IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, August 2026
Impact Factor: 8.2

PayNexus: A Scalable, Multi-Tenant Online Payment Gateway Platform

Devidas Thosar[1], Rajashree Thosar[2], Lakshmi Sharma[3], Akshay Banajgole[4], Chaitanya Rajmane[5], Chetan Mandode[6], Saurabh Sharma[7]

[1345 6]Dept. of CSE – Artificial Intelligence & Machine Learning
[2]Dept. of Electronics & Tele-Communication
[7]Manager, Technical Department Snowflake, Pune, India
G. H. Raisoni College of Engineering and Management, Pune, India

Abstract: *This paper describes PayNexus, an original, production-style build of a multi-tenant online payment gateway in which any number of merchants can onboard, provision API credentials, and accept payments through one shared deployment. Four payment rails common to Indian digital commerce — Cards, UPI (Intent and Collect), Net Banking, and Wallets — are routed through a single order-and-payment lifecycle, while the settlement mechanics unique to each rail are modelled separately underneath. The platform is decomposed into independently deployable microservices that talk to one another synchronously through an API Gateway and asynchronously over a Kafka event bus, treating idempotency, durable event delivery, and PCI-aware card tokenization as core architectural requirements rather than later additions. The intent of this work is to lay out, on one learnable codebase, the architecture, data model, and operational guarantees that a real fintech platform relies on.*

Keywords: Payment Gateway, Microservices, Idempotency, Transactional Outbox, PCI-DSS Tokenization, Multi-Tenant SaaS, UPI, Settlement Engine, Event-Driven Architecture.

I. INTRODUCTION

A payment gateway functions as the intermediary between a merchant and the banking network, taking on the job of authorising and capturing a customer's payment no matter which instrument is used — a debit or credit card, a Unified Payments Interface (UPI) transaction, net banking, or a mobile wallet. In the absence of such an intermediary, each merchant would be forced to build and maintain its own protocol-specific integration with every card network, every issuing bank, and the National Payments Corporation of India (NPCI) switch [8], each governed by a different authentication scheme, message format, and failure behaviour. Established gateways such as Razorpay, Stripe, PayU, and Cashfree already solve this for merchants, but as closed, hosted products, they keep their internal engineering out of view — the mechanisms by which they guarantee exactly-once order creation, retry a failed webhook, reconcile a day's transactions into a bank settlement, or keep a card number away from the rest of the system are simply not visible to a student trying to understand how such a system is actually put together.

PayNexus is an original, ground-up build of a production-style payment gateway, structured as a multi-tenant platform where any number of merchants can onboard, generate API credentials, and start accepting payments under one shared deployment, much as a commercial gateway serves thousands of unrelated businesses at once. It covers Cards, UPI (Intent and Collect), Net Banking, and Wallets, routing all four through one common order-and-payment lifecycle while still accounting for how differently each rail actually settles behind the scenes — direct debits against a wallet's own ledger, UPI's near-real-time interbank settlement through NPCI, batch clearing across card networks, and redirect-based debits for net banking.

DOI: 10.48175/IJARSCT-38112

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, August 2026
Impact Factor: 8.2

Three observations, specific to payment systems and seldom encountered in a typical student project, shaped the direction of this work. The first is that correctness under failure is the defining engineering problem in payments: a dropped connection, a retried request, or a process that crashes mid-flow must never end in a customer being charged twice, or in a merchant losing visibility of a payment that was, in fact, captured — which is why idempotency, explicit state machines, and durable event delivery are treated here as core design constraints rather than something bolted on afterward [1]. The second is that a payment gateway is inherently asynchronous by nature: a payment is authorised in the moment, but settled, reconciled, and reported well after the fact, making the whole system a natural fit for learning event-driven microservice design built around message queues and outbox-style publishing. The third is that touching card data at all pulls a project into PCI-DSS territory, so isolating and encrypting sensitive card data inside a dedicated vault service becomes a necessary security-by-design exercise rather than an optional feature. This same appreciation for cleanly separated, purpose-built processing stages recurs across other AI-driven system engineering efforts — from layered detection pipelines in surveillance systems to modular remote-sensing analysis workflows — where isolating each concern into its own stage improves both maintainability and fault isolation [2], [4].

II. LITERATURE REVIEW

Kleppmann's treatment of what distributed systems can and cannot guarantee remains foundational here: true exactly-once delivery, he argues, is not achievable over a network, and the practical workaround is to make operations idempotent at the application layer — typically by attaching a client-generated idempotency key to every mutating request and de-duplicating against it at the server [1]. This is precisely the mechanism PayNexus's order-creation and payment APIs rely on, each requiring an idempotency key on every write within a 24-hour de-duplication window. Garcia-Molina and Salem's earlier work on the Saga pattern takes this a step further, describing how a long-running transaction can be managed as a chain of local transactions coordinated through compensating actions instead of a single distributed lock [3] — which is essentially the shape of PayNexus's payment-to-settlement flow, and the reason an event-driven, choreographed microservice design was chosen over one large monolithic transaction.

Richardson's catalogue of microservices patterns is directly relevant to how PayNexus is structured, most notably the transactional outbox pattern: a service writes its domain event to an outbox table inside the same local transaction as its business write, and a separate relay process later publishes that event to a broker such as Kafka [5]. This sidesteps the familiar dual-write problem and is exactly what payment-service uses to publish payment and refund events for operations-service to pick up. Separately, the PCI Security Standards Council's PCI-DSS requirements govern any system that stores, processes, or transmits cardholder data, and specifically recommend tokenization as a way to shrink the standard's scope down to one small, isolated component [6] — the reasoning behind vault-service, where the Primary Account Number (PAN) is encrypted at rest, referenced elsewhere in the platform only through an opaque token, and kept out of logs by a dedicated masking filter.

Outside of distributed-systems literature specifically, applied AI-driven system engineering across other domains offers a related lesson about building pipelines that must stay reliable under real-time constraints. Work on unsupervised recurrent optical-flow estimation for multi-frame processing shows how a latency-sensitive pipeline can be structured to remain consistent under continuous, streaming input [9] — a concern that maps closely onto authorising a payment within a few hundred milliseconds. In a similar vein, CNN-based classification pipelines built for tasks such as skin-tone detection and personalised recommendation demonstrate how a structured feature-extraction stage can feed a downstream decision stage as its own isolated, independently testable component [7], which mirrors the separation PayNexus enforces between card tokenization and payment authorisation.

A. Drawbacks of Existing Approaches

- Commercial gateways such as Razorpay, Stripe, PayU, and Cashfree keep their internal engineering closed, leaving a student with no visibility into how idempotency is actually guaranteed, how a failed webhook gets retried, or how a day's transactions are reconciled into a settlement.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, August 2026 Impact Factor: 8.2

- Most academic or portfolio-style payment projects simply call a sandbox payment API from within a single monolithic application, so idempotency handling, webhook reliability, and a settlement layer are never actually built.
- Card data handling in student projects is rarely PCI-aware — the PAN is often passed straight through the main application instead of being tokenized inside a dedicated, isolated vault component.
- None of these smaller projects model several merchants sharing one deployment, so there is nothing to show how a gateway actually keeps one tenant's keys, data, and traffic separated from another's.
- Papers describing individual patterns — Sagas, the transactional outbox, PCI tokenization — tend to study each in isolation rather than combining them into one working, end-to-end payment platform that can be examined as a whole.

III. PROBLEM STATEMENT

Any merchant that wants to accept digital payments today runs into a fragmented set of instruments, each governed by its own protocol and its own failure behaviour. Card payments demand PAN handling, tokenization, and an authorisation-then-capture flow routed through an acquiring bank and a card network. UPI payments require integrating with the NPCI switch and supporting two distinct customer flows — app-to-app Intent and merchant-initiated Collect. Net banking means redirecting the customer's browser to their bank's own site and then safely resuming the transaction once they return. Wallet payments are typically closed-loop, direct integrations with a single wallet provider. Building and maintaining a separate, correct integration for each of these individually is expensive and easy to get wrong.

Beyond simply routing a request to the right rail, a gateway is also expected to hold guarantees far stricter than an ordinary web application would need to: a retried request must never turn into a duplicate order or a duplicate charge; a merchant's backend must reliably learn the instant a payment is captured or a refund processed, even if that backend is briefly unreachable; every rupee collected must eventually be reconciled and paid out to the correct merchant account, net of fees and GST, with a complete and auditable trail; and card numbers must never sit in the clear anywhere in the system, in line with the security expectations laid out by the Reserve Bank of India for digital payment platforms [12] — all while the platform keeps up high throughput and low latency even under partial failure.

A. Objectives

1) To study how Card, UPI, Net Banking, and Wallet payments work and define a unified order and payment lifecycle for all four.

2) To design a secure database and service architecture for order and payment management with tokenization and AES-256 encryption.

3) To implement a secure payment microservices system with idempotent APIs, reliable webhook handling, and an auditable settlement process.

4) To analyze the platform's performance, reliability, and security, and compare it with existing commercial gateways.

IV. PROPOSED METHODOLOGY

PayNexus is built as a multi-tenant, microservices-based payment gateway platform rather than a single deployable application, with the system decomposed along business boundaries into independently deployable services that communicate synchronously through an API Gateway and asynchronously through a Kafka event bus — a decomposition style consistent with established microservices design practice [5], [11]. This section walks through the system architecture, the database design, the payment methodology for each supported instrument, the mathematical model used for settlement, and the tools and technologies behind the implementation.

A. System Architecture

External clients — the Checkout SDK embedded on a merchant's site, the merchant's own backend, and the merchant Analytics Dashboard — reach the platform only through the API Gateway, which acts as the single point of

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, August 2026
Impact Factor: 8.2

authentication (JWT for the dashboard, API-key Basic Auth for server-to-server calls) and rate limiting. The gateway routes requests on to the business services (merchant-service, payment-service, operations-service, vault-service), while discovery-service, config-service, and common-lib handle service registration, centralised configuration, and shared utilities respectively. payment-service publishes domain events to Kafka through an outbox, which operations-service consumes to drive webhooks, settlement, and analytics without ever needing a synchronous call back into payment-service — the same kind of modular, stage-isolated design seen in other layered AI-driven detection systems, where keeping each processing stage independent limits how far a fault in one stage can propagate [4].

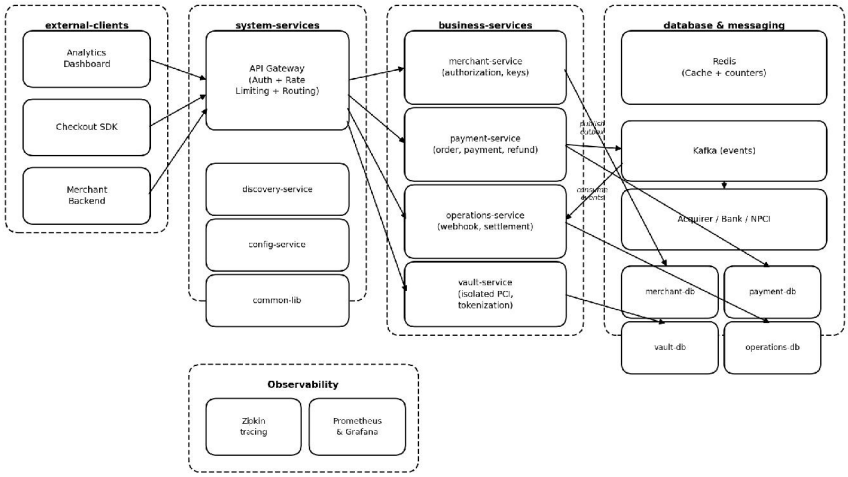

Figure 1: Microservices Architecture of PayNexus

Algorithm 1: Request Routing and Payment Processing

1) START
2) Client (Checkout SDK / Merchant Backend / Analytics Dashboard) sends a request to the API Gateway.
3) API Gateway authenticates the request (JWT for dashboard calls, API-Key + Basic Auth for server-to-server calls) and checks the per-merchant rate limit.
4) If authentication fails, return HTTP 401 and stop; if the rate limit is exceeded, return HTTP 429 and stop.
5) Gateway routes the request to the target service based on the request path.
6) If the request is a write operation, look up its X-Idempotent-Key in the Redis idempotency cache; if the key already exists, return the cached response and stop.
7) The target service executes its business logic and persists the resulting state to its own PostgreSQL database.
8) In the same local transaction, the service writes a domain event to its outbox table.
9) An outbox relay asynchronously publishes the event to Kafka.
10) operations-service consumes the event and triggers webhook delivery, settlement aggregation, or analytics updates as applicable.
11) The Gateway returns the synchronous response to the client.
12) STOP

B. Database Design

The PayNexus entity-relationship model places the merchant at the centre of the system, since virtually every other record ultimately belongs to one. Each merchant can hold multiple API keys for secure server-to-server communication, several app users with dashboard access, a webhook configuration describing where notifications should be sent, and a list of customers who transact with it.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, August 2026
Impact Factor: 8.2

Whenever a customer starts a transaction, an order is created under that merchant, carrying its own idempotency key and expiry time, following the same de-duplication logic used across the platform's write APIs [1]. Every payment is tied to exactly one order and moves through stages such as pending, successful, or failed; rather than overwriting the status in place, each change is appended to a payment transition log, producing a complete, auditable history that supports the same kind of end-to-end traceability the settlement and refund records rely on elsewhere in the platform. Where money needs to be returned, one or more refunds are linked back to the originating payment.

Sensitive card details are kept entirely separate from payment information: the raw card data is stored, encrypted, in the vault card table, and every other part of the system — customers and applications alike — interacts only with an opaque card token that safely references it. Webhook notifications are tracked in much the same auditable fashion: every delivery attempt is recorded, and an event that keeps failing past the retry limit is moved into a dead letter queue (DLQ) for later review and replay. Settlements, finally, combine many successful payments into a single payout record, with a settlement-payment table linking each settlement back to the individual payments it covers, and each settlement recording the gross amount, fees, GST, and net amount actually transferred to the merchant.

C. Payment Flow Methodology

The system is organised around one common order-and-payment lifecycle that every payment method passes through, branching only at the point where the customer actually authorises the debit.

Step 1 – Order Creation: checkout begins on the customer's side, which calls the merchant backend, which in turn calls PayNexus's POST /v1/orders API carrying an X-Idempotent-Key header. payment-service creates an order record — or returns the existing one if that key has already been used — starts a 15-minute expiry timer, and hands back an order_id to the Checkout SDK.

Step 2 – Method-Specific Payment Authorisation: once the customer picks a payment method, the flow branches accordingly —

- Card: vault-service tokenizes the card details on the spot, encrypting the PAN with AES-256 in line with PCI-DSS's tokenization guidance [6], so the PAN itself never reaches payment-service, which then sends an authorisation request carrying only the token to the acquirer bank and card network.
- UPI: for Intent, the checkout page opens a deep link straight into the customer's UPI app; for Collect, the customer enters a VPA and the request is routed through the NPCI switch to the issuing bank [8]; either way, the customer authorises with their UPI PIN.
- Net Banking: the customer is redirected to their issuing bank's site through a signed redirect URL, consistent with the digital-payment security expectations set out by the Reserve Bank of India [12]; once approved, the bank redirects back with a signed status that payment-service verifies before proceeding.
- Wallet: the customer confirms via a one-time password, and the wallet provider debits its own closed-loop ledger directly, producing a near-instant captured state.

Step 3 – State Transition and Event Publication: every transition a payment goes through (CREATED → AUTHORIZED → CAPTURED → SETTLED, or into a FAILED branch) is written to a payment transition log for auditability, and a domain event is published to Kafka through the transactional outbox pattern.

Step 4 – Webhook Notification: on consuming a payment.captured (or refund) event, operations-service assembles an HMAC-SHA256 signed webhook payload and posts it to the merchant's configured URL, following the retry-and-eventual-delivery approach documented in production webhook engineering practice [10], retrying up to seven times on failure before the payload is moved to a dead-letter queue.

Step 5 – Settlement: a nightly scheduled job groups every captured payment per merchant, computes the gateway fee and GST, works out the net settlement amount, creates an auditable settlement record linked to each contributing payment, and simulates the bank transfer.

Step 6 – Refunds and Analytics: a refund request against a captured payment is validated, driven through its own state machine, and processed asynchronously as a compensating action against the original capture rather than a synchronous

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, August 2026
Impact Factor: 8.2

reversal [3], while operations-service continuously aggregates captured, refunded, and settled amounts to power the merchant's real-time analytics dashboard.

D. Mathematical Model for Settlement

For a captured payment of gross amount A, the settlement engine works out the merchant's net payable amount using a fixed percentage gateway fee rate f together with the applicable GST rate g — 18% under current Indian tax law and consistent with the fee-disclosure expectations set out for digital payment operators [12] — as follows:

$$Fee = A \times f$$

$$GST = Fee \times g$$

$$Net\ Settlement\ Amount = A - Fee - GST$$

For a nightly settlement batch containing n captured payments A_1, A_2, …, A□ belonging to the same merchant, the gross settlement amount, total fees, total GST, and net payable amount are all obtained simply by summing the corresponding per-payment quantities across the batch. Card and UPI settlement additionally subtract a network/interchange fee and a transfer fee respectively, before the gateway fee and GST are applied, so as to reflect the actual fee structure each rail imposes.

E. Tools, Technologies, and Resources

Table 1 lists the primary technologies behind PayNexus's implementation and the role each plays; encryption of the vault's cardholder data specifically follows the AES specification published by NIST [13].

TABLE 1. TECHNOLOGIES, TOOLS, AND RESOURCES USED IN PAYNEXUS

Technology	Role in the System
Java & Spring Boot	Primary language and application framework for all backend microservices
Spring Cloud (Gateway, Eureka, Config)	API Gateway routing/auth/rate-limiting, service discovery, and centralised configuration
PostgreSQL	Per-service relational database for merchant, payment, operations, and vault data
Apache Kafka	Asynchronous event bus carrying payment/refund domain events via the outbox pattern
Redis	Caching, idempotency-key look-ups, and atomic rate-limit / usage counters
JWT (JSON Web Token)	Authentication for merchant Analytics Dashboard endpoints
API Key + HTTP Basic Auth	Server-to-server authentication for merchant integration endpoints
AES-256	Encryption of cardholder PAN data at rest inside vault-service
HMAC-SHA256	Signing of outbound webhook payloads for merchant-side verification
Docker	Containerisation of each microservice for consistent local and deployment environments
Zipkin	Distributed tracing of requests across microservices
Prometheus & Grafana	Metrics collection and dashboarding for latency, throughput, and availability
JUnit & Mockito	Unit and integration testing of business logic and state machines

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, August 2026
Impact Factor: 8.2

V. PROJECT PLAN AND TIMELINE

Table 2 outlines the planned execution schedule for PayNexus over five months, from requirement analysis in June through to final documentation and review in October.

TABLE 2. PROJECT PLAN AND TIMELINE (JUNE – OCTOBER)

Activity / Month	Jun '26	Jul '26	Aug '26	Sep '26	Oct '26
Requirement Analysis & Literature Survey	✓				
System & Database Design (ERD, Architecture, Algorithm)	✓	✓			
Core Services: merchant-service, payment-service		✓	✓		
Payment Methods: Card, UPI, Net Banking, Wallet		✓	✓		
vault-service (Tokenization, AES-256)			✓		
operations-service: Webhooks, Settlement, Analytics			✓	✓	
Observability: Zipkin, Prometheus/Grafana				✓	
Load Testing & Chaos Testing				✓	✓
Documentation & Final Review					✓

VI. EXPECTED OUTCOMES

1) Develop a secure multi-tenant payment gateway that enables merchant onboarding, API key generation, order creation, and payment processing through Card, UPI, Net Banking, and Wallet with auditable payment state management.

2) Ensure reliability and security by implementing idempotent APIs, PCI-DSS–aligned card tokenization with AES-256 encrypted vault storage, and a secure tokenize-and-charge payment flow.

3) Provide robust payment operations through reliable webhook delivery with retries and dead-letter replay, automated settlement generation with fee and GST calculation, and real-time merchant analytics dashboards, drawing on the same applied experience in building analytics-facing, pattern-recognition systems that informed the team's prior AI-driven work [2], [7].

4) Meet scalability and observability goals by deploying independently scalable microservices with containerization, distributed tracing, monitoring dashboards, and validating high throughput, low latency, and fault tolerance under load and failure scenarios.

VII. CONCLUSION

PayNexus set out to close a specific gap: commercial payment gateways are widely used but structurally opaque, and most academic projects that touch payments settle for a thin integration against a sandbox API rather than confronting the engineering problems a real gateway has to solve. This project instead builds those problems in directly — idempotent APIs that survive retries without double-charging, a transactional outbox feeding Kafka so that no domain

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, August 2026 Impact Factor: 8.2

event is silently dropped, a dedicated vault service that keeps cardholder data encrypted and tokenized in line with PCI-DSS practice, and a settlement engine that reconciles every captured payment down to a net, GST-adjusted payout. Structuring the system as independently deployable microservices, coordinated through an API Gateway and an asynchronous event bus, keeps each of these concerns — merchant identity, tokenization, payment processing, and operations — isolated in its own service rather than tangled together in one application.

At the current stage, the core order-to-payment path, merchant authentication, and card tokenization are functional across all four supported payment rails, while the cross-service settlement and webhook-delivery layer is still being hardened, and refunds, automated testing, and full containerized deployment remain open work. Even in its present form, the project demonstrates that the operational guarantees a production fintech platform depends on — correctness under failure, auditability, and security-by-design — can be built, understood, and reasoned about on a single, learnable codebase rather than treated as a black box owned by a commercial provider.

REFERENCES

[1] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017.

[2] D. S. Thosar, R. Thosar, P. Kharade, L. Sharma, R. Dalve and P. Nikam, "DRISHTI: GAN-CNN Based SAR Colorization with K-Means Land Cover Analysis," 2025 3rd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIHEI), Wardha, India, 2025, pp. 1-5, doi: 10.1109/IDICAIHEI65991.2025.11379205.

[3] H. Garcia-Molina and K. Salem, "Sagas," in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1987.

[4] D. Thosar, R. Thosar, L. Sharma, M. Chanore, P. Belkhede and S. Pohanerkar, "SmartSurveil: AI-based Intrusion and Suspicious Behavior Detection in CCTV," 2025 3rd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIHEI), Wardha, India, 2025, pp. 1-6, doi: 10.1109/IDICAIHEI65991.2025.11379088.

[5] C. Richardson, Microservices Patterns: With Examples in Java. Manning Publications, 2018.

[6] PCI Security Standards Council, Payment Card Industry Data Security Standard (PCI DSS), Requirements and Testing Procedures, latest version.

[7] L. Sharma, D. Thosar, B. Kumari, S. Nikamline, L. Aher and S. Sayyad, "DeepColor: A CNN-Based Skin Tone Detection and Personalized Color Recommendation System," 2025 3rd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIHEI), Wardha, India, 2025, pp. 1-7, doi: 10.1109/IDICAIHEI65991.2025.11377529.

[8] National Payments Corporation of India, Unified Payments Interface (UPI) Procedural Guidelines, NPCI.

[9] D. S. Thosar, R. D. Thosar, P. B. Dhamdhere, S. B. Ananda, U. D. Butkar and D. S. Dabhade, "Optical Flow Self-Teaching in Multi-Frame with Full-Image Warping via Unsupervised Recurrent All-Pairs Field Transform," 2024 2nd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIEI), Wardha, India, 2024, pp. 1-4, doi: 10.1109/IDICAIEI61867.2024.10842718.

[10] Stripe Engineering and Razorpay Engineering blogs, "Building reliable webhooks," public engineering write-ups on webhook signing, retries, and dead-letter handling.

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

[12] Reserve Bank of India, Master Direction on Digital Payment Security Controls, RBI/2020-21/74, 2021.

[13] National Institute of Standards and Technology, Advanced Encryption Standard (AES), FIPS PUB 197, 2001.