COULD HAVE FEATURES

Polish features for global/premium merchants (Phase 6, 50,000+ customers).

1. Advanced RFM Analytics (Phase 6)

- **Goal**: Provide predictive and industry-specific insights. Success metric: 85%+ adoption of advanced analytics, 80%+ accuracy in churn predictions.
- **Features**: Dynamic RFM thresholds via no-code dashboard (US-MD11), industry-specific benchmarks (e.g., Pet Store: R5 ≤14 days, Electronics: R5 ≤30 days) in `rfm_benchmarks`, predictive churn models with ML (logistic regression, neural networks via xAI API: https://x.ai/api), lifecycle stage forecasting (new lead → repeat buyer).
- **RFM Wizard**: Polaris-compliant wizard for configuring dynamic thresholds, visualized with Chart.js (line type for score trends, bar type for segment distribution), supports A/B testing of thresholds.
- **Insights**: Cross-store RFM comparisons for Shopify Plus multi-store setups, real-time churn risk nudges, and behavioral segment forecasting.
- **Scalability**: Handles 100,000+ customers with Redis Streams (`rfm:{customer_id}`, `analytics:{merchant_id}`), PostgreSQL partitioning, and Kubernetes sharding.
- **Database Design**:
  - **Table**: `rfm_benchmarks`
    - `benchmark_id` (text, PK, NOT NULL): Unique ID.
    - `industry` (text, NOT NULL): e.g., "Pet Store", "Electronics".
    - `segment_name` (text, NOT NULL): e.g., "Champions".
    - `thresholds` (jsonb): e.g., `{"recency": "<=14", "frequency": ">=5", "monetary": ">500"}`.
  - **Table**: `rfm_score_history` (partitioned by `merchant_id`)
    - `history_id` (text, PK, NOT NULL): Unique ID.
    - `customer_id` (text, FK → `users`, NOT NULL): Customer.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `rfm_score` (jsonb): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
    - `lifecycle_stage` (text, CHECK IN ('new_lead', 'repeat_buyer', 'churned', 'vip')): Stage.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `rfm_threshold_updated`, `churn_forecasted`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"industry": "Pet Store", "threshold": "recency<=14"}`.
  - **Indexes**: `idx_rfm_benchmarks_industry` (btree: `industry`), `idx_rfm_score_history_customer_id` (btree: `customer_id`, `created_at`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **PUT** `/v1/api/rfm/thresholds` (REST) | gRPC `/analytics.v1/AnalyticsService/UpdateRFMThresholds`
    - **Input**: `{ merchant_id: string, industry: string, thresholds: object, locale: string }`

        - **Output**: `{ status: string, benchmark_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate thresholds, update `rfm_benchmarks`, cache in Redis Streams (`rfm_benchmarks:{merchant_id}`), enforce RBAC (`admin:analytics`), log in `audit_logs`, track via PostHog (`rfm_threshold_updated`, 85%+ adoption).
    - **POST** `/v1/api/rfm/forecast` (REST) | gRPC `/analytics.v1/AnalyticsService/ForecastChurn`
        - **Input**: `{ merchant_id: string, customer_ids: array, locale: string }`
        - **Output**: `{ status: string, forecasts: [{ customer_id: string, churn_score: number, lifecycle_stage: string }], error: { code: string, message: string } | null }`
        - **Flow**: Run ML model via xAI API (https://x.ai/api), update `users.churn_score`, `users.lifecycle_stage`, cache in Redis Streams (`rfm:{customer_id}`), enforce RBAC (`admin:analytics`), log in `audit_logs`, track via PostHog (`churn_forecasted`, 80%+ accuracy).
- **GraphQL Query Examples**:
    - **Query: Update RFM Thresholds**
        - **Purpose**: Updates industry-specific RFM thresholds, used in `/v1/api/rfm/thresholds`.
        - **Query**:

```
mutation UpdateRFMThresholds($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

        - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "rfm_benchmarks",
    "value": "{\"industry\": \"Pet Store\", \"segment_name\":
\"Champions\", \"thresholds\": {\"recency\": \"<=14\",
\"frequency\": \">=5\", \"monetary\": \">500\"}}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

- **Use Case**: Merchant Dashboard updates `rfm_benchmarks` via RFM Wizard, visualized with Chart.js, cached in Redis Streams (`rfm_benchmarks:{merchant_id}`), and tracked via PostHog (`rfm_threshold_updated`).

- **Query: Fetch Churn Forecast**
  - **Purpose**: Retrieves churn predictions for customers, used in `/v1/api/rfm/forecast`.
  - **Query**:

```
query GetChurnForecast($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key: "churn_score") {
          value
        }
        metafield(namespace: "loyalnest", key: "lifecycle_stage")
{
          value
        }
      }
    }
  }
}
```

  - **Variables**: `{ "first": 50, "query": "tag:at_risk" }`
  - **Use Case**: Merchant Dashboard fetches `users.churn_score` and `lifecycle_stage` for real-time churn risk nudges, powered by xAI API, cached in Redis Streams (`rfm:{customer_id}`), and tracked via PostHog (`churn_forecasted`).

- **Service**: Analytics Service (gRPC: `/analytics.v1/*`, Dockerized).
- **Chart**: RFM Segment Distribution

```
{
  "type": "bar",
  "data": {
    "labels": ["Champions", "Loyal", "At-Risk", "New", "Inactive"],
    "datasets": [{
      "label": "Customer Count",
      "data": [500, 1000, 750, 300, 200],
      "backgroundColor": ["#4CAF50", "#2196F3", "#FF9800", "#9C27B0",
"#F44336"],
      "borderColor": ["#388E3C", "#1976D2", "#F57C00", "#7B1FA2",
"#D32F2F"],
      "borderWidth": 1
    }]
  },
  "options": {
    "scales": {
```

```
        "y": {
          "beginAtZero": true,
          "title": { "display": true, "text": "Customer Count" }
        },
        "x": {
          "title": { "display": true, "text": "RFM Segment" }
        }
      },
      "plugins": {
        "legend": { "display": false },
        "title": { "display": true, "text": "RFM Segment Distribution" }
      }
    }
  }
```

2. Multi-Store Sync (Phase 6)

- **Goal**: Enable seamless loyalty across Shopify Plus multi-store setups. Success metric: 15%+ adoption, 99%+ sync accuracy.
- **Features**: Sync points, VIP tiers, and referrals across stores (US-CW6), managed via no-code dashboard (US-MD2). Supports cross-store RFM analytics and shared leaderboards.
- **Notifications**: Notify customers of shared points across stores via Klaviyo/Postscript (3 retries, `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Scalability**: Handles 100,000+ customers with Redis Streams (`points:{customer_id}`, `tiers:{customer_id}`), PostgreSQL partitioning, and Kubernetes sharding.
- **Database Design**:
  - **Table**: `program_settings`
    - `multi_store_config` (jsonb): e.g., `{"shared_stores": ["store1.myshopify.com", "store2.myshopify.com"], "shared_leaderboard": true}`.
  - **Table**: `users`
    - `shared_customer_id` (text, UNIQUE): Cross-store customer ID.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `multi_store_sync`, `shared_leaderboard_updated`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"store_id": "store1.myshopify.com", "customer_id": "C123"}`.
  - **Indexes**: `idx_program_settings_merchant_id` (btree: `merchant_id`), `idx_users_shared_customer_id` (btree: `shared_customer_id`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/points/sync` (REST) | gRPC `/points.v1/PointsService/SyncPoints`
    - **Input**: `{ customer_id: string, store_ids: array, locale: string }`
    - **Output**: `{ status: string, points_balance: number, error: { code: string, message: string } | null }`
    - **Flow**: Sync `points_balance`, `vip_tiers`, `referrals` across stores, update `points_transactions`, cache in Redis Streams (`points:{customer_id}`), enforce RBAC

(`admin:points`), notify via Klaviyo/Postscript, log in `audit_logs`, track via PostHog (`multi_store_sync`, 15%+ adoption).

- **GraphQL Query Examples**:
  - **Query: Sync Customer Points Across Stores**
    - **Purpose**: Synchronizes customer points across multiple stores, used in `/v1/api/points/sync`.
    - **Query**:

```
mutation UpdateCustomerPoints($input: CustomerInput!) {
  customerUpdate(input: $input) {
    customer {
      id
      metafield(namespace: "loyalnest", key: "points_balance") {
        value
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

    - **Variables**:

```
{
  "input": {
    "id": "gid://shopify/Customer/987654321",
    "metafields": [
      {
        "namespace": "loyalnest",
        "key": "points_balance",
        "value": "500",
        "type": "number_integer"
      },
      {
        "namespace": "loyalnest",
        "key": "shared_customer_id",
        "value": "SHARED_C123",
        "type": "string"
      }
    ]
  }
}
```

    - **Use Case**: Customer Widget syncs `points_balance` across stores in `program_settings.multi_store_config`, cached in Redis Streams (`points:{customer_id}`), and tracked via PostHog (`multi_store_sync`).

    - **Query: Fetch Shared Leaderboard Data**
        - **Purpose**: Retrieves shared leaderboard data for multi-store setups, used in `/v1/api/points/sync`.
        - **Query**:

```
query GetSharedLeaderboard($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key:
"leaderboard_score") {
          value
        }
        metafield(namespace: "loyalnest", key:
"shared_customer_id") {
          value
        }
      }
    }
  }
}
```

        - **Variables**: `{ "first": 10, "query": "tag:shared_leaderboard_2025-07" }`
        - **Use Case**: Merchant Dashboard displays shared leaderboard rankings, using `users.shared_customer_id`, visualized with Chart.js, and cached in Redis Streams (`leaderboard:{merchant_id}:{cycle}`).
- **Service**: Points Service (gRPC: `/points.v1/*`, Dockerized).

3. Merchant Referral Program (Phase 6)

- **Goal**: Encourage merchants to refer others, enhancing Phase 5 groundwork. Success metric: 5%+ referral conversion rate, 80%+ adoption in “LoyalNest Collective” Slack community.
- **Features**: Merchants earn rewards (e.g., 1 month free subscription) for referring new merchants, managed via Admin Module (US-AM9). Referral links generated via `ReferralService`, tracked in Merchant Dashboard with Chart.js (line type for referral trends).
- **Slack Community**: “LoyalNest Collective” for merchants to share tips, access beta features, and receive referral rewards, integrated via Slack API.
- **Notifications**: Notify merchants of referral status via Klaviyo/Postscript (3 retries, `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Scalability**: Handles 10,000+ merchants with Redis Streams (`merchant_referral:{referral_code}`), PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
    - **Table**: `merchant_referrals` (partitioned by `merchant_id`)
        - `merchant_referral_id` (text, PK, NOT NULL): Unique ID.
        - `advocate_merchant_id` (text, FK → `merchants`, NOT NULL): Referring merchant.
        - `referred_merchant_id` (text, FK → `merchants`, NOT NULL): Referred merchant.
        - `reward_id` (text, NOT NULL): Reward (e.g., `1_month_free`).

        - `status` (text, CHECK IN ('pending', 'completed', 'expired')): Status.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `merchant_referral_created`, `merchant_referral_completed`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"merchant_referral_id": "MER123", "reward": "1_month_free"}`.
    - **Indexes**: `idx_merchant_referrals_advocate_merchant_id` (btree: `advocate_merchant_id`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/merchant-referrals/create` (REST) | gRPC `/referrals.v1/ReferralService/CreateMerchantReferral`
        - **Input**: `{ advocate_merchant_id: string, locale: string }`
        - **Output**: `{ status: string, referral_code: string, error: { code: string, message: string } | null }`
        - **Flow**: Insert into `merchant_referrals`, generate referral link, cache in Redis Streams (`merchant_referral:{referral_code}`), notify via Klaviyo/Postscript, enforce RBAC (`admin:referrals`), log in `audit_logs`, track via PostHog (`merchant_referral_created`, 5%+ conversion).
    - **GET** `/v1/api/merchant-referrals/analytics` (REST) | gRPC `/analytics.v1/AnalyticsService/GetMerchantReferralAnalytics`
        - **Input**: `{ merchant_id: string, date_range: { start: string, end: string } }`
        - **Output**: `{ status: string, analytics: { clicks: number, conversions: number, ctr: number }, error: { code: string, message: string } | null }`
        - **Flow**: Query `merchant_referrals`, generate Chart.js data, cache in Redis Streams (`merchant_referral_analytics:{merchant_id}`), enforce RBAC (`admin:analytics`), log in `audit_logs`, track via PostHog (`merchant_referral_analytics_viewed`, 80%+ view rate).
- **GraphQL Query Examples**:
    - **Query: Create Merchant Referral**
        - **Purpose**: Generates a referral link for a merchant, used in `/v1/api/merchant-referrals/create`.
        - **Query**:

```
mutation CreateMerchantReferral($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
```

```
    }
  }
}
```

- **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "merchant_referral",
    "value": "{\"referral_code\": \"MER123\", \"reward_id\":
\"1_month_free\", \"status\": \"pending\"}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

- **Use Case**: Admin Module creates a referral link in `merchant_referrals`, notifies via Klaviyo/Postscript, and tracks via PostHog (`merchant_referral_created`).

- **Query: Fetch Referral Analytics**
  - **Purpose**: Retrieves referral analytics for a merchant, used in `/v1/api/merchant-referrals/analytics`.
  - **Query**:

```
query GetReferralAnalytics($id: ID!) {
  shop(id: $id) {
    id
    metafield(namespace: "loyalnest", key: "referral_analytics") {
      value
    }
  }
}
```

  - **Variables**: `{ "id": "gid://shopify/Shop/123456789" }`
  - **Use Case**: Merchant Dashboard displays referral trends (clicks, conversions) with Chart.js, cached in Redis Streams (`merchant_referral_analytics:{merchant_id}`), and tracked via PostHog (`merchant_referral_analytics_viewed`).

- **Service**: Referrals Service (gRPC: `/referrals.v1/*`, Dockerized).

4. Advanced Gamification (Phase 6)

- **Goal**: Enhance customer engagement with immersive gamification, building on Phase 3 gamification. Success metric: 20%+ engagement rate, 85%+ leaderboard interaction rate.
- **Features**: Dynamic badges (e.g., “Seasonal Star” for holiday purchases), team-based leaderboards (e.g., group challenges), and achievement streaks (e.g., 5 consecutive purchases). Displayed in Customer Widget with Chart.js (line type for streaks, bar type for team rankings).

- **Integration**: Sync with Shopify Flow for automated badge triggers (e.g., "Purchase → Award Badge") and Klaviyo/Postscript for notifications (3 retries, `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Scalability**: Handles 100,000+ customers with Redis Streams (`badge:{customer_id}`, `leaderboard:{merchant_id}:{cycle}`), PostgreSQL partitioning, and Kubernetes sharding.
- **Database Design**:
  - **Table**: `gamification_achievements` (partitioned by `merchant_id`)
    - `achievement_id` (text, PK, NOT NULL): Unique ID.
    - `customer_id` (text, FK → `users`, NOT NULL): Customer.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `badge` (jsonb, CHECK ?| ARRAY['en', `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`, `ko`, `uk`, `hu`, `sv`, `he`]): e.g., `{"en": "Seasonal Star", "ar": "..."}`.
    - `streak_count` (integer, CHECK >= 0): Streak count.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `team_leaderboards` (partitioned by `merchant_id`)
    - `leaderboard_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `team_id` (text, NOT NULL): Team identifier.
    - `score` (integer, CHECK >= 0): Team score.
    - `cycle` (text, NOT NULL): e.g., `2025-07`.
    - `updated_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `streak_earned`, `team_leaderboard_updated`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"streak_count": 5, "team_id": "TEAM123"}`.
  - **Indexes**: `idx_gamification_achievements_customer_id` (btree: `customer_id`, `created_at`), `idx_team_leaderboards_merchant_id_cycle` (btree: `merchant_id`, `cycle`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/gamification/streak` (REST) | gRPC `/analytics.v1/AnalyticsService/UpdateStreak`
    - **Input**: `{ customer_id: string, action_type: string, locale: string }`
    - **Output**: `{ status: string, streak_count: number, badge: string, error: { code: string, message: string } | null }`
    - **Flow**: Update `gamification_achievements.streak_count`, notify via Klaviyo/Postscript, cache in Redis Streams (`streak:{customer_id}`), enforce RBAC (`admin:gamification`), log in `audit_logs`, track via PostHog (`streak_earned`, 20%+ engagement).
  - **POST** `/v1/api/team-leaderboards` (REST) | gRPC `/analytics.v1/AnalyticsService/UpdateTeamLeaderboard`
    - **Input**: `{ merchant_id: string, team_id: string, score: number, cycle: string, locale: string }`
    - **Output**: `{ status: string, leaderboard_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Update `team_leaderboards`, Redis sorted set (`team_leaderboard:{merchant_id}:{cycle}`), generate Chart.js data, cache in Redis Streams, enforce RBAC

(`admin:gamification`), log in `audit_logs`, track via PostHog (`team_leaderboard_updated`, 85%+ interaction).

- **GraphQL Query Examples**:
  - **Query: Update Achievement Streak**
    - **Purpose**: Updates customer streak count for gamification, used in `/v1/api/gamification/streak`.
    - **Query**:

```
mutation UpdateStreak($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

    - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "achievement_streak",
    "value": "{\"streak_count\": 5, \"badge\": {\"en\": \"Seasonal Star\"}}",
    "ownerId": "gid://shopify/Customer/987654321",
    "type": "json"
  }
}
```

    - **Use Case**: Customer Widget updates `gamification_achievements.streak_count`, triggers Shopify Flow for badge awards, and tracks via PostHog (`streak_earned`).
  - **Query: Fetch Team Leaderboard**
    - **Purpose**: Retrieves team-based leaderboard rankings, used in `/v1/api/team-leaderboards`.
    - **Query**:

```
query GetTeamLeaderboard($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
```

```
        node {
          id
          metafield(namespace: "loyalnest", key: "team_leaderboard")
{
            value
          }
        }
      }
    }
}
```

    - **Variables**: `{ "first": 10, "query": "tag:team_leaderboard_2025-07" }`
    - **Use Case**: Customer Widget displays team rankings, visualized with Chart.js, cached in Redis Streams (`team_leaderboard:{merchant_id}:{cycle}`), and tracked via PostHog (`team_leaderboard_updated`).
- **Service**: Analytics Service (gRPC: `/analytics.v1/*`, Dockerized).

5. Theme App Extensions (Phase 6)

- **Goal**: Enhance storefront integration with native Shopify themes. Success metric: 80%+ adoption, <1s render time.
- **Features**: Embed loyalty widgets (rewards panel, points display, sticky bar, post-purchase widget) directly in Shopify themes via Theme App Extensions, reducing JavaScript load (US-CW15). Supports dynamic placement and A/B testing via no-code dashboard (US-MD2).
- **Accessibility**: WCAG 2.1 compliance, ARIA labels (e.g., `aria-label="View points balance"`), RTL for `ar`, `he`, Lighthouse CI score 90+ (LCP, FID, CLS).
- **Scalability**: Renders <1s for 100,000+ customers via Redis Streams (`content:{merchant_id}:{locale}`), supports 10,000 orders/hour with circuit breakers and Chaos Mesh testing.
- **Database Design**:
    - **Table**: `program_settings`
        - `theme_extension_config` (jsonb): e.g., `{"widget": "rewards_panel", "placement": "cart_page"}`.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `theme_extension_configured`, `theme_widget_rendered`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"widget": "rewards_panel", "placement": "cart_page"}`.
    - **Indexes**: `idx_program_settings_merchant_id` (btree: `merchant_id`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **PUT** `/v1/api/theme-extensions` (REST) | gRPC `/frontend.v1/FrontendService/ConfigureThemeExtension`
        - **Input**: `{ merchant_id: string, theme_extension_config: object, locale: string }`
        - **Output**: `{ status: string, preview: object, error: { code: string, message: string } | null }`

        - **Flow**: Update `program_settings.theme_extension_config` using Shopify's Theme App Extension API, cache in Redis Streams (`theme:{merchant_id}:{locale}`), enforce RBAC (`admin:frontend`), log in `audit_logs`, track via PostHog (`theme_extension_configured`, 80%+ adoption).
- **GraphQL Query Examples**:
    - **Query: Configure Theme Extension**
        - **Purpose**: Configures widget placement in Shopify themes, used in `/v1/api/theme-extensions`.
        - **Query**:

```
mutation ConfigureThemeExtension($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

        - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "theme_extension_config",
    "value": "{\"widget\": \"rewards_panel\", \"placement\":
\"cart_page\"}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

        - **Use Case**: Merchant Dashboard configures widget placement via Theme App Extensions, ensuring <1s render time, and tracks via PostHog (`theme_extension_configured`).
    - **Query: Fetch Widget Configuration**
        - **Purpose**: Retrieves theme extension settings for rendering, used in `/v1/api/theme-extensions`.
        - **Query**:

```
query GetThemeExtensionConfig($id: ID!) {
  shop(id: $id) {
    id
    metafield(namespace: "loyalnest", key:
"theme_extension_config") {
      value
    }
  }
}
```

    - **Variables**: `{ "id": "gid://shopify/Shop/123456789" }`
    - **Use Case**: Customer Widget renders loyalty widgets based on `program_settings.theme_extension_config`, cached in Redis Streams (`theme:{merchant_id}:{locale}`).
- **Service**: Frontend Service (gRPC: `/frontend.v1/*`, Dockerized).

6. Slack Community Integration (Phase 6)

- **Goal**: Foster merchant engagement via “LoyalNest Collective”. Success metric: 80%+ merchant participation, 10%+ beta feature adoption.
- **Features**: Slack community for merchants to share tips, access beta features (e.g., new RFM models, gamification streaks), and manage merchant referrals. Supports automated onboarding notifications, referral rewards, and analytics via Slack API.
- **Notifications**: Real-time updates for referral completions, beta feature releases via Slack webhooks, localized (`en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Scalability**: Handles 10,000+ merchants with Redis Streams (`slack:{merchant_id}`), PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
    - **Table**: `slack_community` (partitioned by `merchant_id`)
        - `community_id` (text, PK, NOT NULL): Unique ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `slack_user_id` (text, UNIQUE, NOT NULL): Slack user ID.
        - `status` (text, CHECK IN ('active', 'invited', 'inactive')): Status.
        - `beta_features` (jsonb): e.g., `{"rfm_advanced": true, "streaks": true}`.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `slack_joined`, `beta_feature_enabled`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"slack_user_id": "U123", "beta_feature": "rfm_advanced"}`.
    - **Indexes**: `idx_slack_community_merchant_id` (btree: `merchant_id`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/slack/join` (REST) | gRPC `/admin.v1/AdminService/JoinSlackCommunity`
        - **Input**: `{ merchant_id: string, slack_user_id: string, locale: string }`

        - **Output**: `{ status: string, community_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Insert into `slack_community`, send Slack invite via API, cache in Redis Streams (`slack:{merchant_id}`), enforce RBAC (`admin:community`), log in `audit_logs`, track via PostHog (`slack_joined`, 80%+ participation).
    - **POST** `/v1/api/slack/beta` (REST) | gRPC `/admin.v1/AdminService/EnableBetaFeature`
        - **Input**: `{ merchant_id: string, feature: string, locale: string }`
        - **Output**: `{ status: string, error: { code: string, message: string } | null }`
        - **Flow**: Update `slack_community.beta_features`, notify via Slack webhook, cache in Redis Streams (`beta:{merchant_id}`), enforce RBAC (`admin:community`), log in `audit_logs`, track via PostHog (`beta_feature_enabled`, 10%+ adoption).
- **GraphQL Query Examples**:
    - **Query: Join Slack Community**
        - **Purpose**: Adds a merchant to the Slack community, used in `/v1/api/slack/join`.
        - **Query**:

```
mutation JoinSlackCommunity($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

        - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "slack_community",
    "value": "{\"slack_user_id\": \"U123\", \"status\": \"invited\"}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

        - **Use Case**: Admin Module adds merchant to `slack_community`, sends Slack invite, and tracks via PostHog (`slack_joined`).

    - **Query: Enable Beta Feature**
        - **Purpose**: Enables beta feature access for a merchant, used in `/v1/api/slack/beta`.
        - **Query**:

```
mutation EnableBetaFeature($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

        - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "slack_beta_features",
    "value": "{\"rfm_advanced\": true, \"streaks\": true}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

        - **Use Case**: Admin Module updates `slack_community.beta_features`, notifies via Slack webhook, and tracks via PostHog (`beta_feature_enabled`).
- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).

7. Apple Wallet Export (Phase 6)

- **Goal**: Enhance customer UX with mobile wallet integration. Success metric: 20%+ adoption for Shopify Plus merchants, 90%+ export success rate.
- **Features**: Allows customers to save loyalty balance and referral QR code to Apple Wallet via Storefront API, displayed in Customer Widget (US-CW15). Generates `.pkpass` file with `points_balance` and `referral_code` using Shopify's infrastructure. Supports Google Wallet for broader compatibility.
- **Notifications**: Notify customers of wallet export via Klaviyo/Postscript (3 retries, `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Accessibility**: WCAG 2.1 compliance with ARIA labels (e.g., `aria-label="Export loyalty balance to Apple Wallet"`), RTL for `ar`, `he`, Lighthouse CI score 90+ (LCP, FID, CLS).

- **Scalability**: Handles 100,000+ customers with Redis Streams (`wallet:{customer_id}`), PostgreSQL partitioning, and Kubernetes sharding.
- **Database Design**:
  - **Table**: `wallet_exports` (partitioned by `merchant_id`)
    - `export_id` (text, PK, NOT NULL): Unique ID.
    - `customer_id` (text, FK → `users`, NOT NULL): Customer.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `wallet_type` (text, CHECK IN ('apple', 'google')): Wallet type.
    - `pass_data` (jsonb, AES-256 ENCRYPTED): e.g., `{"points_balance": 500, "referral_code": "REF123"}`.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `wallet_exported`, `wallet_export_failed`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"wallet_type": "apple", "points_balance": 500}`.
  - **Indexes**: `idx_wallet_exports_customer_id` (btree: `customer_id`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/wallet/export` (REST) | gRPC `/frontend.v1/FrontendService/ExportToWallet`
    - **Input**: `{ customer_id: string, wallet_type: string, locale: string }`
    - **Output**: `{ status: string, pass_url: string, error: { code: string, message: string } | null }`
    - **Flow**: Generate `.pkpass` file with `points_balance` (from `users`) and `referral_code` (from `referral_links`) using Shopify's infrastructure, encrypt with AES-256, insert into `wallet_exports`, notify via Klaviyo/Postscript, cache in Redis Streams (`wallet:{customer_id}`), enforce RBAC (`customer:frontend`), log in `audit_logs`, track via PostHog (`wallet_exported`, 20%+ adoption).
- **GraphQL Query Examples**:
  - **Query: Export to Apple Wallet**
    - **Purpose**: Generates a wallet pass for a customer, used in `/v1/api/wallet/export`.
    - **Query**:

```
mutation ExportToWallet($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
```

```
    }
  }
```

- **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "wallet_export",
    "value": "{\"wallet_type\": \"apple\", \"points_balance\":
500, \"referral_code\": \"REF123\"}",
    "ownerId": "gid://shopify/Customer/987654321",
    "type": "json"
  }
}
```

- **Use Case**: Customer Widget generates `.pkpass` file for Apple Wallet, stored in `wallet_exports`, and tracks via PostHog (`wallet_exported`).

- **Query: Fetch Wallet Export Status**
  - **Purpose**: Retrieves wallet export details, used in `/v1/api/wallet/export`.
  - **Query**:

```
query GetWalletExport($id: ID!) {
  customer(id: $id) {
    id
    metafield(namespace: "loyalnest", key: "wallet_export") {
      value
    }
  }
}
```

  - **Variables**: `{ "id": "gid://shopify/Customer/987654321" }`
  - **Use Case**: Customer Widget verifies `wallet_exports` data, ensuring successful export, and notifies via Klaviyo/Postscript.

- **Service**: Frontend Service (gRPC: `/frontend.v1/*`, Dockerized).