System Architecture and Technical Specifications: LoyalNest Shopify App

1. Overview

LoyalNest is a loyalty and rewards platform for Shopify merchants, competing with Smile.io, Yotpo, and LoyaltyLion. It targets small (100–1,000 customers, AOV $20–$50), medium (1,000–10,000 customers, AOV $50–$200), and Shopify Plus merchants (10,000+ customers, multi-store). The system uses a microservices architecture in an Nx monorepo for modularity, scalability, and independent deployments, supporting 5,000+ merchants and 50,000+ customers for Plus, with Black Friday surges (10,000 orders/hour). Built with NestJS (TypeScript) for APIs, Rust/Wasm for Shopify Functions, Vite + React for the frontend, PostgreSQL (JSONB, range partitioning), MongoDB (sharding), TimescaleDB (hypertables), Redis Cluster (Streams), Kafka, and Loki + Grafana, it is deployed on a VPS (Ubuntu, 32GB RAM, 8 vCPUs) with Docker Compose and Nginx. Must Have features for TVP (Phase 3, February 2026) include points (purchases, signups, reviews, birthdays), SMS/email referrals, RFM analytics, Shopify POS (offline mode), checkout extensions, GDPR/CCPA request form, referral status with progress bar, notification templates, customer import, campaign discounts, rate limit monitoring, usage thresholds, upgrade nudges, gamified onboarding, user/role management, and merchant feedback. The system ensures GDPR/CCPA compliance, Shopify App Store requirements, multilingual support (English, Spanish, French, German, Portuguese, Japanese, Arabic with RTL), and disaster recovery with Backblaze B2.

2. System Objectives

- **Scalability**: Support 5,000+ merchants, with Plus merchants handling 50,000+ customers and 10,000 orders/hour (Black Friday).
- **Modularity**: Utilize 15 microservices (API Gateway, Core, Auth, Points, Referrals, Users, Roles, RFM, Event Tracking, AdminCore, AdminFeatures, Campaign, Gamification, Frontend, Products) in an Nx monorepo.
- **Shopify Compliance**: Adhere to Shopify APIs (2025-01), OAuth, webhooks (`orders/create`, GDPR), POS offline mode, Checkout UI Extensions, Flow templates (Phase 5).
- **GDPR/CCPA Compliance**: Encrypt PII (`email`, `credentials`) with AES-256 via `pgcrypto`, handle GDPR webhooks (`customers/data_request`, `customers/redact`) with retries, enforce 90-day retention (`gdpr_requests.retention_expires_at`).
- **Performance**: Achieve API responses <200ms using Redis Cluster caching, PostgreSQL range partitioning, MongoDB sharding, TimescaleDB hypertables, and circuit breakers; manage Shopify API rate limits (2 req/s standard, 40 req/s Plus).
- **Developer Efficiency**: Leverage AI tools (Grok, Copilot, Cursor) for a solo developer (30–40% efficiency), with in-house UI/UX and QA, using Nx monorepo, Docker Compose, and `dev.sh` for mock data, RFM simulation, rate limit simulation, and audit log replay.
- **Reliability**: Implement disaster recovery with `pg_dump`, MongoDB snapshots, Redis snapshotting, and Backblaze B2 backups (90-day retention, RTO: 4 hours, RPO: 1 hour), plus Loki + Grafana logging and Chaos Mesh testing.
- **Merchant Engagement**: Support gamified onboarding, merchant referral program, Slack community (“LoyalNest Collective”), and Typeform feedback.

3. System Architecture

The system uses a microservices architecture, orchestrated with Docker Compose on a VPS (Ubuntu, Nginx with gRPC proxy). It employs REST APIs for UI-facing endpoints, gRPC for inter-service communication, Kafka for async events, WebSocket for real-time updates, and PostgreSQL, MongoDB, TimescaleDB, Redis Cluster, and Loki for storage, caching, and logging.

3.1 Microservices

The system comprises 15 microservices, built with NestJS (TypeScript) and Rust/Wasm for Shopify Functions, managed in an Nx monorepo:

1. **API Gateway**:
    - **Purpose**: Entry point for external API requests, handling routing, load balancing, rate limiting, and authentication verification.
    - **Endpoints**: `/v1/api/*` (proxies to Core, Points, Referrals, Users, Roles, RFM, etc.), `/frontend/*`, `/admin/*`.
    - **Tech**: NestJS, `@nestjs/microservices`, Nginx (reverse proxy, rate limiting), Redis Cluster (`api_rate_limit:{merchant_id}`), OpenAPI/Swagger.
    - **Database Schema**: None (uses Redis for rate limiting).
    - **Interactions**: Validates JWTs with Auth, enforces Shopify API rate limits, routes via gRPC/REST, caches in Redis, logs to Loki.
    - **Inter-Service Communication**:
        - **Consumes**: None.
        - **Produces**: `webhook.received` (Kafka, to Points, Referrals, RFM).
        - **Calls**: `/auth.v1/ValidateToken`, `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: External clients, Frontend, routes to all services.
2. **Core Service**:
    - **Purpose**: Centralizes business logic, configuration management, and plan enforcement (freemium-to-Plus funnel).
    - **Endpoints**:
        - REST: `/v1/api/core/settings`, `/v1/api/core/usage`, `/v1/api/core/plan`
        - gRPC: `/core.v1/CreateCustomer`, `/core.v1/UpdateSettings`
    - **Tech**: NestJS, PostgreSQL (`program_settings`, `customer_import_logs`), Redis Cluster (`config:{merchant_id}`, `usage:{merchant_id}`), Kafka, `socket.io`.
    - **Database Schema**:
        - `program_settings`:
            - `merchant_id`: UUID, primary key
            - `settings`: JSONB (loyalty rules)
            - `created_at`: TIMESTAMP, indexed
            - `updated_at`: TIMESTAMP
        - `customer_import_logs`:
            - `import_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `status`: ENUM(pending, completed, failed)
            - `log_details`: JSONB
            - `created_at`: TIMESTAMP, indexed

- **Interactions**: Provides configuration to all services, coordinates upgrade nudges via WebSocket, logs imports to AdminCore.
- **Inter-Service Communication**:
  - **Consumes**: `rfm.updated` (Kafka, RFM), `gdpr_request.created` (Kafka, AdminCore), `user.created`, `user.updated` (Kafka, Users).
  - **Produces**: `customer.created`, `customer.updated`, `plan_limit_warning` (Kafka, to Points, Referrals, RFM, AdminFeatures).
  - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/auth.v1/ValidateMerchant` (gRPC, Auth).
  - **Called By**: Points, Referrals, RFM, Frontend, AdminFeatures.

3. **Auth Service**:

- **Purpose**: Manages Shopify OAuth, JWT authentication (15-minute expiry, revocation in Redis), MFA via Auth0, and RBAC integration.
- **Endpoints**:
  - REST: `/v1/api/auth/login`, `/v1/api/auth/refresh`, `/v1/api/auth/mfa`, `/admin/v1/auth/revoke`
  - gRPC: `/auth.v1/ValidateToken`, `/auth.v1/ValidateMerchant`
- **Tech**: NestJS, `@shopify/shopify-app-express`, Redis Cluster (`jwt:{merchant_id}`, `revoked_jti:{token_id}`), PostgreSQL (`merchants`, `admin_users`, `admin_sessions`, `impersonation_sessions`).
- **Database Schema**:
  - `merchants`:
    - `merchant_id`: UUID, primary key
    - `shop_domain`: VARCHAR, unique, indexed
    - `access_token`: TEXT (AES-256 encrypted)
    - `created_at`: TIMESTAMP, indexed
  - `admin_users`:
    - `user_id`: UUID, primary key
    - `email`: TEXT (AES-256 encrypted), indexed
    - `merchant_id`: UUID, indexed
    - `created_at`: TIMESTAMP
  - `admin_sessions`:
    - `session_id`: UUID, primary key
    - `user_id`: UUID, indexed
    - `jwt_token`: TEXT
    - `expires_at`: TIMESTAMP, indexed
  - `impersonation_sessions`:
    - `session_id`: UUID, primary key
    - `admin_id`: UUID, indexed
    - `merchant_id`: UUID, indexed
    - `expires_at`: TIMESTAMP
- **Interactions**: Validates tokens, integrates with Roles and Users for RBAC and user validation, supports MFA for Plus merchants.
- **Inter-Service Communication**:
  - **Consumes**: `user.created`, `role.assigned` (Kafka, Users, Roles).
  - **Produces**: `merchant.created`, `auth.revoked` (Kafka, to Core, AdminCore).

        - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/roles.v1/GetRole` (gRPC, Roles).
        - **Called By**: API Gateway, Core, Points, Referrals, RFM, Frontend, AdminCore, AdminFeatures.

4. **Points Service**:
    - **Purpose**: Manages points earning/redemption, Shopify POS (offline mode), checkout extensions, campaign discounts.
    - **Endpoints**:
        - REST: `/v1/api/points/earn`, `/v1/api/points/redeem`, `/v1/api/points/adjust`, `/v1/api/rewards`, `/v1/api/points/sync`
        - gRPC: `/points.v1/GetPointsBalance`
        - WebSocket: `/api/points/stream`
    - **Tech**: NestJS, Rust/Wasm (Shopify Functions), MongoDB (`points_transactions`, `reward_redemptions`, `pos_offline_queue`), Redis Cluster (`points:customer:{id}`), `socket.io`.
    - **Database Schema**:
        - `points_transactions` (MongoDB, sharded):
            - `_id`: ObjectId, primary key
            - `customer_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `points`: NUMBER
            - `type`: ENUM(earn, redeem, adjust)
            - `created_at`: TIMESTAMP, indexed
        - `reward_redemptions` (MongoDB):
            - `_id`: ObjectId, primary key
            - `customer_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `reward_id`: UUID
            - `status`: ENUM(pending, completed)
            - `created_at`: TIMESTAMP
        - `pos_offline_queue` (MongoDB):
            - `_id`: ObjectId, primary key
            - `customer_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `points`: NUMBER
            - `created_at`: TIMESTAMP
    - **Interactions**: Processes `orders/create` webhooks, syncs POS data, applies discounts, streams updates.
    - **Inter-Service Communication**:
        - **Consumes**: `webhook.received` (Kafka, API Gateway), `user.created` (Kafka, Users).
        - **Produces**: `points.earned`, `points.redeemed` (Kafka, to RFM, AdminCore).
        - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/auth.v1/ValidateMerchant` (gRPC, Auth), `/core.v1/GetSettings` (gRPC, Core).
        - **Called By**: API Gateway, Frontend, AdminFeatures.

5. **Referrals Service**:

- **Purpose**: Manages SMS/email referrals, referral status with progress bar, error handling with AWS SES fallback.
- **Endpoints**:
  - REST: `/v1/api/referrals/create`, `/v1/api/referrals/complete`, `/v1/api/referrals/status`, `/v1/api/referrals/progress`
  - gRPC: `/referrals.v1/GetReferralStatus`
- **Tech**: NestJS, Klaviyo/Postscript (SMS/email, 5s timeout, 3 retries), AWS SES (fallback), Bull queues, PostgreSQL (`referrals`), Redis Streams (`referral:{code}`, `referral:status:{id}`).
- **Database Schema**:
  - `referrals`:
    - `referral_id`: UUID, primary key
    - `merchant_id`: UUID, indexed
    - `referrer_id`: UUID, indexed
    - `referral_link_id`: UUID, indexed
    - `status`: ENUM(pending, completed, expired)
    - `created_at`: TIMESTAMP, indexed
    - `updated_at`: TIMESTAMP
- **Interactions**: Generates referral links, sends notifications via Klaviyo/Postscript/AWS SES, tracks 7%+ conversion, logs errors to PostHog.
- **Inter-Service Communication**:
  - **Consumes**: `webhook.received` (Kafka, API Gateway), `user.created` (Kafka, Users).
  - **Produces**: `referral.created`, `referral.completed` (Kafka, to RFM, AdminCore).
  - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/auth.v1/ValidateMerchant` (gRPC, Auth), `/core.v1/GetSettings` (gRPC, Core).
  - **Called By**: API Gateway, Frontend, AdminFeatures.

6. **Users-Service**:

- **Purpose**: Manages merchant and customer accounts with PII encryption and audit logging.
- **Endpoints**:
  - REST: `/v1/api/users/create`, `/v1/api/users/update`, `/v1/api/users/get`
  - gRPC: `/users.v1/GetUser`, `/users.v1/UpdateUser`
- **Tech**: NestJS, PostgreSQL (`users`, `audit_logs`), Redis Cluster (`user:{user_id}`), Kafka.
- **Database Schema**:
  - `users`:
    - `user_id`: UUID, primary key
    - `email`: TEXT (AES-256 encrypted), indexed
    - `merchant_id`: UUID, indexed
    - `role_id`: UUID, indexed
    - `created_at`: TIMESTAMP, indexed
    - `updated_at`: TIMESTAMP
    - Trigger: Logs to `audit_logs`
  - `audit_logs` (shared with AdminCore, Roles):
    - `log_id`: UUID, primary key
    - `merchant_id`: UUID, indexed
    - `user_id`: UUID, indexed
    - `action`: ENUM(create, update, delete)

        - `details`: JSONB
        - `created_at`: TIMESTAMP, indexed
    - **Interactions**: Manages user accounts, logs changes for GDPR/CCPA, caches in Redis.
    - **Inter-Service Communication**:
        - **Consumes**: `merchant.created` (Kafka, Auth).
        - **Produces**: `user.created`, `user.updated` (Kafka, to Core, Points, Referrals, RFM, AdminCore).
        - **Calls**: `/roles.v1/GetRole` (gRPC, Roles), `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: Core, Points, Referrals, RFM, Frontend, AdminCore, AdminFeatures.

7. **Roles-Service**:

    - **Purpose**: Manages RBAC with role-based permissions.
    - **Endpoints**:
        - REST: `/v1/api/roles/create`, `/v1/api/roles/update`, `/v1/api/roles/get`
        - gRPC: `/roles.v1/GetRole`, `/roles.v1/UpdateRole`
    - **Tech**: NestJS, PostgreSQL (`roles`, `audit_logs`), Redis Cluster (`role:{role_id}`), Kafka.
    - **Database Schema**:
        - `roles`:
            - `role_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `permissions`: JSONB (e.g., `["admin:full", "admin:analytics"]`)
            - `created_at`: TIMESTAMP, indexed
            - `updated_at`: TIMESTAMP
            - Trigger: Logs to `audit_logs`
        - `audit_logs`: Same as Users-Service.
    - **Interactions**: Defines roles, supports RBAC for Auth, logs changes.
    - **Inter-Service Communication**:
        - **Consumes**: `merchant.created` (Kafka, Auth).
        - **Produces**: `role.assigned`, `role.updated` (Kafka, to Auth, AdminCore).
        - **Calls**: `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: Auth, AdminCore, AdminFeatures, Frontend.

8. **RFM-Service**:

    - **Purpose**: Provides RFM analytics with time-weighted recency, lifecycle stages, and visualizations.
    - **Endpoints**:
        - REST: `/v1/api/rfm/segments`, `/v1/api/rfm/segments/preview`, `/v1/api/rfm/nudges`
        - gRPC: `/rfm.v1/GetSegments`, `/rfm.v1/GetCustomerRFM`
        - WebSocket: `/api/rfm/visualizations`
    - **Tech**: NestJS, Rust/Wasm, TimescaleDB (`rfm_segment_deltas`, `rfm_segment_counts`, `rfm_score_history`, `customer_segments`), Redis Streams (`rfm:customer:{id}`, `rfm:preview:{merchant_id}`).
    - **Database Schema**:
        - `rfm_segment_deltas` (hypertable):
            - `customer_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `recency`: NUMBER

        - `frequency`: NUMBER
        - `monetary`: NUMBER
        - `created_at`: TIMESTAMP, indexed
    - `rfm_segment_counts` (materialized view):
        - `merchant_id`: UUID, indexed
        - `segment`: ENUM(high_value, at_risk, new)
        - `count`: NUMBER
        - `created_at`: TIMESTAMP, indexed
    - `rfm_score_history`:
        - `history_id`: UUID, primary key
        - `customer_id`: UUID, indexed
        - `merchant_id`: UUID, indexed
        - `rfm_score`: JSONB
        - `created_at`: TIMESTAMP, indexed
    - `customer_segments`:
        - `customer_id`: UUID, indexed
        - `merchant_id`: UUID, indexed
        - `segment`: ENUM(high_value, at_risk, new)
        - `created_at`: TIMESTAMP, indexed
        - Trigger: Updates on `orders/create`, `points.earned`
- **Interactions**: Calculates RFM scores, refreshes daily (`0 1 * * *`) and on `orders/create`, caches in Redis Streams, streams visualizations.
- **Inter-Service Communication**:
    - **Consumes**: `points.earned`, `referral.completed`, `user.created`, `user.updated` (Kafka, Points, Referrals, Users).
    - **Produces**: `rfm.updated` (Kafka, to Core, AdminFeatures, Frontend).
    - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/auth.v1/ValidateMerchant` (gRPC, Auth).
    - **Called By**: Core, Frontend, AdminFeatures.

9. **Event Tracking Service**:

- **Purpose**: Tracks feature usage and events for analytics and merchant engagement.
- **Endpoints**:
    - REST: `/v1/api/events`
    - gRPC: `/event_tracking.v1/CreateTask`
- **Tech**: NestJS, PostHog, PostgreSQL (`queue_tasks`), Kafka.
- **Database Schema**:
    - `queue_tasks`:
        - `task_id`: UUID, primary key
        - `merchant_id`: UUID, indexed
        - `event_type`: VARCHAR (e.g., `points_earned`, `user.created`)
        - `status`: ENUM(pending, completed, failed)
        - `payload`: JSONB
        - `created_at`: TIMESTAMP, indexed
- **Interactions**: Captures events, sends to PostHog via Kafka, queues tasks.
- **Inter-Service Communication**:

- **Consumes**: `points.earned`, `referral.created`, `user.created`, `role.assigned`, `rfm.updated` (Kafka, Points, Referrals, Users, Roles, RFM).
- **Produces**: `task.created`, `task.completed` (Kafka, to AdminCore).
- **Calls**: None.
- **Called By**: API Gateway, AdminCore.

10. **AdminCore Service**:

    - **Purpose**: Manages merchant accounts, GDPR requests, and audit logs.
    - **Endpoints**:
      - REST: `/admin/merchants`, `/admin/logs`
      - gRPC: `/admin_core.v1/GetMerchants`, `/admin_core.v1/GetAuditLogs`, `/admin_core.v1/HandleGDPRRequest`
    - **Tech**: NestJS, PostgreSQL (`gdpr_requests`, `audit_logs`, `webhook_idempotency_keys`), Redis Streams (`audit_logs:{merchant_id}`), Kafka, `socket.io`, Nginx (IP allowlisting, HMAC).
    - **Database Schema**:
      - `gdpr_requests`:
        - `request_id`: UUID, primary key
        - `merchant_id`: UUID, indexed
        - `user_id`: UUID, indexed
        - `status`: ENUM(pending, completed)
        - `retention_expires_at`: TIMESTAMP, indexed
      - `audit_logs` (shared with Users, Roles):
        - Same as Users-Service.
      - `webhook_idempotency_keys`:
        - `key_id`: UUID, primary key
        - `merchant_id`: UUID, indexed
        - `webhook_id`: UUID, indexed
        - `created_at`: TIMESTAMP, indexed
    - **Interactions**: Handles GDPR webhooks with retries, logs audits, streams logs via WebSocket, integrates Typeform feedback.
    - **Inter-Service Communication**:
      - **Consumes**: `user.created`, `role.assigned`, `points.earned`, `referral.created` (Kafka, Users, Roles, Points, Referrals).
      - **Produces**: `gdpr_request.created` (Kafka, to Core).
      - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/roles.v1/GetRole` (gRPC, Roles), `/auth.v1/ValidateMerchant` (gRPC, Auth).
      - **Called By**: AdminFeatures, Frontend.

11. **AdminFeatures Service**:

    - **Purpose**: Manages points adjustments, referrals, RFM segments, customer imports, notification templates, rate limits, integration health, onboarding, multi-currency settings, feedback.
    - **Endpoints**:
      - REST: `/admin/points/adjust`, `/admin/referrals`, `/admin/rfm-segments`, `/admin/rfm/export`, `/admin/notifications/template`, `/admin/rate-limits`, `/admin/customers/import`, `/admin/settings/currency`, `/admin/integrations/square/sync`, `/admin/v1/feedback`

        - gRPC: `/admin_features.v1/UpdateNotificationTemplate`, `/admin_features.v1/GetRateLimits`, `/admin_features.v1/ImportCustomers`, `/admin_features.v1/SyncSquarePOS`
        - WebSocket: `/admin/v1/setup/stream`
    - **Tech**: NestJS, PostgreSQL (`email_templates`, `integrations`, `setup_tasks`, `merchant_settings`), Redis Streams (`setup_tasks:{merchant_id}`), Bull queues, `socket.io`, Nginx (IP allowlisting, HMAC), Klaviyo/Postscript/AWS SES.
    - **Database Schema**:
        - `email_templates`:
            - `template_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `template_data`: JSONB
            - `created_at`: TIMESTAMP, indexed
        - `integrations`:
            - `integration_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `credentials`: TEXT (AES-256 encrypted)
            - `type`: ENUM(shopify, square, klaviyo)
            - `created_at`: TIMESTAMP
        - `setup_tasks`:
            - `task_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `status`: ENUM(pending, completed)
            - `created_at`: TIMESTAMP, indexed
        - `merchant_settings`:
            - `merchant_id`: UUID, primary key
            - `currency`: VARCHAR
            - `settings`: JSONB
            - `created_at`: TIMESTAMP
    - **Interactions**: Manages rate limits, async imports/exports, notification templates via Klaviyo/Postscript/AWS SES, onboarding progress, integration health.
    - **Inter-Service Communication**:
        - **Consumes**: `user.created`, `rfm.updated` (Kafka, Users, RFM).
        - **Produces**: `email_event.created` (Kafka, to Event Tracking).
        - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/roles.v1/GetRole` (gRPC, Roles), `/rfm.v1/GetSegments` (gRPC, RFM), `/points.v1/GetPointsBalance` (gRPC, Points), `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: Frontend, AdminCore.

12. **Campaign Service**:

    - **Purpose**: Manages Shopify Discounts API campaigns.
    - **Endpoints**:
        - REST: `/api/campaigns`, `/api/campaigns/{id}`
        - gRPC: `/campaign.v1/CreateCampaign`, `/campaign.v1/GetCampaign`
    - **Tech**: NestJS, Rust/Wasm, PostgreSQL (`campaigns`), Redis Cluster (`campaign:{campaign_id}`).
    - **Database Schema**:

        - `campaigns`:
            - `campaign_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `details`: JSONB (discount rules)
            - `created_at`: TIMESTAMP, indexed
            - `updated_at`: TIMESTAMP
    - **Interactions**: Creates/applies campaign discounts, caches in Redis.
    - **Inter-Service Communication**:
        - **Consumes**: `user.created` (Kafka, Users).
        - **Produces**: `campaign.created` (Kafka, to AdminCore, Frontend).
        - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: Frontend, AdminFeatures.

13. **Gamification Service**:

    - **Purpose**: Manages badge awards and leaderboards (Phase 6).
    - **Endpoints**:
        - REST: `/api/gamification/badges`, `/api/gamification/leaderboard`
        - gRPC: `/gamification.v1/AwardBadge`, `/gamification.v1/GetLeaderboard`
    - **Tech**: NestJS, PostgreSQL (`customer_badges`, `leaderboard_rankings`), Redis Cluster (`badge:{customer_id}:{badge_id}`, `leaderboard:{merchant_id}:{page}`).
    - **Database Schema**:
        - `customer_badges`:
            - `badge_id`: UUID, primary key
            - `customer_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `badge_type`: VARCHAR
            - `created_at`: TIMESTAMP, indexed
        - `leaderboard_rankings`:
            - `ranking_id`: UUID, primary key
            - `customer_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `score`: NUMBER
            - `created_at`: TIMESTAMP, indexed
    - **Interactions**: Awards badges, ranks customers, caches in Redis.
    - **Inter-Service Communication**:
        - **Consumes**: `user.created` (Kafka, Users).
        - **Produces**: `badge.awarded` (Kafka, to AdminCore, Frontend).
        - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: Frontend, AdminFeatures.

14. **Frontend Service**:

    - **Purpose**: Hosts merchant dashboard, customer widget, and admin module as a single-page app, ensuring Shopify compliance and accessibility.
    - **Endpoints**:
        - REST: `/`, `/customer`, `/admin`

        - WebSocket: `/admin/v1/setup/stream`, `/api/points/stream`, `/api/rfm/visualizations`
    - **Tech**: Vite + React, Polaris, Tailwind CSS, App Bridge, `i18next` (multilingual: en, es, fr, de, pt, ja, ar with RTL, fallback: en).
    - **Database Schema**: None (client-side, data via API Gateway).
    - **Interactions**: Displays dashboards (`UsersPage.tsx`, `RolesPage.tsx`, `AnalyticsPage.tsx`), supports i18n, communicates via API Gateway.
    - **Inter-Service Communication**:
        - **Consumes**: `points.earned`, `referral.created`, `user.created`, `role.assigned`, `rfm.updated` (Kafka, via WebSocket).
        - **Produces**: None.
        - **Calls**: `/points.v1/GetPointsBalance`, `/referrals.v1/GetReferralStatus`, `/rfm.v1/GetSegments`, `/users.v1/GetUser`, `/roles.v1/GetRole`, `/admin_core.v1/GetAuditLogs`, `/admin_features.v1/GetRateLimits`, `/core.v1/GetSettings` (gRPC, via API Gateway).
        - **Called By**: None (client-facing).

15. **Products Service**:
    - **Purpose**: Manages product-related data, product-level RFM analytics (Phase 6), and Shopify Product API integration.
    - **Endpoints**:
        - REST: `/v1/api/products`, `/v1/api/products/rfm`, `/v1/api/products/campaigns`
        - gRPC: `/products.v1/GetProductRFM`, `/products.v1/ApplyCampaign`
    - **Tech**: NestJS, PostgreSQL (`products`, `product_rfm_scores`), Redis Cluster (`product:{product_id}:rfm`), Shopify Product API.
    - **Database Schema**:
        - `products`:
            - `product_id`: UUID, primary key
            - `merchant_id`: UUID, indexed
            - `name`: VARCHAR
            - `created_at`: TIMESTAMP, indexed
        - `product_rfm_scores`:
            - `score_id`: UUID, primary key
            - `product_id`: UUID, indexed
            - `merchant_id`: UUID, indexed
            - `rfm_score`: JSONB
            - `created_at`: TIMESTAMP, indexed
    - **Interactions**: Fetches product data, calculates product-level RFM, integrates with Campaign.
    - **Inter-Service Communication**:
        - **Consumes**: `user.created`, `rfm.updated` (Kafka, Users, RFM).
        - **Produces**: `product_rfm.updated` (Kafka, to AdminCore, Frontend).
        - **Calls**: `/users.v1/GetUser` (gRPC, Users), `/rfm.v1/GetSegments` (gRPC, RFM), `/auth.v1/ValidateMerchant` (gRPC, Auth).
        - **Called By**: Frontend, AdminFeatures, Campaign.

3.2 Data Storage

- **PostgreSQL**:
  - Schema: `loyalnest_full_schema.sql`.
  - Key Tables: `merchants`, `users` (email, merchant_id, role_id, AES-256 encrypted), `roles` (permissions: JSONB), `points_transactions`, `referrals`, `reward_redemptions`, `program_settings`, `gdpr_requests`, `audit_logs`, `email_templates`, `integrations`, `setup_tasks`, `merchant_settings`, `campaigns`, `customer_badges`, `leaderboard_rankings`, `products`, `product_rfm_scores`, `webhook_idempotency_keys`.
  - Indexes: `users(email, merchant_id, role_id)`, `roles(merchant_id)`, `points_transactions(customer_id)`, `referrals(merchant_id, referral_link_id)`, `gdpr_requests(retention_expires_at)`, `audit_logs(merchant_id)`, `products(product_id)`.
  - Partitioning: Range partitioning on `created_at` for `points_transactions`, `referrals`, `audit_logs`.
  - Encryption: PII (`users.email`, `integrations.credentials`) with AES-256 via `pgcrypto`, quarterly key rotation via AWS KMS.
- **MongoDB** (Points Service):
  - Collections: `points_transactions`, `reward_redemptions`, `pos_offline_queue`.
  - Sharding: On `merchant_id` for scalability.
- **TimescaleDB** (RFM Service):
  - Tables: `rfm_segment_deltas`, `rfm_segment_counts` (materialized view), `rfm_score_history`, `customer_segments` (hypertables on `created_at`).
  - Triggers: Update `customer_segments` on `orders/create`, `points.earned`.
  - Refresh: Daily (`0 1 * * *`) and real-time on `orders/create`.
- **Redis Cluster**:
  - Caches: Points balances (`points:customer:{id}`), referral codes (`referral:{code}`), rate limits (`shopify_api_rate_limit:{merchant_id}`, `api_rate_limit:{merchant_id}`), user data (`user:{user_id}`), roles (`role:{role_id}`), RFM previews (`rfm:preview:{merchant_id}`), badges, leaderboards, setup tasks, product RFM (`product:{product_id}:rfm`).
  - Streams: Logs (`logs:{merchant_id}`), queue monitoring, RFM caching.
  - Dead-letter queue for GDPR webhook retries (3 retries).
- **Backblaze B2**:
  - Stores daily backups (`pg_dump`, MongoDB/Redis snapshots) with 90-day retention, validated weekly via `restore.sh`.

3.3 Event Processing

- **Kafka**:
  - Events: `points.earned`, `points.redeemed`, `referral.created`, `referral.completed`, `user.created`, `user.updated`, `role.assigned`, `role.updated`, `rfm.updated`, `gdpr_request.created`, `campaign.created`, `badge.awarded`, `plan_limit_warning`, `product_rfm.updated`, `email_event.created`.
  - Ensures decoupling and reliable delivery with retries.
- **PostHog**:
  - Tracks: `points_earned`, `referral_clicked`, `user.created`, `role.assigned`, `rfm.updated`, `gdpr_request_submitted`, `template_edited`, `customer_import_initiated`, `rate_limit_viewed`, `product_rfm_viewed`.

3.4 Integrations

- **Shopify**: APIs (Orders, Customers, Discounts, Products), webhooks (`orders/create`, GDPR), POS offline mode, Checkout UI Extensions, Flow templates (Phase 5).
- **Klaviyo/Postscript/AWS SES**: SMS/email referrals, notification templates, A/B testing for RFM nudges.
- **Yotpo/Judge.me**: Points for reviews.
- **Square**: POS integration with health checks and manual sync.

3.5 Deployment

- **VPS**: Ubuntu, Docker Compose, Nginx (reverse proxy, gRPC proxy, frontend assets, IP allowlisting, HMAC), `nestjs-circuit-breaker`.
- **CI/CD**: GitHub Actions with change detection, Jest/Cypress/k6 tests, Lighthouse CI, OWASP ZAP, Chaos Mesh (nightly resilience tests), daily backups, Slack alerts.
- **Local Development**: `dev.sh` starts Docker containers, seeds mock data (Faker), simulates RFM scores, rate limits, audit log replay.
- **Disaster Recovery**: `pg_dump`, MongoDB/Redis snapshots, Backblaze B2 backups (90-day retention, RTO: 4 hours, RPO: 1 hour).
- **Feature Flags**: `feature-flags.json` for rollouts (e.g., product-level RFM), canary routing via Nginx.

3.6 Monitoring and Logging

- **Loki + Grafana**: Structured logging, monitors API latency (<200ms), rate limits (alerts at 80%), integration health, SLOs.
- **PostHog**: Tracks feature adoption (80%+ RFM wizard completion, 7%+ SMS referral conversion, 95% user setup success, 100% role assignment accuracy), merchant engagement, campaign performance.

3.7 Feature Prioritization Matrix

Feature	Priority	Phase	Merchant Value	Complexity
Points (purchases, signups, reviews, birthdays)	Must Have	3	High (20%+ redemption rate)	Medium
SMS/Email Referrals	Must Have	3	High (7%+ SMS conversion)	Medium
RFM Analytics	Must Have	3	High (10%+ repeat purchase uplift)	High
User Management	Must Have	3	High (95% setup success)	Medium
Role Management (RBAC)	Must Have	3	High (100% role assignment accuracy)	Medium
Shopify POS (offline mode)	Must Have	3	Medium (POS adoption)	High
Checkout Extensions	Must Have	3	High (85%+ adoption)	Medium

Feature	Priority	Phase	Merchant Value	Complexity
GDPR Request Form	Must Have	3	Medium (50%+ usage)	Low
Referral Status (Progress Bar)	Must Have	3	Medium (60%+ engagement)	Low
Notification Templates	Must Have	3	High (80%+ usage)	Medium
Customer Import	Must Have	3	High (90%+ success rate)	Medium
Campaign Discounts	Must Have	3	High (10%+ redemption)	Medium
Rate Limit Monitoring	Must Have	3	Medium (operational reliability)	Low
Usage Thresholds	Must Have	3	Medium (upgrade funnel)	Low
Upgrade Nudges	Must Have	3	Medium (freemium-to-Plus conversion)	Low
Gamified Onboarding	Must Have	3	High (80%+ completion)	Medium
Product-Level RFM	Should Have	4–5	Medium (targeted campaigns)	High
VIP Tiers	Should Have	4–5	Medium (tier engagement)	Medium
Multi-Store Point Sharing	Should Have	4–5	High (Shopify Plus adoption)	High
Shopify Flow Templates	Should Have	4–5	Medium (automation adoption)	Medium

4. Data Flow

1. **Merchant Authentication**: Merchant logs in via Shopify OAuth, API Gateway routes to Auth, which issues JWT and validates with Roles/Users, Core enforces plan limits.
2. **Points and Rewards**: `orders/create` triggers Points via API Gateway, Users checks customer data, Products checks campaign eligibility, streams updates.
3. **Referrals**: Referrals generates links via API Gateway, sends notifications via Klaviyo/Postscript/AWS SES, tracks status.
4. **RFM Analytics**: RFM calculates scores via API Gateway, uses Users for customer data, Products adds product-level insights.
5. **GDPR Compliance**: AdminCore handles GDPR webhooks via API Gateway, encrypts PII with Users.

6. **Admin Operations**: AdminFeatures manages imports, templates, and Square sync via API Gateway, Frontend displays updates with Users/Roles integration.

5. API Endpoints

- **GET /api/customer/points**: Retrieves points balance (API Gateway → Points).
- **POST /api/redeem**: Redeems points (API Gateway → Points).
- **POST /api/settings**: Updates program settings (API Gateway → Core).
- **POST /api/rewards**: Adds reward (API Gateway → Points).
- **POST /api/referral**: Creates referral link (API Gateway → Referrals).
- **GET /api/referral/status**: Retrieves referral status (API Gateway → Referrals).
- **POST /api/gdpr/request**: Submits GDPR request (API Gateway → AdminCore).
- **POST /api/notifications/template**: Configures templates (API Gateway → AdminFeatures).
- **POST /api/customers/import**: Imports customers (API Gateway → AdminFeatures).
- **GET /api/rate-limits**: Monitors rate limits (API Gateway → AdminFeatures).
- **GET /api/admin/analytics**: Retrieves analytics (API Gateway → RFM).
- **GET /api/nudges**: Retrieves RFM nudges (API Gateway → RFM).
- **POST /api/admin/replay**: Replays actions (API Gateway → AdminFeatures).
- **POST /api/admin/rfm/simulate**: Simulates RFM transitions (API Gateway → AdminFeatures).
- **POST /api/admin/square/sync**: Syncs Square POS (API Gateway → AdminFeatures).
- **GET /api/products**: Retrieves product data (API Gateway → Products).
- **GET /api/products/rfm**: Gets product-level RFM (API Gateway → Products).
- **POST /v1/api/users/create**: Creates user (API Gateway → Users).
- **GET /v1/api/users/get**: Retrieves user (API Gateway → Users).
- **POST /v1/api/roles/create**: Creates role (API Gateway → Roles).
- **GET /v1/api/roles/get**: Retrieves role (API Gateway → Roles).

6. Webhooks

- **orders/create**: Awards points (API Gateway → Points, RFM).
- **orders/cancelled**: Adjusts points (API Gateway → Points).
- **customers/data_request**: Handles GDPR requests (API Gateway → AdminCore).
- **customers/redact**: Handles redaction (API Gateway → AdminCore).
- **pos/offline_sync**: Syncs POS transactions (API Gateway → Points).

7. Notifications

- **Klaviyo/Postscript/AWS SES**: Triggers for points, referrals, GDPR, with Bull queues and PostHog logging.
- **Logic**: Queues via Bull, monitors deliverability (alerts at 90%+ success).

8. Shopify Functions (Rust/Wasm)

- **calculate_points**: Computes points at checkout (Points).
- **update_rfm_score**: Updates RFM scores (RFM).

9. Frontend Components

- **Customer Widget**: `PointsBalance`, `RedeemForm`, `ReferralStatus`, `GDPRRequestForm`.
- **Merchant Dashboard**: `SettingsPage`, `NotificationTemplatePage`, `RateLimitPage`, `CustomerImportPage`, `ActionReplayPage`, `RFMSimulationPage`, `SquareSyncPage`, `UsersPage`, `RolesPage`, `AnalyticsPage`.

10. Deployment

- **Docker Compose**: Configures 15 services (API Gateway, Core, Auth, Points, Referrals, Users, Roles, RFM, Event Tracking, AdminCore, AdminFeatures, Campaign, Gamification, Frontend, Products), db (PostgreSQL, MongoDB, TimescaleDB), redis, nginx.
- **Kubernetes**: Horizontal Pod Autoscaling for Plus merchants (Phase 6).
- **CI/CD**: GitHub Actions, Jest/Cypress/k6 tests, Lighthouse CI, OWASP ZAP, Chaos Mesh, daily backups, Slack alerts.
- **Local Development**: `dev.sh` seeds mock data (Faker), simulates RFM, rate limits, audit log replay.
- **Disaster Recovery**: `pg_dump`, MongoDB/Redis snapshots, Backblaze B2 (90-day retention, RTO: 4 hours, RPO: 1 hour).
- **Feature Flags**: `feature-flags.json` for rollouts, canary routing via Nginx.

11. Security

- **Authentication**: JWTs with MFA, RBAC via Roles, IP allowlisting, HMAC.
- **Encryption**: AES-256 for PII via `pgcrypto`.
- **GDPR/CCPA**: Webhook handlers with retries, 90-day retention.

12. Testing

- **Unit Tests**: Jest for all services.
- **E2E Tests**: Cypress for Frontend.
- **Performance Tests**: k6 for 10,000 orders/hour.
- **Accessibility**: Lighthouse CI (90+ scores).

13. Risks and Mitigations

- **Solo Developer Bandwidth**: AI tools (30–40% efficiency), prioritize Must Have features.
- **Shopify API Rate Limits**: Redis tracking, Bull queues, simulate limits.
- **GDPR/CCPA Complexity**: Redis retries, automated tests.
- **Multilingual Accuracy**: Native speakers via Slack.
- **Integration Reliability**: Fallbacks (AWS SES), health checks.
- **Email Deliverability**: Monitor Klaviyo/Postscript/AWS SES, PostHog alerts.
- **Black Friday Scalability**: MongoDB sharding, Redis clustering, TimescaleDB hypertables, k6 tests.

14. Future Considerations (Phase 6)

- **Should Have Features**: Product-level RFM, VIP tiers, multi-store sharing, Flow templates.
- **Could Have Features**: Gamification, multilingual widget, AI reward suggestions.
- **Scaling**: Migrate to Kubernetes, Elasticsearch.
- **Certification**: Built for Shopify (4.5+ stars).

15. Assumptions and Constraints

- **Assumptions**: Shopify API stability, AI efficiency, VPS support, Shopify Partner feedback.
- **Constraints**: Solo developer, $97,012.50 budget, 39.5-week timeline.

Implementation Steps

1. **Update File**:
   - Replace `E:\loyalnest\system_architecture_and_specifications.md` with this content.
2. **Apply Changes**:
   - Commit to Nx monorepo.
3. **Test Integration**:
   - Run `npx nx test` to verify services.
4. **Commit Changes**:

```
git add system_architecture_and_specifications.md
git commit -m "Update system architecture with Users, Roles, RFM services
and schemas"
```