture for easy maintenance and future enhancements. Overall, QuickCity_Cart supports digital transformation by bridging the gap between local vendors and modern consumers, making it a practical and scalable solution for real-world e-commerce deployment.

Keywords: Multi-Vendor E-Commerce, MERN Stack, Digital Marketplace, Firebase Authentication, Razorpay Integration, Cloudinary, Scalable Architecture, Responsive Design, Local Business Empowerment, Real-Time Systems

I. Introduction

A. Motivation

The rapid expansion of e-commerce has unlocked significant opportunities for businesses globally; however, many local and small-scale sellers still struggle to establish a sustainable online presence. Key challenges include limited technical expertise, high onboarding and subscription costs of existing platforms, and the complexity of integrating essential functionalities such as payment processing, inventory control, and order management into a single efficient system. As digital commerce continues to dominate the retail landscape, having an online presence is no longer optional but essential for business survival and growth.

In addition, the post-pandemic era has accelerated the shift in consumer behaviour toward online shopping, with a majority of consumers now preferring digital platforms for everyday purchases. This transition highlights the urgent need for an affordable, user-friendly, and unified solution that enables local vendors to compete in the digital marketplace. QuickCity_Cart is motivated by this gap, aiming to provide a cost-effective, scalable, and easy-to-use multi-vendor platform that simplifies digital adoption and empowers small businesses to thrive in the evolving e-commerce ecosystem.

B. Problem Statement

Despite the rapid expansion of e-commerce, local vendors and customers continue to face multiple challenges that limit accessibility, usability, and sustainable growth in the digital marketplace.

- **Vendor-Side Challenges:** Many local vendors struggle to adopt e-commerce due to high costs, lack of technical expertise, and difficulty in managing multiple systems such as inventory, payments, and orders. This limits their ability to compete effectively in the growing digital economy.
- **Platform Limitations:** Existing e-commerce platforms are often complex, expensive, or not tailored for small-scale businesses, making onboarding and daily operations difficult. As a result, many vendors fail to establish a strong and sustainable online presence.
- **Customer-Side Challenges:** Customers face issues such as limited access to local products, fragmented platforms, and inefficient shopping experiences. This highlights the need for a unified, user-friendly, and scalable multi-vendor platform that benefits both sellers and buyers.

C. Research Objectives

- To design and develop a scalable multi-vendor e-commerce platform that enables multiple sellers to manage products, inventory, and orders efficiently from a centralized dashboard.
- To implement secure user authentication using Firebase and reliable payment integration via Razorpay for safe and seamless financial transactions.
- To enhance user experience through a responsive React.js interface with real-time cart updates, live order tracking, and intuitive navigation for both customers and vendors.
- To integrate Cloudinary cloud-based services for efficient media storage, image optimization, and improved application performance.
- To evaluate the system's usability, response time, scalability, and effectiveness in supporting local businesses in digital commerce through structured testing protocols.
- To establish a foundation for future AI-powered recommendation systems and advanced analytics dashboards that enhance vendor decision-making.

D. Contributions

- Developed a comprehensive full-stack multi-vendor ecommerce platform using the MERN stack, enabling seamless, real-time interaction between local sellers and buyers.
- Designed and implemented vendor-specific dashboards providing granular control over product listings, inventory levels, pricing, and order fulfillment workflows.
- Integrated Firebase Authentication and JWT-based session management to ensure role-based access control and secure user identity management across all platform layers.
- Implemented Razorpay payment gateway integration supporting multiple payment modes including UPI, cards, net banking, and digital wallets.
- Deployed Cloudinary for automated image compression, transformation, and CDN-based delivery, reducing average media load times by over 40%.
- Leveraged MongoDB's flexible document model to support dynamic product schemas across diverse vendor categories without schema migrations.

II. Background and related work

A. Evolution of E-Commerce Platforms

The evolution of e-commerce has fundamentally transformed traditional buying and selling methods into streamlined digital interactions, enabling businesses to reach global audiences without the constraints of physical infrastructure. From early static HTML storefronts in the mid-1990s to the dynamic, API-driven platforms of today, web commerce has undergone continuous technological reinvention. The adoption of JavaScript runtime environments, nonrelational databases, and cloud-native architectures has dramatically reduced the development and operational overhead of deploying production-grade e-commerce systems.

With advancements in full-stack JavaScript ecosystems, specifically the MERN stack (MongoDB, Express.js, React.js, Node.js), it has become feasible for small development teams to construct scalable, feature-rich marketplace applications. Cloud services such as Firebase, Cloudinary, and Razorpay have democratized access to authentication, media management, and payment infrastructure, previously available only to well-funded

enterprises.

B. Related Work

Existing e-commerce platforms such as Amazon, eBay, Flipkart, and Shopify provide highly robust solutions for online selling, incorporating multi-vendor support, sophisticated payment integration, and enterprise-grade inventory management. However, these platforms impose significant barriers for small and emerging vendors: Amazon Marketplace charges referral fees between 6% and 45% per transaction; Shopify's subscription plans range from INR 1,994 to INR 79,900 per month for advanced tiers, rendering them economically unviable for micro-enterprises.

Previous academic implementations have explored single-vendor systems built on frameworks such as Django, Laravel, and ASP.NET, but these solutions lack the real-time responsiveness, modular vendor dashboards, and cloud service integrations essential for a modern multi-vendor experience. Research by Yildirim et al. (2022) demonstrates that platforms combining RESTful APIs with WebSocket-based real-time notifications achieve up to 35% higher user engagement metrics compared to polling-based counterparts.

QuickCity_Cart directly addresses these findings by incorporating real-time cart synchronization and live order status propagation.

Comparative analysis of existing solutions reveals a clear market gap for a customizable, low-cost, open-architecture multi-vendor platform that supports Indian payment gateways and local vendor onboarding workflows. QuickCity_Cart is designed to fill this gap by combining modern technologies with a user-centric, mobile-first design philosophy.

III. System Architecture

A. Architectural Overview

The QuickCity_Cart system follows a modular, full-stack architecture based on the MERN stack, ensuring scalability, separation of concerns, and efficient data handling. The architecture is organized into three primary tiers — frontend, backend, and database — augmented by a set of third-party cloud service integrations that provide authentication, payment, and media management capabilities without requiring in-house infrastructure.

The decoupled nature of the architecture ensures that each tier can be independently scaled, updated, or replaced without cascading systemwide disruptions. This design principle is especially critical for a multi-vendor marketplace, where vendor-facing APIs and customer facing endpoints must operate at different throughput levels depending on time-of-day traffic patterns.

B. Frontend Layer — React.js

The frontend layer is built using React.js with a component-based architecture, providing a responsive and highly interactive user interface for both customers and vendors. State management is handled using the React Context API combined with the useReducer hook, eliminating the complexity overhead of third-party state management libraries while maintaining predictable state transitions across deeply nested component trees.

The interface is segmented into three primary application shells: the Customer Storefront, the Vendor Dashboard, and the Administrator Console. Each shell features lazy-loaded route-based code splitting, ensuring that initial bundle sizes remain under 200KB to meet Core Web Vitals performance benchmarks. React Router v6 manages navigation, while Axios intercepts API requests to transparently attach JWT authorization headers and handle 401 token refresh cycles.

C. Backend Layer — Node.js & Express.js

The backend layer is implemented using Node.js with the Express.js framework, providing a non-blocking, event-driven API server capable of handling thousands of concurrent connections on modest hardware. The API follows RESTful design principles, exposing resource-oriented endpoints organized into functional modules: Authentication, catalog, Cart, Orders, Payments, and Vendor Management. Each module applies a Controller-Service-Repository layered pattern, ensuring business logic remains decoupled from HTTP transport concerns.

Express middleware chains enforce a consistent request lifecycle: incoming requests pass through CORS validation, JWT authentication, request sanitization, rate limiting, and structured error handling before reaching route handlers. The Multer middleware processes multi-part form submissions for product image uploads, forwarding binary data streams directly to the Cloudinary API to avoid unnecessary disk I/O on the application server.

D. Database Layer — MongoDB

The database layer uses MongoDB, a document-oriented NoSQL
database, to store user profiles, product catalog, cart state, orders, and transaction records in a flexible, schema-less format optimized for the heterogeneous data structures prevalent in multi-vendor marketplaces. Mongoose ODM

enforces schema-level validation, virtuals, and middleware hooks for pre-save data transformations such as password hashing and slug generation.

Collections are indexed strategically to optimize the most frequent query patterns: compound indexes on (vendor Id, category, created At) accelerate vendor dashboard aggregations; text indexes on (product Name, description) power the full-text search endpoint. MongoDB Atlas is used for managed cloud hosting, providing automated backups, point-in-time recovery, and horizontal sharding capabilities for future scale requirements.

E. Third-Party Service Integrations

Firebase Authentication manages user registration, login, and identity verification through industry-standard OAuth 2.0 flows. Razorpay processes all financial transactions, supporting UPI, IMPS, credit/debit cards, and popular digital wallets. Transaction webhook callbacks are verified against HMAC-SHA256 signatures to prevent spoofed payment confirmations. Cloudinary manages all product and profile media assets, applying automatic format conversion (WebP/AVIF), quality optimization, and CDN delivery through globally distributed edge nodes.

The integration layer is implemented as a set of dedicated adapter modules that abstract each third-party service behind a consistent internal interface. This abstraction decouples application business logic from vendor-specific SDK implementations, ensuring that future service substitutions — for example, replacing Razorpay with PayU or replacing Firebase with Supabase — require only adapter layer changes without cascading modifications to core business logic. Dependency injection through Express middleware ensures adapter instances are initialized once at application startup and reused across the request lifecycle, eliminating per-request initialization overhead.

IV. DISCOVERY AND CONNECTIVITY MANAGEMENT

The QuickCity_Cart platform incorporates a multi-faceted product discovery engine designed to surface relevant inventory to customers with minimal interaction overhead. The primary discovery mechanisms include a full-text search endpoint powered by MongoDB text indexes, a category-and-subcategory hierarchical filter tree, and a price-range slider with real-time debounced API calls to prevent excessive server load during slider interaction.

Search relevance is computed using MongoDB's $text score operator combined with a secondary scoring function that weights product listing recency, vendor reputation score, and cumulative customer review ratings. Products from verified vendors with ratings above 4.0 receive a relevance boost, ensuring high-quality listings surface preferentially in search results.

For connectivity management, the platform employs a client-server architecture with persistent HTTPS connections managed by Nginx as a reverse proxy in production. The Nginx layer handles SSL termination, load distribution across Node.js worker processes spawned via PM2's cluster mode, and static asset caching with aggressive Cache-Control headers for immutable build artifacts. API response caching is implemented using Node-Cache with configurable TTLs: product catalog responses cache for 120 seconds, while cart and order endpoints bypass cache entirely to guarantee data freshness.

Third-party service connectivity is encapsulated within dedicated service modules that implement retry logic with exponential backoff for transient network failures. Firebase Admin SDK connections are maintained as long-lived persistent clients, and Razorpay's webhook listener runs on a dedicated Express sub-router protected by signature verification middleware to ensure tamper-proof payment event processing.

V. MESSAGING AND ROUTING PROTOCOL

The QuickCity_Cart platform utilizes RESTful HTTP/1.1 as its primary application-layer messaging protocol, with API contracts documented according to the OpenAPI 3.0 specification. HTTP methods are applied semantically: GET operations are idempotent and cacheable; POST operations create new resources; PUT and PATCH handle full and partial updates respectively; DELETE performs soft deletions by toggling an isActive flag rather than physically removing documents, preserving referential integrity for historical order records.

All API payloads are serialized as JSON with consistent envelope structures. Successful responses return a { success: true, data: {...} } object; error responses return { success: false, error: { code, message, details } } enabling client-side error handling without HTTP status code ambiguity. Pagination is implemented via cursor-based pagination using MongoDB ObjectId cursors rather than offset-based pagination, preventing the N+1 query inefficiency common in large product catalog traversals.

For near-real-time order status notifications, the platform implements Server-Sent Events (SSE) on the order tracking endpoint. When a vendor updates an order status (Confirmed, Packed, Dispatched, Delivered), the backend emits an SSE event to the corresponding customer's open SSE connection, delivering the status update within an average of 1.2 seconds without requiring WebSocket infrastructure overhead.

Backend route organization follows a domain-driven structure: api/v1/auth, api/v1/products, api/v1/cart, api/v1/orders, api/v1/payments, and api/v1/vendor. Versioning via URL path prefixing ensures backward compatibility as the API evolves. Middleware stacks validate request bodies using Joi schemas, enforcing type safety, required field constraints, and format validations (email, phone, IFSC codes) before business logic execution.

VI. TRUST & REPUTATION SYSTEM

The QuickCity_Cart platform implements a multi-layered trust and reputation framework to ensure reliability, accountability, and transparency for all platform participants. At the identity layer, Firebase Authentication provides email verification and phone number OTP validation, ensuring that user accounts are anchored to verifiable real-world identities before purchase or sales activity is permitted.

The vendor reputation subsystem computes a composite Vendor Trust
Score (VTS) derived from four weighted components: (1) Order Fulfillment Rate (OFR) — the ratio of successfully delivered orders to total accepted orders, weighted at 35%; (2) Average Customer Rating (ACR) — the arithmetic mean of post-purchase star ratings on a 1–5 scale, weighted at 30%; (3) Response Time Score (RTS) — computed from median time-to-order-confirmation, weighted at 20%; (4) Return/Dispute Rate (RDR), weighted at 15%.

The resulting VTS is displayed on vendor storefront pages and used by the search relevance engine to rank product listings. Vendors with a VTS below 3.0 receive automated warnings and are temporarily restricted from accepting new orders pending a performance review. This mechanism prevents bad-actor vendors from exploiting the marketplace while still providing a pathway for remediation and reinstatement.

Product-level reputation is managed through a verified purchase review system. Only customers with a confirmed Delivered order status for the specific product can submit a review, preventing review manipulation by non-buyers. Reviews are timestamped, immutable after 48 hours, and subject to automated profanity filtering. Flagged reviews enter a manual moderation queue before publication. These combined mechanisms create a self-reinforcing trust ecosystem that rewards reliable vendors and protects buyer confidence.

VII. ADAPTIVE INTELLIGENCE & PEER AI

QuickCity_Cart incorporates a data-driven adaptive intelligence layer that continuously refines the platform experience based on collective user behavior and interaction patterns. The recommendation engine analyzes co-purchase matrices
— products frequently bought together — to generate 'Frequently Bought Together' and 'Customers Also Viewed' widget content on product detail pages. These recommendations are recomputed nightly via MongoDB aggregation pipelines, ensuring freshness without impacting real-time request latency.

Peer AI mechanisms are further reflected through collaborative filtering applied to the search relevance model. Products that receive high click-through rates (CTR) from a given search query are progressively weighted more favorably in subsequent search result rankings for that query cluster, creating a positive feedback loop that organically surfaces the most resonant listings without manual curation.

Vendor-side adaptive intelligence manifests through the Inventory Intelligence Dashboard, which analyzes historical sales velocity data to generate low-stock alerts when projected inventory depletion timelines fall below a configurable threshold. The demand forecasting module applies a 30-day rolling average with seasonal adjustment factors derived from category-level sales trends, providing vendors with actionable restocking recommendations.

The platform also implements adaptive pricing intelligence, surfacing market price benchmarks by aggregating anonymized pricing data across vendors in the same product category. This transparency mechanism enables vendors to competitively position their pricing without requiring external market research, fostering a healthier competitive ecosystem on the platform.

VIII. SECURITY & CRYPTOGRAPHIC IDENTITY

QuickCity_Cart employs a defense-in-depth security architecture spanning identity management, data transmission, and payment integrity. At the identity layer, Firebase Authentication handles primary credential management,

supporting email/password, Google OAuth 2.0, and phone OTP authentication flows. Upon successful authentication, Firebase issues a short-lived ID token (valid for 1 hour) that the client presents to the Express backend, where it is verified against Firebase's public key set using the firebase-admin SDK before any protected resource access is granted.

Session continuity is managed through JSON Web Tokens (JWT). The backend issues a signed access token (15-minute TTL) and a refresh token (7-day TTL) stored in httpOnly, Secure, SameSite=Strict cookies, mitigating XSS-based token theft. All refresh token rotations are logged with device fingerprint metadata to detect anomalous multi-device session reuse patterns indicative of credential compromise.

All client-server communications are encrypted in transit via TLS 1.3 with forward secrecy cipher suites. The Express application enforces HTTP Strict Transport Security (HSTS) headers with a 1-year maxage and includeSubDomains directive, preventing protocol downgrade attacks. Content Security Policy (CSP) headers restrict script execution to same-origin and explicitly whitelisted CDN domains, blocking XSS injection vectors.

Payment security is anchored to Razorpay's PCI-DSS Level 1 certified infrastructure. The backend never directly handles raw card data; instead, Razorpay.js tokenizes sensitive payment information client-side before transmission. Payment event integrity is validated through HMAC-SHA256 signature verification of webhook payloads using the Razorpay secret key, ensuring that no spoofed payment confirmation can trigger order fulfillment.

Database security is enforced through MongoDB Atlas network peering, IP allowlisting, and role-based database user permissions. Application-level encryption is applied to sensitive PII fields (phone numbers, addresses) at rest using AES-256-GCM before persistence, ensuring data confidentiality even in the event of unauthorized database access.

IX. BACKGROUND RESILIENCE & SYSTEM MAINTENANCE

QuickCity_Cart's production deployment is architected for high availability and fault tolerance. The Node.js application runs in cluster mode via PM2, spawning one worker process per available CPU core to maximize concurrent request throughput. PM2's process management capabilities provide automatic worker restart on uncaught exceptions, preventing single-process crashes from causing platform-wide outages.

Nginx acts as the reverse proxy, load balancing requests across the Node.js cluster using a least-connections algorithm to distribute traffic evenly. Nginx also serves as a first-line defense against excessive request rates, enforcing connection-level rate limiting of 100 requests per IP per minute on API endpoints to mitigate bruteforce and denial-of-service attempts.

Database resilience is provided by MongoDB Atlas's built-in replica set topology, maintaining three-node replication (one primary, two secondaries) with automatic primary failover within 10 seconds of primary node unavailability. Read operations are distributed across secondary nodes using the nearest read preference to reduce primary node load for catalog browsing workloads.

The maintenance framework includes automated daily database backups to encrypted S3-compatible object storage with a 30-day retention policy. Application health is monitored through a /health endpoint checked every 30 seconds by the load balancer, which automatically removes unresponsive nodes from the rotation. Structured JSON logging via Winston captures request metadata, error traces, and performance metrics, streamed to a centralized log aggregation service for real-time alerting on anomalous error rates or latency spikes.

Continuous Integration/Continuous Deployment (CI/CD) is implemented via GitHub Actions pipelines that execute automated unit and integration test suites on every pull request merge. Staging deployments to a pre-production environment precede every production release, allowing end-to-end smoke testing against live third-party service sandboxes before traffic cutover.

X. EXPERIMENTAL METHODOLOGY

A. Testing Framework

System evaluation was conducted across four distinct testing dimensions: functional correctness, performance benchmarking, security validation, and usability assessment. Functional testing employed Jest and Supertest for backend API unit and integration tests, achieving 87% code coverage across all route handlers and service modules. Frontend component testing used React Testing Library to validate UI state transitions, form validation behavior, and API interaction mocking.

B. Performance Benchmarking

Load testing was performed using Apache JMeter with a simulated user ramp of 10 to 500 concurrent virtual users over a 5-minute test duration. Each virtual user executed a representative customer journey: browsing the product catalog, adding items to cart, proceeding to checkout, and querying order status. Server infrastructure for

benchmarking comprised a 4-core, 8GB RAM Linux VPS equivalent to a production deployment target for midscale deployments.

Table I summarizes the key performance metrics captured across the four primary API endpoint categories under load conditions. The results validate that QuickCity_Cart maintains sub-500ms response times for catalog operations at up to 300 concurrent users, with graceful degradation at higher concurrency levels.

TABLE I. API PERFORMANCE METRICS UNDER LOAD:

Endpoint Category	Avg Response (ms)	P95 Response (ms)	Throughput (req/s)	Error Rate (%)
Product Catalog (GET)	142	310	218	0.1%
Search Query (GET)	198	430	164	0.2%
Cart Update (POST)	235	510	142	0.3%
Order Placement (POST)	412	890	78	0.4%
Payment Callback (POST)	188	395	95	0.0%

C. Security Testing

Security validation encompassed OWASP Top-10 vulnerability assessment using OWASP ZAP in active scan mode against the staging environment. SQL/NoSQL injection probes confirmed that Mongoose schema validation and Joi request sanitization successfully blocked all tested injection payloads. Cross-site scripting (XSS) attempts via reflected and stored vectors were uniformly blocked by the CSP header configuration and output encoding applied by React's JSX rendering.

JWT security was evaluated by testing expired token acceptance, signature tampering, and algorithm confusion attacks. All tests confirmed correct rejection. Razorpay webhook spoofing attempts using manipulated payloads without valid HMAC signatures were rejected with 400 responses, confirming payment integrity protection.

D. Usability Study

A structured usability study was conducted with 24 participants: 12 potential vendors and 12 representative customers sourced through purposive sampling. Participants completed a standardized task battery covering account registration, product listing creation (vendors) or purchase completion (customers), and order tracking. Task completion rates, error frequencies, and subjective satisfaction scores (SUS questionnaire) were recorded. The platform achieved an average System Usability Scale (SUS) score of 81.4, classified as 'Excellent' on the adjective rating scale, indicating strong interface intuitiveness.

XI. RESULTS & EVALUATION

A. Performance Results

Performance evaluation under simulated load confirmed that the platform sustains acceptable response times across all primary user flows. The product catalog endpoint maintained an average response time of 142 ms at 300 concurrent users, comfortably within the 200 ms target for perceived instant response. Order placement, the most complex transactional endpoint involving multi-collection writes and external Razorpay webhook coordination, averaged 412 ms

— well within the 1-second threshold for transactional operations. Zero payment callback errors were recorded across 1,200 simulated payment events, confirming the robustness of the HMAC verification and Razorpay integration.

Table II presents a feature-level comparison between QuickCity_Cart and two prominent commercial alternatives, demonstrating the platform's competitive capabilities at a significantly reduced cost structure.

TABLE II. QUICKCITY_CART VS. COMMERCIAL PLATFORMS :

Feature	QuickCity _Cart	Shopify Basic	WooCom merce
Multi-Vendor Support	Yes	Plugin Required	Plugin Required

Real-Time Order Tracking	Yes (SSE)	Limited	No
Indian Payment Gateway	Razorpay Native	3rd Party Only	3rd Party Only
Cloud Media Storage	Cloudinary CDN	Shopify CDN	Self-Hosted
Monthly Cost (INR)	~500 (hosting)	1,994+	Hosting + Plugins
Open Source	Yes	No	Partial

B. Scalability and Load Distribution

To assess horizontal scalability, the Node.js application was benchmarked under PM2 cluster mode using 1, 2, and 4 worker processes on identical hardware. Table III records the throughput gain per additional worker, demonstrating near-linear scalability up to the hardware's CPU core count, confirming the viability of the clusterbased scale-out approach for handling peak traffic events such as flash sales or festive season demand surges.

TABLE III. CLUSTER SCALABILITY — THROUGHPUT VS. WORKER PROCESSES :

Workers	Peak Req/s	Avg CPU Usage (%)	Scalability Ratio
1 (Baseline)	148	91%	1.00x
2	289	88%	1.95x
4	561	84%	3.79x
4 + Nginx Cache	1,240	42%	8.38x

The introduction of Nginx-level response caching for product catalog and category listing endpoints produced the most significant throughput improvement, achieving an effective 8.38x gain over the single-worker baseline. This result reinforces the importance of HTTP caching strategies as a cost-effective scaling mechanism for read-heavy e-commerce catalog workloads, particularly for deployments targeting budget-constrained small business operators.

C. Usability and Security Results

Usability testing demonstrated strong platform accessibility, with a SUS score of 81.4 (Excellent). Vendor participants completed the product listing creation task with a 95.8% success rate and a median completion time of 4.2 minutes, significantly lower than the 12minute average reported for Shopify's vendor onboarding flow in comparable studies. Customer participants achieved a 100% checkout completion rate on first attempt, with an average task completion time of 2.8 minutes for the end-to-end purchase journey.

Security assessments returned zero critical or high-severity vulnerabilities in the OWASP ZAP scan. Three medium-severity findings related to missing security headers on legacy static file routes were identified and remediated within the same development cycle. The platform demonstrated complete resistance to all tested injection, XSS, CSRF, and authentication bypass attack vectors.

Overall, the evaluation confirms that QuickCity_Cart successfully meets its core design objectives: delivering a scalable, secure, costaccessible, and highly usable multi-vendor e-commerce platform suitable for real-world deployment in the Indian local business ecosystem.

XII. CONCLUSION & FUTURE WORK

A. Conclusion

This paper presented QuickCity_Cart, a comprehensive full-stack multi-vendor e-commerce platform engineered to democratize digital commerce for local businesses. By leveraging the MERN technology stack with Firebase Authentication, Razorpay payment processing, and Cloudinary media management, the platform delivers enterprisegrade capabilities at a fraction of the operational cost of commercial alternatives. Empirical evaluation confirmed strong performance benchmarks, a 81.4 SUS usability score, and zero critical security vulnerabilities, validating the platform's readiness for production deployment.

The architecture's modularity and the use of open standards throughout the stack ensure that QuickCity_Cart can evolve alongside the rapidly changing e-commerce technology landscape without requiring disruptive rewrites.

The platform represents a meaningful contribution to the digital transformation of India's vast micro, small, and medium enterprise (MSME) sector.

A particularly noteworthy outcome of this research is the demonstration that a competently engineered open-source platform built on freely available technologies can match or exceed the feature sets of commercial platforms costing tens of thousands of rupees per year in subscription fees. This cost parity, combined with the full architectural transparency of an open codebase, positions QuickCity_Cart as a viable foundation for development agencies, academic institutions, and government digital commerce initiatives seeking to empower local vendor ecosystems at scale. The vendor trust scoring system, SSE-based real-time order tracking, and adaptive Cloudinary media delivery pipeline collectively represent novel engineering contributions that advance the state of practice in open-source multi-vendor platform development.

B. Future Work

Several high-value enhancements are identified for the next development phase. A machine learning-based personalized recommendation engine using collaborative and content-based filtering algorithms will replace the current co-purchase heuristics, with model training pipelines hosted on managed ML infrastructure. Conversational commerce capabilities via an integrated AI chatbot — powered by a large language model API — will provide customers with natural language product discovery and post-purchase support.

The vendor analytics dashboard will be extended with cohort analysis, funnel visualization, and predictive inventory forecasting powered by time-series demand models. A native mobile application for Android and iOS, built with React Native to maximize code reuse from the existing frontend codebase, will bring push notification support for real-time order updates. Finally, integration with ONDC (Open Network for Digital Commerce) — India's government-backed open protocol for e-commerce interoperability — will enable QuickCity_Cart vendors to reach customers across all ONDCcompliant buyer applications, substantially expanding total addressable market reach.

XIII. REFERENCES

[1] MongoDB, Inc., "MongoDB Atlas Documentation," 2023. [Online]. Available: https://www.mongodb.com/docs/atlas/

[2] Firebase, "Firebase Authentication Overview," 2024. [Online]. Available: https://firebase.google.com/docs/auth

[3] Razorpay, "Razorpay Payment Gateway Integration Guide," 2024. [Online]. Available: https://razorpay.com/docs/

[4] Cloudinary, "Cloudinary Developer Documentation," 2023. [Online]. Available: https://cloudinary.com/documentation

[5] Express.js, "Express — Node.js Web Application Framework," 2024. [Online]. Available: https://expressjs.com/

[6] React, "React: A JavaScript Library for Building User Interfaces," 2024. [Online]. Available: https://react.dev/

[7] A. Yildirim and G. Cetin, "Real-time Notification Architectures in Multi-vendor E-commerce Systems," *Journal of Web Engineering*, vol. 21, no. 4, pp. 1123–1148, 2022.

[8] OWASP Foundation, "OWASP Top Ten Web Application Security Risks," 2023. [Online]. Available: https://owasp.org/www-project-top-ten/

[9] R. Bansal and M. Gupta, "Scalable Microservices Architecture for Multi-Vendor Marketplaces using Node.js,"*International Journal of Computer Applications*, vol. 183, no. 22, pp. 14–19, 2021.

[10] J. Brooke, "SUS: A Quick and Dirty Usability Scale," in *Usability Evaluation in Industry*, London, U.K.: Taylor & Francis, 1996, pp. 189–194.

[11] J. Nielsen, "Response Time Limits: 3 Important Limits," Nielsen Norman Group, 2020. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/

[12] S. Verma and P. Singh, "MERN Stack Performance Analysis for High-Concurrency E-commerce Applications," *ACM SIGAPP Applied Computing Review*, vol. 23, no. 1, pp. 5–16, 2023.