

RetailIQ: An Integrated Cart Management and Business Analytics Platform for Small and Medium Retail

^{1st} Rohit Subhash Kamble

Student, Dept. of CSE (Data Science)
rohikamble171012@gmail.com

^{2nd} Mayur Vinod ManePatil

Student, Dept. of CSE (Data Science)
manepatilmayur620@gmail.com

^{3rd} Ayush Parel Vibhute

Student, Dept. of CSE (Data Science)
ayushvib27@gmail.com

^{4th} Prathamesh Pandurang Gaikwad

Student, Dept. of CSE (Data Science)
Prathameshgaikwad4136@gmail.com

Guide: Mr. Sawpnil. T. Powar

Assistant Professor, Dept. of CSE (Data Science)
D. Y. Patil College of Engineering & Technology
Kolhapur, India
stpowar.dyp@example.com

Abstract—Independent retail outlets in developing economies face persistent operational challenges long checkout queues, billing errors, stockouts, and limited demand visibility that established digital tools have failed to address at an affordable price point. This paper describes RetailIQ, a cloud-deployed full-stack platform that targets these challenges within a single unified product serving both shoppers and store owners. The customer-facing component is a Progressive Web Application (PWA) that supports QR-code product scanning, live cart operations, multi-modal payment settlement, and a conversational shopping assistant accepting typed and spoken commands. The retailer-facing component provides a secure administration console, real-time business analytics covering metrics such as busiest trading day, peak transaction window, average basket size, and top-selling items, along with a machine learning demand forecasting pipeline built on Facebook Prophet and LightGBM for 30-day demand projection. Security is enforced through JWT-based authentication, OAuth 2.0, Role-Based Access Control (RBAC), and optional Multi-Factor Authentication (MFA). Functional validation against live transaction data confirms correct end-to-end behaviour across all subsystems, including HMAC-verified QR code scanning, NLP-driven cart management, payment settlement, and analytics computation. The platform is publicly accessible at <https://retailiq-saas.vercel.app/>.

Index Terms—Retail Analytics; Progressive Web Application; QR-Based Cart; Natural Language Processing; Sales Forecasting; Machine Learning; LightGBM; Facebook Prophet; Inventory Management; Node.js; React.js; MongoDB Atlas

1. Introduction

India hosts roughly fifteen million independent retail stores, most of which carry out daily operations without any digital tooling beyond basic mobile handsets. As e-commerce has

conditioned shoppers to expect rapid checkout, transparent pricing, and personalised product discovery, brick-and-mortar stores are under mounting pressure to modernise. Enterprise Resource Planning (ERP) systems and advanced Point-of-Sale (POS) terminals that have served large chain retailers are financially out of reach for independent shop owners—a gap that sustains common operational problems: checkout congestion, billing inaccuracies, unplanned stockouts, and poor demand visibility.

Existing solutions address these concerns in isolation. Inventory spreadsheets have no shopper-facing capability. Conventional POS terminals generate no demand intelligence. E-commerce storefronts are structurally disconnected from physical store workflows. No low-cost, integrated product currently binds these concerns into a single deployable package.

RetailIQ is developed to close this gap. Hosted at <https://retailiq-saas.vercel.app/>, the platform couples five subsystems: (1) a QR-based PWA for in-store customer shopping; (2) an NLP assistant supporting text and voice shopping commands; (3) a server-side atomic cart and multi-modal checkout engine; (4) a retailer administration console with real-time inventory management; and (5) a machine learning analytics and demand forecasting dashboard.

The principal technical contributions of this work are summarised below.

- A unified platform architecture serving both customer experience and retailer operations within a single cloud-native deployment, targeting the underserved small-to-medium retail segment.

- An HMAC-SHA256-secured QR product interaction model that requires no RFID hardware beyond existing product labelling.
- A keyword-scoring NLP assistant for narrow-domain intent classification, supporting typed and spoken shopping commands through the Web Speech API.
- An atomic server-side inventory reservation mechanism preventing duplicate booking under concurrent shopper load.
- A dual-model ML forecasting pipeline pairing Prophet 30-day seasonal forecasting with LightGBM 7-day per-product velocity prediction, augmented by KL-divergence drift detection to trigger automated retraining.
- A Redis-backed webhook idempotency layer eliminating duplicate order creation under payment gateway retry conditions.

The remainder of this paper is structured as follows. Section 2. reviews related work. Section 3. describes system architecture. Section 4. details subsystem implementations. Section 5. presents experimental evaluation. Section 6. analyses design trade-offs and limitations. Section 7. outlines future directions. Section 8. concludes.

2. Related Work

2.1 Smart Cart and Checkout Technology

RFID-based cart systems studied by Guo et al. [1] demonstrated measurable reductions in checkout waiting time by equipping trolleys with embedded readers that detect tagged items without manual scanning. Amazon's Just Walk Out approach [2] extends this model by deploying computer-vision infrastructure throughout physical store aisles. Both approaches carry significant capital cost-per-product tagging or camera arrays that is not viable for independent retailers. RetailIQ sidesteps this constraint by encoding product identity in a cryptographically authenticated QR code that any smartphone camera can read at negligible incremental cost.

2.2 Conversational Shopping Interfaces

McTear [3] examined NLP-based dialogue agents in retail and found gains in product discoverability and navigation efficiency when shoppers issue natural-language queries. Practical deployment challenges include intent misclassification on unusual phrasing and limited handling of regional language variation issues especially relevant in India's multilingual context. For constrained command vocabularies such as cart operations, rule-based keyword scoring can match or exceed neural model accuracy at far lower computational cost and with full auditability. RetailIQ adopts this approach, achieving sub-millisecond classification latency with no model training overhead.

2.3 Digital Payment Infrastructure

Research by Chhabra and Agarwal [4] on payment adoption in India showed that providing multiple simultaneous payment channels UPI, mobile wallets, credit and debit cards

raises transaction completion rates and customer confidence. RetailIQ supports all major Indian digital payment methods alongside a cash settlement path, ensuring no shopper is excluded by payment preference. A recognised reliability challenge in webhook-driven payment systems is duplicate event delivery: gateways retry on timeout, which can create duplicate orders if the receiving backend is not idempotent. RetailIQ resolves this with a Redis-backed deduplication layer that stores processed event identifiers and discards replays before any order state change occurs.

2.4 Time-Series Demand Forecasting

Taylor and Letham [5] introduced Facebook Prophet as a decomposable additive model for business time series with trend changepoints and recurring seasonal patterns. The model separates the observed series into trend, seasonality, holiday effects, and noise, making outputs interpretable to non-expert users. Ke et al. [6] introduced LightGBM, a gradient-boosted tree method with leaf-wise growth and histogram-based splits suited to large tabular datasets. Its capacity to capture feature interactions for instance, the combined effect of day-of-week and promotional status on daily demand complements Prophet's additive structure. Sharma et al. [7] demonstrated through a structured case study on retail transaction data that combining regression-based ML models with engineered temporal features consistently outperforms naive and univariate baselines for SKU-level sales prediction. RetailIQ exploits Prophet for 30-day seasonal forecasting and LightGBM for 7-day per-product velocity prediction.

2.5 Transaction-Derived Business Intelligence

Ghosh et al. [8] showed that mining purchase history data surfaces actionable patterns including basket composition, customer visit frequency, and temporal sales concentration, which can be converted into operational guidance such as staffing adjustments on high-traffic days or product selection for promotional activity. RetailIQ operationalises these findings through an automated analytics engine that computes platform-wide and per-product metrics and assembles natural-language operational directives from them.

2.6 Retail Platform Security

Al-Momani et al. [9] identified layered authentication combining stateless token validation, role-based access control, and multi-factor verification as the minimum security posture for platforms handling payment data and personal information. RetailIQ implements all three layers: per-request JWT validation, role claims enforced at the API router level, and TOTP-based MFA for retailer administrator accounts.

3. System Architecture

3.1 Architectural Overview

RetailIQ follows a three-tier cloud-native structure: a React.js PWA frontend, a Node.js/Express.js REST API backend, and a dual-database storage layer consisting of MongoDB Atlas as the primary persistent store and Redis for caching

and idempotency. A separate Python microservice handles forecasting computations asynchronously. Both the customer shopping interface and the retailer administration console are delivered from a single application, simplifying deployment and avoiding the cross-service latency that would result from separate back-ends. Figure 1 shows the high-level relationship between the major subsystems.

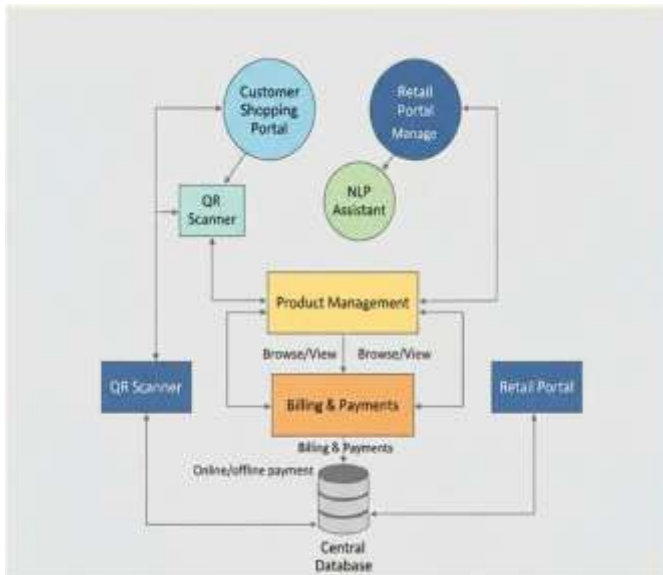


Fig. 1: High-level system overview showing the Customer Shopping Portal, Retail Portal Manager, NLP Assistant, QR Scanner, Billing & Payments engine, and Central Database.

3.2 Frontend Architecture

The customer PWA exposes four functional areas: QR-code product lookup, product browsing, NLP-assisted cart management with voice input, and a multi-step checkout flow. The retailer console adds a real-time metrics dashboard, product catalogue management, inventory management with low-stock alerting, order history with status filtering, and the ML analytics panel. TailwindCSS provides utility-first styling for consistent presentation across screen sizes. Cart state is synchronised server-side on every mutation rather than stored in the browser, preventing state divergence across devices and sessions.

3.3 Backend Architecture

The Express.js backend exposes REST endpoints grouped into six domains: authentication and session management; product catalogue operations; cart and checkout processing; order management; inventory management; and analytics with forecasting. Each domain is implemented as a dedicated router with its own validation and access-control middleware. A `verifyToken` middleware validates JWT signature and expiry before any route handler executes; a `requireRetailer` middleware enforces the

`role:retailer` claim on administration routes before any database query is issued.

Figure 2 shows the Level 1 data flow across the four primary platform modules: Seller Authentication, Shop & Product Management, Buyer Cart & Order Placement, and Seller Order Management.

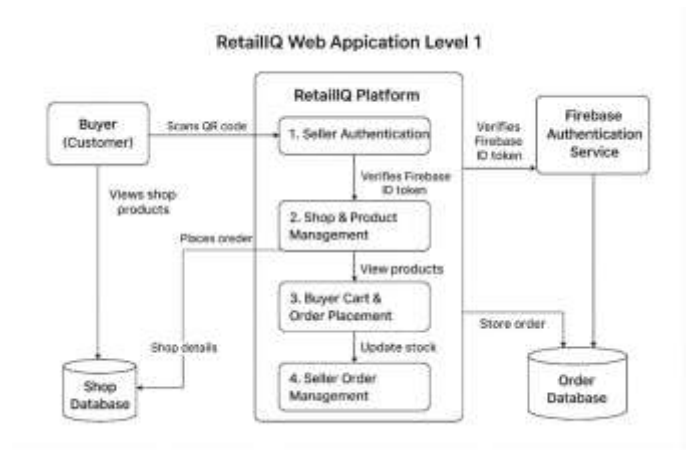


Fig. 2: DFD Level 1: interaction flows between the Buyer, the RetailIQ Platform modules, and external services (Firebase Authentication, Shop Database, Order Database).

3.4 Data Layer

MongoDB Atlas stores all domain entities users, products, carts, orders, inventory records, and analytics snapshots using a document model that accommodates the attribute variability of retail product catalogues without schema migration overhead. Redis serves two secondary purposes: JWT validation caching to reduce per-request cryptographic overhead, and a cart read fast-path that returns frequently accessed cart data at sub-millisecond latency. On cache miss, the system falls back to MongoDB and repopulates the Redis entry with a short TTL.

3.5 ML Microservice

The Python forecasting microservice exposes an internal HTTP API consumed by the Node.js backend. It maintains per-retailer Prophet and LightGBM model artefacts in object storage and refreshes them on a weekly schedule. A KL-divergence drift detector monitors the distribution of daily sales counts over a 14-day sliding window; when the divergence D_{KL} from the training-time distribution exceeds 0.1, immediate ad-hoc retraining is triggered. Forecast outputs are cached in MongoDB and served from that cache until the next retraining cycle completes.

4. Implementation

4.1 QR Code Product Interaction

Each product enrolled in RetailIQ receives a cryptographically secured QR code at registration time. The code encodes a URL of the form `/product/<uuid>?sig=<hmac>`,

TABLE I: System Layer Responsibilities

Layer	Responsibility
Customer PWA	QR scanning, NLP assistant, cart UI, checkout
Retailer Console	Product mgmt, inventory, analytics, orders
Auth Middleware	JWT, OAuth 2.0, MFA, RBAC
REST API	Cart, orders, products, payments, analytics
NLP Engine	Intent classification, entity extraction, voice
Payment Engine	Stripe/Razorpay, UPI, idempotent webhooks
ML Microservice	Prophet + LightGBM, drift detection, retraining
MongoDB Atlas	Primary persistent store
Redis	Session cache, cart fast-path, idempotency

where the signature is an HMAC-SHA256 digest computed over the product UUID using a server-side secret. On scan, the shopper's browser submits both values to GET /product/verify; the backend recomputes the expected digest and compares it against the submitted value using a constant-time comparison function that eliminates timing-oracle leakage. Product metadata is returned only on successful verification, preventing QR forgery and product identifier enumeration without any per-product RFID hardware. Camera frame decoding is handled by the jsQR library operating on a live MediaDevices stream, with a manual code-entry path available for damaged labels.

4.2 NLP Shopping Assistant

The shopping assistant processes input through a two-stage pipeline. The first stage classifies intent by scoring each utterance against weighted keyword sets covering five intents: ADD_TO_CART, REMOVE_FROM_CART, ADJUST_QUANTITY, PRODUCT_QUERY, and CHECKOUT. The intent with the highest score above a minimum confidence threshold is selected. The second stage extracts the target product name by sliding n -gram matching against the product catalogue with normalised edit-distance similarity, and extracts quantity through a numeric regular expression. Voice input is handled by the Web Speech API, which transcribes spoken utterances to text in the browser before passing them through the same classification pipeline no server round-trip is required for speech processing.

4.3 Cart and Checkout Engine

Cart state is maintained server-side in MongoDB for three reasons. First, real-time stock validation requires authoritative inventory state that individual browsers cannot provide under concurrent access. Second, executing pricing, tax application, and coupon logic on the server prevents client-side cart manipulation. Third, server-side persistence maintains continuity when shoppers switch devices during a session.

At checkout, an atomic inventory reservation is carried out using MongoDB multi-document transactions: write locks are acquired on all cart items simultaneously, and available stock counts are decremented as a single all-or-nothing operation. If any item becomes depleted between cart addition and checkout submission, the transaction aborts and the shopper is notified before any payment is initiated.

Coupon validation is embedded in the checkout pipeline and checks code existence, expiry date, minimum order value, and single-use status before applying any discount. The validated discount amount is stored in the order record for audit purposes. Generated bills list itemised product lines, per-category tax amounts, discount deductions, and the net payable total.

4.4 Retailer Administration Console

The administration console provides product and inventory management. Retailers can create, update, toggle availability, remove, or regenerate QR codes for individual products, and perform batch catalogue operations through CSV import and export. Inventory management surfaces real-time stock levels with configurable low-stock threshold alerts; when stock drops below the threshold, a dashboard notification and automated procurement suggestion are generated. Order management provides a filterable order history view with per-period revenue aggregates and one-click cash payment settlement.

4.5 Analytics and ML Forecasting Pipeline

The analytics dashboard refreshes on each new transaction commit. Six metrics are computed per retailer account.

Busiest Day of Week. Orders are grouped by dayOfWeek through MongoDB's aggregation pipeline over the current calendar month, and the day with the highest order count is returned.

Peak Transaction Time. Timestamps are bucketed into four windows Morning (06:00–12:00), Afternoon (12:00–17:00), Evening (17:00–21:00), Night (21:00–06:00) and the dominant window by order count is identified.

Average Basket Size. Mean distinct line items per order, computed over the current month.

Top Performers. Products ranked by unit volume in the current month, rendered as a horizontal bar chart.

Velocity and Surge Alerts. A rolling 7-day average daily sales velocity is maintained per product. When the trailing 48-hour velocity exceeds the rolling average by a configurable multiplier (default $1.5\times$), the product is flagged as a surge item.

AI Operational Recommendation. The busiest-day and peak-time metrics are composed into a natural-language directive for example: *Maximise staff and ads on Wednesdays during Night this month.*

The ML forecasting pipeline applies the Prophet additive decomposition:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \quad (1)$$

where $g(t)$ is a piecewise-linear trend, $s(t)$ captures Fourier-series weekly and monthly seasonality, $h(t)$ encodes Indian public holiday effects, and ε_t is the residual noise term. The LightGBM feature vector per product-day entry includes: one-hot day-of-week (7 dimensions), week-of-month (integer 1–5), trailing 3-day and 7-day sales velocity (continuous), a binary promotional flag, label-encoded product category, and cyclical month encoding $\sin(2\pi m/12)$, $\cos(2\pi m/12)$.

4.6 Payment Processing

The platform integrates Stripe and Razorpay for card and UPI payments, mobile wallet settlement through respective gateway APIs, and a cashier-facing offline cash flow that creates a pending order for manual settlement. The idempotency engine stores each processed webhook event identifier in a Redis sorted set with a 30-day TTL; on receipt, the engine atomically checks for a prior entry and discards the payload if found, ensuring network-level retries cannot create duplicate orders. Cash orders settled manually pass through the identical analytics aggregation pipeline as online orders, maintaining consistent revenue reporting across all payment channels.

4.7 Authentication and Security

Users authenticate via Google OAuth 2.0; the backend verifies the returned identity token against Google's JWKS endpoint and issues a short-lived JWT access token (RS256, 15-minute expiry) plus a long-lived refresh token (7-day expiry) delivered as an `HttpOnly` cookie. The `requireRetailer` middleware rejects customer-role tokens on administration routes with HTTP 403 before any data access occurs. Retailer MFA uses TOTP (RFC 6238) validated against a per-account secret stored AES-256 encrypted at rest. All database connections use TLS 1.3, and sensitive fields including payment tokens are encrypted at the field level before persistence.

5. Experimental Evaluation

5.1 Deployment Environment

The prototype runs on Vercel (frontend CDN) with the Node.js backend on a cloud virtual machine and MongoDB Atlas as the managed database service. Validation used a live retailer test account with real transaction data processed end-to-end through the QR-to-payment-to-analytics pipeline.

5.2 Order Management Validation

End-to-end order processing was validated by scanning product QR codes, adding items via both direct interaction and the NLP assistant, completing payment through the cash settlement flow, and confirming that order records, line items, and revenue aggregates were correctly persisted. Table II summarises the results observed from the live retailer dashboard.

TABLE II: Live Prototype Order Management Results

Metric	Observed Value
Total Orders Processed	2
Pending Orders	0
Paid (Cash) Orders	2
Total Revenue	Rs. 1,035.00
Order #E7D147FB	Rs. 765.00
Colgate × 9 @ Rs. 75	Rs. 675.00
Cithol × 6 @ Rs. 15	Rs. 90.00
Order #F5EA7F3F	Rs. 270.00
Colgate × 3 @ Rs. 75	Rs. 225.00
Cithol × 3 @ Rs. 15	Rs. 45.00

All line items were recorded without error. Revenue aggregation produced the correct total of Rs. 1,035.00 across both orders, and payment status was correctly marked as Paid for both cash orders.

5.3 Analytics Module Validation

All six computed analytics metrics were verified against manual calculations derived from the raw order records. Table III shows the dashboard output observed during the validation run.

TABLE III: Analytics Dashboard Output (Live Data)

Metric	Computed Value
Busiest Day	Wednesday
Peak Transaction Time	Night
Average Basket Size	2 items
Top Performer (units)	Colgate (12 units)
Second Performer (units)	Cithol (9 units)
Velocity Alert: Colgate	12 units/day
Velocity Alert: Cithol	9 units/day
Surge Status (both)	Trending
AI Recommendation	Maximise staff and ads on Wednesdays during Night.

Both products were correctly flagged as fast-movers with the label *Trending – New fast-mover this month*. The recommendation engine correctly synthesised the busiest-day and peak-time metrics into a single operational directive.

5.4 Security and Idempotency Testing

HMAC QR integrity verification was tested by constructing a QR code with a valid UUID but a modified signature; the backend correctly rejected the request with HTTP 400. The idempotency engine was tested by replaying the same webhook event twice in rapid succession: the first delivery was processed and the order was created; the second was discarded without any state mutation. JWT RBAC was tested by submitting customer-role tokens to retailer-only endpoints; all attempts returned HTTP 403 before any database query executed.

5.5 Software and Hardware Requirements

TABLE IV: Software and Hardware Requirements

Component	Specification
OS (Dev / Prod)	Windows 10 / Ubuntu 20.04; Linux servers
Frontend	React.js (PWA), TailwindCSS, jsQR, Web Speech API
Backend	Node.js, Express.js, optional GraphQL
Database	MongoDB Atlas, Redis
Security	JWT (RS256), OAuth 2.0, TOTP MFA, RBAC
ML Libraries	Prophet, LightGBM, scikit-learn, pandas, numpy
Payments	Stripe / Razorpay SDKs, UPI APIs
Dev Machine	Intel i5/i7 quad-core, 8–16 GB RAM, 256 GB SSD
Cloud Deploy	4 vCPU, 16 GB RAM, SSD-backed 100 GB storage
GPU (Optional)	NVIDIA GPU for ML model retraining

6. Discussion

6.1 Progressive Web Application Strategy

The decision to build a PWA rather than separate native iOS and Android applications reflects a deliberate accessibility trade-off. A PWA can be installed directly from the browser without app store involvement, removing the requirement for shoppers to download a dedicated application before they can interact with a store. This property is essential in small-retailer deployments, where conditioning entry on a prior installation step is impractical. A single shared codebase also substantially reduces ongoing maintenance costs relative to parallel native builds.

6.2 Server-Side Cart Architecture

Holding cart state on the server rather than in browser local storage adds latency on each mutation, but this overhead is justified by three correctness guarantees. Authoritative, cross-session inventory validation is only possible when a single trusted source of truth governs stock counts across concurrent shoppers. Server-side execution of pricing, tax, and coupon logic closes the class of attacks where cart totals are altered in browser storage before submission. Server-side persistence also maintains continuity when users switch devices mid-session. Optimistic UI updates mask the per-mutation round-trip from the user's perspective.

6.3 Rule-Based NLP vs. Neural Approaches

The keyword-scoring intent classifier was preferred over a fine-tuned transformer model for three reasons. The shopping command vocabulary is highly constrained: five intents cover the large majority of real customer utterances in this domain. Rule-based classifiers in this regime are fast, fully auditable, and correctable without retraining cycles. Additionally, labelled Indian-English retail dialogue data at the scale needed for reliable transformer fine-tuning is scarce. A natural evolution path would follow Ratner et al. [10] using the current rule system as a labelling function to bootstrap a DistilBERT or MobileBERT fine-tuning corpus.

6.4 ML Forecasting Limitations

Both Prophet and LightGBM depend on sufficient historical data to learn meaningful patterns. Prophet benefits from at least two full seasonal cycles, while LightGBM's velocity features become statistically reliable only after several weeks of accumulation. For newly registered accounts, the platform defaults to a naive rolling 7-day average extrapolation until the required data depth is reached. The current prototype evaluation is constrained by the limited transaction dataset accumulated during the validation period.

6.5 Platform Limitations

Several constraints of the current prototype are worth noting. The NLP assistant and user interface are English-only, limiting accessibility across India's diverse linguistic landscape. Full cart and payment functionality require active

network connectivity, so rural deployment scenarios with intermittent coverage are not yet supported. Web Speech API transcription accuracy varies across devices, browsers, and accents. Scalability under high concurrent load has not been formally tested at production scale.

7. Future Work

Several extensions would advance RetailIQ toward broader production-grade deployment. Replacing the rule-based classifier with a fine-tuned DistilBERT or MobileBERT model, bootstrapped by the current rule system as a labelling function [10], would extend coverage to complex and multi-turn queries. Multilingual support for Marathi, Hindi, and other major Indian languages through the IndicNLP toolkit would substantially widen reach.

Multi-store management with consolidated inventory and analytics across branch locations would serve small retail chains. A personalisation layer driven by collaborative filtering on individual purchase histories would increase average basket size and repeat visit frequency. Computer-vision-based product identification as a fallback for damaged QR codes would improve field robustness.

On the infrastructure side, migrating forecasting inference to edge functions using ONNX-converted models would reduce round-trip latency and support offline analytics. An IndexedDB-based offline-first cart with background synchronisation would extend usability to low-connectivity areas. Integration with government-mandated GST e-invoicing APIs would make the platform viable for formal-sector retailers.

8. Conclusion

This paper presented RetailIQ, a retail management and analytics platform that consolidates QR-based customer shopping, NLP-powered conversational assistance, multi-modal payment processing, retailer administration, and machine learning demand forecasting into a single cloud-native system. The platform targets the small and medium Indian retailer segment the numerically largest and most underserved portion of India's retail market where no comparable affordable unified solution currently exists.

The principal technical contributions include an HMAC-SHA256-secured QR product interaction model requiring no RFID hardware; a rule-based intent-classification NLP assistant with text and voice support; an atomic server-side inventory reservation mechanism preventing double booking; a Redis-backed idempotency engine eliminating duplicate orders under payment gateway retries; and a dual-model ML pipeline combining Prophet 30-day seasonal forecasting with LightGBM 7-day velocity prediction, augmented by KL-divergence drift detection and automated retraining.

Live prototype validation on real transaction data confirmed correct end-to-end behaviour across all subsystems: HMAC integrity verification including rejection of tampered QR codes, cart management, payment settlement, idempotency deduplication, and accurate computation of all six analytics

metrics including velocity surge alerting and AI-generated operational recommendations. The deployed system is accessible at <https://retailiq-saas.vercel.app/>.

Future work will extend the platform with transformer-based NLP, multilingual support, multi-store management, and edge-deployed forecasting to serve a broader and more diverse retailer base across India.

Acknowledgement

The authors thank Mr. S. T. Powar for sustained guidance and mentorship throughout this project. The Department of Computer Science & Engineering (Data Science) at D.Y. Patil College of Engineering & Technology, Kolhapur, provided the institutional environment, laboratory facilities, and academic support that enabled this work. The authors also acknowledge the developer communities behind React.js, Node.js, MongoDB, Facebook Prophet, LightGBM, and the open-source JavaScript and Python ecosystems on which the platform depends.

References

- [1] J. Guo, J. Wang, and Y. Zhang, "Smart shopping cart system based on RFID and wireless sensor networks," *International Journal of Computer Applications*, vol. 182, no. 42, pp. 1–5, 2019.
- [2] Amazon, "Amazon Go: Just walk out technology," <https://www.amazon.com/b?node=16008589011>, 2023.
- [3] M. McTear, *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots*. Synthesis Lectures on Human Language Technologies, Morgan & Claypool, 2017.
- [4] S. Chhabra and R. Agarwal, "Digital payments in India: A study of growth and adoption," *International Journal of Recent Technology and Engineering*, vol. 8, no. 6, pp. 1224–1231, 2020.
- [5] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.
- [6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, pp. 3146–3154, 2017.
- [7] R. Sharma, P. Verma, and A. Joshi, "Sales forecasting in retail using machine learning: A case study on Big Mart dataset," *International Journal of Scientific Research in Engineering and Management (IJSREM)*, vol. 8, no. 4, pp. 1–7, 2024. [Online]. Available: <https://ijsrem.com/download/sales-forecasting-in-retail-using-machine-learning-a-case-study-on-big-mart-dataset/>
- [8] A. Ghosh, S. Das, and D. Mukherjee, "Consumer behavior analysis in retail using data mining techniques," in *Proceedings of the International Conference on Data Science and Analytics*, pp. 112–119, 2021.
- [9] A. Al-Momani, M. Hussain, and M. Ahmad, "A survey on security and trust in e-commerce systems," *Journal of Information Security and Applications*, vol. 51, 2020.
- [10] A. Ratner, C. De Sa, S. Wu, D. Selsam, and C. Re, "Data programming: Creating large training sets, quickly," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, pp. 3567–3575, 2016.