Real-Time POS–Loyalty Integration Framework for Merchant–Banking

Sumedh Khedekar[1] Mr. Yogeshchandra Puranik[2]
[1]Student [2]Assistant professor
[1,2]Master of Computer Applications
[1,2]PES's Modern College of Engineering, Pune, Maharashtra, India

Abstract — The increasing convergence of retail and banking ecosystems demands seamless and intelligent customer engagement solutions. This research proposes a Real-Time POS– Loyalty Integration Framework that establishes a standardized API-based orchestration layer to integrate Android Point-of-Sale (POS) systems with centralized loyalty engines. The objective is to enable instant reward accrual and redemption during live transactions, ensuring a frictionless checkout experience. The proposed framework introduces a secure middleware layer that facilitates real-time communication between POS terminals, banking networks, and loyalty management platforms. Built on RESTful APIs and event-driven architecture, the system supports dynamic rule evaluation for reward calculation, including tier-based incentives and promotional campaigns. The framework exposes 56 REST API endpoints across Customer Management, Campaign Management, Tier Configuration, Analytics, and Transaction modules — all secured through JWT-based token authentication with refresh token rotation. Security is maintained through token-based authentication, encrypted data exchange, and role-based access control, ensuring compliance with financial-grade standards. The framework supports interoperability with payment gateways and core banking systems, enabling unified ecosystem participation. By enabling real-time reward visibility and redemption at checkout, the system enhances customer satisfaction and retention while providing retailers and banks with actionable transaction analytics across five dimensions: Member Health, Points Economy, Customer Engagement, Financial Impact, and Tier Dynamics.

Keywords: Android POS, Flutter SDK, Loyalty Platform, RESTful API, JWT Authentication, Event-Driven Architecture, Tier-Based Rewards, Spring Boot, PostgreSQL, Campaign Management

I. Introduction

Point-of-Sale (POS) terminals are the primary touchpoint between merchants and their customers in retail and banking environments. Despite advances in digital payment infrastructure, the majority of POS systems continue to process payments in isolation — without any mechanism to reward customers, track repeat visits, or surface behavioural insights at the moment of transaction. This gap represents a significant missed opportunity for merchants who wish to build loyalty without increasing operational friction.

Existing loyalty solutions either operate independently of the payment terminal — requiring double-entry of transaction data — or are tightly coupled to specific hardware vendors with no customization capability. Third-party loyalty applications introduce adoption barriers by requiring separate installations on customer devices. Neither model is viable for merchants embedded in banking ecosystems, where loyalty programs must conform to financial-grade security, multi-tenant isolation, and centralized management.

This research presents the DUX Loyalty Platform, a production-oriented implementation of a Real-Time POS–Loyalty Integration Framework. The system is built as three interconnected components: a Flutter-based Loyalty SDK embedded inside an Android POS application, a Flutter Merchant Admin Dashboard for loyalty configuration and analytics, and a Spring Boot backend with PostgreSQL persistence exposing 56 REST API endpoints. Together, these components enable a merchant to intercept the payment lifecycle, offer customers real-time point redemption before checkout, and automatically finalize earned points after payment — all without interrupting the native POS payment flow.

The objective of this research is to document the design decisions, architectural patterns, and implementation outcomes of this framework, and to identify how it bridges documented gaps in the existing literature on loyalty systems, cross- platform mobile SDK development, event- driven architectures, and API security.

II. Literature Review

The development of mobile-first loyalty platforms has gained significant traction in recent years, particularly in the context of banking and merchant ecosystems. Bhagat et al. (2022) conducted a comprehensive review of Flutter as a cross-platform mobile development framework, highlighting its widget-based architecture, single codebase compilation to native ARM code, and its Dart runtime as key enablers of consistent UI/UX across Android and iOS environments. While the authors affirm Flutter's suitability for enterprise-grade applications, they acknowledge a notable gap in documented frameworks that leverage Flutter as an embedded SDK within third-party Android POS environments — a deployment model that introduces non-trivial lifecycle and context management challenges. The present system addresses this gap by deploying the Flutter loyalty module as an Activity-embedded SDK within an existing Android POS application, enabling seamless UI integration without disrupting the host payment workflow.

The communication architecture between POS terminals and loyalty backends is a critical design consideration in real-time transaction systems. The IEEE ICADEIS (2022) study on event-driven architecture demonstrates that decoupling services through asynchronous event streams significantly improves throughput and fault isolation in microservices environments. However, the study focuses primarily on server-side inter-service communication and does not address the hybrid synchronous- asynchronous pattern required when a POS device must await a loyalty engine response within a bounded transaction window. The proposed framework bridges this gap by implementing a synchronous REST callback from the POS module to the

Spring Boot loyalty backend, while isolating downstream enrichment operations — such as tier recalculation and analytics emission — as asynchronous post-transaction events, thereby balancing real-time responsiveness with scalability.

Tier-based reward structures are a foundational component of loyalty program design. Jasmi et al. (2022) examined tiered loyalty membership programs delivered through mobile applications and established that tier differentiation based on cumulative spend thresholds drives measurable increases in customer retention and transaction frequency. Their work, however, models tier logic as a static, configuration-time decision baked into the application layer. The present system extends this by externalizing tier definitions — including spend thresholds, multipliers, and eligibility conditions — into a merchant-configurable rule engine exposed through a REST API, enabling banks and merchants to update tier logic dynamically without redeployment, a requirement critical in regulated banking ecosystems.

Securing inter-service communication between POS devices, mobile clients, and backend APIs is a non-trivial challenge in multi-stakeholder environments. Bucko et al. (2023) examined JWT authentication and authorization mechanisms in web applications, emphasizing the importance of short-lived access tokens, refresh token rotation, and role-claim scoping as countermeasures against token interception and privilege escalation. They note, however, that most JWT implementations operate within single-service or same- domain contexts. The architecture implemented in this framework extends JWT security across a multi-role hierarchy — encompassing Bank Admin, Merchant Admin, and POS terminal identities — where each token carries role-scoped claims validated by Spring Security filter chains, ensuring that POS-issued transaction requests cannot escalate into administrative operations.

The backend design of the loyalty platform draws upon established patterns for scalable microservices in financial systems. Ranjani (2021) identified service decomposition, repository pattern abstraction, and layered controller-service-repository architectures as essential design patterns for maintainability and regulatory auditability in banking and insurance platforms. While Ranjani's work provides theoretical grounding for these patterns, it does not address the specific challenge of orchestrating a multi-tenant loyalty engine where merchant-level rule sets, configuration thresholds, and customer transaction histories must be isolated yet processed through shared infrastructure. The Spring Boot backend in this system operationalizes these patterns through tenant- scoped service delegation and merchant- specific configuration objects, fulfilling the auditability and isolation requirements unique to a bank-managed loyalty ecosystem.

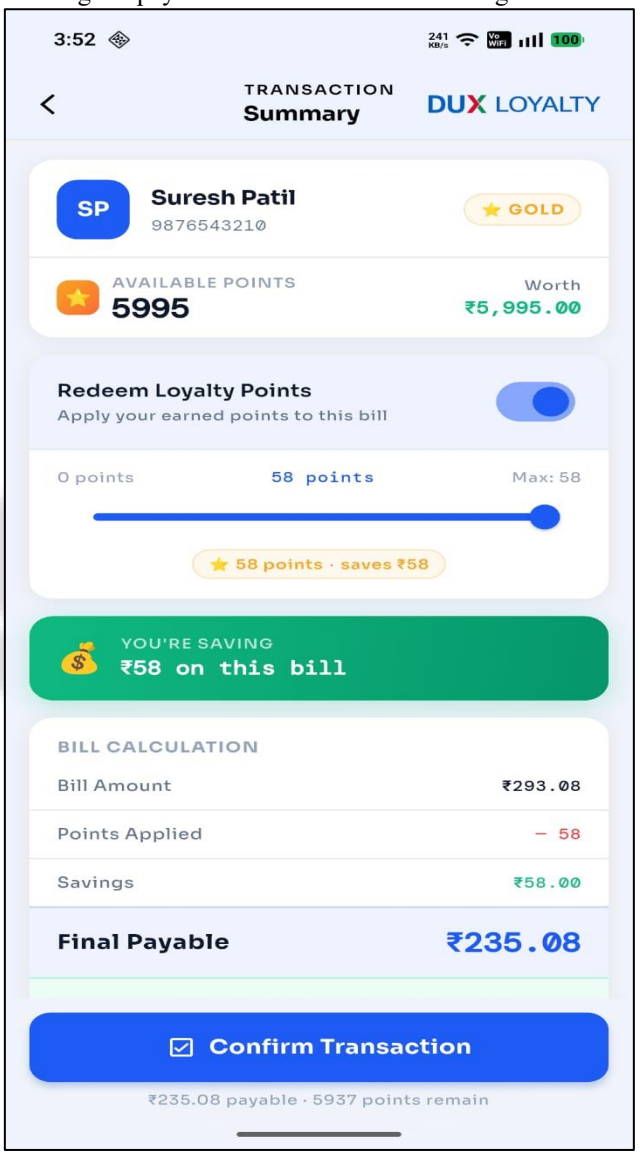
III. Methodology

A. System Architecture:

The proposed system is designed as a three- layer architecture, where each layer has a clearly defined responsibility and communicates through well-defined API contracts. The three layers are the Customer- Facing POS SDK, the Merchant Admin Dashboard, and the RESTful Backend Service.

1) POS Host Application (Android)

The POS host application is a native Android application responsible for hardware payment processing, including EMV card, UPI, and contactless NFC flows. It manages the physical terminal session and passes transaction metadata — amount, terminal ID, merchant ID — to the loyalty SDK through Android's platform channel mechanism. Upon receiving a redemption response from the loyalty module, the host application adjusts the final payable amount before initiating the payment instruction to the banking network.



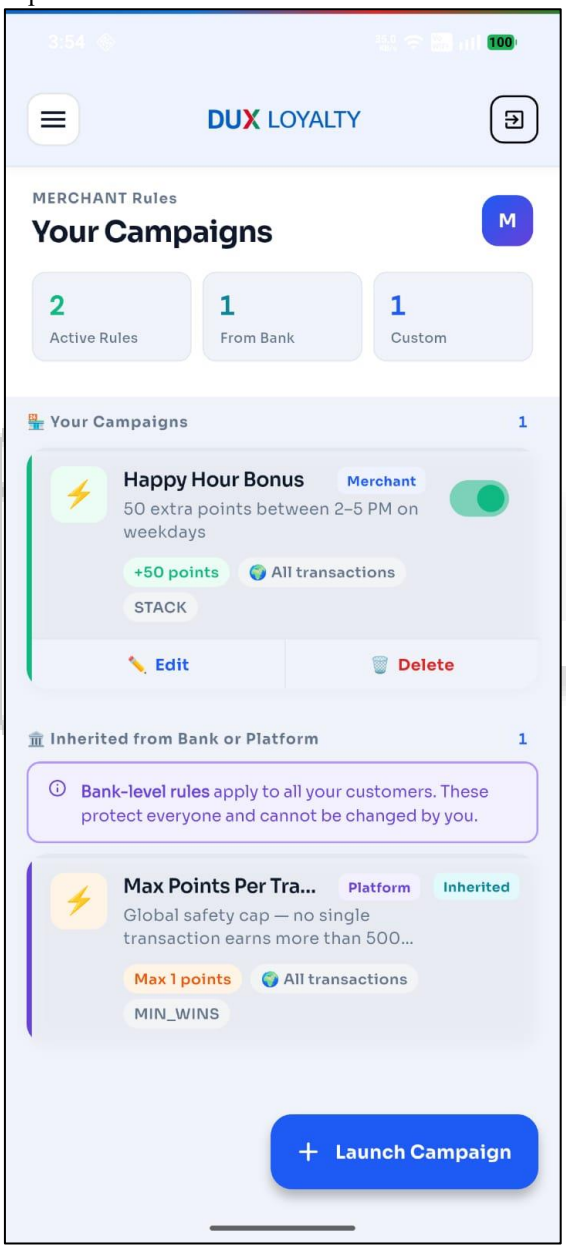
2) Loyalty SDK (Flutter — Embedded Module)

The loyalty module is a Flutter application compiled as a reusable SDK and embedded inside the POS host as an Android Activity. It is activated via platform channels when a payment session begins. The SDK supports two entry modes: a Customer Flow for transactional POS interactions, and a Dashboard Entry Point for merchant administrators. The mode is determined at initialization from parameters passed by the host application. The SDK manages customer lookup by mobile number, real- time eligibility checking, point redemption preview, and post-payment point

finalization — all communicating with the Spring Boot backend via REST APIs.

3) Merchant Admin Dashboard (Flutter — Standalone App)

The Merchant Admin Dashboard is the same Flutter codebase launched in Dashboard mode. It provides 16 screens covering Tier Configuration, Campaign Management, Rules Engine, Customer Management, Analytics (with four tabs and custom date range support), Redemption Flow, and Transaction History. All business rule parameters — earn rates, redemption thresholds, tier spend requirements, campaign windows — are configurable through this interface without developer intervention.



4) Backend API (Spring Boot + PostgreSQL)

The backend is a Spring Boot application with a PostgreSQL database, exposing 56 REST API endpoints organized into five functional modules: Authentication and Session Management, Customer and Membership Management, Transaction and Points Engine, Campaign and Rules Configuration, and Analytics Aggregation. The controller-service-repository layered architecture ensures separation of business logic from persistence concerns. JWT tokens with role-based claims govern all API access, with Spring Security filter chains enforcing role-level endpoint restrictions.

5) Data Flow

The end-to-end transaction data flow proceeds as follows: the POS terminal initiates a payment session and invokes the Loyalty SDK with transaction context. The SDK performs customer lookup via the backend, evaluates applicable loyalty rules and campaigns, presents a real-time redemption preview to the customer, and returns the final payable amount to the POS host. After payment confirmation, the POS notifies the SDK, which triggers the backend to finalize point accrual and update the customer's tier standing. The entire lifecycle is completed within a single payment session.

IV. Implementation & Technical Complexity:

The implementation addresses several non- trivial engineering challenges that distinguish it from conventional client-server mobile applications.

A. Cross-Platform Embedded SDK Deployment

The loyalty module operates as a Flutter SDK embedded inside a native Android POS application, activated through Android's platform channel mechanism. This deployment model requires careful management of the Flutter engine lifecycle within a foreign Activity context, including handling engine initialization, background/foreground transitions, and teardown without affecting the host POS session.

B. Android Lifecycle Debouncing

The Android POS operating system fires lifecycle events (onResume) twice within approximately 428 milliseconds during application startup. Without mitigation, this causes duplicate SDK initializations and race conditions in the loyalty session. The implementation resolves this through a timestamp-based debounce mechanism (_lastInitTime), accepting only the first initialization event within a configurable window and discarding subsequent duplicate firings.

C. Multi-Role JWT Authentication

The system implements a three-tier role hierarchy — Bank Admin, Merchant Admin, and POS Terminal — each authenticated via JWT tokens signed by the Spring Boot backend. POS terminals authenticate using Terminal ID and Merchant ID as programmatic credentials. Access tokens are short-lived, with refresh token rotation ensuring transparent re-authentication when sessions expire. Spring Security filter chains enforce role-level access control on all 56 API endpoints, preventing privilege escalation between roles.

D. Configurable Rule Engine

Loyalty earning rules are decoupled from application code and stored as merchant- level configurations in PostgreSQL. Each rule carries attributes including earn rate, minimum transaction threshold, usage limits per customer, limit reset period, applicable day-of-week constraints, and transaction date ranges. Rules are grouped into time-bound campaigns with scheduled activation and deactivation, preventing unintended point issuance outside promotional windows.

E. Modular Screen Architecture

The Flutter codebase implements 16 screens across both operation modes. Large screen files were refactored from monolithic implementations into modular component hierarchies, separating models, utilities, widgets, and screen logic into independent directories. This structure enables independent feature development and testability per screen without cross- module side effects.

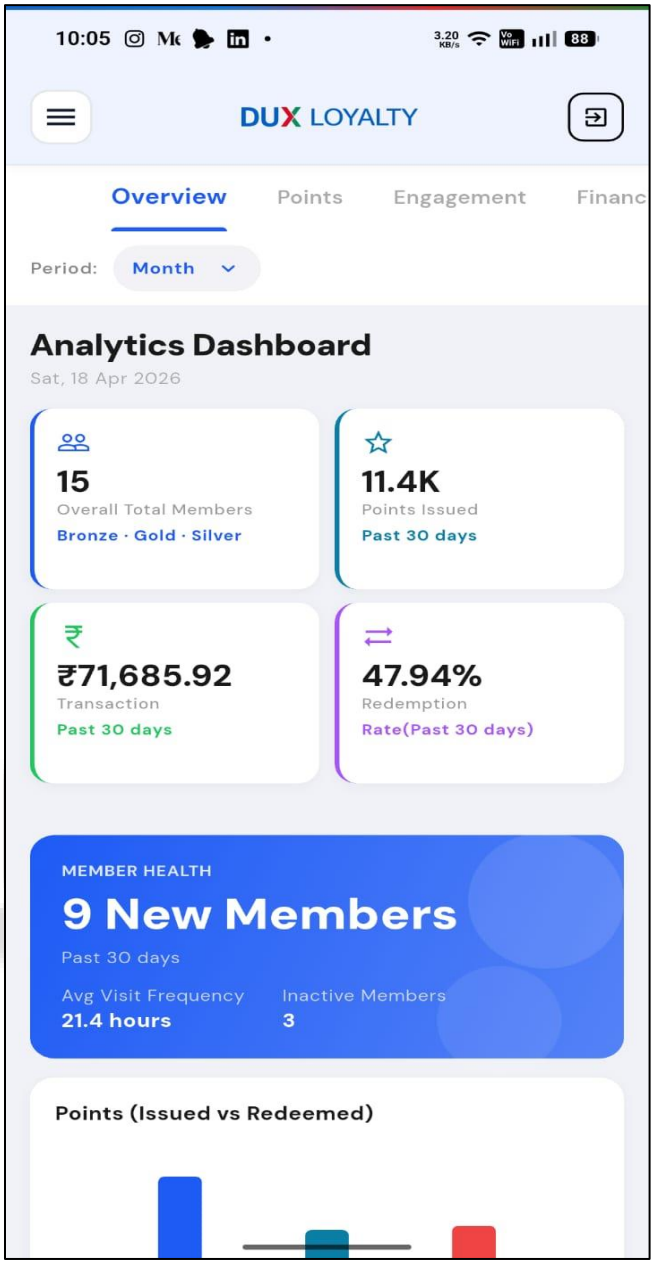
V. Results & Analysis:

The system was validated through functional testing on a real Android POS device, with end-to-end transaction flows exercised across all 16 application screens.

1) End-to-End Transaction Integration: The system successfully intercepts, processes, and finalizes loyalty points within the same payment lifecycle — from mobile number lookup through points redemption to post-payment point issuance — without customer drop-off.
2) Configurable Without Code Changes: All merchant-specific parameters (earn rates, redemption caps, tier thresholds, campaign windows) are fully editable through the admin dashboard. No deployment is required for business rule changes.
3) Analytics Across Five Dimensions: The dashboard tracks Member Health, Points Economy (issued, redeemed, expired, outstanding liability), Customer Engagement (top spenders, at-risk members), Financial Impact (revenue, discounts, refunds), and Tier Dynamics (upgrades, downgrades, tier-wise revenue).
4) Refund and Points Reversal: The system accurately reverses loyalty points when a payment is refunded, maintaining integrity of the points economy and preventing fraudulent point retention.
5) At-Risk and Tier-Upgrade Identification: The platform proactively identifies members at risk of churning and customers close to tier upgrade thresholds, surfacing them in the merchant dashboard for targeted intervention.
6) White-Label Theming: Merchant brand colors and logos are applied to the customer-facing UI at runtime from host application initialization parameters, requiring no SDK redeployment per merchant.

VI. Advantages:

1) Embedded SDK Architecture eliminates the need for a separate customer application — the loyalty experience is native to the existing payment terminal.
2) White-label theming enables commercial deployment across multiple merchant brands from a single codebase.
3) Fully configurable rule engine and campaign scheduler allow non- technical merchant administrators to manage the loyalty program independently.
4) Multi-role JWT security ensures financial-grade access control across Bank Admin, Merchant Admin, and POS Terminal identities without shared credentials.
5) Modular Flutter architecture supports independent screen development and reduces regression risk during feature additions.
6) Real-time redemption preview at checkout reduces decision friction and increases redemption adoption rates.



VII. Limitations:

- The current implementation does not include a caching layer (e.g., Redis), which may affect backend throughput under high concurrent transaction volumes in large-scale merchant deployments.
- The loyalty SDK operates exclusively on Android POS devices. iOS POS hardware is not currently supported, limiting deployment scope to Android-based terminal ecosystems.
- Load and performance testing under sustained concurrency has not been conducted. Response time guarantees under peak transaction loads remain uncharacterized.
- The rule engine supports a defined set of rule types (earn rate, birthday bonus, usage limits, day-of-week rules).

Addition of custom rule types such as product-specific multipliers requires backend schema extension.

– Offline resilience is not implemented. Loss of network connectivity during a payment session will interrupt the loyalty flow; a local fallback queue mechanism is absent from the current version.

VIII. CONCLUSION:

This research designed and implemented a Real-Time POS–Loyalty Integration Framework that addresses documented gaps in embedded SDK deployment, event-driven POS communication, configurable tier logic, multi-role JWT security, and scalable microservices architecture for banking environments. The resulting platform — comprising a Flutter loyalty SDK embedded in an Android POS application, a Flutter merchant admin dashboard, and a Spring Boot backend with PostgreSQL — demonstrates that a fully embedded, white- labeled loyalty system can be delivered as a cohesive product within the existing payment terminal infrastructure, without requiring hardware changes or separate customer applications.

The system successfully intercepts the payment lifecycle, enables real-time point redemption at checkout, and finalizes earned points post-payment — all within a single transaction session on a real Android POS device. The configurable rule engine and campaign scheduler empower non-technical merchants to manage their loyalty programs without development involvement, while the five-dimension analytics dashboard provides actionable retention intelligence.

The framework establishes a scalable, secure, and standardized foundation for next- generation loyalty orchestration across retail and banking environments, with clear extension points for future enhancements.

IX. FUTURE WORK:

1) Integration of an in-memory caching layer (Redis) to support sub-millisecond rule evaluation and session state management under high transaction concurrency.
2) Development of an offline queue mechanism in the POS SDK to buffer loyalty transactions during network interruptions and synchronize them upon reconnection.
3) Extension of the rule engine to support product-specific earning multipliers, referral point programs, and geofence- triggered bonus campaigns.
4) Introduction of machine learning-based churn prediction models that feed directly into the at-risk customer.

REFERENCES:

Research Papers

[1] S. A. Bhagat, S. G. Dudhalkar, P. D. Kelapure, A. S. Kokare, and S. A. Bachwani, "Review on Mobile Application Development Based on Flutter Platform," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 2022. DOI: 10.22214/ijraset.2022.39920.

[2] IEEE ICADEIS, "Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems," in *2022 International Conference on Advancement in Data Science, E-Learning and Information Systems (ICADEIS)*, 2022. DOI: 10.1109/ICADEIS56544.2022.10037390.

[3] N. A. Jasmi et al., "The Implementation of Tiered Loyalty Membership Program in Mobile Application via Behavioural Science for Customer Retention in Businesses," in *2022 IEEE 13th Control and System Graduate Research Colloquium (ICSGRC)*, Shah Alam, Malaysia, 2022. DOI: 10.1109/ICSGRC55096.2022.9845178.

[4] A. Bucko, "Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History," *Computers*, vol. 12, no. 4, p. 78, 2023. DOI: 10.3390/computers12040078.

[5] S. Ranjani, "Design Patterns for Scalable Microservices in Banking and Insurance Systems: Insights and Innovations," *International Journal of Emerging Research in Engineering and Technology*, year/details to be verified.

[6] F. F. Reichheld and W. E. Sasser, "Zero Defections: Quality Comes to Services," *Harvard Business Review*, vol. 68, no. 5, pp. 105-111, 1990.

[7] V. Kumar and D. Reinartz, *Customer Relationship Management: Concept, Strategy, and Tools*, 3rd ed. Berlin, Germany: Springer, 2018.

[8] [P. Kotler and K. L. Keller, *Marketing Management*, 15th ed. Pearson, 2016.

[9] Project source code, API definitions, database schema, generated loyalty reports, and POS-integrated Flutter dashboard components used during system design and implementation.

