RFM Configuration Implementation Plan: Herethere Loyalty App

Contents

1 Overview

This document outlines the task list for implementing the RFM (Recency, Frequency, Monetary) configuration feature in the `Herethere Loyalty` Shopify app, targeting small and medium-sized merchants (100–10,000 customers, AOV $20–$200). The implementation leverages the existing tech stack (`NestJS` with TypeScript, `Vite + React`, Rust with Shopify Functions, `PostgreSQL`, `Redis`) and is broken into actionable steps across backend, frontend, Shopify Functions, and testing. The tasks prioritize delivering a minimum viable feature quickly while allowing for iterative improvements.

2 Phase 1: Planning and Setup

Goal: Establish the foundation for RFM configuration, ensuring alignment with merchant needs and tech stack.

2.1 Enhancements & Best Practices

- Interview 2–3 merchants before finalizing requirements to ensure default thresholds and UI match real-world needs.

2.2 Define Feature Requirements

Description: Finalize the scope of RFM configuration, focusing on small and medium-sized merchants.

Tasks:

- Document RFM threshold inputs (e.g., Recency: days, Frequency: orders, Monetary: spend).
- Specify tier configuration (e.g., 2–5 tiers, name, RFM criteria, rewards).
- Define adjustment frequency options (daily, weekly, monthly, quarterly, event-based).
- Include notification settings (e.g., `Klaviyo` integration, email templates).
- Confirm analytics preview (e.g., segment size, repeat purchase rate).

Deliverable: Requirements document (e.g., in Notion or Google Docs).

2.3 Analyze Merchant Data Patterns

Description: Study typical purchase cycles and AOV for small/medium merchants to suggest default RFM thresholds.

Tasks:

- Use Shopify Analytics or test store data to calculate median purchase interval and AOV.
- Example: For AOV $50, suggest Monetary thresholds (5: $250+, 4: $100–$249, etc.).
- Validate with 2–3 merchant personas (e.g., pet store, fashion boutique).

Deliverable: Default RFM thresholds and tier suggestions.

2.4 Set Up Development Environment

Description: Configure the NestJS (TypeScript) backend and Vite + React frontend for development.

Tasks:

- Initialize a new feature branch (feature/rfm-config).
- Set up Vite with React, TypeScript, and Shopify Polaris for the admin UI.
- Configure NestJS server with TypeScript and PostgreSQL (using herethere_full_schema.sql).
- Install Shopify CLI for Rust-based Shopify Functions.
- Set up Redis for caching RFM scores and segment sizes.

Deliverable: Working dev environment with basic app scaffold.

3 Phase 2: Backend Development (NestJS/TypeScript)

Goal: Build backend logic to calculate RFM scores, manage tiers, and handle adjustments.

3.1 Enhancements & Best Practices

- Use API versioning for RFM endpoints (e.g., /api/v1/rfm-config) for future compatibility.
- Add input validation and robust error handling for all config endpoints.

3.2 Integrate Shopify APIs

Description: Fetch customer and order data for RFM calculations.

Tasks:

- Set up Shopify GraphQL Admin API client in NestJS (e.g., using @shopify/shopify-api).
- Create endpoints to query Customer and Order data (e.g., GET /api/customers).
- Cache results in PostgreSQL (customers table) and Redis to reduce API calls.
- Handle pagination for stores with >100 customers.

Deliverable: API service to fetch purchase data.

3.3 Implement RFM Calculation Logic

Description: Calculate RFM scores based on merchant-defined thresholds.

Tasks:

- Define TypeScript interfaces for RFM config and scores in NestJS.
- Write a NestJS service to compute RFM scores from order data and store in customers.rfm_score (JSONB).
- Cache scores in Redis for fast retrieval.

Deliverable: RFM calculation service.

3.4 Develop Tier Assignment Logic

Description: Assign customers to tiers based on RFM scores and merchant criteria.

Tasks:

- Create a `NestJS` service to map RFM scores to tiers.
- Update customer records in `PostgreSQL` (`customers.rfm_score`) with tier assignments.

Deliverable: Tier assignment service.

3.5 Set Up Adjustment Scheduling

Description: Implement scheduled and event-based tier adjustments.

Tasks:

- Use `@nestjs/schedule` for scheduled updates (e.g., monthly on the 1st).
- Subscribe to Shopify `orders/create` webhook in `NestJS` for event-based updates.
- Implement grace period logic (e.g., delay demotions by 30 days) in `program_settings` (JSONB).

Deliverable: Scheduling and webhook handlers.

3.6 Integrate Notifications

Description: Enable tier change notifications via `Klaviyo` or Shopify email.

Tasks:

- Create `NestJS` API endpoints to send emails (e.g., `POST /api/notifications`).
- Integrate with `Klaviyo` API for email templates (e.g., "Welcome to Gold!").
- Allow merchants to customize notification templates in `program_settings.email_templates` (JSONB).

Deliverable: Notification service.

4 Phase 3: Shopify Functions (Rust)

Goal: Use Rust for performance-critical RFM updates.

4.1 Enhancements & Best Practices

- Add logging and error reporting to Rust functions for easier debugging in production.

4.2 Develop RFM Score Update Function

Description: Create a Shopify Function to update RFM scores in real-time.

Tasks:

- Set up a Rust project with Shopify Function CLI.
- Implement logic to calculate RFM scores on `orders/create` events (see `herethere_full_schema.s` for `rfm_score` structure).

- Update customer records in PostgreSQL via webhook callbacks.
- Test with Shopify's Function testing tools.

Deliverable: Deployed Shopify Function.

4.3 Optimize for Large Stores

Description: Ensure scalability for merchants with 5,000–10,000 customers.

Tasks:

- Use batch processing for RFM calculations in Rust.
- Cache results in Redis to reduce API calls.
- Test performance with simulated data (e.g., 10,000 customers).

Deliverable: Optimized test report.

5 Phase 4: Frontend Development (Vite/React)

Goal: Build an intuitive configuration UI for merchants.

5.1 Enhancements & Best Practices

- Ensure React/Polaris UI is accessible (labels, keyboard navigation, color contrast).
- Confirm the RFM config UI is mobile responsive for merchants using the Shopify app.

5.2 Design RFM Configuration UI

Description: Create a React form using Polaris for RFM thresholds, tiers, and frequency.

Tasks:

- Extend the provided React component with Vite, TypeScript, and Tailwind CSS.
- Add inputs for:
  - RFM thresholds (e.g., sliders or text fields for days/orders/spend).
  - Tier definitions (name, RFM criteria, rewards).
  - Adjustment frequency (dropdown: daily, weekly, monthly, quarterly, event-based).
  - Notification settings (email templates, integration toggle).
- Use Polaris components (e.g., TextField, Select, FormLayout).
- Implement validation (e.g., ensure Monetary 5 > Monetary 4).

Deliverable: RFM configuration form component.

5.3 Add Analytics Preview

Description: Display segment sizes and metrics to guide merchants.

Tasks:

- Create a `NestJS` API endpoint to calculate segment sizes based on draft config.
- Use `Chart.js` in the `React` frontend to visualize segment distribution (e.g., bar chart of customers per tier).
- Example: ”Gold: 100 customers (10%), Silver: 300 customers (30%)”.

Deliverable: Analytics preview component.

5.4 Enhancements & Best Practices

- Add analytics to track how merchants interact with the RFM config UI (e.g., which fields are changed, where they drop off).
- Track the performance of RFM segments over time (e.g., repeat purchase rate by tier) and surface this in analytics.

6 Phase 5: Testing and Validation

Goal: Ensure the feature works reliably for small and medium-sized merchants.

6.1 Enhancements & Best Practices

- Test with edge-case data (e.g., merchants with no orders, very high/low AOV, or customers with no email).
- For batch processing, simulate even larger stores (e.g., 20,000+ customers) to future-proof.

6.2 Unit Test Backend Logic

Description: Test RFM calculations and tier assignments.

Tasks:

- Write `Jest` tests for `NestJS` RFM calculation and tier assignment services.
- Mock Shopify API responses to simulate customer data.

Deliverable: Test suite with 80%+ coverage.

6.3 Test Shopify Function

Description: Validate Rust function for real-time updates.

Tasks:

- Use Shopify CLI to test Function with sample order data.
- Verify `PostgreSQL` updates for tier assignments.

Deliverable: Tested Shopify Function.

6.4 Test UI and UX

Description: Ensure the configuration UI is intuitive.

Tasks:

- Conduct usability testing with 2–3 mock merchants (e.g., pet store, fashion).

- Verify validation (e.g., prevent invalid thresholds).
- Test form submission and API integration.

Deliverable: Bug-free UI.

6.5 End-to-End Testing

Description: Test the full workflow from configuration to tier updates.

Tasks:

- Simulate a merchant configuring RFM thresholds and tiers.
- Trigger order events to test real-time updates.
- Verify notifications (e.g., `Klaviyo` emails).
- Test with 1,000 customers to ensure scalability.

Deliverable: End-to-end test report.

7 Phase 6: Deployment and Documentation

Goal: Launch the feature and support merchants.

7.1 Enhancements & Best Practices

- For deployment, have a rollback plan in case the new RFM config causes issues.
- Prepare support scripts/FAQs for common merchant questions about RFM.
- Generate and publish `OpenAPI/Swagger` docs for RFM endpoints.
- Include a ”Best Practices” section in the help article (e.g., how to choose thresholds, common mistakes).

7.2 Deploy Feature

Description: Release the RFM configuration to production.

Tasks:

- Deploy `NestJS` backend, `Vite + React` frontend, and Shopify Function updates to VPS using `Docker Compose`.
- Configure `Nginx` to route API requests and serve frontend assets.
- Monitor logs for errors using tools like `Sentry`.

Deliverable: Live feature deployment.

7.3 Create Documentation

Description: Provide guides for merchants to use the feature.

Tasks:

- Write a help article explaining RFM setup (e.g., ”How to Configure RFM Tiers”).
- Include examples for common merchant types (e.g., pet store: monthly, AOV $40).

- Add screenshots of the UI.

Deliverable: Help article in the app's support portal.

7.4 Pilot with Merchants

Description: Test with real merchants to gather feedback.

Tasks:

- Recruit 3–5 small/medium merchants for a beta test.
- Monitor usage (e.g., segment sizes, engagement rates).
- Collect feedback via surveys or calls.

Deliverable: Beta feedback report.

8 Phase 7: Pricing and Rollout

Goal: Make the feature accessible and profitable.

8.1 Enhancements & Best Practices

- Consider using feature flags to enable RFM config for select merchants before a full rollout.

8.2 Define Pricing Strategy

Description: Price the feature to appeal to small/medium merchants.

Tasks:

- Offer basic RFM config (2–3 tiers, monthly adjustments) in free plan.
- Include advanced features (e.g., real-time updates, custom notifications) in a paid plan ($15–$29/month).
- Compare with competitors (e.g., Loyal: free, LoyaltyLion: $399/month).

Deliverable: Pricing model.

8.3 Roll Out to All Merchants

Description: Launch the feature to all users.

Tasks:

- Announce via email and in-app banner.
- Provide a setup wizard for new merchants.
- Monitor adoption and support queries.

Deliverable: Launch campaign.

9 Timeline and Resource Estimate

Total Duration: Approximately 30–35 days (assuming 1–2 developers).

Breakdown:

- Phase 1: 5–6 days
- Phase 2: 13–14 days
- Phase 3: 5–6 days
- Phase 4: 6–7 days
- Phase 5: 9 days
- Phase 6: 3 days
- Phase 7: 3 days

Resources: 1 full-stack developer (`NestJS`/`React`, Rust), Grok AI (handling both).

10 Considerations for Small/Medium Merchants

- **Simplicity**: Provide a "Quick Setup" wizard with pre-filled thresholds (e.g., based on AOV $50, purchase cycle 30 days).
- **Affordability**: Keep advanced features affordable ($15–$29/month) to compete with Loyal (free) and Rivo (free plan).
- **Usability**: Use `Polaris` for a familiar Shopify admin experience; include tooltips and examples.
- **Scalability**: Optimize for 100–10,000 customers, as medium-sized merchants may grow quickly.
- **Support**: Offer live chat or email support to guide setup, critical for small merchants.

11 Example Merchant Workflow

For a pet store (AOV $40, 1,000 customers):

- **Configuration**:
  - Recency: 5 = <30 days, 4 = 31–60 days, etc.
  - Frequency: 5 = 5+ orders, 4 = 3–4 orders.
  - Monetary: 5 = $200+, 4 = $100–$199.
  - Tiers: Gold (5-4-4), Silver (3-2-2), Bronze (1-1-1).
  - Frequency: Monthly, 30-day grace period.
- **Outcome**: 10% in Gold (100 customers), 25% increase in repeat purchases.

12 Next Steps

1. Start with Phase 1: Finalize requirements and analyze merchant data (5–6 days).
2. Prioritize Backend: Build Shopify API integration and RFM calculation first to validate data flow.
3. Test Early: Use a test store to simulate RFM configs before building the full UI.
4. Seek Feedback: Share the UI prototype with 1–2 friendly merchants for early input.

13 Summary Table of Key Suggestions

Area	**Suggestion**
Planning	Interview merchants before finalizing requirements
Backend	API versioning, input validation, error handling
Rust	Add logging/error reporting to Shopify Functions
Frontend	Ensure accessibility and mobile responsiveness
Analytics	Track UI usage and segment performance
Testing	Test edge cases and larger stores
Deployment	Prepare rollback plan, support scripts/FAQs
Docs	Generate `OpenAPI` docs, add ”Best Practices” to merchant docs
Rollout	Use feature flags for gradual enablement

Table 1: Summary of Key Suggestions for RFM Configuration