RFM Configuration

LoyalNest App

Overview

This document outlines the RFM (Recency, Frequency, Monetary) configuration feature for the LoyalNest Shopify app, targeting small (100–1,000 customers, AOV $20), medium (1,000–10,000 customers, AOV $100), and Shopify Plus merchants (50,000+ customers, AOV $500, 10,000 orders/hour during Black Friday surges). The feature is implemented in a dedicated `rfm-service`, integrated with `AdminCore`, `AdminFeatures`, `Users`, `Roles`, `Points`, `Referrals`, `Auth`, and `Frontend` services using NestJS, Vite + React, Rust/Wasm with Shopify Functions, PostgreSQL (JSONB, range partitioning), TimescaleDB, Redis Cluster/Streams, Bull queues, Kafka, and Loki + Grafana, managed via an Nx monorepo. It delivers a minimum viable product with iterative improvements, focusing on usability, performance, GDPR/CCPA compliance, and multilingual support (`en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6). Enhancements include lifecycle-based segments, industry benchmarks, multi-segment support, score history, explainable scores, time-weighted recency, smart nudges, persona-based defaults, A/B test templates, incremental updates, queue monitoring, and robust testing to enhance retention and scalability within the 39.5-week TVP timeline (ending February 17, 2026).

Microservices Architecture

- **RFM Service**: Handles RFM calculations, segment exports, nudge tracking, A/B testing, lifecycle stage tagging, score history, churn prediction, and visualizations; exposes REST (`/api/v1/rfm/*`) and gRPC APIs (`/rfm.v1/RFMService/UpdateThresholds`, `/rfm.v1/RFMService/GetSegmentCounts`, `/rfm.v1/RFMService/ExportSegments`, `/rfm.v1/RFMService/GetNudges`, `/rfm.v1/RFMService/PreviewRFMSegments`, `/rfm.v1/RFMService/PredictChurn`, `/rfm.v1/RFMService/SimulateRFM`, `/rfm.v1/RFMService/GetRFMVisualizations`); manages PostgreSQL tables (`customer_segments`, `rfm_segment_counts`, `rfm_segment_deltas`, `rfm_score_history`, `rfm_benchmarks`, `nudges`, `nudge_events`, `email_templates`, `email_events`); uses Redis Streams (`rfm:customer:{id}`, `rfm:preview:{merchant_id}`, `rfm:burst:{merchant_id}`).
- **Users Service**: Manages customer data (`users` table: id, email ENCRYPTED, first_name, last_name, metadata JSONB); exposes gRPC APIs (`/users.v1/UsersService/GetCustomers`, `/users.v1/UsersService/UpdateCustomer`).
- **Roles Service**: Manages RBAC (`roles` table: role_id, role_name, permissions JSONB); exposes gRPC APIs (`/roles.v1/RolesService/GetRoles`, `/roles.v1/RolesService/GetPermissions`).
- **AdminCore Service**: Manages RFM configuration, audit logging, and user management; exposes REST (`/admin/v1/rfm/*`, `/admin/v1/users/*`) and gRPC APIs (`/admin.v1/AdminCoreService/GetOverview`, `/admin.v1/AdminCoreService/ImportCustomers`).
- **AdminFeatures Service**: Handles scheduling, reprocessing failed jobs, rate limit/queue monitoring, event simulation, and onboarding progress; exposes REST (`/admin/v1/queues`, `/admin/v1/events/simulate`, `/admin/v1/setup/stream`) and gRPC APIs (`/admin.v1/AdminFeaturesService/GetQueueMetrics`, `/admin.v1/AdminFeaturesService/SimulateEvent`).

- **Frontend Service**: Delivers RFM configuration UI, customer widget nudges, industry benchmark visualizations, and onboarding progress; uses Vite + React with Polaris, Tailwind CSS, i18next (RTL for `ar`, `he`); communicates via REST (`/api/v1/rfm/*`) and WebSocket (`/admin/v1/setup/stream`).
- **Points Service**: Integrates with RFM for reward assignments (e.g., points for Champions); exposes gRPC APIs (`/points.v1/PointsService/RedeemCampaignDiscount`).
- **Referrals Service**: Supports referral-based nudges (e.g., "Invite a friend!" for At-Risk); exposes gRPC APIs.
- **Auth Service**: Handles authentication and RBAC enforcement; exposes gRPC APIs (`/auth.v1/AuthService/CreateAdminUser`).
- **Communication**: gRPC for inter-service communication (`RFM ↔ AdminCore`, `RFM ↔ AdminFeatures`, `RFM ↔ Points`, `RFM ↔ Referrals`, `RFM ↔ Users`, `RFM ↔ Roles`), REST/GraphQL for `Frontend ↔ RFM/AdminCore`, Redis Streams for caching (`rfm:preview:{merchant_id}`, `campaign_discount:{campaign_id}`, `rfm:burst:{merchant_id}`, `setup_tasks:{merchant_id}`), Kafka for events (`rfm_tier_assigned`, `rfm_nudge_clicked`), WebSocket for real-time updates (`/admin/v1/imports/stream`, `/admin/v1/setup/stream`).
- **Deployment**: Docker Compose for service containers (`rfm`, `users`, `roles`, `admin-core`, `admin-features`, `frontend`, `redis`, `postgres`, `kafka`, `timescaledb`), Nx monorepo for build management, Kubernetes for Plus-scale orchestration (Phase 6), Chaos Mesh for resilience testing.

Task List for Implementing RFM Configuration

Phase 1: Planning and Setup

Goal: Establish a robust foundation for RFM configuration in `rfm-service`, aligning with merchant needs, Shopify Plus scalability, GDPR/CCPA compliance, and the 39.5-week TVP timeline.

Enhancements & Best Practices:

- Interview 5–10 merchants (2–3 Plus) via Typeform surveys and "LoyalNest Collective" Slack to validate RFM thresholds, lifecycle stages, usability, and multilingual accuracy (90%+ for `en`, `es`, `fr`, `de`, `pt`, `ja`).
- Ensure GDPR/CCPA compliance (AES-256 encryption for `users.email`, `email_events.recipient_email`, webhook handling for `customers/redact` with 3 retries).
- Support multilingual UI and notifications (JSONB, i18next for `en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6, validated by 2–3 native speakers per language).
- Use PostHog to track interactions (e.g., `rfm_config_field_changed`, `rfm_wizard_badge_earned`, `rfm_segment_filtered`, `rfm_preview_viewed`, `rfm_nudge_clicked`).
- Implement LaunchDarkly feature flags (`rfm_advanced`, `rfm_nudges`, `rfm_benchmarks`, `rfm_simulation`, `rfm_incremental_updates`).
- Conduct monthly security audits for npm, cargo, Docker dependencies using OWASP ZAP (ECL: 256).
- Plan predictive analytics (churn prediction) and product-level RFM as Phase 6 stretch goals, leveraging xAI API (https://x.ai/api).

1. **Define Feature Requirements** (AdminCore, RFM Services)
    - *Description*: Finalize RFM configuration scope for small, medium, and Plus merchants.
    - *Tasks*:
        - Document RFM thresholds (weighted: 40% Recency, 30% Frequency, 30% Monetary):

        - Recency: Days since last order (1: >90 days, 2: 61–90 days, 3: 31–60 days, 4: 8–30 days, 5: ≤7 days).
        - Frequency: Number of orders (1: 1, 2: 2–3, 3: 4–5, 4: 6–10, 5: >10).
        - Monetary: Total spend, normalized by AOV (1: <0.5x AOV, 2: 0.5–1x AOV, 3: 1–2x AOV, 4: 2–5x AOV, 5: >5x AOV).
    - Support time-weighted recency for subscription models (e.g., slower decay: R5 ≤14 days, R1 >180 days, stored in `program_settings.rfm_thresholds.recency_decay` JSONB).
    - Define 2–5 tiers (e.g., Champions: R5, F4–5, M4–5; At-Risk: R1–2, F1–2, M1–2) with multi-segment support (e.g., "VIP" and "At-Risk High-Value" in `customer_segments.segment_ids` JSONB array).
    - Add lifecycle stages (e.g., "new lead," "repeat buyer," "churned") in `users.metadata` JSONB for marketing flows via Klaviyo/Postscript.
    - Specify adjustment frequencies: Real-time (`orders/create`), daily (<10,000 customers), weekly (10,000+), monthly, quarterly.
    - Support incremental updates via `rfm_segment_deltas` on `orders/create`, with daily refresh of `rfm_segment_counts` materialized view (`0 1 * * *`) in `rfm-service`.
    - Include multilingual notification templates (`email_templates.body`, `nudges.description` as JSONB, e.g., `{"en": "Welcome to Gold!", "es": "¡Bienvenido a Oro!", "ar": "مرحبًا بك في الذهب!"}`), with RTL support (`ar`, `he`).
    - Implement GDPR/CCPA webhooks (`customers/data_request`, `customers/redact`, 3 retries, Redis dead-letter queue) with cascade deletes in `rfm-service`.
    - Define success metrics: 85%+ wizard completion rate, 15%+ repeat purchase rate increase, 90%+ query performance under 1s, 80%+ nudge interaction rate, 90%+ export completion under 5s.
    - Handle edge cases: Zero orders (R1, F1, M1), high AOV ($10,000+ capped at M5), negative AOV (returns, M1), partial orders (exclude cancelled), inactive customers (>365 days, flag for nudges).
    - Define Phase 6 stretch goals: Churn prediction using `orders`, `nudge_events`, `rfm_score_history`, `rfm_segment_deltas` via xAI API; product-level RFM using `orders.lineItems`.
    - Create Typeform survey for 5–10 merchants (2–3 Plus) on RFM thresholds, lifecycle stages, notification preferences, and persona-based defaults (e.g., "Pet Store," "Electronics"), validated via "LoyalNest Collective" Slack, logging responses in Notion.
    - Store requirements in `program_settings.rfm_thresholds` (JSONB) and `merchant_settings.currencies` (JSONB for multi-currency) via `AdminCore`.
    - Initialize `rfm_segment_counts`, `rfm_segment_deltas`, `rfm_benchmarks` tables in `rfm-service` with daily refresh (`0 1 * * *`) for segment analytics.
    - Initialize `rfm_benchmarks` table for anonymized industry benchmarks (e.g., % customers in Champions by merchant size/AOV).
  - *Deliverable*: Requirements document (Notion) with Plus, GDPR/CCPA, multilingual, RTL, lifecycle, benchmark, and incremental update considerations.

2. **Analyze Merchant Data Patterns** (RFM Service)

  - *Description*: Study purchase cycles, AOV, and industry benchmarks to suggest default RFM thresholds and personas.
  - *Tasks*:

        - Use Shopify GraphQL Admin API (2025-01) to calculate median purchase interval and AOV:
            - Small: AOV $20, Monetary 5 = $100+.
            - Medium: AOV $100, Monetary 5 = $500+.
            - Plus: AOV $500, Monetary 5 = $2,500+.
        - Calculate anonymized benchmarks (e.g., % in Champions, average RFM score) by merchant size/AOV, stored in `rfm_benchmarks` via `rfm-service`.
        - Validate with 5–10 merchant personas (e.g., Pet Store, Fashion, Electronics, Plus-scale retailer) via Typeform surveys and "LoyalNest Collective" Slack (2–3 native speakers per language for `en`, `es`, `fr`, `de`, `pt`, `ja`).
        - Store defaults in `program_settings.rfm_thresholds` (JSONB, e.g., `{"monetary_5": 2500, "recency_decay": "standard"}`) via `AdminCore`.
        - Cache AOV and benchmark analysis in Redis Streams (`rfm:aov:{merchant_id}`, TTL 7d; `rfm:benchmarks:{size}`, TTL 30d).
        - Cache configuration previews in Redis Streams (`rfm:preview:{merchant_id}`, TTL 1h).
        - Track analysis via PostHog (`rfm_aov_analyzed`, `rfm_benchmarks_analyzed`, `rfm_persona_selected`).
    - *Deliverable*: Default RFM thresholds, persona-based defaults, and industry benchmarks.

3. **Set Up Development Environment** (All Services)

    - *Description*: Configure microservices for RFM development.
    - *Tasks*:
        - Initialize branch (`feature/rfm-service`) in Nx monorepo.
        - **Frontend Service**: Set up Vite + React with Shopify Polaris, TypeScript, Tailwind CSS, App Bridge, i18next for multilingual support (`en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6).
        - **RFM Service**: Configure NestJS with GraphQL client (`@shopify/shopify-api`, 2025-01), PostgreSQL (TypeORM, partitioned `customer_segments`, `rfm_segment_counts`, `rfm_segment_deltas`, `rfm_score_history`, `rfm_benchmarks`, `nudges`, `nudge_events`, `email_templates`, `email_events`), Redis Cluster (ioredis), PostHog SDK, Bull queues, and Rust/Wasm for Shopify Functions.
        - **Users Service**: Configure NestJS with PostgreSQL (TypeORM, `users`), gRPC server for customer data.
        - **Roles Service**: Configure NestJS with PostgreSQL (TypeORM, `roles`), gRPC server for RBAC.
        - **AdminCore Service**: Configure NestJS with PostgreSQL (TypeORM, `program_settings`, `audit_logs`), gRPC server, and LaunchDarkly SDK.
        - **AdminFeatures Service**: Configure NestJS for queue monitoring, event simulation, and onboarding progress, with Bull queues and WebSocket support.
        - Install Shopify CLI for Rust Functions (`cargo shopify`) in `rfm-service`.
        - Set up gRPC proto files for `RFM ↔ AdminCore`, `RFM ↔ AdminFeatures`, `RFM ↔ Points`, `RFM ↔ Referrals`, `RFM ↔ Users`, `RFM ↔ Roles`, `RFM` (`/rfm.v1/RFMService/*`).
        - Configure Docker Compose for service containers (`rfm`, `users`, `roles`, `admin-core`, `admin-features`, `frontend`, `redis`, `postgres`, `kafka`, `timescaledb`).
        - Add `init.sql` to initialize `rfm_segment_counts`, `rfm_segment_deltas`, `rfm_benchmarks`, `nudges`, `nudge_events`, `email_templates`, `email_events` tables with daily refresh (`0 1 * * *`).

- *Deliverable*: Dev environment with microservices, GraphQL, gRPC, partitioned database, PostHog, feature flags, and Kafka.

Phase 2: Backend Development (NestJS/TypeScript)

Goal: Build scalable backend logic for RFM calculations, tier assignments, notifications, lifecycle tagging, score history, smart nudges, and incremental updates in `rfm-service`.

Enhancements & Best Practices:

- Use API versioning (`/api/v1/rfm/*` for RFM, `/admin/v1/rfm/*` for AdminCore/AdminFeatures).
- Implement input validation (e.g., `recency < 365 days`) and GDPR-compliant AES-256 encryption (pgcrypto).
- Optimize for Plus-scale with PostgreSQL range partitioning (`customer_segments`, `rfm_segment_deltas`), materialized views (`rfm_segment_counts`, daily refresh at `0 1 * * *`), Redis Streams (`rfm:preview:{merchant_id}`, `campaign_discount:{campaign_id}`, `rfm:burst:{merchant_id}`), and Bull queues (`rate_limit_queue:{merchant_id}`).
- Log errors to Sentry, monitor performance with Loki + Grafana (alerts for median >1s, P95 >3s).
- Use gRPC for inter-service communication, circuit breakers (`nestjs-circuit-breaker`), and dead letter queues (DLQ) for resilience.
- Use token bucket algorithm (`rate-limiter-flexible`) for Shopify API burst handling (2 req/s standard, 40 req/s Plus).

4. **Integrate Shopify APIs** (RFM Service)
    - *Description*: Fetch customer/order data for RFM calculations.
    - *Tasks*:
        - Set up GraphQL Admin API client (2025-01) in `rfm-service`:

```
query {
  customer(id: "gid://shopify/Customer/123") {
    id
    email
    orders(first: 100, after: $cursor) {
      edges {
        node { totalPrice, createdAt, status, lineItems {
productId, quantity } }
      }
    }
  }
}
```

        - Create REST endpoints: `GET /api/v1/rfm/customers`, `GET /api/v1/rfm/orders` with pagination (batch 100).
        - Fetch customer data via gRPC (`/users.v1/UsersService/GetCustomers`).
        - Cache results in PostgreSQL (`users`, `orders`) and Redis Streams (`rfm:customer:{id}`, TTL 24h).

        - Verify webhook signatures (HMAC-SHA256) for `orders/create`, `customers/data_request`, `customers/redact` with 3 retries (2s initial delay, exponential backoff, Redis DLQ).
        - Handle rate limits (2 req/s REST, 40 req/s Plus, 1–4 req/s Storefront) with token bucket algorithm (`rate-limiter-flexible`), caching bursts in Redis Streams (`rfm:burst:{merchant_id}`, TTL 1h) and queuing non-critical tasks in Bull (`rate_limit_queue:{merchant_id}`).
        - Prioritize Plus merchants in Bull queues for burst scenarios (10,000 orders/hour).
        - Track API calls via PostHog (`shopify_api_called`, `rate_limit_breach`).
        - Handle service downtime: Fallback to cached data in Redis if Shopify API unavailable, log to Sentry (`shopify_api_failed`).
    - *Deliverable*: GraphQL-based API service with webhook verification, burst caching, queue handling, and error logging.

5. **Implement RFM Calculation Logic** (RFM Service)

    - *Description*: Calculate RFM scores with time-weighted recency, lifecycle tagging, score history, and incremental updates.
    - *Tasks*:
        - Define TypeScript interfaces:

```
interface RFMConfig {
  recency: { [key: number]: { maxDays: number } };
  frequency: { [key: number]: { minOrders: number } };
  monetary: { [key: number]: { minSpend: number } };
  recency_decay: string; // e.g., "standard", "subscription"
}
interface RFMScore {
  recency: number;
  frequency: number;
  monetary: number;
  score: number; // Weighted average (40% Recency, 30% Frequency, 30% Monetary)
}
```

        - Write NestJS service in `rfm-service` to compute RFM scores:
            - Recency: Compare `orders.createdAt` to current date, adjust for decay (e.g., subscription: R5 ≤14 days, R1 >180 days).
            - Frequency: Count valid orders (`status = 'completed'`).
            - Monetary: Sum `totalPrice`, normalize by AOV, support multi-currency via `merchant_settings.currencies` (JSONB).
            - Weighted score: `(0.4 * recency + 0.3 * frequency + 0.3 * monetary)`.
            - Store in `users.rfm_score` (JSONB, AES-256 encrypted, e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.1}`) via `users-service`.
            - Log history in `rfm_score_history` (customer_id, rfm_score JSONB, timestamp).
            - Log incremental updates in `rfm_segment_deltas` (merchant_id, customer_id, segment_change, updated_at) on `orders/create`.

        - Tag lifecycle stages (e.g., “new lead,” “repeat buyer,” “churned”) in `users.metadata` JSONB via `users-service`.
    - Add constraints: `CHECK (rfm_score->>'recency' IN ('1', '2', '3', '4', '5'))`, `CHECK (rfm_score->>'score' BETWEEN 1 AND 5)`.
    - Add partial index: `idx_users_rfm_score_at_risk` on `users` (WHERE `rfm_score->>'score' < 2`) for At-Risk nudges.
    - Handle edge cases: Zero orders (R1, F1, M1), high AOV ($10,000+ → M5), negative AOV (returns, M1), partial orders (exclude cancelled), inactive (>365 days, R1).
    - Cache scores in Redis Streams (`rfm:customer:{id}`, TTL 24h).
    - Support multilingual nudges via gRPC (`/rfm.v1/RFMService/GetNudges`, `nudges.title`, `nudges.description` as JSONB) for `Frontend Service`.
    - Use Bull queues for async calculations, priority for Plus merchants, monitored via `AdminFeatures` (`/admin/v1/queues`).
    - Handle service failures: Retry gRPC calls to `Points Service` (`/points.v1/PointsService/RedeemCampaignDiscount`) 3 times with circuit breakers (`nestjs-circuit-breaker`).
    - Log errors to Sentry (`rfm_calculation_failed`), track via PostHog (`rfm_tier_assigned`, `rfm_segment_updated`).
  - *Deliverable*: RFM calculation service with time-weighted recency, lifecycle tagging, score history, incremental updates, constraints, caching, and edge case handling.

6. **Develop Tier Assignment Logic** (RFM Service)

  - *Description*: Assign customers to multiple segments based on RFM scores and lifecycle stages.
  - *Tasks*:
    - Create NestJS service in `rfm-service` to map RFM scores to segments (`program_settings.rfm_thresholds` via gRPC from `AdminCore`):
      - Example: `{"name": "Gold", "rules": {"recency": ">=4", "frequency": ">=3", "monetary": ">=4"}}`, `{"name": "VIP", "rules": {"monetary": ">=5"}}`.
      - Segments: Champions (R5, F4–5, M4–5), Loyal (R3–5, F3–5, M3–5), At-Risk (R1–2, F1–2, M1–2), New (R4–5, F1, M1–2), Inactive (R1, F1, M1), VIP (M5).
      - Support multi-segment membership (e.g., “VIP” and “At-Risk High-Value”) in `customer_segments.segment_ids` (JSONB array).
    - Update `users.rfm_score`, `users.metadata` (lifecycle stages) via `users-service`, and `customer_segments` (JSONB) in `rfm-service`.
    - Update `rfm_segment_deltas` (merchant_id, customer_id, segment_change, updated_at) on `orders/create`.
    - Partition `customer_segments` and `rfm_segment_deltas` by `merchant_id` for Plus-scale.
    - Enforce RBAC via gRPC call to `Roles Service` (`/roles.v1/RolesService/GetPermissions`, e.g., `["admin:full", "admin:analytics"]`).
    - Notify `Points Service` via gRPC (`/points.v1/PointsService/RedeemCampaignDiscount`) for reward assignments (e.g., 500 points for Champions, discounts based on `bonus_campaigns.conditions`).
    - Trigger smart nudges on tier drops (e.g., Champion → At-Risk) via Klaviyo/Postscript/AWS SES, using `rfm_score_history` and `rfm_segment_deltas`.

        - Track assignments via PostHog (`rfm_tier_assigned`, `rfm_tier_dropped`, `rfm_nudge_clicked`).
        - Log tier changes in `audit_logs` (`AdminCore`, action: `tier_assigned`, `tier_dropped`, `rfm_export`) with `reverted` flag for undo.
    - *Deliverable*: Tier assignment service with multi-segment support, lifecycle tagging, smart nudges, incremental updates, RBAC, partitioning, and audit logging.

7. **Set Up Adjustment Scheduling** (AdminFeatures Service)

    - *Description*: Implement scheduled and event-based tier adjustments.
    - *Tasks*:
        - Use `@nestjs/schedule` for cron jobs: Real-time (`orders/create`), daily (`0 0 * * *` for <10,000 customers), weekly (`0 0 * * 0` for 10,000+), monthly/quarterly options.
        - Subscribe to `orders/create` webhook for event-based updates, 3 retries (2s initial delay, exponential backoff, Redis DLQ).
        - Implement grace period in `program_settings.config` (JSONB, e.g., `{"grace_period_days": 30}`).
        - Handle GDPR webhooks (`customers/data_request`, `customers/redact`, 3 retries, Redis DLQ) with cascade deletes (`users`, `customer_segments`, `rfm_score_history`, `rfm_segment_deltas`) in `rfm-service`.
        - Use Bull queues with priority for Plus merchants, cache schedules in Redis Streams (`rfm:schedule:{merchant_id}`, TTL 7d), monitor via `AdminFeatures` (`/admin/v1/queues`).
        - Notify `rfm-service` via gRPC (`/rfm.v1/RFMService/PreviewRFMSegments`) to trigger RFM calculations.
        - Track scheduling via PostHog (`rfm_schedule_triggered`, `rfm_segment_updated`).
        - Handle service downtime: Queue jobs in Bull if `rfm-service` unavailable, with DLQ and `/admin/v1/queues/reprocess` endpoint (RBAC: `admin:full`).
    - *Deliverable*: Scheduling service with retries, GDPR compliance, caching, queue monitoring, and DLQ.

8. **Integrate Notifications** (RFM Service)

    - *Description*: Enable tier change and smart nudges via Klaviyo, Postscript, and AWS SES fallback.
    - *Tasks*:
        - Create endpoint: `POST /api/v1/rfm/notifications` with input validation (e.g., regex for `nudge.title`).
        - Integrate Klaviyo API (`POST /api/v2/campaigns`), Postscript API (`POST /sms/messages`), and AWS SES (`SendEmail`) for multilingual templates (`email_templates.body`, `nudges.description` as JSONB, including `ar`, `he` for RTL, `fallback_language: en`).
        - Encrypt `email_events.recipient_email` (AES-256 via pgcrypto) for GDPR/CCPA.
        - Implement retries (5 attempts, 2s initial delay, exponential backoff) via Bull queues, with DLQ for failed jobs, monitored via `AdminFeatures` (`/admin/v1/queues`).
        - Trigger referral-based nudges via gRPC to `Referrals Service` (e.g., “Invite a friend!” for At-Risk).
        - Trigger smart nudges on tier drops (e.g., Champion → At-Risk) using `rfm_score_history`, `rfm_segment_deltas`, via Klaviyo/Postscript/AWS SES.

        - Support A/B testing of nudges with predefined templates (e.g., “Urgency vs. Discount,” “Social Proof vs. Direct Ask”) in `nudges.variants` (JSONB), tracked via PostHog (`nudge_variant_clicked`).
        - Fetch nudges via gRPC (`/rfm.v1/RFMService/GetNudges`).
        - Track via PostHog (`notification_sent`, `sms_nudge_sent`, `rfm_tier_dropped_nudge`, `referral_fallback_triggered`).
        - Add default templates: “Welcome to {tier}!” (Klaviyo), “Stay Active!” (Postscript/AWS SES for At-Risk).
    - *Deliverable*: Notification service with multilingual support, smart nudges, A/B test templates, retries, GDPR compliance, AWS SES fallback, and queue monitoring.

Phase 3: Shopify Functions (Rust/Wasm)

Goal: Optimize RFM updates for performance-critical scenarios in `rfm-service`.

Enhancements & Best Practices:

- Add Sentry logging for Rust function errors (`rfm_function_failed`).
- Handle Shopify API rate limits (40 req/s for Plus) with exponential backoff and Bull queue (`rate_limit_queue:{merchant_id}`).
- Use feature flags (LaunchDarkly: `rfm_incremental_updates`, `rfm_nudges`, `rfm_simulation`) for real-time updates, A/B testing, and smart nudges.

9. **Develop RFM Score Update Function** (RFM Service)
    - *Description*: Update RFM scores in real-time via Shopify Functions.
    - *Tasks*:
        - Set up Rust project with Shopify Function CLI (`cargo shopify`) in `rfm-service`.
        - Implement logic:

```
#[shopify_function]
fn update_rfm_score(input: Input) -> Result<Output> {
    let order = input.order;
    let config = input.rfm_config; // Includes recency_decay
    let score = calculate_rfm(&order, input.merchant_aov,
config.recency_decay)?;
    update_customer(&score, &input.customer_id)?;
    log_history(&score, &input.customer_id)?; // Log to
rfm_score_history
    log_delta(&score, &input.customer_id, &input.merchant_id)?; //
Log to rfm_segment_deltas
    log::info!("RFM updated for customer {}", input.customer_id);
    Ok(Output { score })
}
```

        - Update `users.rfm_score`, `rfm_score_history`, and `rfm_segment_deltas` via webhook callbacks (`orders/create`) and gRPC to `users-service`.
        - Handle edge cases: Partial orders (exclude cancelled), negative AOV (M1), multi-currency via `merchant_settings.currencies`.

        - Support A/B testing of nudges, storing variants in `nudges.variants` (JSONB) and tracking via PostHog (`nudge_variant_clicked`).
        - Log errors to Sentry (`rfm_function_failed`), handle rate limits (40 req/s Plus) with Bull queue (`rate_limit_queue:{merchant_id}`).
        - Cache results in Redis Streams (`rfm:customer:{id}`, TTL 24h).
        - Notify `Points Service` via gRPC (`/points.v1/PointsService/RedeemCampaignDiscount`) for reward updates based on `bonus_campaigns.conditions`.
    - *Deliverable*: Deployed Shopify Function with logging, caching, A/B testing, incremental updates, queue handling, and gRPC integration.

10. **Optimize for Large Stores** (RFM Service)
    - *Description*: Ensure scalability for 50,000+ customers and Black Friday surges (10,000 orders/hour).
    - *Tasks*:
        - Implement batch processing in Rust (1,000 customers/batch) for RFM calculations, leveraging Shopify Functions.
        - Cache batch results in Redis Streams (`rfm:batch:{merchant_id}`, TTL 1h) and campaign discounts (`campaign_discount:{campaign_id}`, TTL 24h).
        - Optimize PostgreSQL with range partitioning on `customer_segments`, `rfm_segment_deltas`, `rfm_score_history` by `merchant_id`, and materialized views (`rfm_segment_counts`, daily refresh at `0 1 * * *`) for segment analytics.
        - Use Bull queues (`rate_limit_queue:{merchant_id}`) for non-critical tasks (e.g., batch updates, exports), with priority for Plus merchants, monitored via `AdminFeatures` (`/admin/v1/queues`).
        - Test with simulated data (50,000 customers, 10,000 orders/hour) using k6, targeting 90%+ query performance under 1s and 95%+ queue operation success rate.
        - Scale `rfm-service` independently using Kubernetes for Plus merchants (Phase 6), with auto-scaling based on CPU/memory usage (Loki + Grafana).
        - Implement circuit breakers (`nestjs-circuit-breaker`) for gRPC calls (`/rfm.v1/RFMService/PreviewRFMSegments`, `/points.v1/PointsService/RedeemCampaignDiscount`) to prevent cascading failures.
        - Log performance metrics to Loki + Grafana (alerts for median >1s, P95 >3s), errors to Sentry (`rfm_batch_failed`).
        - Track via PostHog (`rfm_batch_processed`, `queue_metrics_viewed`).

- *Deliverable*: Optimized test report for Plus-scale performance, batch processing, queue handling, and monitoring.

Phase 4: Frontend Development (Vite/React)

Goal: Build an accessible, multilingual UI for RFM configuration with benchmarks, explainability, persona-based defaults, A/B test templates, and onboarding progress in `Frontend Service`.

Enhancements & Best Practices:

- Ensure WCAG 2.1 AA compliance and mobile responsiveness with Polaris and Tailwind CSS.

- Track UI interactions via PostHog (e.g., `rfm_config_field_changed`, `rfm_wizard_badge_earned`, `rfm_explain_viewed`, `rfm_benchmarks_viewed`, `setup_progress_viewed`).
- Use i18next for multilingual support (`en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6), validated by 2–3 native speakers per language via "LoyalNest Collective" Slack for 90%+ accuracy.
- Handle service downtime with fallback messages in Polaris `Banner` components.
- Add dynamic locale detection, interactive Chart.js visualizations, industry benchmarks, A/B test template selectors, gamification, and onboarding progress tracking.

11. **Design RFM Configuration UI** (Frontend Service)
    - *Description*: Create a React form using Polaris for RFM settings with persona-based defaults, explainability, and onboarding integration.
    - *Tasks*:
        - Extend React component (`RFMConfigPage.tsx`) with Polaris, TypeScript, Tailwind CSS, App Bridge, and i18next (`en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6, `dir=auto` for RTL).
        - Add inputs:
            - Persona selector (e.g., "Pet Store," "Fashion Retailer," "Electronics") to pre-fill thresholds based on Typeform survey data and `rfm_benchmarks`.
            - RFM thresholds (sliders/text fields, e.g., Recency 5: ≤7 days, Monetary 5: $2,500+ for Plus).
            - Recency decay mode (dropdown: "standard," "subscription").
            - Segments (name, RFM criteria, multi-segment support, rewards: discounts, free shipping via `bonus_campaigns.conditions`).
            - Adjustment frequency (dropdown: real-time, daily, weekly, monthly, quarterly).
            - Notification settings (multilingual templates, toggle for Klaviyo/Postscript/AWS SES, A/B test template selector: "Urgency vs. Discount," "Social Proof vs. Direct Ask").
        - Use Polaris components (`TextField`, `Select`, `FormLayout`) with ARIA labels for WCAG 2.1 AA compliance.
        - Implement real-time validation (e.g., "Monetary 5 must be > Monetary 4") and feedback via Polaris `Banner` (e.g., "Invalid Recency value").
        - Add explainability: Polaris `Tooltip`/`Modal` for RFM score breakdowns (e.g., "Last purchase 150 days ago (R1), 2 orders (F2), spent $50 (M2)"), tracked via PostHog (`rfm_explain_viewed`).
        - Add progress checklist (e.g., "3/5 steps completed") integrated with `setup_tasks` table, visualized in `RFMConfigPage.tsx` with Polaris `ProgressBar`, updated via WebSocket (`/admin/v1/setup/stream`).
        - Implement gamification: Badges (e.g., "RFM Pro" for 5/5 steps, PostHog: `rfm_wizard_badge_earned`) and one-time 10% discount for setup within 24 hours (PostHog: `rfm_discount_claimed`).
        - Implement dynamic locale detection using Shopify Storefront API (`shop.locale`), with manual override via Polaris `Select`, cached in Redis (`admin:locale:{user_id}`, TTL 7d), fallback to `en`.
        - Handle `rfm-service`, `AdminCore`, `AdminFeatures` downtime: Display fallback message ("Configuration temporarily unavailable") in Polaris `Banner`.
        - Track via PostHog (`rfm_config_field_changed`, `rfm_config_saved`, `rfm_persona_selected`, `setup_progress_viewed`).

- *Deliverable*: Accessible, multilingual RFM configuration form with persona-based defaults, explainability, A/B test templates, validation, gamification, onboarding progress, and fallback.

12. **Add Analytics Preview** (Frontend Service)
    - *Description*: Display segment sizes, benchmarks, and time-series with Chart.js.
    - *Tasks*:
        - Create endpoint in `rfm-service`: `GET /api/v1/rfm/preview` via gRPC (`/rfm.v1/RFMService/PreviewRFMSegments`, `/rfm.v1/RFMService/GetRFMVisualizations`) for real-time segment sizes and benchmarks.
        - Use Chart.js for interactive visualizations (bar chart for segment sizes, line chart for time-series, scatter plot for Recency vs. Monetary):

```
[
  {
    type: "bar",
    data: {
      labels: ["Champions", "Loyal", "At-Risk", "New", "Inactive",
"VIP"],
      datasets: [
        {
          label: "Your Customers",
          data: [100, 300, 600, 200, 400, 50],
          backgroundColor: ["#FFD700", "#C0C0C0", "#FF4500",
"#32CD32", "#808080", "#800080"],
          borderColor: ["#DAA520", "#A9A9A9", "#B22222",
"#228B22", "#696969", "#4B0082"],
          borderWidth: 1
        },
        {
          label: "Industry Average",
          data: [120, 280, 580, 220, 380, 60],
          backgroundColor: ["#FFFACD", "#D3D3D3", "#FFA07A",
"#90EE90", "#A9A9A9", "#E6E6FA"],
          borderWidth: 1
        }
      ]
    },
    options: {
      scales: { y: { beginAtZero: true } },
      plugins: {
        tooltip: { enabled: true },
        legend: { position: 'top' }
      },
      onClick: (event, elements) => {
        if (elements.length) {
          const segment = elements[0].index; // e.g., 0 for
Champions
          postHog.capture('rfm_segment_filtered', { segment });
          openModal(segment); // Polaris Modal for customer list
        }
```

```
      }
    }
  },
  {
    type: "line",
    data: {
      labels: ["2025-06", "2025-07", "2025-08"],
      datasets: [
        {
          label: "Champions Count",
          data: [90, 100, 110],
          borderColor: "#FFD700",
          fill: false
        },
        {
          label: "At-Risk Count",
          data: [650, 600, 580],
          borderColor: "#FF4500",
          fill: false
        }
      ]
    },
    options: {
      scales: { y: { beginAtZero: true } },
      plugins: { legend: { position: 'top' } }
    }
  },
  {
    type: "scatter",
    data: {
      datasets: [
        {
          label: "Customers",
          data: [
            { x: 10, y: 500, segment: "Champions" },
            { x: 60, y: 200, segment: "At-Risk" }
          ],
          backgroundColor: (context) => context.raw.segment ===
"Champions" ? "#FFD700" : "#FF4500",
          pointRadius: 5
        }
      ]
    },
    options: {
      scales: {
        x: { title: { display: true, text: "Recency (days)" } },
        y: { title: { display: true, text: "Monetary ($)" } }
      },
      plugins: { legend: { position: 'top' } }
    }
  }
]
```

        - Cache previews in Redis Streams (`rfm:preview:{merchant_id}`, TTL 1h).
        - Implement CSV/JSON/PNG export for visualizations via `POST /api/v1/rfm/export`, async processing with Bull queues, tracked in `audit_logs` (`rfm_export`).
        - Display filtered customer lists in Polaris `Modal`, with pagination (100 customers/page), tracked via PostHog (`rfm_segment_filtered`).
        - Fetch data via REST/gRPC from `rfm-service`, fallback to cached data in Redis if service unavailable.
        - Display industry benchmarks from `rfm_benchmarks` table, with Polaris `Tooltip` for comparison (e.g., “Your Champions: 10% vs. industry 12%”).
        - Track via PostHog (`rfm_preview_viewed`, `rfm_segment_filtered`, `rfm_benchmarks_viewed`, `rfm_preview_exported`, `visualization_viewed`).

- *Deliverable*: Interactive analytics preview with Chart.js, industry benchmarks, time-series, scatter plots, caching, export, and fallback.

Phase 5: Testing and Validation

Goal: Ensure reliability for small, medium, and Plus merchants across microservices.

Enhancements & Best Practices:

- Test edge cases (zero orders, high AOV, negative AOV, GDPR scenarios, service downtime).
- Simulate Plus-scale stores (50,000+ customers) and Black Friday surges (10,000 orders/hour) with k6.
- Conduct concurrency tests for simultaneous RFM calculations across services.
- Run penetration tests with OWASP ZAP (ECL: 256) for security (XSS, SQL injection, API vulnerabilities).
- Add performance alerts for RFM calculations (median >1s, P95 >3s) via Loki + Grafana.
- Support simulation mode for 30/60/90-day RFM histories via `AdminFeatures` (`/admin/v1/events/simulate`).
- Validate multilingual accuracy (90%+ for `en`, `es`, `fr`, `de`, `pt`, `ja`) with 2–3 native speakers per language via “LoyalNest Collective” Slack.

13. **Unit Test Backend Logic** (RFM, AdminCore, AdminFeatures, Users, Roles Services)
    - *Description*: Test RFM calculations, tier assignments, lifecycle tagging, smart nudges, and incremental updates.
    - *Tasks*:
        - Write Jest tests for `rfm-service` (RFM calculations, multi-segment support, lifecycle tagging, smart nudges, incremental updates via `rfm_segment_deltas`), `AdminCore`/`AdminFeatures` (configuration, scheduling, queue monitoring, event simulation), `Users Service` (customer data), and `Roles Service` (RBAC):
            - Edge cases: Zero orders (R1, F1, M1), high AOV ($10,000+ → M5), negative AOV (M1), partial orders (exclude cancelled), inactive customers (>365 days), multi-currency conversions.
            - Validation: Invalid thresholds (e.g., Recency <0), duplicate segment names, recency decay modes, queue retry limits, error codes (`RFM_CONFIG_INVALID`, `EXPORT_FAILED`).
            - Service failures: Simulate `rfm-service` downtime, test gRPC retries (`/points.v1/PointsService/RedeemCampaignDiscount`,

`/rfm.v1/RFMService/GetNudges`) with circuit breakers, and Bull queue DLQ handling.
    - Mock Shopify API (2025-01) for edge cases (e.g., invalid emails, cancelled orders).
    - Test GDPR webhook handling (`customers/redact` with cascade deletes in `users`, `customer_segments`, `rfm_score_history`, `rfm_segment_deltas`).
    - Test AWS SES fallback for Klaviyo/Postscript failures, tracked via PostHog (`referral_fallback_triggered`).
    - Run OWASP ZAP penetration tests for `/api/v1/rfm/*`, `/admin/v1/rfm/*`, and gRPC endpoints (`/rfm.v1/*`, `/users.v1/*`, `/roles.v1/*`), validating RBAC and AES-256 encryption (`users.rfm_score`, `users.metadata`, `email_events.recipient_email`).

- *Deliverable*: Test suite with 85%+ coverage across services.

14. **Test Shopify Function** (RFM Service)
    - *Description*: Validate Rust/Wasm function for real-time RFM updates.
    - *Tasks*:
        - Use Shopify CLI to test with sample (100 customers) and Plus-scale data (50,000 customers, 10,000 orders/hour).
        - Verify PostgreSQL updates (`users.rfm_score`, `rfm_score_history`, `rfm_segment_deltas`, `rfm_segment_counts`) and rate limit handling (40 req/s Plus) with Bull queue (`rate_limit_queue:{merchant_id}`).
        - Test edge cases: Partial orders, negative AOV, multi-currency, service downtime (fallback to Bull queues).
        - Test A/B nudge variants and smart nudges, ensuring PostHog tracking (`nudge_variant_clicked`, `rfm_tier_dropped_nudge`).
        - Validate manual Square POS sync integration (`/admin/integrations/square/sync`, `square_sync_triggered`).

- *Deliverable*: Tested Shopify Function for Plus-scale with A/B testing, smart nudges, incremental updates, and Square sync.

15. **Test UI and UX** (Frontend Service)
    - *Description*: Ensure intuitive, accessible UI with explainability, benchmarks, and onboarding progress.
    - *Tasks*:
        - Conduct usability testing with 5–10 merchants (2–3 Plus) via Typeform surveys and "LoyalNest Collective" Slack, focusing on RFM setup, nudge interactions, benchmark visualizations, and RTL support (`ar`, `he`), iterating via Notion.
        - Verify WCAG 2.1 AA compliance (ARIA labels, keyboard navigation) and multilingual rendering (`en`, `es`, `fr`, `de`, `pt`, `ja`; Phase 6: `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)) with 2–3 native speakers per language.
        - Test form submission, validation, persona selector, A/B test templates, visualizations, and API integration (`POST /api/v1/rfm/config`, `/rfm.v1/RFMService/PreviewRFMSegments`, `/rfm.v1/RFMService/GetRFMVisualizations`) using Cypress.
        - Test explainability modals/tooltips, benchmark visualizations, and onboarding progress (`setup_tasks`, WebSocket `/admin/v1/setup/stream`) in `RFMConfigPage.tsx`.

        - Test service downtime scenarios: Display fallback messages in Polaris `Banner` for `rfm-service`, `AdminCore`, `AdminFeatures` unavailability.

- *Deliverable*: Accessible, multilingual UI with usability feedback, explainability, benchmarks, and onboarding integration.

16. **End-to-End Testing** (All Services)
    - *Description*: Test full workflow with simulation mode across microservices.
    - *Tasks*:
        - Simulate RFM configuration (`AdminCore`/`AdminFeatures`), calculations (`rfm-service`), and UI rendering (`Frontend`) for 50,000 customers using `test/factories/*.ts`.
        - Implement simulation mode: `POST /admin/v1/events/simulate` (RBAC: `admin:full`) for 30/60/90-day RFM histories using mock data (`rfm_score_history`, `rfm_segment_deltas`), tracked via PostHog (`admin_event_simulated`).
        - Trigger `orders/create` and GDPR webhooks (`customers/redact`) in `rfm-service`, validating cascade deletes.
        - Verify Klaviyo/Postscript/AWS SES notifications (including smart nudges), Redis caching (`rfm:preview:{merchant_id}`, `campaign_discount:{campaign_id}`), Bull queue handling (`rate_limit_queue:{merchant_id}`), and segment accuracy (`rfm_segment_counts`, `rfm_segment_deltas`, `rfm_benchmarks`).
        - Test concurrency: Simultaneous RFM calculations for 5,000+ merchants across services.
        - Extend k6 tests for Black Friday load (10,000 orders/hour, 100 concurrent RFM calculations), validating Redis Streams (`rfm:burst:{merchant_id}`), Bull queue prioritization, and circuit breakers.
        - Test inter-service communication: gRPC calls between `rfm-service`, `AdminCore`, `AdminFeatures`, `Points` (`/points.v1/PointsService/RedeemCampaignDiscount`), `Referrals`, `Users`, and `Roles` Services.
        - Validate manual Square POS sync (`/admin/integrations/square/sync`) and AWS SES fallback (`referral_fallback_triggered`).

- *Deliverable*: End-to-end test report for Plus-scale, Black Friday, GDPR compliance, simulation mode, and Square sync.

Phase 6: Deployment and Documentation

Goal: Launch with rollback plan, comprehensive docs, and disaster recovery across microservices.

Enhancements & Best Practices:

- Use feature flags (LaunchDarkly: `rfm_advanced`, `rfm_nudges`, `rfm_benchmarks`, `rfm_simulation`, `rfm_incremental_updates`) for gradual rollout.
- Include GDPR/CCPA, multilingual, benchmark, and disaster recovery (RTO: 4 hours, RPO: 1 hour) guidance in docs.
- Monitor deployment with Sentry (errors), Loki + Grafana (performance, alerts for median >1s, P95 >3s), and PostHog (events).
- Implement chaos testing with Chaos Mesh and API rate-limiting (100 req/min per merchant).
- Ensure 90-day backup retention in Backblaze B2 for `audit_logs`, `nudge_events`, `rfm_score_history`, `rfm_segment_deltas`.

17. **Deploy Feature** (All Services)
    - *Description*: Release RFM configuration to production.
    - *Tasks*:
        - Deploy using Docker Compose for `rfm`, `users`, `roles`, `admin-core`, `admin-features`, `frontend`, `redis`, `postgres`, `kafka`, `timescaledb`; use Kubernetes for Plus-scale orchestration (Phase 6).
        - Enable feature flags (LaunchDarkly: `rfm_advanced`, `rfm_nudges`, `rfm_benchmarks`, `rfm_simulation`, `rfm_incremental_updates`) for phased rollout.
        - Implement rate-limiting for `/api/v1/rfm/*`, `/admin/v1/rfm/*` (100 req/min per merchant) using `rate-limiter-flexible`, caching in Redis Streams (`admin:rate_limits:{merchant_id}`, `admin:endpoint_limits:{merchant_id}:{endpoint}`, TTL 1h), logging violations to Sentry (`rfm_api_rate_limited`).
        - Implement chaos testing with Chaos Mesh in VPS (Kubernetes in Phase 6), simulating `rfm`, `users`, `roles`, `admin-core`, `admin-features`, Redis, and PostgreSQL failures, validating circuit breakers, DLQs, and fallback UI messages.
        - Monitor via Sentry (errors, e.g., `rfm_calculation_failed`), Loki + Grafana (API latency, queue performance, RFM calculation alerts), PostHog (`rfm_wizard_completed`, `rfm_nudge_clicked`), and AWS SNS (alerts for rate limit breaches, integration failures).
        - Implement disaster recovery: PostgreSQL point-in-time recovery (RTO: 4 hours, RPO: 1 hour), Redis AOF persistence, and Backblaze B2 backups (90-day retention), validated weekly via `restore.sh`.
        - Implement rollback plan: Revert if errors >1% or latency >5s, using blue-green deployment via Docker Compose.

- *Deliverable*: Live deployment with monitoring, rate-limiting, chaos testing, disaster recovery, and backup.

18. **Create Documentation** (AdminCore Service)
    - *Description*: Provide merchant and developer guides.
    - *Tasks*:
        - Write multilingual help article (`en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6) with GDPR/CCPA tips (e.g., "Log customer consent for exports in `audit_logs`"), benchmark guidance (e.g., "Compare Champions % to industry averages"), and best practices (e.g., "Use subscription decay for infrequent buyers").
        - Include screenshots and 1–2 minute YouTube videos for RFM setup, nudges, benchmarks, simulation mode, queue monitoring, and onboarding progress.
        - Generate OpenAPI specs for `/api/v1/rfm/*` (`rfm-service`) and `/admin/v1/rfm/*` (`AdminCore`/`AdminFeatures`) using NestJS decorators.
        - Document gRPC proto files for `RFM ↔ AdminCore`, `RFM ↔ AdminFeatures`, `RFM ↔ Points`, `RFM ↔ Referrals`, `RFM ↔ Users`, `RFM ↔ Roles`, `RFM` (`/rfm.v1/RFMService/*`).
        - Document RFM calculation logic (weighted scoring, time-weighted recency, lifecycle stages, incremental updates, edge cases, queue handling) in developer guide.

- *Deliverable*: Multilingual help article, OpenAPI specs, gRPC proto files, and developer guide.

19. **Pilot with Merchants** (All Services)
    - *Description*: Test with real merchants.

    - *Tasks*:
        - Recruit 5–10 merchants (2–3 Plus) via Shopify Reddit/Discord and "LoyalNest Collective" Slack.
        - Monitor metrics (segment sizes via `rfm_segment_counts`, `rfm_segment_deltas`, repeat purchases, nudge interactions, benchmark engagement, onboarding completion) via PostHog.
        - Collect feedback via Typeform surveys and Slack, focusing on Plus usability, RFM effectiveness, RTL support (`ar`, `he`), persona defaults, benchmarks, and queue monitoring, iterating via Notion.

- *Deliverable*: Beta feedback report with Plus, RTL, benchmark, and onboarding insights.

Phase 7: Pricing and Rollout

Goal: Ensure accessibility and profitability across microservices.

Enhancements & Best Practices:

- Prioritize Plus merchants for advanced features (real-time updates, RBAC, custom notifications, benchmarks, predictive analytics, simulation mode).
- Use feature flags for phased rollout to minimize disruptions.
- Track adoption and support queries via PostHog and Zendesk.
- Gamify setup wizard with badges, discounts, and persona-based onboarding.

20. **Define Pricing Strategy** (AdminCore Service)
    - *Description*: Price for small, medium, and Plus merchants.
    - *Tasks*:
        - Basic RFM (2–3 tiers, monthly updates) in free plan (300 orders).
        - Advanced RFM (real-time updates via `rfm_segment_deltas`, RBAC, custom notifications, export, nudges, A/B testing, benchmarks, simulation mode) in paid plans ($29/month for 500 orders, $49/month for 50,000–100,000 customers, $99/month for 100,000+ Plus).
        - Compare with competitors (e.g., LoyaltyLion: $399/month for similar features).
        - Redirect to https://x.ai/grok for pricing details.

- *Deliverable*: Tiered pricing model with Plus tiers.

21. **Roll Out to All Merchants** (Frontend and AdminCore Services)
    - *Description*: Launch to all users.
    - *Tasks*:
        - Announce via email, in-app Polaris `Banner`, and Klaviyo/Postscript/AWS SES campaigns (`Frontend Service`).
        - Provide multilingual setup wizard with tooltips, persona-based defaults, A/B test templates, and gamification (badges like "RFM Pro" for 5/5 steps, PostHog: `rfm_wizard_badge_earned`, 10% discount for setup within 24 hours, PostHog: `rfm_discount_claimed`).
        - Track onboarding progress in `setup_tasks` table, visualized with Polaris `ProgressBar` and WebSocket (`/admin/v1/setup/stream`).
        - Monitor adoption via PostHog (`rfm_wizard_completed`, `rfm_nudge_clicked`, `rfm_benchmarks_viewed`, `setup_progress_viewed`) and support queries via Zendesk

(`AdminCore Service`).
    - Prioritize Plus merchants for support and advanced feature rollout (`AdminCore Service`).
- *Deliverable*: Phased rollout campaign with gamification, onboarding progress, and Plus prioritization.

Timeline and Resource Estimate

Total Duration: ~35–40 days (1–2 developers), aligned with 39.5-week TVP timeline (ending February 17, 2026).

- Phase 1: 6–7 days (Planning, environment setup, merchant surveys).
- Phase 2: 15–16 days (Backend APIs, calculations, notifications, lifecycle tagging, smart nudges, incremental updates, circuit breakers, DLQ).
- Phase 3: 5–6 days (Rust Shopify Functions, A/B testing, smart nudges, incremental updates).
- Phase 4: 7–8 days (Frontend UI, analytics preview, benchmarks, persona defaults, explainability, A/B templates, onboarding progress).
- Phase 5: 10–11 days (Testing, Black Friday simulation, penetration testing, simulation mode, performance alerts).
- Phase 6: 4–5 days (Deployment, documentation, chaos testing, disaster recovery).
- Phase 7: 3 days (Pricing, rollout).

Resources: 1 full-stack developer (NestJS/React, Rust), 1 QA tester ($3,000), Grok AI for code review and documentation, GitHub Copilot/Cursor for coding efficiency. Total cost: ~$10,000 (4 weeks at $2,500/week), within $97,012.50 budget.

Considerations for Merchants

- *Simplicity*: "Quick Setup" wizard (`Frontend Service`) with persona-based/AOV-based thresholds, progress checklist, gamification (badges, discounts), real-time validation, and onboarding progress tracking.
- *Affordability*: Basic RFM free (300 orders), advanced RFM $29–$99/month, competitive with LoyaltyLion ($399/month).
- *Usability*: Polaris UI with multilingual tooltips (`en`, `es`, `fr`, `de`, `pt`, `ja`; Phase 6: `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)), WCAG 2.1 AA compliance, mobile responsiveness, explainable RFM scores, industry benchmarks, and onboarding progress (`Frontend Service`).
- *Scalability*: Optimized for 100–50,000+ customers with microservices (`rfm-service` for calculations, `AdminCore`/`AdminFeatures` for config), range partitioning (`customer_segments`, `rfm_segment_deltas`), materialized views (`rfm_segment_counts`, daily refresh `0 1 * * *`), Redis Streams (`rfm:preview:{merchant_id}`, `campaign_discount:{campaign_id}`, `rfm:burst:{merchant_id}`), and Bull queues (`rate_limit_queue:{merchant_id}`).
- *Support*: Live chat/email via Zendesk (`AdminCore Service`) with GDPR/CCPA guidance (e.g., "Log customer consent for exports in `audit_logs`") and disaster recovery documentation (RTO: 4 hours, RPO: 1 hour).

Example Merchant Workflow

Pet Store (AOV $40, 1,000 customers):

- Configuration: Persona: “Pet Store”; Recency: 5 = <30 days (standard decay), Frequency: 5 = 5+ orders, Monetary: 5 = $200+; Tiers: Gold (R5, F4–5, M4–5), Silver (R3–4, F2–3, M2–3), Bronze (R1–2, F1, M1); Multi-segments: “VIP” (M5); Daily updates, 30-day grace period; Klaviyo/AWS SES notifications (“Welcome to Gold!”), smart nudges on tier drops.
- Outcome: 10% in Gold, 25% repeat purchase rate increase, 15% nudge interaction rate, benchmark comparison (e.g., “Your Champions: 10% vs. industry 12%”), 90%+ onboarding completion.

Electronics Retailer (AOV $500, 50,000 customers):

- Configuration: Persona: “Electronics”; Recency: 5 = <60 days (subscription decay), Frequency: 5 = 10+ orders, Monetary: 5 = $2,500+; Tiers: Platinum (R5, F5, M5), Gold (R4–5, F4–5, M4–5); Multi-segments: “VIP” (M5), “At-Risk High-Value” (R1–2, M5); Real-time updates via `rfm_segment_deltas`, RBAC for staff (`admin:analytics`, `admin:full`), Postscript/AWS SES SMS nudges (“Stay Active!” for At-Risk, A/B tested).
- Outcome: 5% in Platinum, 20% engagement increase, 90%+ query performance under 1s, benchmark comparison (e.g., “Your At-Risk: 15% vs. industry 18%”), 95%+ queue operation success rate.

Database Schema

- **Tables**:
  - `users` (id, email ENCRYPTED, first_name, last_name, rfm_score ENCRYPTED JSONB, metadata JSONB) [Users Service]
  - `customer_segments` (segment_id, merchant_id, rules JSONB, name JSONB, segment_ids JSONB ARRAY) [RFM Service]
  - `rfm_score_history` (customer_id, rfm_score JSONB, timestamp) [RFM Service]
  - `rfm_segment_deltas` (merchant_id, customer_id, segment_change JSONB, updated_at) [RFM Service]
  - `rfm_benchmarks` (merchant_size, aov_range, segment_name, customer_percentage, avg_rfm_score, last_updated) [RFM Service]
  - `rfm_segment_counts` (merchant_id, segment_name, customer_count, last_refreshed, INDEX `idx_rfm_segment_counts_merchant_id`) [RFM Service]
  - `nudges` (nudge_id, merchant_id, type CHECK('at-risk', 'loyal', 'new', 'inactive', 'tier_dropped'), title JSONB, description JSONB, is_enabled BOOLEAN, variants JSONB) [RFM Service]
  - `nudge_events` (event_id, customer_id, nudge_id, action CHECK('view', 'click', 'dismiss'), created_at) [RFM Service]
  - `program_settings` (merchant_id, config JSONB, rfm_thresholds JSONB, branding JSONB) [AdminCore Service]
  - `merchant_settings` (merchant_id, currencies JSONB) [AdminCore Service]
  - `email_templates` (template_id, merchant_id, type CHECK('tier_change', 'nudge'), subject JSONB, body JSONB, fallback_language TEXT) [RFM Service]
  - `email_events` (event_id, merchant_id, event_type CHECK('sent', 'failed'), recipient_email ENCRYPTED) [RFM Service]
  - `audit_logs` (id UUID, admin_user_id, action CHECK('tier_assigned', 'tier_dropped', 'rfm_export', 'customer_import', 'customer_import_completed', 'config_updated', 'rate_limit_viewed', 'undo_action', 'referral_fallback_triggered', 'square_sync_triggered'), target_table, target_id, created_at, reverted BOOLEAN) [AdminCore Service]

    - `bonus_campaigns` (campaign_id, merchant_id, name, type, multiplier, conditions JSONB) [Points Service]
    - `setup_tasks` (merchant_id, task_name, status, completed_at) [AdminFeatures Service]
    - `roles` (role_id, role_name UNIQUE, permissions JSONB with `["admin:full", "admin:analytics", "admin:support", "admin:points", "admin:merchants:view:shopify_plus", "admin:merchants:edit:plan"]`, created_at) [Roles Service]
- **Indexes**: `users(rfm_score, metadata)`, `customer_segments(merchant_id)`, `rfm_score_history(customer_id)`, `rfm_segment_deltas(merchant_id, updated_at)`, `rfm_benchmarks(merchant_size)`, `rfm_segment_counts(merchant_id)`, `nudges(merchant_id)`, `nudge_events(customer_id)`, `email_templates(merchant_id)`, `email_events(merchant_id)`, `audit_logs(admin_user_id)`, `bonus_campaigns(merchant_id, type)`, `setup_tasks(merchant_id)`, `roles(role_name)` [RFM/Users/Roles/AdminCore/AdminFeatures/Points Services]
- **Partial Indexes**: `idx_users_rfm_score_at_risk` on `users` (WHERE `rfm_score->>'score' < 2`) for At-Risk nudges [Users Service].
- **Materialized Views**: `rfm_segment_counts` for analytics performance, refreshed daily (`0 1 * * *`) [RFM Service].
- **Partitioning**: `customer_segments`, `rfm_segment_deltas`, `nudge_events`, `email_events`, `bonus_campaigns`, `rfm_score_history` by `merchant_id` [RFM/Points Services].
- **Encryption**: AES-256 via pgcrypto for `users.email`, `users.rfm_score`, `email_events.recipient_email`, quarterly key rotation via AWS KMS.

Next Steps

1. Start Phase 1: Finalize requirements (`AdminCore`, `rfm-service`), analyze data with benchmarks (`rfm-service`), set up environment with `rfm_segment_counts`, `rfm_segment_deltas`, `rfm_benchmarks` initialization in `init.sql` (All Services), and conduct Typeform surveys via “LoyalNest Collective” Slack (6–7 days).
2. Prioritize Backend: Build GraphQL integration (`GET /api/v1/rfm/customers` in `rfm-service`), RFM calculation with lifecycle tagging, score history, smart nudges, and incremental updates (`rfm-service`), and notification services with gRPC (`/rfm.v1/RFMService/GetNudges