RFM Configuration

LoyalNest App

Overview

This document outlines the RFM (Recency, Frequency, Monetary) configuration feature for the LoyalNest 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, 1,000 orders/hour). The implementation uses a microservices architecture (auth, points, referrals, analytics, admin, frontend services) with NestJS (Analytics and Admin Services), Vite + React (Frontend Service), Rust with Shopify Functions (Analytics Service), PostgreSQL, and Redis Streams, managed via Nx monorepo. The plan delivers a minimum viable feature with iterative improvements, focusing on usability, performance, GDPR/CCPA compliance, and multilingual support (English, Spanish, French, Arabic). 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, and robust testing to enhance retention and scalability.

Microservices Architecture

- **Analytics Service**: Handles RFM calculations, segment exports, nudge tracking, A/B testing, lifecycle stage tagging, score history, and churn prediction; exposes REST (`/api/v1/rfm/*`), gRPC APIs (`/analytics.v1/AnalyticsService/GetNudges` , `/analytics.v1/AnalyticsService/PreviewRFMSegments` , `/analytics.v1/AnalyticsService/PredictChurn` , `/analytics.v1/AnalyticsService/SimulateRFM`); uses PostgreSQL (`customers` , `customer_segments` , `rfm_segment_counts` , `rfm_score_history` , `rfm_benchmarks`), Redis Streams (`rfm:customer:{id}` , `rfm:preview:{merchant_id}` , `rfm:burst:{merchant_id}`).
- **Admin Service**: Manages RFM configuration, scheduling, audit logging, and reprocessing failed jobs; exposes REST (`/admin/v1/rfm/*`) and gRPC APIs; uses PostgreSQL (`program_settings` , `audit_logs`).
- **Frontend Service**: Delivers RFM configuration UI, customer widget nudges, and industry benchmark visualizations; uses Vite + React with Polaris, Tailwind CSS, i18next (RTL support for `ar`); communicates via REST (`/api/v1/rfm/*`).
- **Points Service**: Integrates with RFM for reward assignments (e.g., points for Champions) and campaign discounts (`/points.v1/PointsService/RedeemCampaignDiscount`); exposes gRPC APIs.
- **Referrals Service**: Supports referral-based nudges (e.g., “Invite a friend!” for At-Risk); exposes gRPC APIs.
- **Auth Service**: Handles RBAC for RFM configuration (`merchants.staff_roles`) and customer authentication; exposes gRPC APIs.
- **Communication**: gRPC for inter-service communication (e.g., Analytics ↔ Admin for RFM config), REST/GraphQL for Frontend ↔ Analytics/Admin, Redis Streams for cross-service caching (`rfm:preview:{merchant_id}` , `campaign_discount:{campaign_id}` , `rfm:burst:{merchant_id}`).

- **Deployment**: Docker Compose for service containers, Nx monorepo for build management, Kubernetes for Plus-scale orchestration, Chaos Mesh for resilience testing.

Task List for Implementing RFM Configuration

Phase 1: Planning and Setup

Goal: Establish a robust foundation for RFM configuration across microservices, aligning with merchant needs, Shopify Plus scalability, and GDPR/CCPA compliance.

Enhancements & Best Practices:

- Interview 5–10 merchants (2–3 Plus) with Typeform surveys to validate RFM thresholds, lifecycle stages, and usability, iterating via Notion.
- Ensure GDPR/CCPA compliance (encrypted `rfm_score`, `customers.metadata`, webhook handling for `customers/redact`).
- Support multilingual UI and notifications (JSONB, i18next for `en`, `es`, `fr`, `ar` with RTL).
- Use PostHog to track interactions (e.g., `rfm_config_field_changed`, `rfm_wizard_badge_earned`, `rfm_segment_filtered`).
- Implement LaunchDarkly feature flags for phased rollouts (e.g., `rfm_advanced`, `rfm_nudges`, `rfm_benchmarks`).
- Conduct monthly security audits for npm, cargo, Docker dependencies.
- Plan predictive analytics (churn prediction) and product-level RFM as Phase 6 stretch goals.

1. **Define Feature Requirements** (Admin Service)
   - *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`).
     - 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”).
     - Add lifecycle stages (e.g., “new lead,” “repeat buyer,” “churned”) in `customers.metadata` JSONB for marketing flows.

    - Specify adjustment frequencies: Daily (<10,000 customers), weekly (10,000+), monthly, quarterly, event-based (`orders/create`).
    - Include multilingual notification templates (`email_templates.body` as JSONB, e.g., `{"en": "Welcome to Gold!", "es": "¡Bienvenido a Oro!", "ar": "مرحبًا بك في الذهب!"}`).
    - Plan RTL support (`ar` , `he`) in `email_templates.body` , `nudges.description` , and Polaris UI (`dir=auto`).
    - Implement GDPR/CCPA webhooks (`customers/data_request` , `customers/redact`) with cascade deletes in Analytics Service.
    - Define success metrics: 85%+ wizard completion rate, 15%+ repeat purchase rate increase, 90%+ query performance under 1s, 80%+ nudge interaction rate.
    - Add 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` ; 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”), logging responses in Notion for Phase 6 iteration.
    - Store requirements in Admin Service (`program_settings.rfm_thresholds` JSONB).
    - Initialize `rfm_segment_counts` materialized view (US-MD12, US-BI5, I24a) 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/Google Docs) with Plus, GDPR, multilingual, RTL, lifecycle, and benchmark considerations.

2. **Analyze Merchant Data Patterns** (Analytics Service)
  - *Description*: Study purchase cycles, AOV, and industry benchmarks to suggest default RFM thresholds and personas.
  - *Tasks*:
    - Use Shopify GraphQL Admin API 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` .
    - Validate with 5–10 merchant personas (e.g., Pet Store, Fashion, Electronics, Plus-scale retailer) via Typeform surveys.
    - Store defaults in `program_settings.rfm_thresholds` (JSONB, e.g., `{"monetary_5": 2500, "recency_decay": "standard"}`) via Admin Service.
    - 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) for US-MD12.
    - Track analysis via PostHog (`rfm_aov_analyzed` , `rfm_benchmarks_analyzed`).
  - *Deliverable*: Default RFM thresholds and persona-based defaults (e.g., Pet Store, Electronics) with industry benchmarks.
3. **Set Up Development Environment** (All Services)
  - *Description*: Configure microservices for RFM development.
  - *Tasks*:
    - Initialize branch (`feature/rfm-config`) in Nx monorepo.
    - **Frontend Service**: Set up Vite + React with Shopify Polaris, TypeScript, Tailwind CSS, i18next for multilingual support (`en` , `es` , `fr` , `ar` with RTL).
    - **Analytics Service**: Configure NestJS with GraphQL client (`@shopify/shopify-api`), PostgreSQL (TypeORM, partitioned `customer_segments` , `rfm_segment_counts` , `rfm_score_history` , `rfm_benchmarks`), Redis (ioredis), PostHog SDK, and Bull queues.
    - **Admin Service**: Configure NestJS with PostgreSQL (TypeORM, `program_settings`), gRPC server, and LaunchDarkly SDK.
    - Install Shopify CLI for Rust Functions (`cargo shopify`) in Analytics Service.
    - Set up gRPC proto files for Analytics ↔ Admin, Analytics ↔ Points (`/points.v1/PointsService/RedeemCampaignDiscount`), Analytics ↔ Referrals, Analytics (`/analytics.v1/AnalyticsService/GetNudges` , `/analytics.v1/AnalyticsService/PreviewRFMSegments` , `/analytics.v1/AnalyticsService/PredictChurn` , `/analytics.v1/AnalyticsService/SimulateRFM`).
    - Configure Docker Compose for service containers (analytics, admin, frontend, redis, postgres).
    - Add `init.sql` to initialize `rfm_segment_counts` materialized view and `rfm_benchmarks` table with daily refresh (`0 1 * * *`).
  - *Deliverable*: Dev environment with microservices, GraphQL, gRPC, partitioned database, PostHog, and feature flags.

Phase 2: Backend Development (NestJS/TypeScript)

Goal: Build scalable backend logic for RFM calculations, tier assignments, notifications, lifecycle tagging, score history, and smart nudges across Analytics and Admin Services.

Enhancements & Best Practices:

- Use API versioning (`/api/v1/rfm/*` for Analytics, `/admin/v1/rfm/*` for Admin).
- Implement input validation (e.g., `recency < 365 days`) and GDPR-compliant encryption (AES-256).
- Optimize for Plus-scale with PostgreSQL partitioning, materialized views (`rfm_segment_counts` , daily refresh at `0 1 * * *`), and Redis Streams (`rfm:preview:{merchant_id}` , `campaign_discount:{campaign_id}` , `rfm:burst:{merchant_id}`).
- Log errors to Sentry, monitor performance with Prometheus/Grafana (alerts for median >1s, P95 >3s).

- Use gRPC for Analytics ↔ Admin, Analytics ↔ Points, Analytics ↔ Referrals.
- Implement circuit breakers (`nestjs-circuit-breaker`) and dead letter queues (DLQ) for resilience.
- Use token bucket algorithm (`rate-limiter-flexible`) for Shopify API burst handling.

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

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

     - Create REST endpoints: `GET /api/v1/rfm/customers` , `GET /api/v1/rfm/orders` with pagination (batch 100).
     - Cache results in PostgreSQL (`customers` , `orders`) and Redis Streams (`rfm:customer:{id}` , TTL 24h).
     - Verify webhook signatures (HMAC-SHA256) for `orders/create` , `customers/data_request` , `customers/redact` with 5 retries (2s initial delay, exponential backoff).
     - 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).
     - Prioritize Plus merchants in Bull queues for burst scenarios.

- Track API calls via PostHog (`shopify_api_called`).
- Handle service downtime: Fallback to cached data in Redis if Shopify API unavailable.

- *Deliverable*: GraphQL-based API service with webhook verification, burst caching, and error handling.

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

- *Description*: Calculate RFM scores with time-weighted recency, lifecycle tagging, and history tracking.
- *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 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.
    - Weighted score: `(0.4 * recency + 0.3 * frequency + 0.3 * monetary)` .
    - Store in `customers.rfm_score` (JSONB, e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.1}`).

    - Log history in `rfm_score_history` (customer_id, rfm_score JSONB, timestamp).
    - Tag lifecycle stages (e.g., “new lead,” “repeat buyer,” “churned”) in `customers.metadata` JSONB based on RFM and order patterns.
  - Add constraints: `CHECK (rfm_score->>'recency' IN ('1', '2', '3', '4', '5'))`, `CHECK (rfm_score->>'score' BETWEEN 1 AND 5)`.
  - Add partial index: `idx_customers_rfm_score_at_risk` on `customers` (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 (`/analytics.v1/AnalyticsService/GetNudges`, `nudges.title`, `nudges.description` as JSONB) for Frontend Service.
  - Use Bull queues for async calculations, priority for Plus merchants.
  - 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`).
- *Deliverable*: RFM calculation service with time-weighted recency, lifecycle tagging, history tracking, constraints, caching, and edge case handling.

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

- *Description*: Assign customers to multiple segments based on RFM scores and lifecycle stages.
- *Tasks*:
  - Create NestJS service to map RFM scores to segments (`program_settings.rfm_thresholds` via gRPC from Admin Service):
    - 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 `customers.rfm_score`, `customers.metadata` (lifecycle stages), and `customer_segments` (JSONB).
  - Partition `customer_segments` by `merchant_id` for Plus-scale.
  - Enforce RBAC via gRPC call to Auth Service (`merchants.staff_roles` JSONB, e.g., `{"role": "admin:full"}`).
  - 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, using `rfm_score_history` to detect changes.
- Track assignments via PostHog (`rfm_tier_assigned` , `rfm_tier_dropped`).
- Log tier changes in `audit_logs` (Admin Service, action: `tier_assigned` , `tier_dropped`).

- *Deliverable*: Tier assignment service with multi-segment support, lifecycle tagging, smart nudges, RBAC, partitioning, and audit logging.

7. **Set Up Adjustment Scheduling** (Admin Service)
   - *Description*: Implement scheduled and event-based tier adjustments.
   - *Tasks*:
     - Use `@nestjs/schedule` for cron jobs: Daily (`0 0 * * *`) for <10,000 customers, weekly (`0 0 * * 0`) for 10,000+ (Plus), monthly/quarterly options.
     - Subscribe to `orders/create` webhook for event-based updates, 5 retries (2s initial delay, exponential backoff).
     - Implement grace period in `program_settings.config` (JSONB, e.g., `{"grace_period_days": 30}`).
     - Handle GDPR webhooks (`customers/data_request` , `customers/redact`) with cascade deletes (`customers` , `customer_segments` , `rfm_score_history`) in Analytics Service.
     - Use Bull queues with priority for Plus merchants, cache schedules in Redis Streams (`rfm:schedule:{merchant_id}` , TTL 7d).
     - Notify Analytics Service via gRPC to trigger RFM calculations.
     - Track scheduling via PostHog (`rfm_schedule_triggered`).
     - Handle service downtime: Queue jobs in Bull if Analytics Service unavailable, with DLQ for failed jobs and `/admin/v1/rfm/reprocess` endpoint for manual retries.
   - *Deliverable*: Scheduling service with retries, GDPR compliance, caching, and DLQ.

8. **Integrate Notifications** (Analytics Service)
   - *Description*: Enable tier change and smart nudges via Klaviyo and Postscript.
   - *Tasks*:
     - Create endpoint: `POST /api/v1/rfm/notifications` with input validation (e.g., regex for `nudge.title`).
     - Integrate Klaviyo API (`POST /api/v2/events`) and Postscript API (`POST /sms/messages`) for multilingual templates (`email_templates.body` , `nudges.description` as JSONB, including `ar` for RTL).
     - Encrypt `email_events.recipient_email` (AES-256) for GDPR/CCPA.
     - Implement retries (5 attempts, 2s initial delay, exponential backoff) via Bull queues, with DLQ for failed jobs.
     - 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`, via Klaviyo/Postscript.
- 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 (`/analytics.v1/AnalyticsService/GetNudges`).
- Track via PostHog (`notification_sent`, `sms_nudge_sent`, `rfm_tier_dropped_nudge`).
- Add default templates: "Welcome to {tier}!" (Klaviyo), "Stay Active!" (Postscript for At-Risk).

- *Deliverable*: Notification service with multilingual support, smart nudges, A/B test templates, retries, GDPR compliance, and A/B testing.

Phase 3: Shopify Functions (Rust)

Goal: Optimize RFM updates for performance-critical scenarios in Analytics Service.

Enhancements & Best Practices:

- Add Sentry logging for Rust function errors.
- Handle Shopify API rate limits (40 req/s for Plus) with exponential backoff.
- Use feature flags (LaunchDarkly) for real-time RFM updates, A/B testing, and smart nudges.

9. **Develop RFM Score Update Function** (Analytics Service)
   - *Description*: Update RFM scores in real-time via Shopify Functions.
   - *Tasks*:
     - Set up Rust project with Shopify Function CLI (`cargo shopify`).
     - 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::info!("RFM updated for customer {}", input.customer_id);
    Ok(Output { score })
}
```

- Update `customers.rfm_score` and `rfm_score_history` via webhook callbacks (`orders/create`).
- Handle edge cases: Partial orders (exclude cancelled), negative AOV (M1).
- 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).
- 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, and gRPC integration.

10. **Optimize for Large Stores** (Analytics Service)
    - *Description*: Ensure scalability for 50,000+ customers.
    - *Tasks*:
        - Implement batch processing in Rust (1,000 customers/batch).
        - Cache batch results in Redis Streams (`rfm:batch:{merchant_id}` , TTL 1h).
        - Cache campaign discounts in Redis Streams (`campaign_discount:{campaign_id}` , TTL 24h).
        - Test with simulated data (50,000 customers, 1,000 orders/hour) using k6.
        - Optimize PostgreSQL queries with materialized views (`rfm_segment_counts` , refreshed daily via `0 1 * * *`).
        - Scale Analytics Service independently using Kubernetes for Plus merchants.

- *Deliverable*: Optimized test report for Plus-scale performance.

Phase 4: Frontend Development (Vite/React)

Goal: Build an accessible, multilingual UI for RFM configuration with benchmarks, explainability, and persona-based defaults in Frontend Service.

Enhancements & Best Practices:

- Ensure WCAG 2.1 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`).
- Use i18next for multilingual support (`en`, `es`, `fr`, `ar` with RTL).
- Handle service downtime gracefully with fallback messages.
- Add dynamic locale detection, interactive Chart.js previews, industry benchmarks, and A/B test template selectors.

11. **Design RFM Configuration UI** (Frontend Service)
    - *Description*: Create a React form using Polaris for RFM settings with persona-based defaults and explainability.
    - *Tasks*:
        - Extend React component (`RFMConfigPage.tsx`) with Polaris, TypeScript, Tailwind CSS, and i18next (`en`, `es`, `fr`, `ar` with RTL via `dir=auto`).
        - Add inputs:
            - Persona selector (e.g., “Pet Store,” “Fashion Retailer,” “Electronics”) to pre-fill thresholds based on Typeform survey data.
            - 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: daily, weekly, monthly, quarterly, event-based).
            - Notification settings (multilingual templates, toggle for Klaviyo/Postscript, A/B test template selector: “Urgency vs. Discount,” “Social Proof vs. Direct Ask”).
        - Use Polaris components (`TextField`, `Select`, `FormLayout`) with ARIA labels.
        - Implement real-time validation (e.g., “Monetary 5 must be > Monetary 4”) and feedback (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”), “Reset to Defaults” button (persona/AOV-based), and gamification (badges like “RFM Pro” for 5/5 steps, PostHog: `rfm_wizard_badge_earned`).
        - Offer one-time 10% discount for setup within 24 hours, tracked via PostHog (`rfm_discount_claimed`).
        - Implement dynamic locale detection using Shopify Storefront API (`shop.locale`), with manual override via Polaris `Select` and fallback to `en`.
        - Handle Analytics/Admin Service downtime: Display fallback message (“Configuration temporarily unavailable”).
        - Track via PostHog (`rfm_config_field_changed`, `rfm_config_saved`, `rfm_persona_selected`).

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

12. **Add Analytics Preview** (Frontend Service)
    - *Description*: Display segment sizes, benchmarks, and time-series with Chart.js (US-MD12, I24a).
    - *Tasks*:
        - Create endpoint in Analytics Service: `GET /api/v1/rfm/preview` via gRPC (`/analytics.v1/AnalyticsService/PreviewRFMSegments`) for real-time segment sizes and benchmarks.
        - Use Chart.js for interactive bar chart and time-series:

```
[
  {
    type: "bar",
    data: {
      labels: ["Champions", "Loyal", "At-Risk", "New", "Inactive"],
      datasets: [
        {
          label: "Your Customers",
          data: [100, 300, 600, 200, 400],
          backgroundColor: ["#FFD700", "#C0C0C0", "#FF4500", "#32CD32", "#808080"],
          borderColor: ["#DAA520", "#A9A9A9", "#B22222", "#228B22", "#696969"],
          borderWidth: 1
        },
        {
          label: "Industry Average",
          data: [120, 280, 580, 220, 380],
          backgroundColor: ["#FFFACD", "#D3D3D3", "#FFA07A", "#90EE90", "#A9A9A9"],
          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' } }
    }
  }
]
```

  - Cache previews in Redis Streams (`rfm:preview:{merchant_id}` , TTL 1h).
  - Implement CSV/PNG export for Chart.js preview, with Polaris `Modal` for filtered customer lists, tracked via PostHog (`rfm_preview_exported`).
  - Fetch data via REST/gRPC from Analytics Service, fallback to cached data if service unavailable.
  - Track via PostHog (`rfm_preview_viewed` , `rfm_segment_filtered` , `rfm_benchmarks_viewed`).
- *Deliverable*: Interactive analytics preview with Chart.js, industry benchmarks, time-series, 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 for security.
- Add performance alerts for RFM calculations (median >1s, P95 >3s).
- Support simulation mode for 30/60/90-day RFM histories.

13. **Unit Test Backend Logic** (Analytics and Admin Services)
    - *Description*: Test RFM calculations, tier assignments, lifecycle tagging, and smart nudges.
    - *Tasks*:
        - Write Jest tests for Analytics Service (RFM calculations, multi-segment support, lifecycle tagging, smart nudges) and Admin Service (configuration, scheduling):
            - Edge cases: Zero orders (R1, F1, M1), high AOV ($10,000+ → M5), negative AOV (M1), partial orders (exclude cancelled), inactive customers (>365 days).
            - Validation: Invalid thresholds (e.g., Recency <0), duplicate segment names, recency decay modes.
            - Service failures: Simulate Analytics Service downtime, test gRPC retries (`/points.v1/PointsService/RedeemCampaignDiscount` , `/analytics.v1/AnalyticsService/GetNudges`) with circuit breakers.
        - Mock Shopify API for edge cases (e.g., invalid emails, cancelled orders).
        - Test GDPR webhook handling (`customers/redact` with cascade deletes in `customers` , `customer_segments` , `rfm_score_history`).
        - Run OWASP ZAP penetration tests for `/api/v1/rfm/*` and gRPC endpoints, validating RBAC (`merchants.staff_roles`) and encryption (AES-256 for `rfm_score` , `customers.metadata`).
- *Deliverable*: Test suite with 85%+ coverage across services.

14. **Test Shopify Function** (Analytics Service)
    - *Description*: Validate Rust function for real-time updates.
    - *Tasks*:
        - Use Shopify CLI to test with sample (100 customers) and Plus-scale data (50,000 customers).
        - Verify PostgreSQL updates (`customers.rfm_score` , `rfm_score_history` , `rfm_segment_counts`) and rate limit handling (40 req/s).
        - Test edge cases: Partial orders, negative AOV, service downtime (fallback to Bull queues).
        - Test A/B nudge variants and smart nudges, ensuring PostHog tracking (`nudge_variant_clicked` , `rfm_tier_dropped_nudge`).
- *Deliverable*: Tested Shopify Function for Plus-scale with A/B testing and smart nudges.

15. **Test UI and UX** (Frontend Service)
    - *Description*: Ensure intuitive, accessible UI with explainability and benchmarks.
    - *Tasks*:

    - Conduct usability testing with 5–10 merchants (2–3 Plus) via Typeform surveys/calls, iterating via Notion.
    - Verify WCAG 2.1 compliance (ARIA labels, keyboard navigation) and multilingual rendering (`en` , `es` , `fr` , `ar` with RTL).
    - Test form submission, validation, persona selector, A/B test templates, and API integration (`POST /api/v1/rfm/config` , `/analytics.v1/AnalyticsService/PreviewRFMSegments`).
    - Test explainability modals/tooltips and benchmark visualizations in `RFMConfigPage.tsx` .
    - Test service downtime scenarios: Display fallback messages for Analytics/Admin Service unavailability.
- *Deliverable*: Accessible, multilingual UI with usability feedback, explainability, and benchmarks.

16. **End-to-End Testing** (All Services)
    - *Description*: Test full workflow with simulation mode across microservices.
    - *Tasks*:
        - Simulate RFM configuration (Admin Service), calculations (Analytics Service), and UI rendering (Frontend Service) for 50,000 customers.
        - Implement simulation mode: `POST /api/v1/rfm/simulate` for 30/60/90-day RFM histories using mock data (`test/factories/*.ts` , `rfm_score_history`).
        - Trigger `orders/create` and GDPR webhooks (`customers/redact`) in Analytics Service.
        - Verify Klaviyo/Postscript notifications (including smart nudges), Redis caching (`rfm:preview:{merchant_id}` , `campaign_discount:{campaign_id}`), and segment accuracy (`rfm_segment_counts` , `rfm_benchmarks`).
        - Test concurrency: Simultaneous RFM calculations for 5,000+ merchants across services.
        - Extend k6 tests to simulate Black Friday load (10,000 orders/hour, 100 concurrent RFM calculations), validating Redis Streams (`rfm:burst:{merchant_id}`) and Bull queue prioritization.
        - Test inter-service communication: gRPC calls between Analytics, Admin, Points (`/points.v1/PointsService/RedeemCampaignDiscount`), and Referrals Services.
- *Deliverable*: End-to-end test report for Plus-scale, Black Friday, GDPR compliance, and simulation mode.

Phase 6: Deployment and Documentation

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

Enhancements & Best Practices:

- Use feature flags (LaunchDarkly) for gradual rollout of advanced RFM, nudges, benchmarks, and simulation mode.
- Include GDPR/CCPA, multilingual, and benchmark guidance in docs.

- Monitor deployment with Sentry (errors), Prometheus/Grafana (performance, alerts for median >1s, P95 >3s).
- Implement chaos testing with Chaos Mesh and API rate-limiting.

17. **Deploy Feature** (All Services)
    - *Description*: Release RFM configuration to production.
    - *Tasks*:
        - Deploy using Docker Compose for Analytics, Admin, Frontend Services, Redis, and PostgreSQL; use Kubernetes for Plus-scale orchestration.
        - Enable feature flags (LaunchDarkly: `rfm_advanced` , `rfm_nudges` , `rfm_benchmarks` , `rfm_simulation`) for phased rollout.
        - Implement rate-limiting for `/api/v1/rfm/*` (100 req/min per merchant) using `rate-limiter-flexible` , logging violations to Sentry (`rfm_api_rate_limited`).
        - Implement chaos testing with Chaos Mesh in Kubernetes, simulating Analytics Service failures, network latency, and Redis outages. Validate circuit breakers, DLQs, and fallback UI messages.
        - Monitor via Sentry (errors, e.g., `rfm_calculation_failed`), Prometheus/Grafana (API latency, queue performance, RFM calculation alerts).
        - Implement rollback plan: Revert if errors >1% or latency >5s.
        - Set up 90-day backup retention for `audit_logs` , `nudge_events` , `rfm_score_history` in Admin/Analytics Services.
- *Deliverable*: Live deployment with monitoring, rate-limiting, chaos testing, and backup.

18. **Create Documentation** (Admin Service)
    - *Description*: Provide merchant and developer guides.
    - *Tasks*:
        - Write multilingual help article (`en` , `es` , `fr` , `ar`) with GDPR tips (e.g., “Ensure customer consent for exports”), benchmark guidance (e.g., “Compare your Champions % to industry averages”), and best practices (e.g., “Use subscription decay for infrequent buyers”).
        - Include screenshots, 1–2 minute videos for RFM setup, nudges, benchmarks, and simulation mode.
        - Generate OpenAPI specs for `/api/v1/rfm/*` (Analytics Service) and `/admin/v1/rfm/*` (Admin Service) using NestJS decorators.
        - Document gRPC proto files for Analytics ↔ Admin, Analytics ↔ Points (`/points.v1/PointsService/RedeemCampaignDiscount`), Analytics ↔ Referrals, Analytics (`/analytics.v1/AnalyticsService/GetNudges` , `/analytics.v1/AnalyticsService/PreviewRFMSegments` , `/analytics.v1/AnalyticsService/PredictChurn` , `/analytics.v1/AnalyticsService/SimulateRFM`).
        - Document RFM calculation logic (weighted scoring, time-weighted recency, lifecycle stages, edge cases) 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.
        - Monitor metrics (segment sizes via `rfm_segment_counts` , repeat purchases, nudge interactions, benchmark engagement) via PostHog across services.
        - Collect feedback via Typeform surveys/calls, focusing on Plus usability, RFM effectiveness, RTL support, persona defaults, and benchmark value, iterating via Notion.

- *Deliverable*: Beta feedback report with Plus, RTL, and benchmark 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).
- 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** (Admin Service)
    - *Description*: Price for small, medium, and Plus merchants.
    - *Tasks*:
        - Basic RFM (2–3 tiers, monthly updates) in free plan.
        - Advanced RFM (real-time updates, RBAC, custom notifications, export, nudges, A/B testing, benchmarks, simulation mode) in paid plans ($15/month for small, $29/month for medium, $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 Admin Services)
    - *Description*: Launch to all users.
    - *Tasks*:
      - Announce via email, in-app banner, and Klaviyo/Postscript 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`).
      - Monitor adoption via PostHog (`rfm_wizard_completed` , `rfm_nudge_clicked` , `rfm_benchmarks_viewed`) and support queries via Zendesk (Admin Service).
      - Prioritize Plus merchants for support and advanced feature rollout (Admin Service).

- *Deliverable*: Phased rollout campaign with gamification and Plus prioritization.

Timeline and Resource Estimate

Total Duration: ~35–40 days (1–2 developers).

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

Considerations for Merchants

- *Simplicity*: “Quick Setup” wizard (Frontend Service) with persona-based/AOV-based thresholds, progress checklist, gamification (badges, discounts), and real-time validation feedback.
- *Affordability*: Basic RFM free, advanced RFM $15–$99/month, competitive with LoyaltyLion ($399/month).

- *Usability*: Polaris UI with multilingual tooltips (`en` , `es` , `fr` , `ar` with RTL), WCAG 2.1 compliance, mobile responsiveness, explainable RFM scores, and industry benchmarks (Frontend Service).
- *Scalability*: Optimized for 100–50,000+ customers with microservices (Analytics Service for calculations, Admin Service for config), partitioned `customer_segments` , materialized views (`rfm_segment_counts` , daily refresh `0 1 * * *`), and Redis Streams (`rfm:preview:{merchant_id}` , `campaign_discount:{campaign_id}` , `rfm:burst:{merchant_id}`).
- *Support*: Live chat/email via Zendesk (Admin Service) with GDPR/CCPA guidance (e.g., "Log customer consent for exports in `audit_logs` ").

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); Monthly updates, 30-day grace period; Klaviyo notifications ("Welcome to Gold!"), smart nudges on tier drops.
- Outcome: 10% in Gold, 25% repeat purchase increase, 15% nudge interaction rate, benchmark comparison (e.g., "Your Champions: 10% vs. industry 12%").

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, RBAC for staff, Postscript 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%").

Database Schema

- **Tables**:
  - `customers` (customer_id, email ENCRYPTED, first_name, last_name, rfm_score ENCRYPTED JSONB, metadata JSONB, vip_tier_id JSONB) [Analytics Service]
  - `customer_segments` (segment_id, merchant_id, rules JSONB, name JSONB, segment_ids JSONB ARRAY) [Analytics Service]
  - `rfm_score_history` (customer_id, rfm_score JSONB, timestamp) [Analytics Service]
  - `rfm_benchmarks` (merchant_size, aov_range, segment_name, customer_percentage, avg_rfm_score, last_updated) [Analytics Service]

  - `nudges` (nudge_id, merchant_id, type CHECK('at-risk', 'loyal', 'new', 'inactive', 'tier_dropped'), title JSONB, description JSONB, is_enabled BOOLEAN, variants JSONB) [Analytics Service]
  - `nudge_events` (event_id, customer_id, nudge_id, action CHECK('view', 'click', 'dismiss'), created_at) [Analytics Service]
  - `program_settings` (merchant_id, config JSONB, rfm_thresholds JSONB, branding JSONB) [Admin Service]
  - `email_templates` (template_id, merchant_id, type CHECK('tier_change', 'nudge'), subject JSONB, body JSONB) [Analytics Service]
  - `email_events` (event_id, merchant_id, event_type CHECK('sent', 'failed'), recipient_email ENCRYPTED) [Analytics Service]
  - `audit_logs` (id UUID, admin_user_id, action CHECK('tier_assigned', 'tier_dropped', ...), target_table, target_id, created_at) [Admin Service]
  - `bonus_campaigns` (campaign_id, merchant_id, name, type, multiplier, conditions JSONB) [Points Service]
- **Indexes**: `customers(rfm_score, metadata)` , `customer_segments(merchant_id)` , `rfm_score_history(customer_id)` , `rfm_benchmarks(merchant_size)` , `nudges(merchant_id)` , `nudge_events(customer_id)` , `email_templates(merchant_id)` , `audit_logs(admin_user_id)` , `bonus_campaigns(merchant_id, type)` [Analytics/Admin/Points Services]
- **Partial Indexes**: `idx_customers_rfm_score_at_risk` on `customers` (WHERE `rfm_score->>'score' < 2`) for At-Risk nudges [Analytics Service].
- **Materialized Views**: `rfm_segment_counts` (merchant_id, segment_name, customer_count, last_refreshed, INDEX `idx_rfm_segment_counts_merchant_id`) for analytics performance, refreshed daily (`0 1 * * *`) (I24a, US-MD12, US-BI5) [Analytics Service].
- **Partitioning**: `customer_segments` , `nudge_events` , `email_events` , `bonus_campaigns` , `rfm_score_history` by `merchant_id` [Analytics/Points Services].

Next Steps

1. Start Phase 1: Finalize requirements (Admin Service), analyze data with benchmarks (Analytics Service), set up environment with `rfm_segment_counts` , `rfm_benchmarks` initialization in `init.sql` (All Services), and conduct Typeform surveys (6–7 days).
2. Prioritize Backend: Build GraphQL integration (`GET /api/v1/rfm/customers` in Analytics Service), RFM calculation with lifecycle tagging, score history, and smart nudges (Analytics Service), and notification services with gRPC (`/analytics.v1/AnalyticsService/GetNudges` , `/points.v1/PointsService/RedeemCampaignDiscount`), including circuit breakers and DLQ.
3. Test Early: Simulate RFM configs in Shopify sandbox with 100–50,000 customers, including Black Friday surges (10,000 orders/hour) and 30/60/90-day histories.
4. Seek Feedback: Share UI prototype (`RFMConfigPage.tsx` in Frontend Service) with 5–10 merchants (2–3 Plus) via Shopify Reddit/Discord, focusing on usability, Plus-scale performance, RTL support, persona defaults, and benchmarks.

Future Enhancements (Phase 6)

- **Product-Level RFM**: Calculate RFM per product/category using `orders.lineItems`, stored in `product_rfm_scores` table, for retention insights.
- **AI-Powered Churn Prediction**: Use `orders`, `nudge_events`, `rfm_score_history` with xAI API for predictive analytics, integrated via gRPC (`/analytics.v1/AnalyticsService/PredictChurn`).

Summary Table of Key Suggestions

Area	Suggestion
Planning	Include Plus merchants, GDPR/CCPA, multilingual support (`ar` RTL), PostHog tracking, Typeform surveys, lifecycle stages, benchmarks, predictive analytics roadmap
Microservices	Analytics (RFM calculations, lifecycle tagging, score history, nudges, benchmarks, simulation), Admin (configuration, reprocessing), Frontend (UI, benchmarks, explainability), Points, gRPC communication
Backend	GraphQL/REST APIs, partitioning, RBAC, input validation, time-weighted recency, multi-segment support, smart nudges, Redis Streams, Bull queues, circuit breakers, DLQ, token bucket for bursts
Rust	Shopify Functions with Sentry logging, rate limit handling, batch processing, A/B testing, smart nudges
Frontend	WCAG 2.1 compliance, i18next multilingual UI (`ar` RTL), Chart.js previews with benchmarks/time-series (US-MD12), Polaris UI, service fallback, dynamic locale detection, gamification, persona defaults, A/B templates, explainability

Analytics	Real-time previews, materialized views, A/B testing, churn prediction, benchmarks, simulation mode
Testing	Edge cases, Plus-scale, Black Friday, concurrency, OWASP ZAP, performance alerts (median >1s, P95 >3s), simulation mode
Deployment	Docker Compose, Kubernetes for Plus, feature flags, Sentry/Prometheus monitoring, Chaos Mesh, API rate-limiting, 90-day backups
Docs	Multilingual guides (`ar` RTL), GDPR tips, benchmark guidance, OpenAPI specs, gRPC proto files, developer guide
Rollout	Phased rollout with Plus prioritization, PostHog/Zendesk tracking, gamified wizard with persona defaults, tiered Plus pricing ($49–$99/month)