Enda Tamweel HTS-Based Loyalty Tokenization Platform

Architecture Overview

Version 1.0, 2025-12-17

Table of Contents

Chapter 1. Introduction and Goals

1.1. Executive Summary

The Enda Tamweel HTS-Based Loyalty Tokenization Platform is a loyalty and client fidelization system that manages client points using traditional databases combined with Hedera Hashgraph's HTS (Hedera Token Service). Points are tokenized as mintable and burnable HTS tokens that reflect the points accumulated by each client.

The platform integrates Enda Tamweel's existing T24 and Works systems through an intermediate consolidated database (Middle DB), updated daily (J-1). Based on extracted data, daily points are calculated for each client, stored in the application database, and reflected in their on-chain token balances.

Key capabilities include:

- Automatic Hedera account creation for each registered client
- Daily points calculation based on client activity from the Middle DB
- Automatic minting of HTS loyalty tokens to client accounts
- Gift redemption workflow with admin approval and token burning
- Multi-tier role management (Client, Admin, Super Admin)
- Secure JWT-based authentication with token rotation

1.2. Stakeholders

Role	Contact	Expectations
Enda Tamweel Management	Enda Tamweel Team	Successful loyalty program that increases client engagement and retention
Product Owner	Dar Blockchain Team	Feature completeness and seamless integration with existing T24/Works systems
Project Manager	Dar Blockchain Team	On-time delivery within budget with clear milestone tracking
Tech Lead	Dar Blockchain Team	Code quality, Hedera integration, and technical implementation
Backend Developers	Dar Blockchain Team	Clear API specifications and maintainable codebase
Frontend Developers	Dar Blockchain Team	Intuitive UI/UX for clients and administrators
System Administrators	Enda Tamweel IT	Easy deployment, monitoring, and maintenance

Role	Contact	Expectations
Clients (End Users)	Enda Tamweel Customers	Easy access to loyalty points, transparent balance tracking, and simple gift redemption

1.3. Requirements Overview

1.3.1. Functional Requirements

ID	Requirement
R1	Automatic Hedera wallet creation upon client registration with secure private key storage
R2	Daily extraction of client activity data from Middle DB (J-1 refresh cycle)
R3	Rule-based points calculation engine processing Middle DB data
R4	Automatic minting of HTS loyalty tokens based on calculated points
R5	Gift redemption workflow with Admin approval and token burning
R6	Super Admin panel for Admin lifecycle management
R7	JWT-based authentication with refresh token rotation
R8	Regular reconciliation between database and on-chain token balances
R9	Storage of historical point calculations and redemption logs
R10	Web and Mobile application interfaces for clients and administrators

1.3.2. Integration Requirements

ID	Requirement
IR1	Integration with Enda Tamweel’s Middle DB (consolidated T24 and Works data)
IR2	Integration with Hedera Token Service (HTS) for token management
IR3	Hedera Account Service for wallet creation and management

1.4. Quality Goals

ID	Quality	Description
Q1	Security	Private keys encrypted at rest; JWT tokens with short expiry; refresh tokens stored hashed in DB; HTTPS-only communication; HTTP-only secure cookies for refresh tokens
Q2	Reliability	Daily batch processing must complete successfully; reconciliation between DB and on-chain balances; comprehensive audit logging
Q3	Performance	API responses within 2 seconds for 95% of requests; batch minting operations optimized for large user bases

ID	Quality	Description
Q4	Scalability	Asynchronous job queue for daily point calculations; batch mint operations to handle growing client base
Q5	Availability	System available during business hours with scheduled maintenance windows for batch operations
Q6	Maintainability	Clean separation between business logic and Hedera integration; comprehensive logging for troubleshooting
Q7	Auditability	Complete audit trail for all token minting, burning, and redemption operations; login/logout event logging
Q8	Data Integrity	Regular reconciliation between application database and on-chain token balances

1.5. Glossary

Term	Description
HTS	Hedera Token Service - A native tokenization service on Hedera Hashgraph for creating and managing fungible and non-fungible tokens
DLT	Decentralised Ledger Technology
JWT	JSON Web Token - A compact, URL-safe means of representing claims to be transferred between two parties
Middle DB	Intermediate consolidated database that aggregates data from Enda Tamweel's T24 and Works systems, updated daily (J-1)
T24	Temenos T24 - Core banking system used by Enda Tamweel
Works	Enda Tamweel's operational system integrated with T24
J-1	Day minus one - refers to data from the previous business day
HBAR	The native cryptocurrency of the Hedera network used to pay for transaction fees
Minting	The process of creating new tokens on the Hedera network
Burning	The process of permanently removing tokens from circulation on the Hedera network
[[Refresh Token]]Refresh Token	A long-lived token used to obtain new access tokens without requiring the user to re-authenticate

Chapter 2. Architecture

NOTE	List key constraints that influence architectural decisions. Keep this list updated as project constraints evolve.

2.1. Technical Constraints

- Must integrate with Hedera Token Service (HTS) for token minting and burning
- Must integrate with Enda Tamweel's existing Middle DB (read-only access)
- Must support secure storage of client Hedera private keys
- Must implement JWT-based authentication with secure refresh token handling
- Daily batch processing must align with Middle DB J-1 update cycle
- Token operations must be performed by platform-controlled admin keys only

2.2. Organizational Constraints

- Integration with existing Enda Tamweel T24 and Works systems
- Must align with Enda Tamweel's security and compliance requirements
- Development by Dar Blockchain Team with defined delivery timeline
- Points cannot be used externally - closed ecosystem

2.3. Conventions

- Backend developed using NestJS framework
- Frontend using React (Web) and React Native (Mobile)
- PostgreSQL for application database
- RESTful API design principles
- Comprehensive logging and audit trails
- Asynchronous job processing for batch operations

NOTE	This section presents the static structure of the system using C4 model diagrams. Each level provides increasing detail about the software architecture. Update these diagrams when adding, removing, or significantly changing system components.

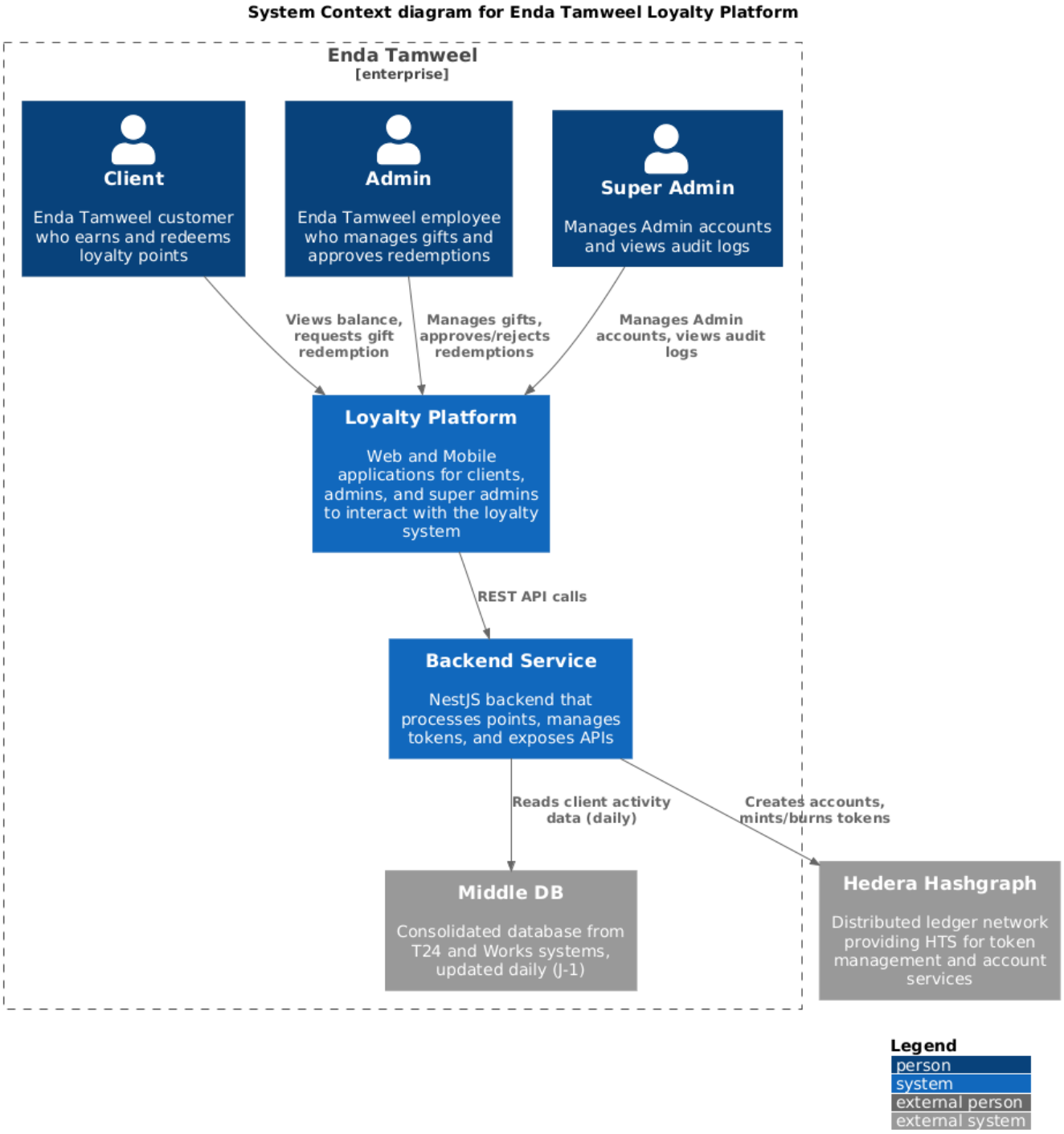
2.4. System Context

2.4.1. Overview

This diagram shows the systems that are communicating and interacting within the scope of the Enda Tamweel Loyalty Tokenization Platform. The platform bridges traditional banking systems

with Hedera's blockchain-based tokenization.

2.4.2. Context Diagram



2.4.3. Key Elements

Client

Enda Tamweel customers who participate in the loyalty program. Clients can:

- View their current loyalty point balance (both in-app and on-chain)
- Browse available gifts in the catalog

- Request gift redemptions
- Track their redemption history

Admin

Enda Tamweel employees responsible for managing the loyalty program. Admins can:

- View and manage the gift catalog
- Review and approve/reject client redemption requests
- Manually create client accounts when needed
- View client transaction history

Super Admin

Senior administrators with elevated privileges. Super Admins can:

- Create, modify, and deactivate Admin accounts
- View comprehensive audit logs
- Access system-wide reports and analytics

Loyalty Platform (Frontend)

The user-facing applications built with React (Web) and React Native (Mobile). The frontend provides:

- Responsive web application for desktop and tablet users
- Native mobile applications for iOS and Android
- Role-based interfaces for Clients, Admins, and Super Admins
- Real-time balance display and transaction history

Backend Service

The core NestJS application that orchestrates all platform operations:

- Fetches and processes data from Middle DB
- Calculates daily points based on business rules
- Manages Hedera account creation and token operations
- Exposes secure REST APIs for all frontend operations
- Handles authentication and authorization

Middle DB (External)

Enda Tamweel's consolidated database that aggregates data from:

- T24 core banking system

- Works operational system

Updated daily with J-1 data cycle. The platform has read-only access for extracting client activity information.

Hedera Hashgraph (External)

The distributed ledger network providing:

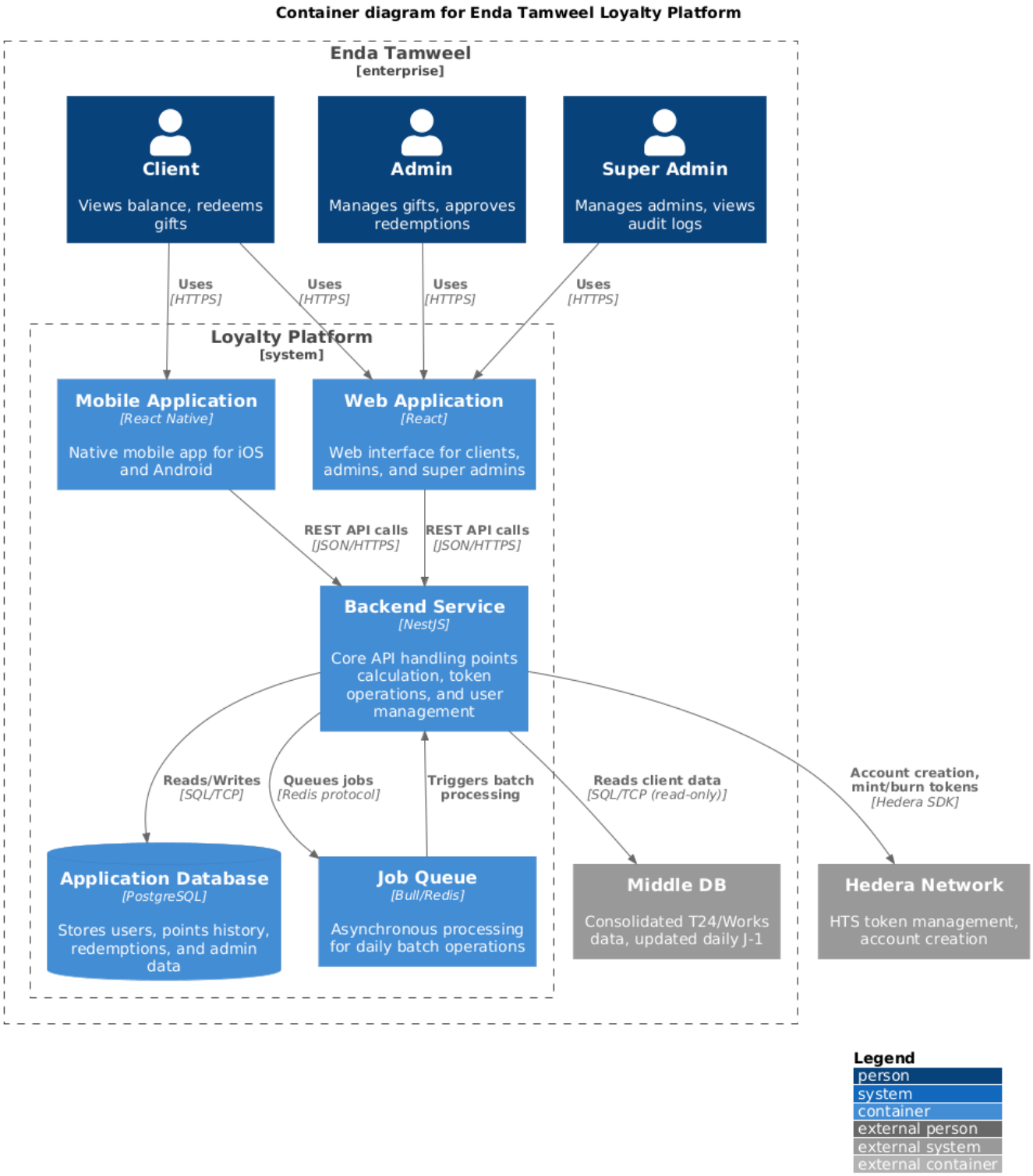
- Hedera Token Service (HTS) for the loyalty token
- Account creation for each registered client
- Immutable transaction records for minting and burning operations

2.5. Container View

2.5.1. Overview

This diagram shows the container images and their interactions within the Enda Tamweel Loyalty Tokenization Platform.

2.5.2. Container Diagram



2.5.3. Description of Containers

Web Application

Built with React, the web application provides:

- Responsive design for desktop and tablet browsers
- Role-based dashboards (Client, Admin, Super Admin)
- Real-time balance display

- Gift catalog and redemption workflow
- Admin management interface for Super Admins

Key design choices:

- JWT access tokens stored in memory only (not localStorage) to prevent XSS attacks
- Refresh tokens handled via HTTP-only secure cookies
- Modern React patterns with hooks and context

Mobile Application

Built with React Native for cross-platform deployment:

- Native iOS and Android applications
- Shared codebase with platform-specific optimizations
- Push notifications for redemption status updates
- Offline-capable balance caching

Backend Service

The core NestJS application responsible for:

- **User Management**: Registration, authentication, Hedera wallet creation
- **Points Engine**: Daily calculation based on Middle DB data
- **Token Operations**: Minting loyalty tokens to user accounts, burning on redemption
- **Gift Management**: Catalog CRUD, redemption workflow
- **Admin Management**: Admin lifecycle for Super Admins

Key design choices:

- Modular NestJS architecture for separation of concerns
- Hedera SDK integration for all blockchain operations
- Encrypted private key storage for user wallets
- Comprehensive audit logging

Job Queue

Asynchronous job processing using Bull with Redis:

- Daily scheduled jobs for Middle DB data extraction
- Batch points calculation processing
- Batch token minting operations
- Retry logic for failed Hedera transactions

Application Database

PostgreSQL database storing:

- User accounts and encrypted wallet credentials
- Points calculation history (daily snapshots and cumulative values)
- Gift catalog and redemption requests
- Admin and Super Admin accounts
- Audit logs for all significant operations

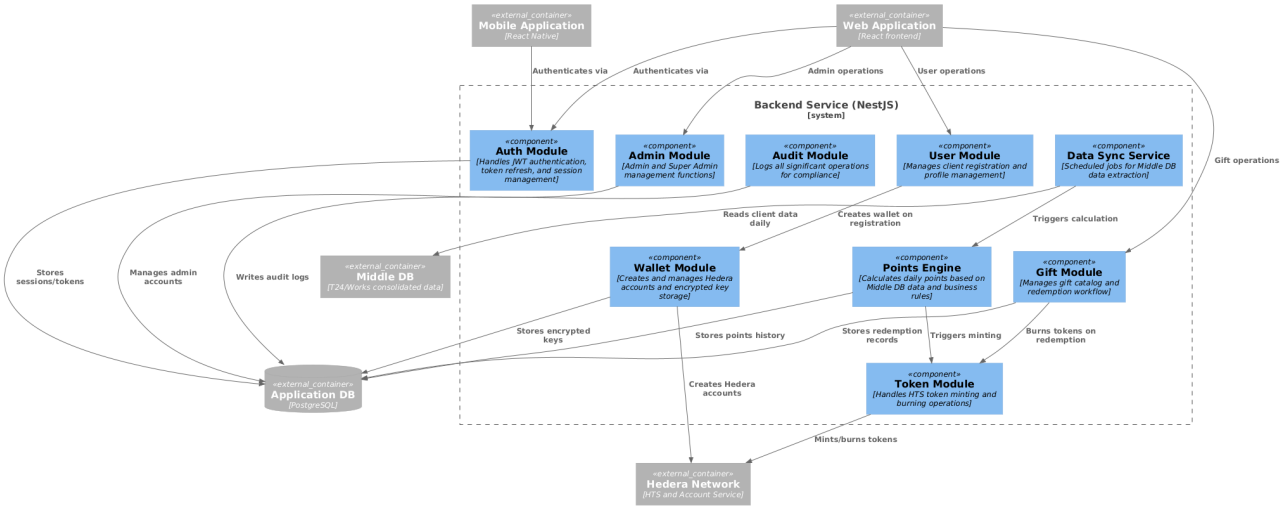
2.6. Component Diagram

2.6.1. Overview

This document provides a component-level view (C3) of the Enda Tamweel Loyalty Platform, detailing the internal structure of the backend service and its interactions with external systems.

2.6.2. Component Diagram - Backend Service

The Backend Service is the core of the platform, handling all business logic, data processing, and blockchain interactions.



Auth Module

Handles all authentication and authorization:

- Login with username/password validation
- JWT access token generation (short-lived, stored in memory only on client)
- Refresh token generation (long-lived, HTTP-only secure cookie)
- Token rotation on refresh (invalidate old, issue new)
- Session management and logout
- Role-based access control (Client, Admin, Super Admin)

User Module

Manages client user accounts:

- Client registration with automatic Hedera wallet creation
- Profile management and updates
- Password management
- Account status tracking

Wallet Module

Handles Hedera account operations:

- Creates new Hedera accounts for registered clients
- Generates and securely stores encrypted private keys
- Manages account associations with the loyalty token
- Provides signing capabilities for token operations

Points Engine

Core business logic for loyalty points:

- Processes daily data extracted from Middle DB
- Applies configurable business rules to calculate points
- Tracks loan behavior, user activity, payment events
- Stores daily snapshots and cumulative point values
- Triggers token minting for calculated points

Token Module

Manages all HTS token operations:

- Mints loyalty tokens to client Hedera accounts
- Burns tokens when gifts are redeemed
- Maintains reconciliation between database and on-chain balances
- Uses platform admin key for all mint/burn operations

Gift Module

Manages the gift redemption workflow:

- Gift catalog CRUD operations
- Redemption request creation and tracking
- Admin approval/rejection workflow

- Triggers token burning on approved redemptions
- Logs all redemption transactions

Admin Module

Administrative functions:

- Admin account CRUD for Super Admins
- Admin role and permission management
- System configuration management

Data Sync Service

Scheduled data synchronization:

- Nightly cron job for Middle DB data extraction
- Read-only access to Middle DB
- Logging for each import session
- Error handling and retry logic

Audit Module

Compliance and audit logging:

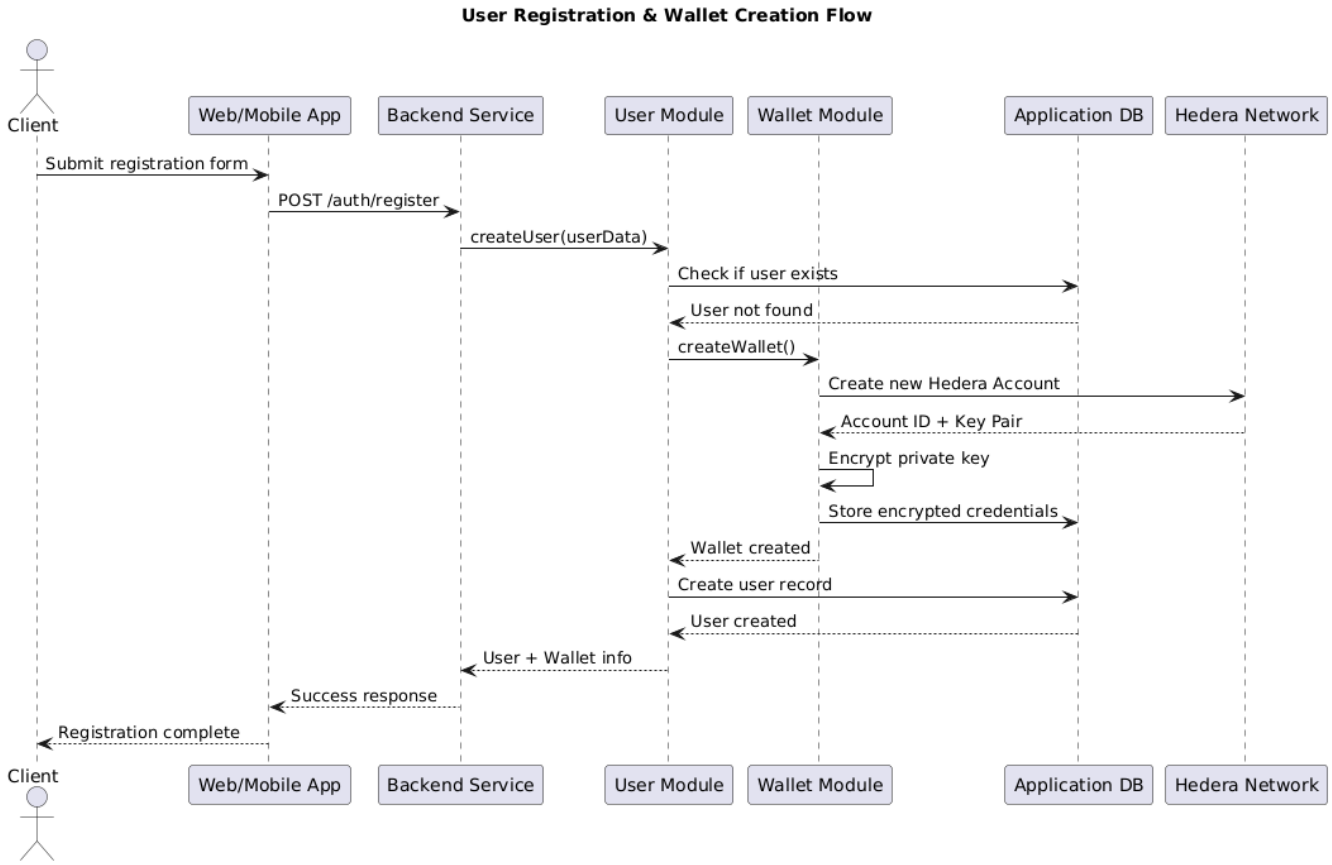
- Logs all authentication events (login, logout, refresh)
- Logs all token operations (mint, burn)
- Logs all redemption transactions
- Logs admin actions
- Provides audit trail query interface

2.7. Runtime View

This section describes the key runtime scenarios of the Enda Tamweel Loyalty Platform, illustrating how components interact during important operations.

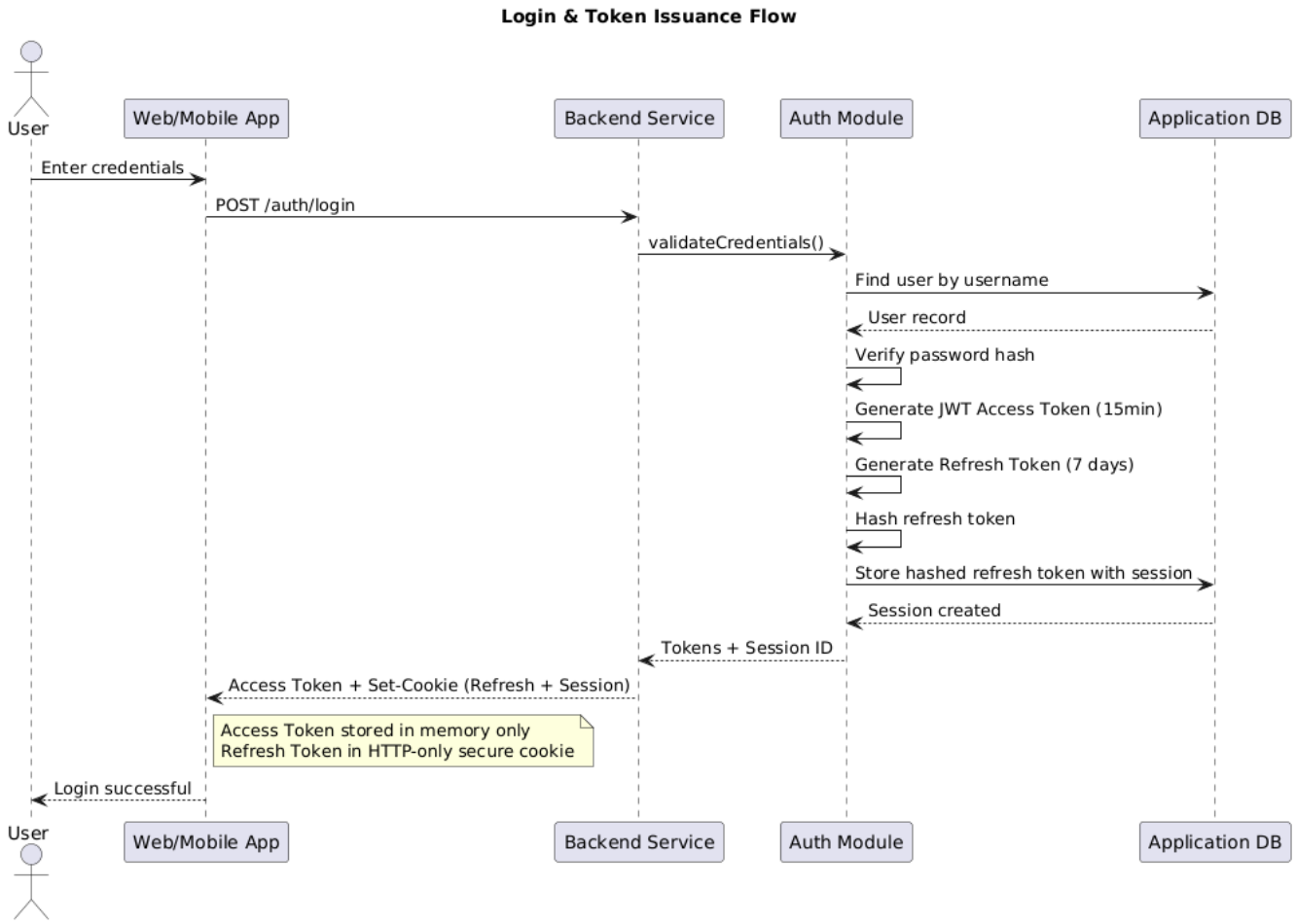
2.7.1. User Registration and Wallet Creation

This sequence shows how a new client is registered and their Hedera wallet is created.

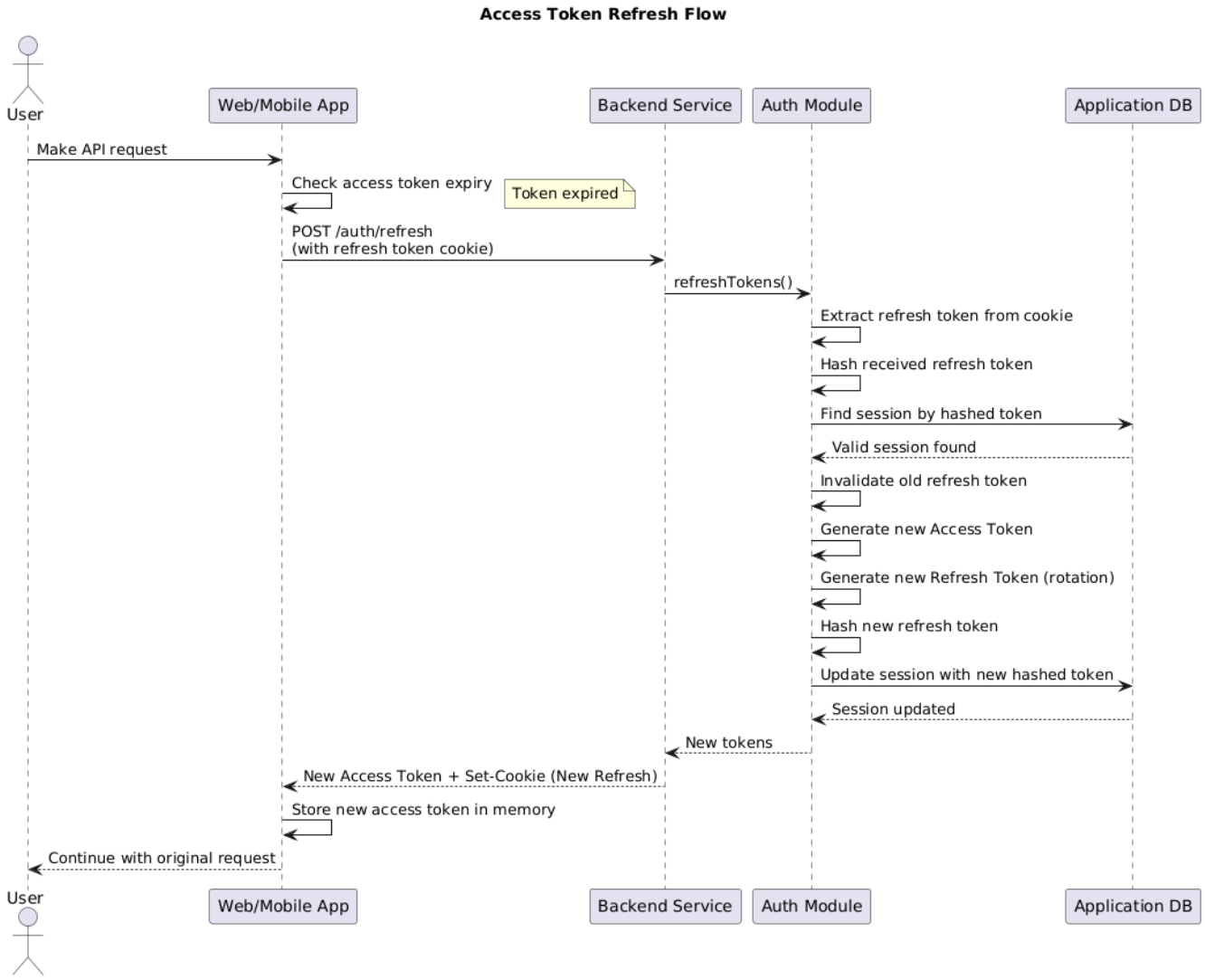


2.7.2. Authentication Flow

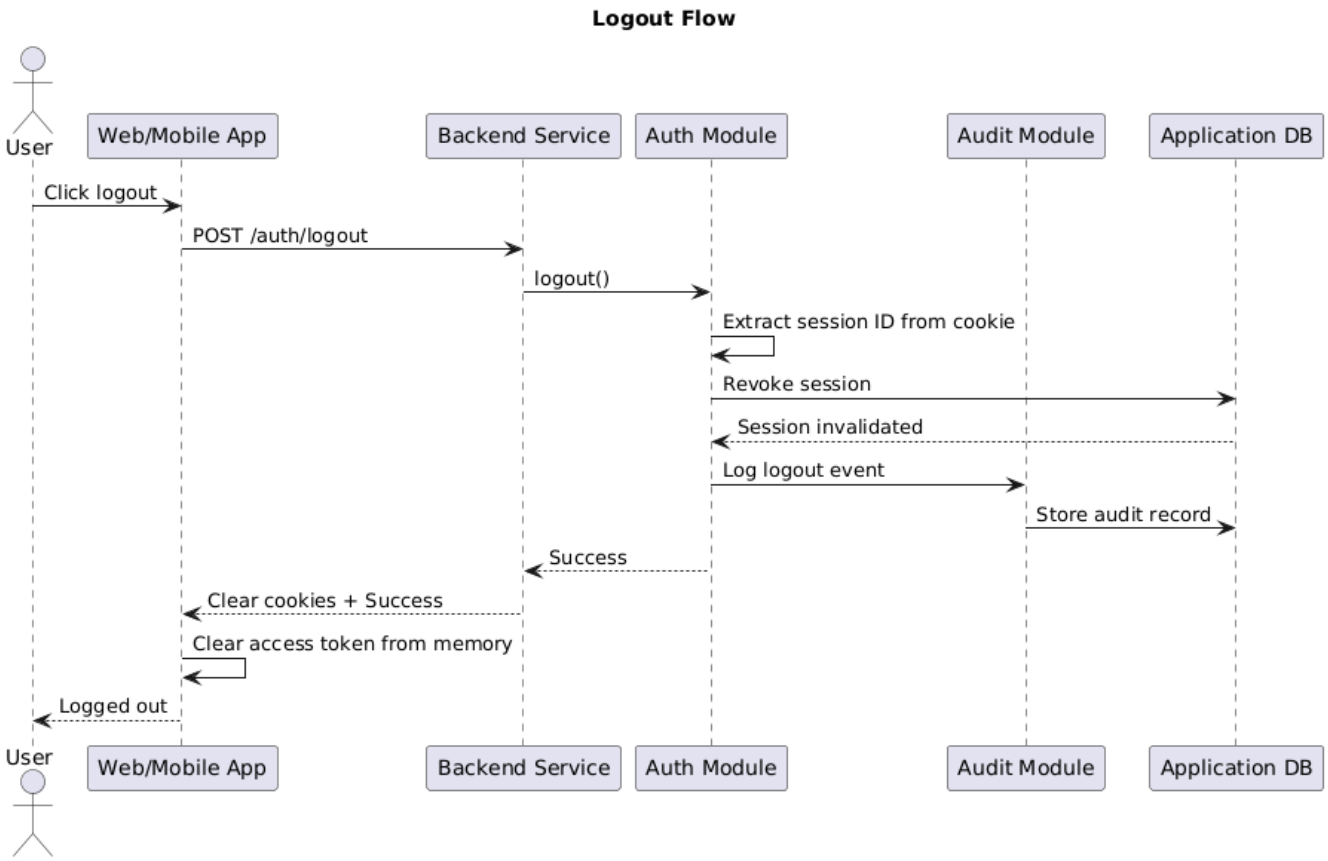
Login & Token Issuance



Access Token Expiry & Refresh

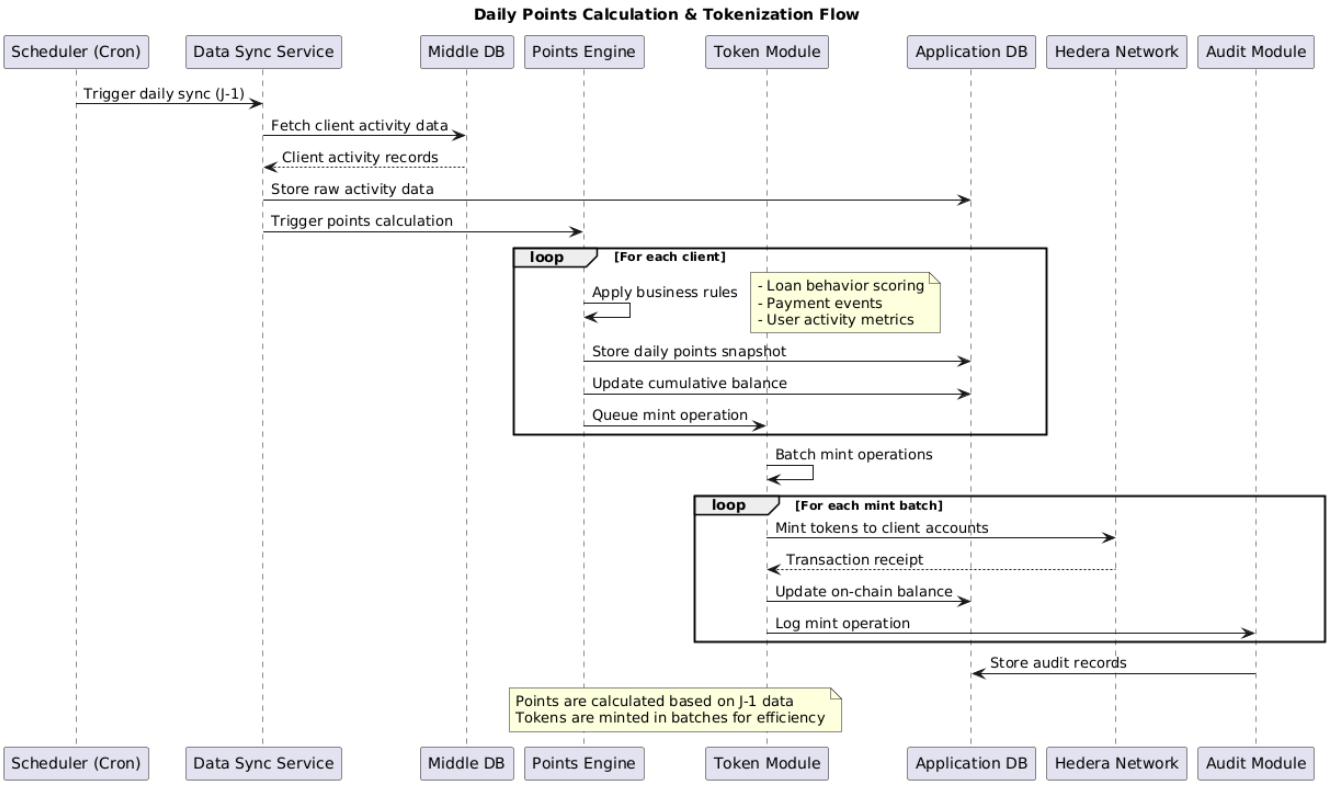

Access Token Refresh Flow
User
Web/Mobile App
Backend Service
Auth Module
Application DB
Make API request
Check access token expiry
Token expired
POST /auth/refresh
(with refresh token cookie)
refreshTokens()
Extract refresh token from cookie
Hash received refresh token
Find session by hashed token
Valid session found
Invalidate old refresh token
Generate new Access Token
Generate new Refresh Token (rotation)
Hash new refresh token
Update session with new hashed token
Session updated
New tokens
New Access Token + Set-Cookie (New Refresh)
Store new access token in memory
Continue with original request
User
Web/Mobile App
Backend Service
Auth Module
Application DB

Logout Flow

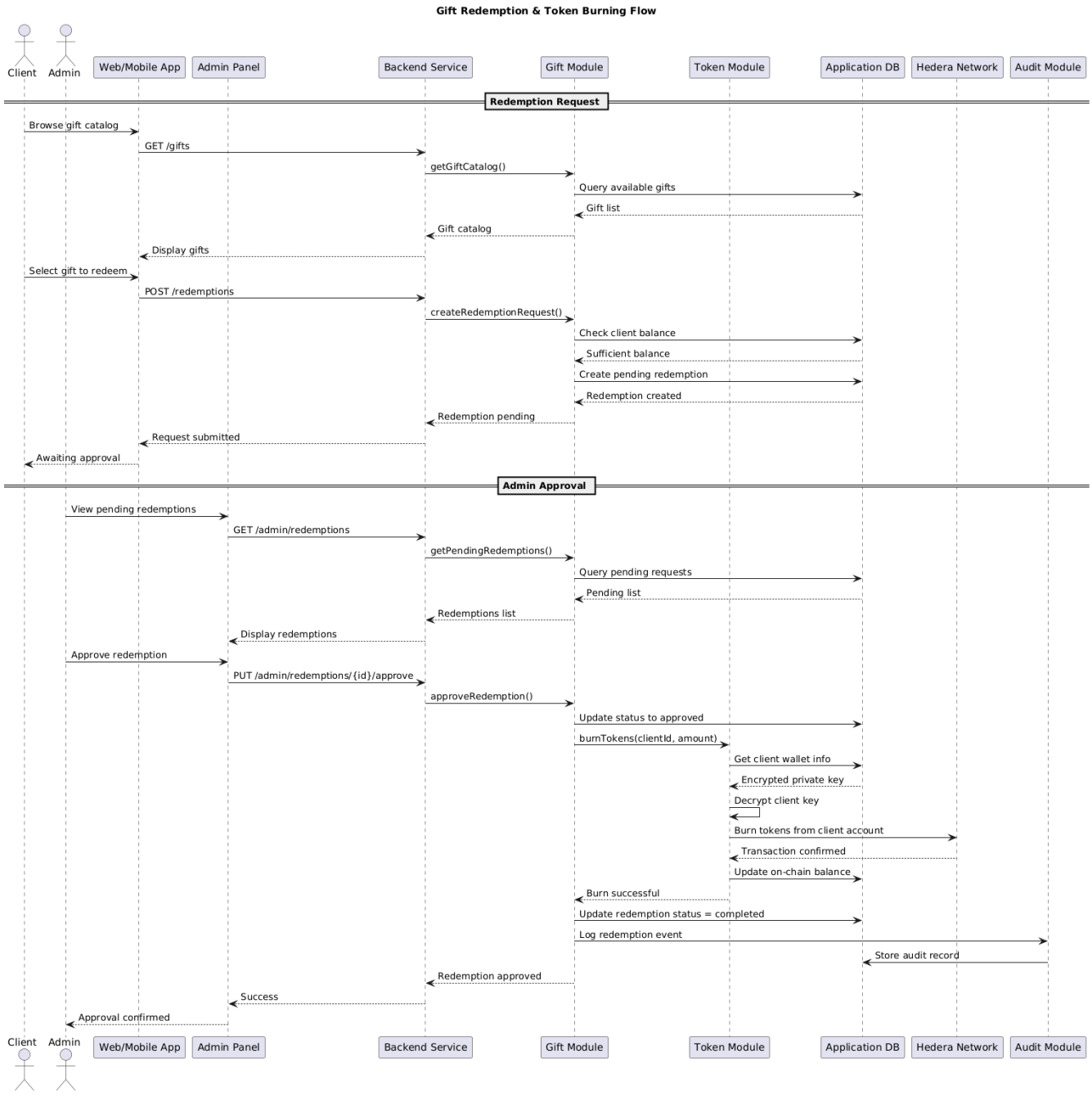


2.7.3. Daily Points Calculation & Tokenization

This is the core business process that runs daily to calculate and mint loyalty tokens.

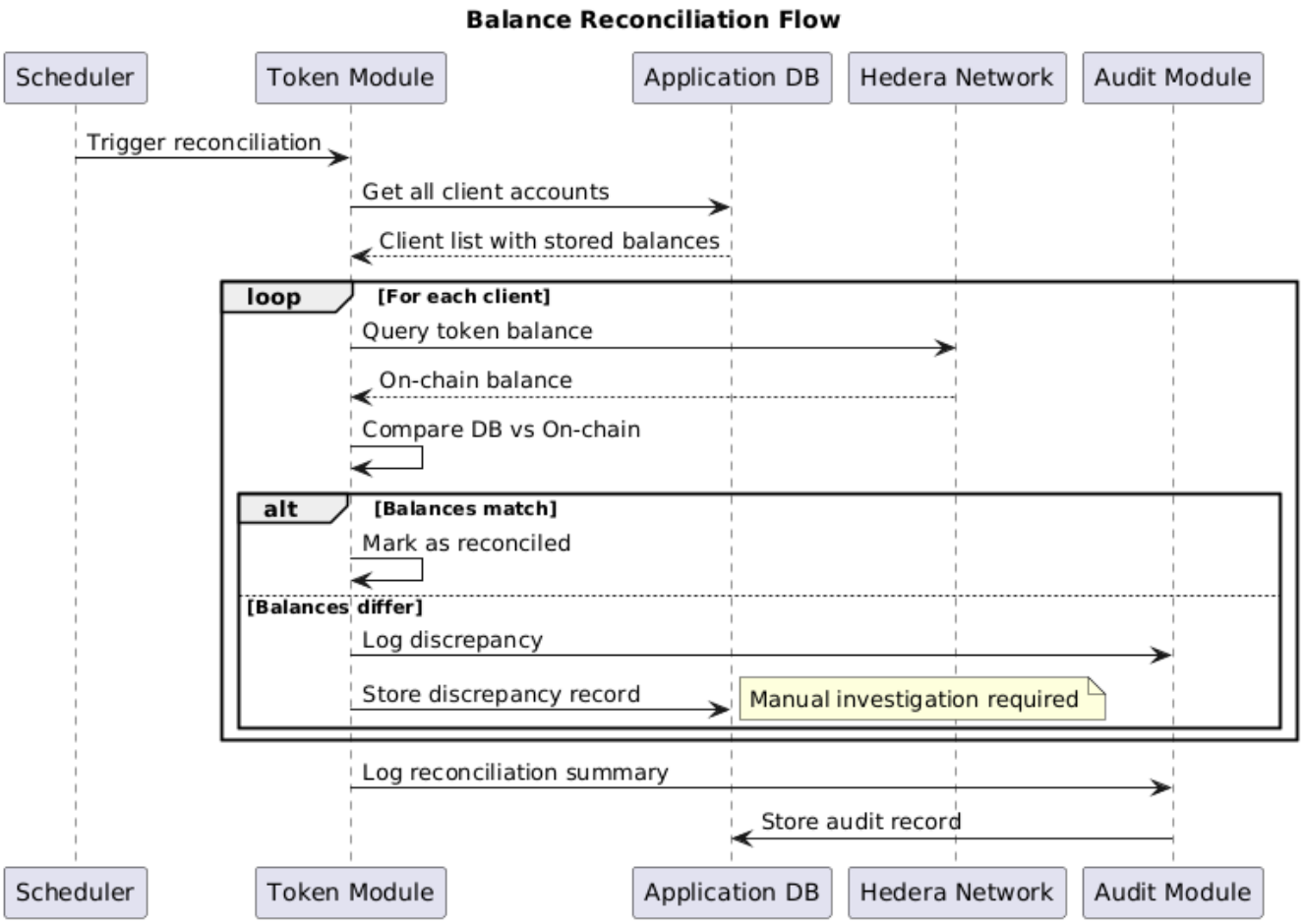


2.7.4. Gift Redemption & Token Burning



2.7.5. Balance Reconciliation

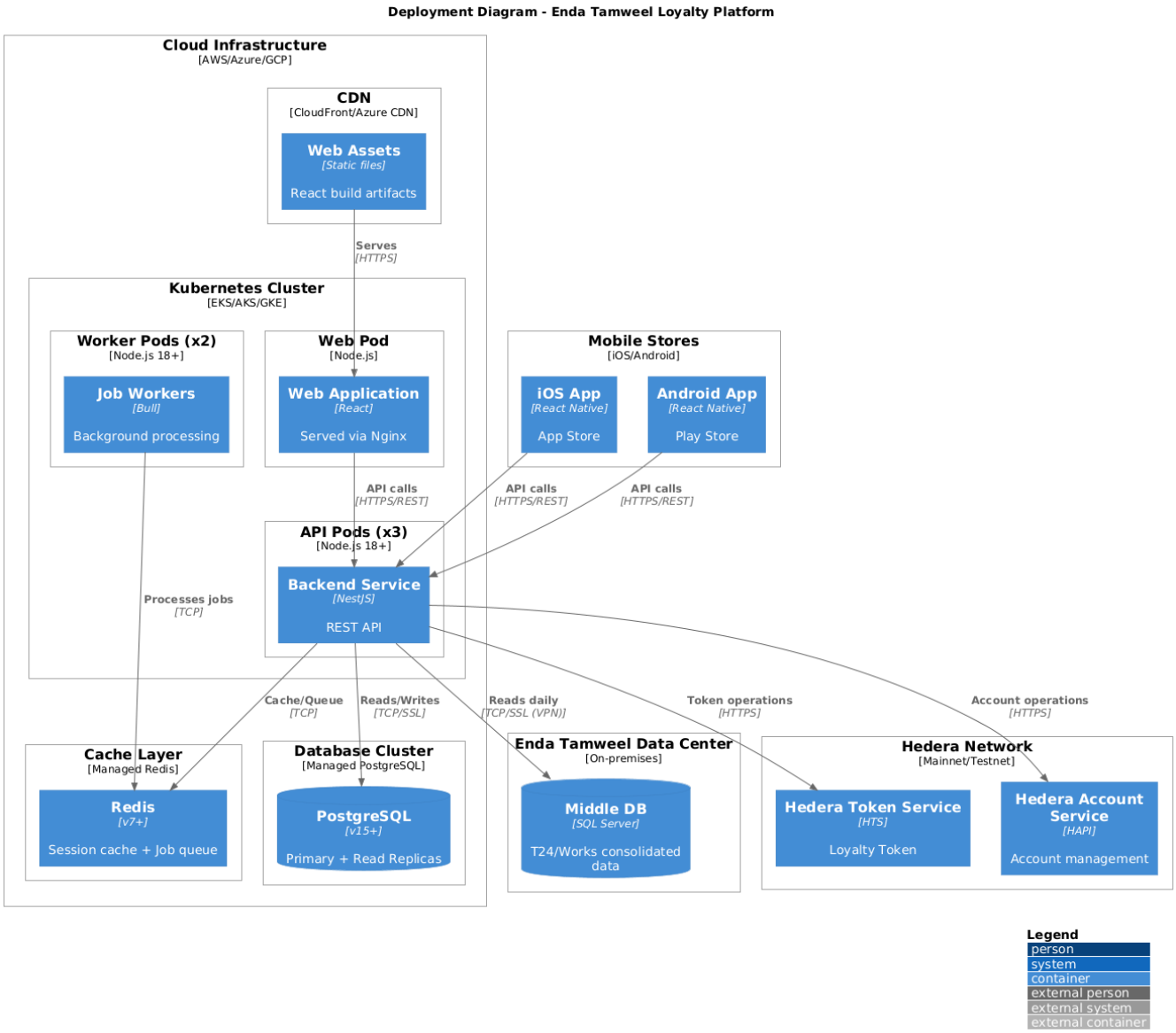
Periodic process to ensure database and on-chain balances are synchronized.



2.8. Deployment View

This section describes the technical infrastructure and deployment architecture for the Enda Tamweel Loyalty Platform.

2.8.1. Infrastructure Overview



2.8.2. Environment Configuration

Production Environment

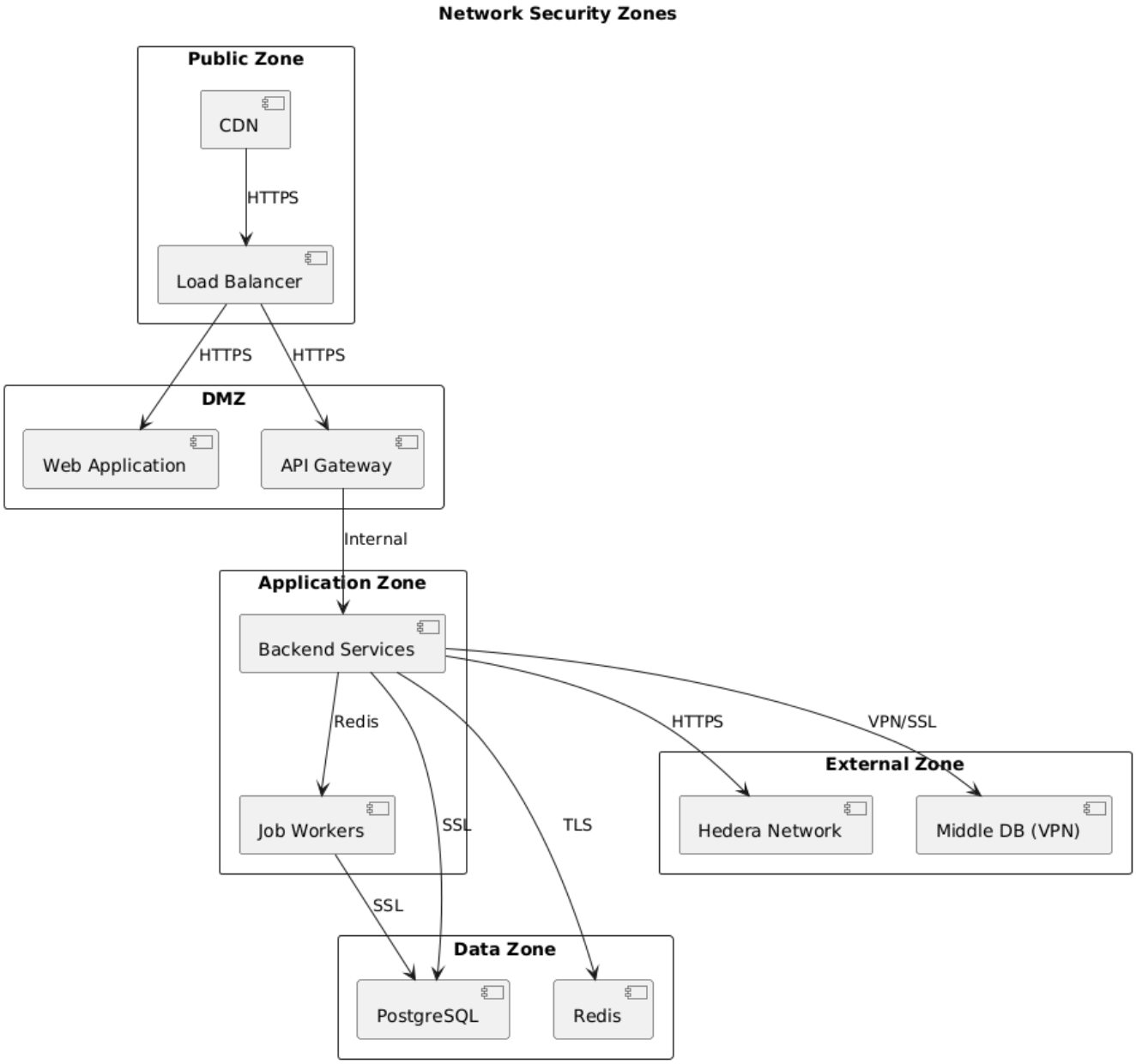
Component	Configuration	Scaling
Web Application	3 pod replicas behind load balancer	HPA: 3-10 pods based on CPU
Backend Service	3 pod replicas with health checks	HPA: 3-15 pods based on requests/sec
Job Workers	2 dedicated worker pods	Manual scaling for batch operations
PostgreSQL	Managed service with read replicas	Vertical scaling + read replicas
Redis	Managed cluster mode	Vertical scaling

Staging Environment

Component	Configuration
Web Application	1 pod replica
Backend Service	2 pod replicas
Job Workers	1 worker pod
PostgreSQL	Single instance (development tier)
Redis	Single instance
Hedera Network	Testnet

2.8.3. Network Architecture

Security Zones



Firewall Rules

Source	Destination	Port	Purpose
Internet	CDN/Load Balancer	443	HTTPS traffic
Load Balancer	Web/API Pods	8080	Internal HTTP
API Pods	PostgreSQL	5432	Database connections
API Pods	Redis	6379	Cache and queue
API Pods	Hedera	443	Token operations
API Pods	Middle DB	1433	Data sync (VPN only)

2.8.4. Branching Strategy

Overview

The branching strategy ensures stability, collaboration, and smooth release cycles aligned with Enda Tamweel's deployment requirements.

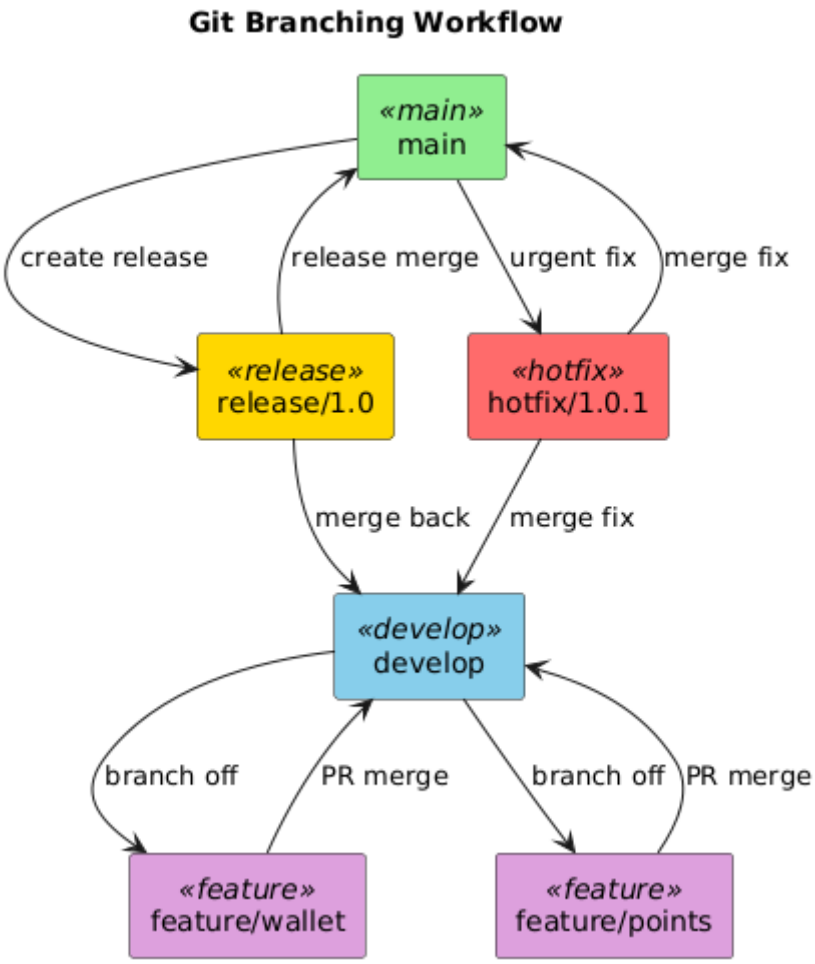
Branch Types

Branch	Purpose	Deployed To
`main`	Production-ready code	Production environment
`develop`	Integration branch for features	Staging environment
`release/*`	Release preparation and stabilization	Pre-production testing
`feature/*`	New feature development	Developer environments
`hotfix/*`	Production bug fixes	Fast-track to production

Branch Naming Convention

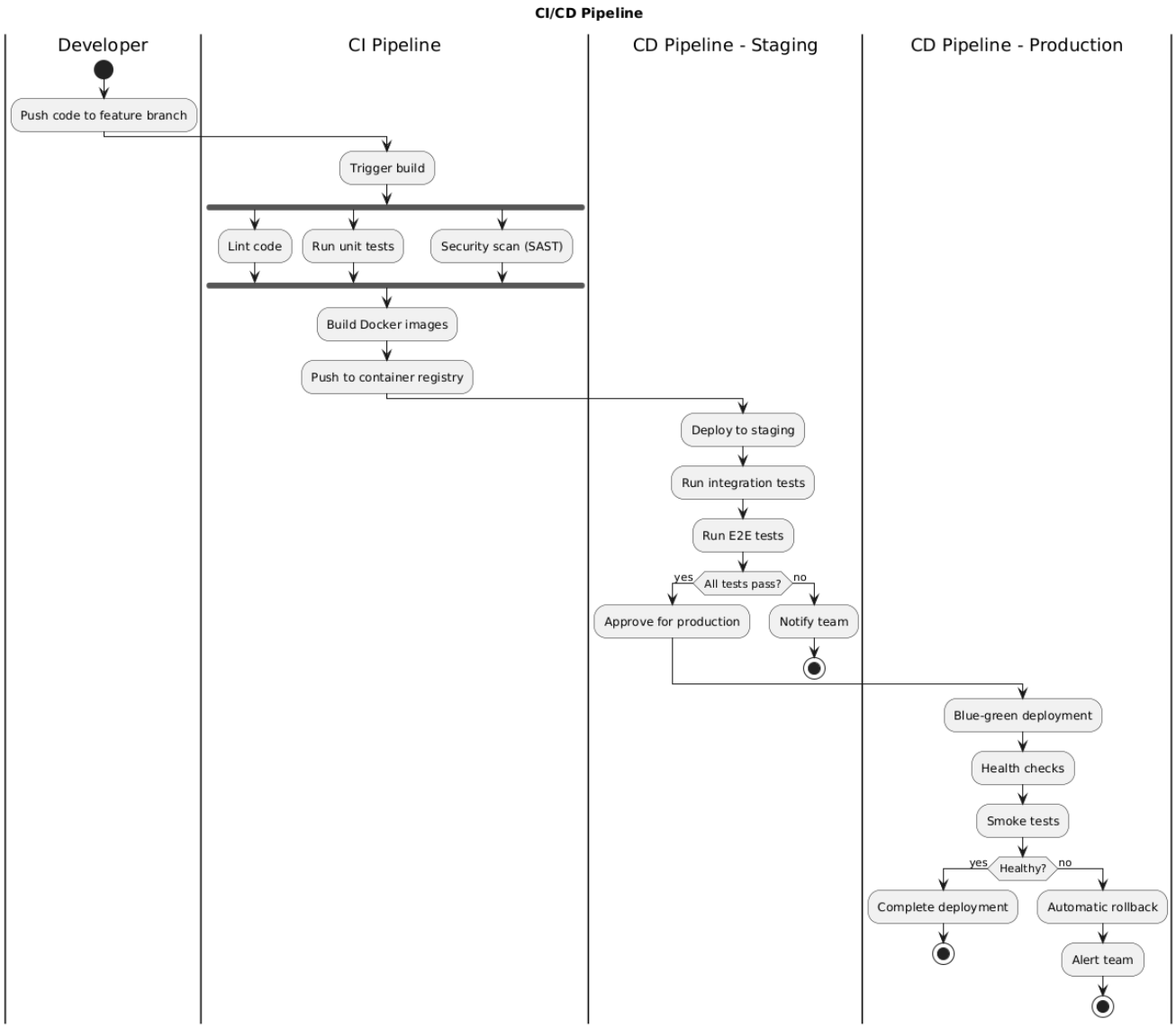
- `feature/TICKET-123-user-registration`
- `feature/loyalty-points-calculation`
- `release/1.2.0`
- `hotfix/1.2.1-token-burn-fix`

Branch Workflow



2.8.5. CI/CD Pipeline

Pipeline Overview



Pipeline Stages

Stage	Actions	Success Criteria
Build	Lint, compile, unit tests, build Docker images	All tests pass, image built successfully
Security	SAST scan, dependency vulnerability check	No critical/high vulnerabilities
Deploy Staging	Deploy to staging K8s cluster	Pods healthy, endpoints responding
Integration Tests	API integration tests, Hedera testnet tests	All integration tests pass
E2E Tests	Full user journey tests	All scenarios complete successfully
Deploy Production	Blue-green deployment to production	Zero-downtime deployment complete
Post-Deploy	Smoke tests, monitoring verification	All health checks pass

Deployment Success Criteria

- All pods reach `Ready` state within 5 minutes
- Health check endpoints return 200 OK
- No error spike in monitoring (>1% error rate triggers rollback)
- Database migrations complete successfully
- Hedera connectivity verified

2.8.6. Secrets Management

Secret Categories

Category	Secrets	Storage
Database	PostgreSQL credentials, connection strings	Kubernetes Secrets / Vault
Authentication	JWT signing keys, refresh token secrets	Vault / KMS
Hedera	Operator account ID, private keys, token IDs	Vault with strict access control
External	Middle DB credentials, API keys	Vault / Sealed Secrets

Key Rotation Policy

Secret Type	Rotation Frequency
JWT Signing Keys	90 days
Database Passwords	90 days
Hedera Operator Keys	Manual (with audit)
API Keys	180 days

Chapter 3. Cross-cutting Concepts

This section describes the overarching concepts and design decisions that apply across multiple components of the Enda Tamweel Loyalty Platform. These cross-cutting concerns ensure consistency and quality throughout the system.

3.1. Overview

Concept	Description
Security	Authentication, authorization, encryption, and key management strategies
Performance	Optimization techniques for batch processing, caching, and API response times
Monitoring & Logging	Observability infrastructure including logs, metrics, traces, and alerting
Data Protection	Privacy controls, data classification, and regulatory compliance measures

3.2. Security Concept

This section describes the security architecture and controls implemented in the Enda Tamweel Loyalty Platform to protect user data, credentials, and token operations.

3.2.1. Authentication Architecture

JWT Token Strategy

The platform implements a dual-token authentication strategy to balance security with user experience:

Token Type	Characteristics	Storage
Access Token	Short-lived (15 minutes), contains user ID, role, username	Frontend memory only (never localStorage/sessionStorage)
Refresh Token	Long-lived (7 days), bound to specific session	HTTP-only, Secure cookie (inaccessible to JavaScript)

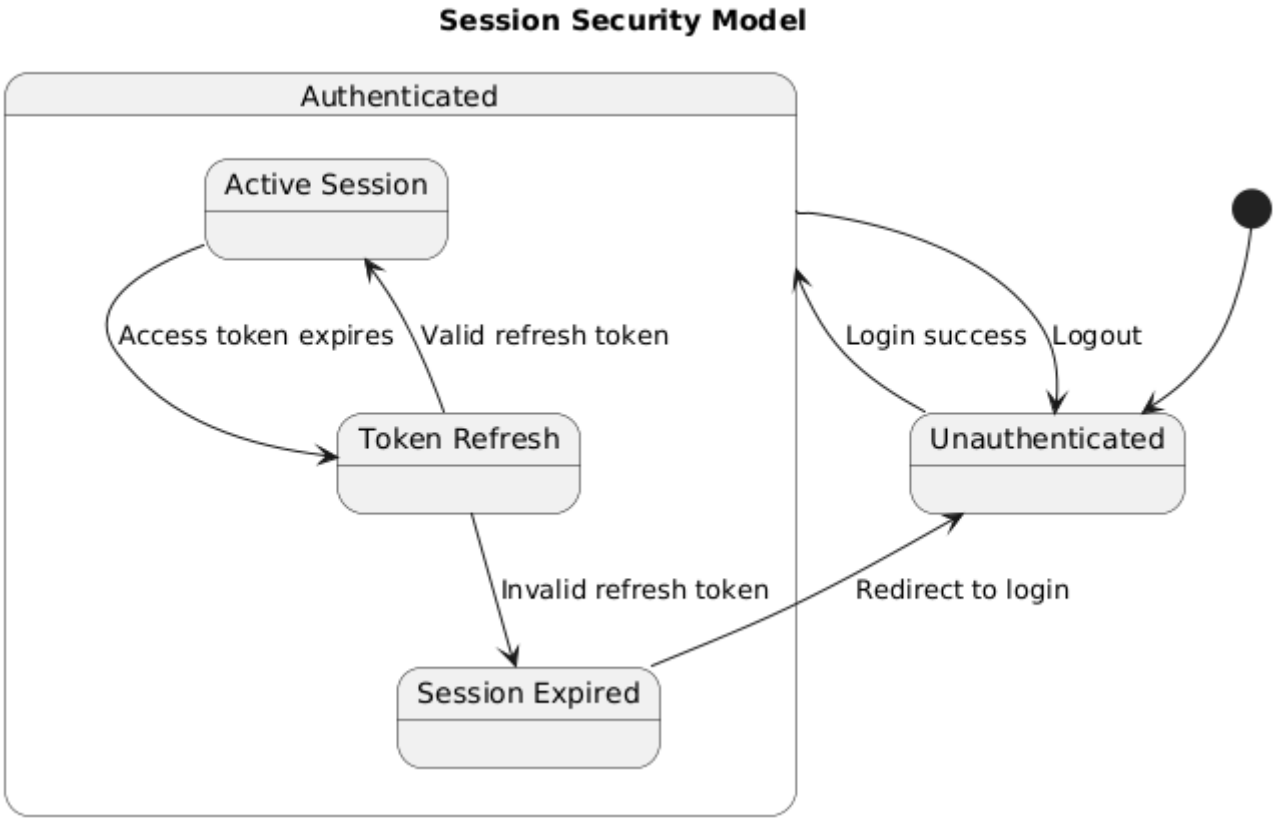
Token Security Measures

- **Access Token in Memory**: Stored only in frontend application memory to prevent XSS attacks. Token is lost when page closes, requiring re-authentication or refresh.
- **Refresh Token Rotation**: Each refresh request generates a new refresh token pair:
  1. Backend validates the existing refresh token
  2. Old refresh token is invalidated in database
  3. New refresh token and access token issued

4. New refresh token sent as HTTP-only cookie

- **Refresh Token Hashing**: Refresh tokens are hashed (bcrypt) before storage in the database. Even if the database is compromised, tokens cannot be used directly.

Session Management



3.2.2. Hedera Wallet Security

Private Key Protection

Client Hedera private keys require special security handling:

Control	Implementation
Encryption at Rest	Private keys encrypted using AES-256-GCM before database storage
Key Derivation	Encryption keys derived using PBKDF2 with unique salts per user
Access Control	Private keys decrypted only during token operations, never exposed via API
Memory Handling	Keys cleared from memory immediately after use
Backup Security	Database backups encrypted; keys stored separately from encrypted data

Token Operation Authorization

Operation	Authorization
Mint Tokens	Platform admin key only (backend-controlled)

Operation	Authorization
Burn Tokens	Platform admin key + verified redemption approval
View Balance	Authenticated user (own balance) or Admin (any user)
Transfer	Disabled - tokens are non-transferable between users

3.2.3. API Security

Transport Security

- All external communication encrypted using **TLS 1.3**
- HSTS (HTTP Strict Transport Security) enabled
- Certificate pinning for mobile applications
- Internal service-to-service communication within Kubernetes uses mTLS

Request Authentication

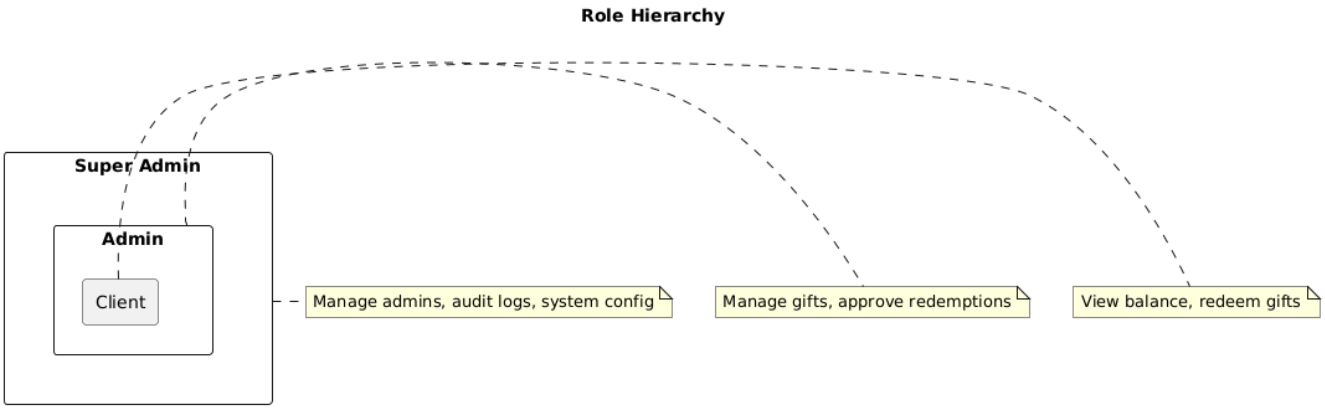
Endpoint Type	Authentication Method
Public	None (login, registration, public info)
Protected	Bearer token (JWT access token) in Authorization header
Admin	Bearer token + role verification (Admin or Super Admin)
Internal	Service mesh authentication (mTLS)

Rate Limiting

Endpoint Category	Rate Limit	Window
Authentication	5 requests	1 minute
Token Refresh	10 requests	1 minute
API General	100 requests	1 minute
Admin Operations	50 requests	1 minute

3.2.4. Role-Based Access Control (RBAC)

Role Hierarchy



Permission Matrix

Feature	Client	Admin	Super Admin
View Own Balance	✔	✔	✔
Request Redemption	✔	✔	✔
View Gift Catalog	✔	✔	✔
Approve Redemptions	✘	✔	✔
Manage Gift Catalog	✘	✔	✔
View All Users	✘	✔	✔
Manage Admin Accounts	✘	✘	✔
View Audit Logs	✘	✘	✔
System Configuration	✘	✘	✔

3.2.5. Security Monitoring

Audit Events

All security-relevant events are logged:

- Authentication events (login, logout, failed attempts)
- Token refresh events
- Role changes
- Admin account modifications
- Token mint/burn operations
- Redemption approvals/rejections
- Permission denied attempts

Security Alerts

Automated alerts for:

- Multiple failed login attempts (>5 in 10 minutes)
- Unusual token refresh patterns
- Admin actions outside business hours
- Large token operations (burn > threshold)
- Database access anomalies

3.2.6. Cookie Security Settings

Attribute	Value
HttpOnly	true (prevents JavaScript access)
Secure	true (HTTPS only)
SameSite	Strict (CSRF protection)
Path	/api/auth (restricted scope)
Max-Age	604800 (7 days)

3.3. Performance Optimization

This section describes the performance optimization strategies implemented in the Enda Tamweel Loyalty Platform to ensure responsive user experience and efficient batch processing.

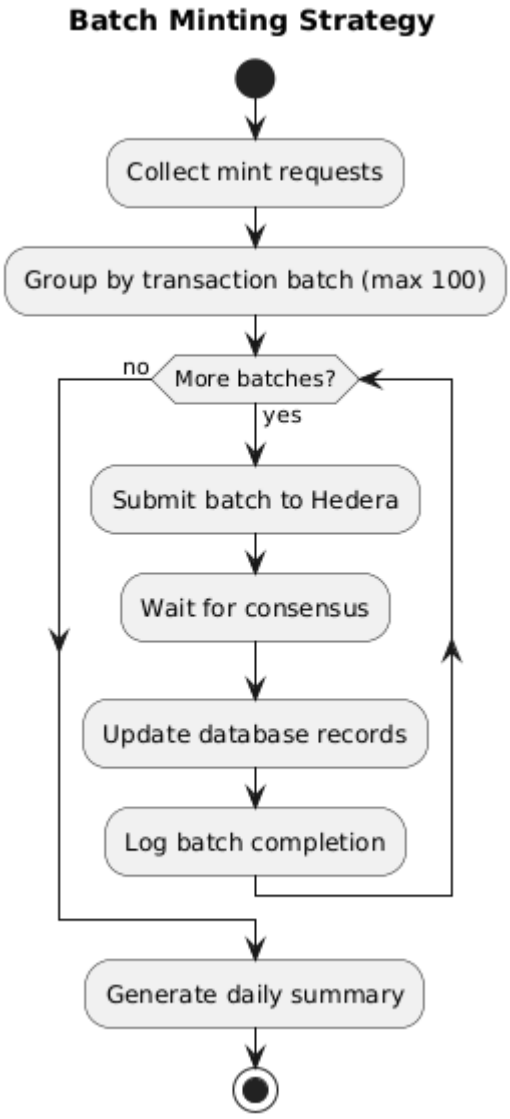
3.3.1. Batch Processing Architecture

Daily Points Calculation

The daily points calculation processes thousands of client records efficiently:

Strategy	Implementation
Chunked Processing	Client records processed in chunks of 100 to prevent memory overflow
Parallel Calculation	Points calculation runs in parallel worker threads (4 workers default)
Database Batching	Bulk INSERT/UPDATE operations instead of individual queries
Progress Tracking	Checkpoint system to resume from failure point

Token Minting Optimization



Key optimizations:

- **Batch Size**: Maximum 100 operations per Hedera transaction to balance throughput and fees
- **Rate Limiting**: Controlled submission rate to avoid network throttling
- **Retry Logic**: Exponential backoff for failed transactions (max 3 retries)
- **Fee Optimization**: Transactions scheduled during lower-fee periods when possible

3.3.2. Caching Strategy

Cache Layers

Layer	Technology	TTL	Data Cached
Application Cache	In-memory (Node.js)	Request lifetime	Parsed tokens, config
Distributed Cache	Redis	Variable	Sessions, user data, gift catalog

Layer	Technology	TTL	Data Cached
Database Cache	PostgreSQL	Query plan cache	Frequent queries

Redis Caching Implementation

Cache Key Pattern	TTL	Purpose
`user:{id}:profile`	30 minutes	User profile data
`user:{id}:balance`	5 minutes	Cached token balance
`gifts:catalog`	1 hour	Gift catalog (all users)
`session:{id}`	7 days	Active session data
`rate:{ip}:{endpoint}`	1 minute	Rate limiting counters

Cache Invalidation

Event	Invalidation Action
Token mint/burn	Invalidate `user:{id}:balance`
Profile update	Invalidate `user:{id}:profile`
Gift catalog change	Invalidate `gifts:catalog`
Logout	Delete `session:{id}`

3.3.3. Database Optimization

Indexing Strategy

Table	Index	Purpose
users	`idx_users_username`	Login lookup
users	`idx_users_hedera_account`	Hedera account lookup
points_history	`idx_points_user_date`	Daily points query
redemptions	`idx_redemptions_status`	Pending redemptions query
audit_logs	`idx_audit_timestamp`	Time-based audit queries

Query Optimization

- **Pagination**: All list endpoints use cursor-based pagination
- **Projection**: SELECT only required columns, avoid SELECT *
- **Connection Pooling**: pgBouncer for connection management (max 100 connections)
- **Read Replicas**: Read-heavy queries routed to replicas

3.3.4. API Response Optimization

Response Time Targets

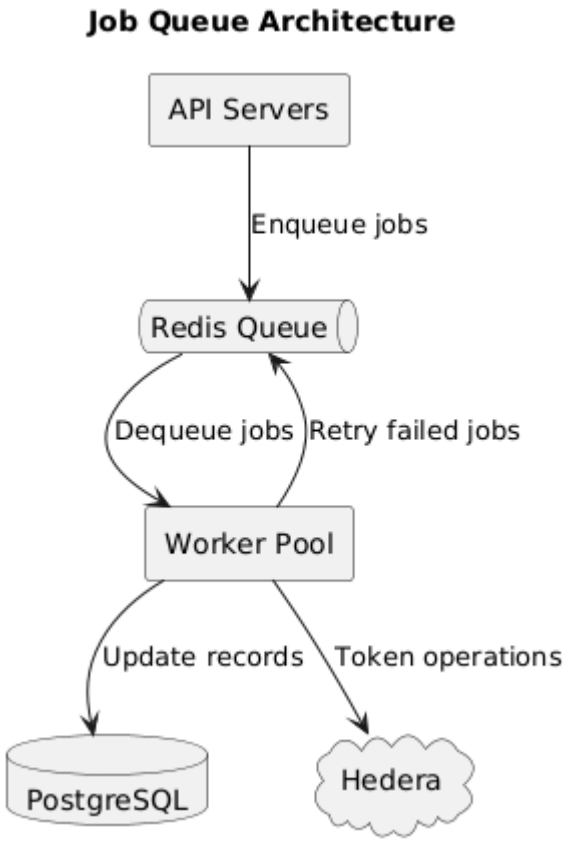
Endpoint Category	P50 Target	P95 Target
Authentication	100ms	300ms
User Profile	50ms	150ms
Balance Query	100ms	250ms
Gift Catalog	50ms	100ms
Redemption Create	200ms	500ms

Optimization Techniques

- **Response Compression**: gzip compression for responses > 1KB
- **JSON Serialization**: Fast JSON serializer (fastify-json)
- **Connection Keep-Alive**: HTTP/2 with connection reuse
- **Early Response**: Return success immediately, process async when possible

3.3.5. Asynchronous Processing

Job Queue Architecture



Job Types and Priorities

Job Type	Priority	Timeout	Retry Policy
Daily Sync	High	2 hours	3 retries, exponential backoff
Points Calculation	High	1 hour	3 retries
Token Minting	Medium	30 minutes	5 retries, exponential backoff
Balance Reconciliation	Low	1 hour	2 retries
Notification	Low	5 minutes	1 retry

3.3.6. Hedera Network Optimization

Transaction Efficiency

- **Account Consolidation**: Platform uses single operator account for all mint/burn operations
- **Auto Token Association**: Tokens auto-associated during account creation
- **Transaction Batching**: Multiple operations combined where possible
- **Testnet vs Mainnet**: Development uses testnet to minimize costs

Network Monitoring

Metric	Threshold
Transaction success rate	>99%
Average consensus time	<5 seconds
Daily transaction count	Monitor for anomalies
Fee spend (HBAR)	Budget alerting

3.3.7. Performance Monitoring

Key Performance Indicators (KPIs)

KPI	Target	Alert Threshold
API Response Time (P95)	<500ms	>1000ms
Daily Batch Completion	<4 hours	>6 hours
Database Query Time (P95)	<100ms	>500ms
Cache Hit Rate	>90%	<80%
Error Rate	<0.1%	>1%

3.4. Monitoring and Logging

This section describes the monitoring, logging, and observability architecture for the Enda Tamweel Loyalty Platform.

3.4.1. Logging Architecture

Log Levels and Usage

Level	Usage	Examples
ERROR	System failures requiring immediate attention	Database connection failure, Hedera transaction failure, unhandled exceptions
WARN	Potential issues that don't prevent operation	Retry attempts, degraded performance, cache misses
INFO	Normal operational events	User login, token minting, redemption approval
DEBUG	Detailed information for troubleshooting	Request/response payloads, calculation steps
TRACE	Very detailed debugging information	Function entry/exit, variable values

Structured Logging Format

All logs follow a structured JSON format for easy parsing:

```
{
  "timestamp": "2025-12-16T10:30:45.123Z",
  "level": "INFO",
  "service": "backend-api",
  "traceId": "abc123def456",
  "spanId": "span789",
  "userId": "user-001",
  "action": "token.mint",
  "details": {
    "amount": 100,
    "accountId": "0.0.12345"
  },
  "duration": 245
}
```

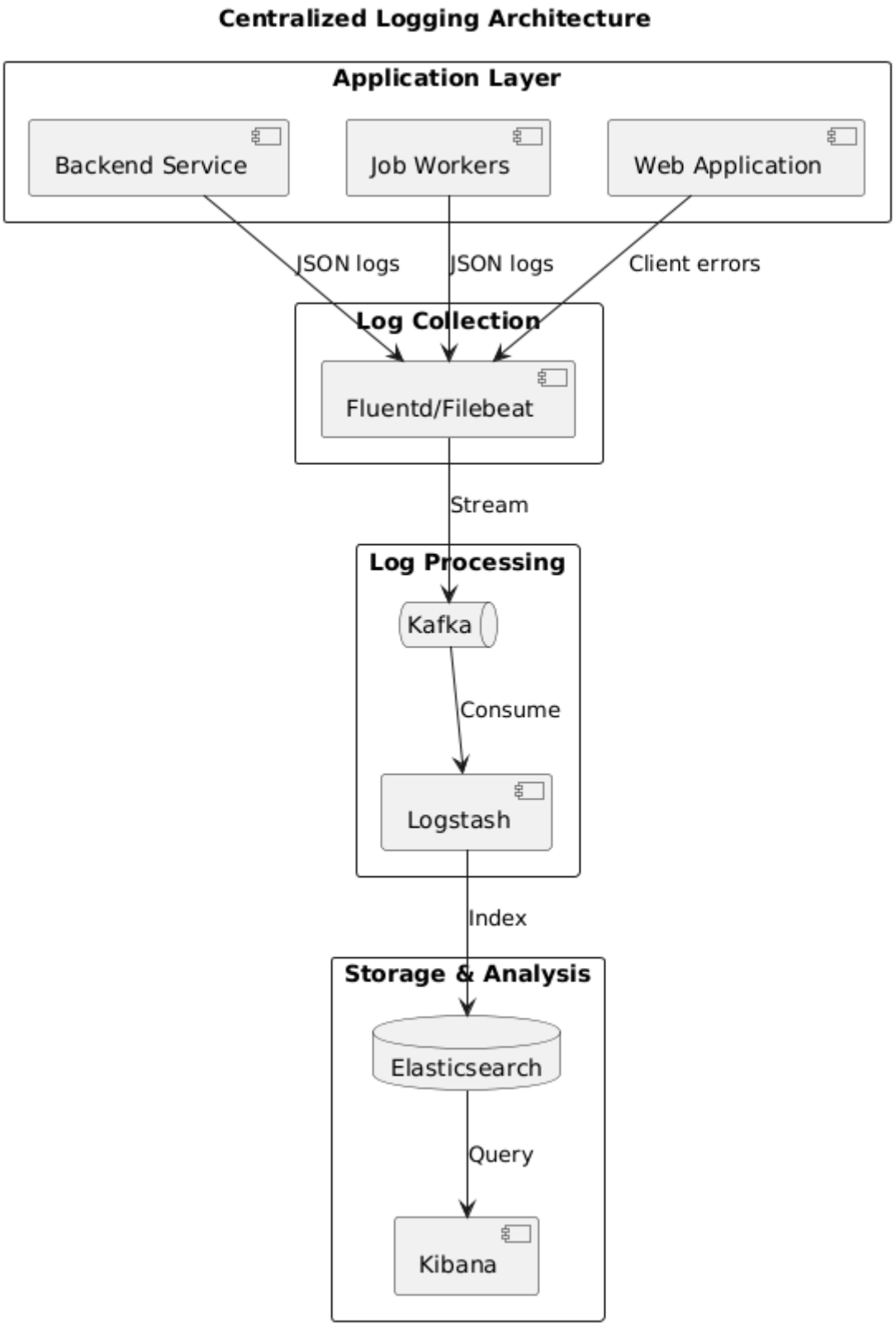
Log Categories

Category	Events Logged	Retention
Security	Authentication, authorization, access denied	1 year

Category	Events Logged	Retention
Audit	Token operations, redemptions, admin actions	7 years (compliance)
Application	Business logic, errors, warnings	90 days
Performance	Response times, queue metrics	30 days
Debug	Detailed troubleshooting	7 days

3.4.2. Centralized Logging Stack

Architecture



Kibana Dashboards

Dashboard	Content
Operations Overview	Request volume, error rates, response times, active users
Security Dashboard	Login attempts, failed authentications, permission denials
Token Operations	Minting volume, burning events, balance changes

Dashboard	Content
Audit Trail	All audit events with filtering by user, action, time
Error Analysis	Error trends, stack traces, affected users

3.4.3. Audit Logging

Mandatory Audit Events

All of the following events are logged with full context:

Category	Events	Data Captured
Authentication	Login, logout, token refresh, failed login	User ID, IP, timestamp, user agent, result
User Management	Registration, profile update, password change	User ID, changed fields, admin (if applicable)
Token Operations	Mint, burn, balance query	User ID, amount, transaction ID, account ID
Redemption	Request, approval, rejection, completion	User ID, gift ID, approver, status change
Admin Actions	Admin creation, role change, system config	Admin ID, target, changes made
Data Access	Export, bulk query, report generation	User ID, data type, scope

Audit Log Format

```
{
  "auditId": "audit-20251216-001234",
  "timestamp": "2025-12-16T10:30:45.123Z",
  "eventType": "TOKEN_MINT",
  "actorId": "system-batch",
  "actorType": "SYSTEM",
  "targetId": "user-12345",
  "targetType": "USER",
  "action": "MINT",
  "details": {
    "amount": 150,
    "hederaTxId": "0.0.1234@1702721445.123456789",
    "previousBalance": 500,
    "newBalance": 650,
    "reason": "DAILY_POINTS"
  },
  "result": "SUCCESS",
  "ipAddress": "10.0.1.50",
  "correlationId": "batch-2025-12-16"
}
```

3.4.4. Metrics and Monitoring

Prometheus Metrics

Application Metrics

Metric	Type	Description
`http_requests_total`	Counter	Total HTTP requests by endpoint, method, status
`http_request_duration_seconds`	Histogram	Request duration distribution
`active_users_total`	Gauge	Currently active user sessions
`token_operations_total`	Counter	Token mints and burns by type
`job_queue_size`	Gauge	Current job queue depth
`job_processing_duration_seconds`	Histogram	Job processing time

Hedera-Specific Metrics

Metric	Type	Description
`hedera_transactions_total`	Counter	Hedera transactions by type and result
`hedera_transaction_duration_seconds`	Histogram	Time from submission to consensus
`hedera_fee_hbar_total`	Counter	Total HBAR spent on fees
`hedera_account_balance_hbar`	Gauge	Operator account balance

Grafana Dashboards

Dashboard	Panels
System Overview	Request rate, error rate, latency percentiles, active connections
Hedera Operations	Transaction success rate, consensus time, fee tracking, daily volume
Business Metrics	Points calculated, tokens minted, redemptions processed, user growth
Infrastructure	CPU, memory, disk, network for all services
Database	Query performance, connection pool, table sizes

3.4.5. Alerting

Alert Rules

Alert	Condition	Severity	Response
High Error Rate	Error rate > 1% for 5 minutes	Critical	On-call page
API Latency	P95 > 2 seconds for 10 minutes	Warning	Slack notification
Hedera Failure	Transaction failures > 5% for 5 minutes	Critical	On-call page
Database Connection	Available connections < 10%	Critical	Auto-scale + page
Batch Job Delay	Daily job not complete by 8 AM	Warning	Slack notification
Low HBAR Balance	Operator balance < 100 HBAR	Warning	Email to finance
Security Event	> 10 failed logins from same IP	Warning	Security team alert

Alert Channels

Channel	Alert Types
PagerDuty/On-call	Critical production issues
Slack #alerts	Warnings, performance degradation
Email	Daily summaries, non-urgent issues
Security Team	Security-related events

3.4.6. Distributed Tracing

Implementation

- **Technology**: OpenTelemetry with Jaeger backend
- **Trace Propagation**: W3C Trace Context headers
- **Sampling**: 10% of requests in production, 100% for errors

Trace Points

Service	Instrumented Operations
API Gateway	All incoming requests
Backend Service	Database queries, Hedera calls, cache operations
Job Workers	Job processing, external calls
Database	Query execution time

3.4.7. Log Retention Policy

Log Type	Hot Storage	Cold Storage
Audit Logs	90 days (Elasticsearch)	7 years (S3 Glacier)
Application Logs	30 days (Elasticsearch)	90 days (S3)
Security Logs	90 days (Elasticsearch)	1 year (S3)
Debug Logs	7 days (Elasticsearch)	Not retained
Metrics	30 days (Prometheus)	1 year (Thanos)

3.5. Data Protection and Privacy

This section describes the data protection measures and privacy controls implemented in the Enda Tamweel Loyalty Platform to ensure compliance with applicable regulations and protect user data.

3.5.1. Regulatory Framework

Applicable Regulations

The platform is designed to comply with:

Regulation	Applicability
Tunisia Personal Data Protection Law (Law 2004-63)	Primary regulation for Enda Tamweel operations in Tunisia
GDPR Principles	Applied as best practice for data protection
Financial Services Regulations	Microfinance and banking data requirements
Hedera Network Compliance	Public blockchain data considerations

3.5.2. Data Classification

Data Categories

Category	Examples	Sensitivity	Storage
Personal Identifiable Information (PII)	Name, email, phone, national ID	High	Encrypted at rest
Financial Data	Loan details, point balances, transactions	High	Encrypted at rest

Category	Examples	Sensitivity	Storage
Authentication Credentials	Password hashes, refresh tokens	Critical	Hashed + encrypted
Cryptographic Keys	Hedera private keys	Critical	AES-256 encrypted
Public Blockchain Data	Hedera account IDs, transaction IDs	Low	Inherently public

Data Handling by Location

Location	Data Type	Protection
Application Database	User profiles, points history, redemptions	Encryption at rest (AES-256), TLS in transit
Redis Cache	Session data, temporary balances	In-memory, TLS, auto-expiry
Hedera Network	Account IDs, token balances, transaction records	Public by design (no PII stored on-chain)
Log Storage	Audit logs, application logs	Encryption, access control, retention limits

3.5.3. On-Chain Data Considerations

What is Stored On-Chain

Data	On-Chain	Rationale
Hedera Account ID	Yes	Required for token operations; not personally identifiable alone
Token Balance	Yes	Core functionality; linked to account ID only
Transaction Records	Yes	Immutable audit trail; contains only account IDs and amounts
User Name/Email	No	PII never stored on-chain
Private Keys	No	Stored encrypted in application database
Redemption Details	No	Stored in application database only

Blockchain Data Privacy

The platform ensures privacy by:

- **Account Pseudonymity**: Hedera account IDs are not linked to user identity on-chain

- **Off-Chain PII**: All personally identifiable information stored in traditional database
- **Link Protection**: The mapping between user ID and Hedera account is protected in the application database
- **Minimal On-Chain Data**: Only essential token operations recorded on Hedera

3.5.4. Encryption Standards

Encryption at Rest

Data Type	Algorithm	Key Management
Database Fields	AES-256-GCM	AWS KMS / Azure Key Vault
Hedera Private Keys	AES-256-GCM with PBKDF2 key derivation	Unique salt per user, master key in KMS
Backup Files	AES-256	Separate backup encryption key
Log Archives	AES-256	Log-specific encryption key

Encryption in Transit

Connection	Protocol
Client to API	TLS 1.3
Service to Database	TLS/SSL
Service to Redis	TLS
Service to Hedera	HTTPS/TLS
Internal Services	mTLS (Kubernetes)

3.5.5. User Rights Management

Supported User Rights

Right	Implementation	Limitations
Right to Access	User can view all their data via profile and history endpoints	Audit logs access restricted to admins
Right to Rectification	Users can update profile information	Historical transaction data immutable
Right to Erasure	Account deactivation with data anonymization	On-chain data cannot be deleted (public ledger)
Right to Data Portability	Export endpoint provides user data in JSON format	Token balance viewable on Hedera explorer
Right to Restriction	Account can be suspended by admin	Existing on-chain balance preserved

Data Retention

Data Type	Retention Period	Deletion Method
Active User Data	Duration of account	Anonymized on closure
Deactivated Accounts	5 years (regulatory requirement)	Hard delete after period
Transaction History	7 years	Archived then deleted
Audit Logs	7 years	Archived to cold storage
Session Data	7 days after logout	Auto-deleted
On-Chain Data	Permanent	Cannot be deleted (Hedera design)

3.5.6. Data Anonymization

Anonymization for Account Closure

When a user requests account deletion:

1. **Profile Data**: Name, email, phone replaced with anonymous values
2. **Transaction History**: User ID replaced with anonymized identifier
3. **Audit Logs**: User reference replaced with "DELETED_USER_{hash}"
4. **Hedera Account**: Account remains but association removed from system
5. **Tokens**: Remaining balance burned before closure

Anonymization for Testing

Original Field	Anonymized Form
Name	Faker-generated name
Email	hash@test.example.com
Phone	Random valid format number
National ID	Random valid format ID
Hedera Account	Testnet account

3.5.7. Access Control

Data Access Matrix

Data	User	Admin	Super Admin	System
Own Profile	Read/Write	Read	Read	Read
Own Balance	Read	Read	Read	Read/Write
Own Transactions	Read	Read	Read	Read/Write

Data	User	Admin	Super Admin	System
Other User Data	None	Read	Read	Read/Write
Hedera Keys	None	None	None	Decrypt for signing
Audit Logs	None	Limited	Full	Write

3.5.8. Data Protection Impact Assessment (DPIA)

Risk Assessment Summary

Risk	Likelihood	Impact	Mitigation
PII Exposure	Low	High	Encryption, access control, monitoring
Key Compromise	Low	Critical	HSM/KMS, encryption, key rotation
On-Chain Privacy	Medium	Low	No PII on-chain, pseudonymous accounts
Unauthorized Access	Low	High	RBAC, MFA for admins, audit logging
Data Breach	Low	High	Encryption, incident response plan

3.5.9. Incident Response

Data Breach Protocol

1. **Detection**: Automated monitoring alerts security team
2. **Containment**: Isolate affected systems, revoke compromised credentials
3. **Assessment**: Determine scope of breach, data affected
4. **Notification**: Notify authorities within 72 hours if required
5. **User Communication**: Inform affected users with guidance
6. **Remediation**: Patch vulnerabilities, enhance controls
7. **Post-Incident**: Review and update security measures

Chapter 4. Architecture Decisions

Architecture Decision Records (ADRs) serve several important purposes and follow a specific process. Here's an explanation of their purpose and process:

Purpose of ADRs:

1. Document key decisions: ADRs capture significant architectural decisions made during the design and development of a software system 1 2.
2. Provide context: They explain the context, rationale, and consequences of each decision 3.
3. Create a decision log: ADRs collectively form a decision log that provides project context and detailed implementation information 3.
4. Facilitate communication: They help team members understand and communicate design choices 4.
5. Support future reference: ADRs serve as a reference for code and architectural reviews 3.
6. Track evolution: They allow documentation to reflect specific states of a system over time 4.
7. Evidence thinking process: ADRs demonstrate the reasoning behind decisions, which can be valuable for audits or future team members 4.

Process of creating and using ADRs:

1. Identification: Recognize an architecturally significant decision that needs to be made 1 3.
2. Authorship: An team member (ADR owner) creates a new ADR document 3.
3. Content creation: The ADR owner fills in the required sections, typically including:
    - Title
    - Status (e.g., proposed, accepted, rejected)
    - Context and problem statement
    - Decision drivers
    - Considered options
    - Decision outcome
    - Pros and cons
    - Mitigation strategy 4
4. Review: The team reviews the proposed ADR, often using a pull request process in a version control system 3 4.
5. Revision: If necessary, the ADR owner makes revisions based on team feedback 3.
6. Acceptance or Rejection: The team decides to accept, reject, or request further work on the ADR 3.
7. Finalization: If accepted, the ADR owner updates the status, adds a timestamp, version, and list of stakeholders 3.
8. Storage: ADRs are typically stored as plain text (often Markdown) files in a version control

system, usually in a `/docs/adrs` folder within the project repository 4.

9. Ongoing use: The team refers to ADRs during development, code reviews, and future decision-making processes 3.
10. Superseding: If a decision changes, a new ADR is created to supersede the old one, rather than modifying the original ADR 3.

By following this process, teams can maintain a clear, versioned history of important decisions throughout a project's lifecycle, improving communication, understanding, and long-term maintainability of the software architecture.

4.1. ADR 001: Use Hedera Token Service (HTS) for Loyalty Points Tokenization

Status: Accepted

Date: December 2025

Context: Enda Tamweel requires a loyalty points system that provides:

- Transparent and immutable record of point balances
- Ability to mint and burn tokens based on business rules
- Low transaction costs for frequent operations
- Integration with existing banking systems
- Regulatory compliance for financial operations

Alternative approaches considered:

1. **Traditional Database Only**: Store points in PostgreSQL without blockchain
2. **Ethereum ERC-20 Token**: Deploy custom smart contract on Ethereum
3. **Private Blockchain**: Deploy on Hyperledger or private network
4. **Hedera Token Service (HTS)**: Use Hedera's native tokenization service

Decision: We will use **Hedera Token Service (HTS)** for tokenizing loyalty points.

Rationale:

- **Native Tokenization**: HTS provides built-in token functionality without custom smart contracts
- **Low Fees**: Predictable, low-cost transactions ($0.001 or less per operation)
- **Fast Finality**: 3-5 second consensus time for all transactions
- **Enterprise Grade**: Governed by major enterprises, suitable for financial applications
- **Built-in Controls**: Native support for minting, burning, and admin keys
- **Compliance**: Transaction fees in USD-equivalent, easy for accounting
- **SDK Support**: Official JavaScript SDK integrates well with NestJS backend

Consequences:

Positive:

- Transparent, auditable record of all token operations on public ledger
- No smart contract vulnerabilities to manage
- Low operational costs compared to Ethereum
- Fast transaction confirmation for good user experience

Negative:

- Dependency on Hedera network availability
- On-chain data is public (mitigated by not storing PII on-chain)
- Requires HBAR for transaction fees
- Team needs to learn Hedera SDK

4.2. ADR 002: Use NestJS as Backend Framework

Status: Accepted

Date: December 2025

Context: The backend service needs to handle:

- RESTful API endpoints for web and mobile clients
- Scheduled batch jobs for daily data processing
- Integration with Hedera SDK (JavaScript/TypeScript)
- Integration with PostgreSQL and Redis
- Secure authentication and authorization
- Scalable architecture for growing user base

Alternative frameworks considered:

1. **Express.js**: Minimal, flexible Node.js framework
2. **Fastify**: High-performance Node.js framework
3. **NestJS**: Enterprise-grade TypeScript framework
4. **Django (Python)**: Full-featured Python web framework
5. **Spring Boot (Java)**: Enterprise Java framework

Decision: We will use **NestJS** as the backend framework.

Rationale:

- **TypeScript Native**: Strong typing reduces bugs and improves maintainability

- **Modular Architecture**: Built-in module system aligns with our component design
- **Dependency Injection**: Enterprise patterns for testable, maintainable code
- **Ecosystem**: Excellent packages for PostgreSQL (TypeORM), Redis (Bull), JWT
- **Hedera Compatibility**: JavaScript SDK works seamlessly with TypeScript
- **Team Expertise**: Development team has strong TypeScript experience
- **Documentation**: Comprehensive documentation and active community

Consequences:

Positive:

- Clean, modular codebase with clear separation of concerns
- Strong typing catches errors at compile time
- Easy to write unit and integration tests
- Excellent support for background job processing
- Guards and interceptors simplify authentication/authorization

Negative:

- Steeper learning curve than Express.js
- More boilerplate code for simple endpoints
- Requires TypeScript build step

4.2.1. Related Requirements

- R7 - JWT-based authentication
- Q6 - Maintainability

4.3. ADR 003: JWT Authentication with Refresh Token Rotation

Status: Accepted

Date: December 2025

Context: The platform requires secure authentication for three user types (Client, Admin, Super Admin) with:

- Protection against common web vulnerabilities (XSS, CSRF)
- Seamless user experience without frequent re-authentication
- Support for web and mobile applications
- Ability to revoke sessions immediately
- Audit trail of authentication events

Alternative approaches considered:

1. **Session-based Authentication**: Traditional server-side sessions
2. **Simple JWT**: Single long-lived JWT token
3. **OAuth 2.0 with External Provider**: Delegate authentication to Auth0, Okta
4. **JWT with Refresh Token Rotation**: Short-lived access tokens with rotating refresh tokens

Decision: We will implement **JWT authentication with refresh token rotation**.

Implementation details:

- **Access Token**:
  - Short-lived (15 minutes)
  - Contains user ID, role, username
  - Stored in frontend memory only (not localStorage)
- **Refresh Token**:
  - Long-lived (7 days)
  - Stored as HTTP-only, Secure cookie
  - Hashed before database storage
  - Rotated on each refresh (old token invalidated)

Rationale:

- **XSS Protection**: Access token in memory cannot be stolen by XSS attacks
- **CSRF Protection**: HTTP-only cookies with SameSite=Strict prevent CSRF
- **Token Theft Mitigation**: Rotation means stolen refresh token becomes invalid quickly
- **Session Control**: Database storage allows immediate revocation
- **Mobile Support**: Works with both web (cookies) and mobile (secure storage)
- **Audit Capability**: All authentication events can be logged

Consequences:

Positive:

- Strong security against common attack vectors
- Users stay logged in for extended periods without re-authentication
- Immediate session revocation capability
- Clear audit trail of authentication events

Negative:

- More complex implementation than simple JWT
- Requires database lookup for refresh token validation

- Cookie handling differs between web and mobile

4.3.1. Security Controls

- Refresh tokens hashed with bcrypt before storage
- Cookie settings: HttpOnly=true, Secure=true, SameSite=Strict
- Rate limiting on authentication endpoints
- Account lockout after failed attempts

4.3.2. Related Requirements

- R7 - JWT-based authentication
- Q1 - Security

4.4. ADR 004: Custodial Wallet Architecture for Client Hedera Accounts

Status: Accepted

Date: December 2025

Context: Each client needs a Hedera account to hold their loyalty tokens. The key management approach affects:

- User experience (onboarding complexity)
- Security responsibility distribution
- Operational requirements
- Recovery procedures

Alternative approaches considered:

1. **Non-Custodial (User Holds Keys)**: Users manage their own Hedera keys
2. **Hardware Wallet Integration**: Users use hardware wallets
3. **MPC (Multi-Party Computation)**: Keys split between user and platform
4. **Fully Custodial**: Platform manages all client keys

Decision: We will implement a **fully custodial wallet architecture** where the platform manages all client Hedera account keys.

Implementation:

- Hedera accounts created automatically during user registration
- Private keys encrypted with AES-256-GCM using per-user derived keys
- Encryption master key stored in cloud KMS (AWS KMS / Azure Key Vault)

- Platform admin key controls all mint/burn operations
- Users cannot transfer tokens independently

Rationale:

- **User Experience**: Clients interact with familiar web/mobile interface
- **Target Audience**: Microfinance clients may not be blockchain-savvy
- **Operational Control**: Platform maintains full control over token lifecycle
- **Key Recovery**: Platform can recover access if database is restored
- **Closed Ecosystem**: Tokens cannot be used outside the platform anyway
- **Simplified Support**: No need for user key recovery procedures

Consequences:

Positive:

- Seamless onboarding - users don't need to understand blockchain
- No risk of users losing access due to lost keys
- Simplified mobile app without wallet complexity
- Full control over token operations

Negative:

- Platform is fully responsible for key security
- Single point of failure if platform is compromised
- Users must trust platform completely
- Not "true" decentralization

4.4.1. Security Requirements

- Private keys encrypted at rest with AES-256-GCM
- Encryption keys managed by cloud KMS with strict IAM
- Keys decrypted only in memory during signing operations
- Database backups encrypted separately
- Regular key rotation for encryption keys

4.4.2. Related Requirements

- R1 - Automatic Hedera wallet creation
- Q1 - Security

4.5. ADR 005: Daily Batch Processing for Points Calculation and Token Minting

Status: Accepted

Date: December 2025

Context: The platform needs to calculate loyalty points based on client activity data from Middle DB and mint corresponding tokens. Key considerations:

- Middle DB is updated daily with J-1 (previous day) data
- Thousands of clients need points calculated
- Token minting has associated Hedera network fees
- System should handle processing failures gracefully
- Results need to be auditable

Alternative approaches considered:

1. **Real-time Processing**: Calculate and mint on each transaction
2. **Hourly Batch**: Process every hour
3. **Daily Batch**: Process once daily aligned with Middle DB refresh
4. **Weekly Batch**: Aggregate and process weekly

Decision: We will implement **daily batch processing** for points calculation and token minting.

Processing schedule:

- **Trigger**: Scheduled cron job at 2:00 AM (after Middle DB J-1 refresh)
- **Data Extraction**: Read all client activity from Middle DB
- **Points Calculation**: Apply business rules to compute daily points
- **Token Minting**: Batch mint tokens to client accounts
- **Completion Target**: Process complete by 6:00 AM

Rationale:

- **Data Alignment**: Matches Middle DB J-1 update cycle
- **Cost Efficiency**: Batch minting reduces per-transaction overhead
- **System Load**: Processing during off-peak hours
- **Business Fit**: Daily points aligns with Enda Tamweel's operational model
- **Error Handling**: Daily cycle allows time for retry and manual intervention

Implementation Details:

Job Queue Architecture

```
Daily Sync Job
 ├──── Data Extraction (from Middle DB)
 ├──── Points Calculation (parallel workers)
 │     ├──── Chunk 1 (100 clients)
 │     ├──── Chunk 2 (100 clients)
 │     └──── ...
 ├──── Token Minting (batch operations)
 │     ├──── Batch 1 (max 100 operations)
 │     ├──── Batch 2 (max 100 operations)
 │     └──── ...
 ├──── Reconciliation Check
 └──── Summary Report
```

Checkpointing

- Progress saved after each chunk
- Failed jobs resume from last checkpoint
- Maximum 3 automatic retries with exponential backoff

Monitoring

- Job start/completion alerts
- Error rate monitoring
- Processing time tracking
- Automatic escalation if not complete by deadline

Consequences:

Positive:

- Efficient use of Hedera network with batched operations
- Clear processing window with defined completion time
- Easy to monitor and troubleshoot
- Aligns with existing data refresh cycles

Negative:

- Points not reflected in real-time (24-hour delay)
- Large processing window if client base grows significantly
- Failure affects all clients for that day

4.5.4. Related Requirements

- R2 - Daily extraction from Middle DB
- R3 - Rule-based points calculation
- R4 - Automatic minting
- Q4 - Scalability

Chapter 5. Quality Assurance Testing Strategy

5.1. Testing Scope

The testing team will execute validation of the changes that include but not limited to the following activities:

- Integration
- System Security
- Authorization & Authentication

5.1.1. Test Platform

- Website
- Mobile App

5.1.2. Tools

- Website & Mobile Manual Testing Tools: Test Link, MS Azure DevOps
- Website Automation Test Tools: JMETER, Selenium & Python
- Mobile Automation Testing Tools: Appium, Java and BrowserStack (Multiple devices)

5.1.3. Test Types

- End-to-End Testing: Testing entire product from beginning to end to ensure the application flow with expected behaviours.
- Functional Testing: Aligns & validate the system functional requirements / specifications.
- Non-Functional Testing: Load & Performance Testing.

5.1.4. Performance Testing

- Website Performance Testing Tools: JMETER
    - Up to max. 1000 concurrent users
    - Up to 95% Loading Performance

5.1.5. Product

- All

5.1.6. Supported Mobile Devices

- iOS 17+

- Android 14+

5.2. Testing Levels

This section describes the different levels of testing implemented for the Enda Tamweel Loyalty Platform.

5.2.1. Unit Testing

Scope

Individual functions, methods, and classes in isolation.

Tools

- **Framework**: Jest
- **Mocking**: jest-mock, ts-mockito
- **Coverage**: Istanbul/nyc

Coverage Targets

Module	Minimum Coverage	Priority
Points Engine	90%	Critical - core business logic
Auth Module	85%	Critical - security
Token Module	80%	High - Hedera operations
Gift Module	80%	High - business workflow
API Controllers	70%	Medium

Unit Test Examples

Component	Test Cases
Points Calculator	Points calculation rules, edge cases, error handling
JWT Service	Token generation, validation, expiry handling
Wallet Service	Key encryption/decryption, account creation
Redemption Service	Request validation, status transitions

5.2.2. Integration Testing

Scope

Interaction between modules and external dependencies.

Tools

- **Framework**: Jest with supertest
- **Database**: Testcontainers (PostgreSQL)
- **Hedera**: Hedera Testnet
- **Redis**: Testcontainers (Redis)

Integration Test Scenarios

Scenario	Validations
User Registration Flow	Database record created, Hedera account created, wallet encrypted
Authentication Flow	Login creates session, refresh rotates tokens, logout invalidates
Points Calculation	Middle DB data processed, database updated, tokens minted
Gift Redemption	Request persisted, approval triggers burn, balance updated

External System Integration

System	Test Approach	Environment
PostgreSQL	Testcontainers with schema migration	Isolated container
Redis	Testcontainers	Isolated container
Hedera Network	Hedera Testnet with test accounts	Public testnet
Middle DB	Mock data or staging replica	Mock/Staging

5.2.3. End-to-End (E2E) Testing

Scope

Complete user journeys through the entire system.

Tools

- **Web**: Playwright
- **Mobile**: Detox (React Native)
- **API**: Postman/Newman

E2E Test Scenarios

Client Journey

Test Case	Steps
New Client Onboarding	Register → Verify account created → View initial balance (0)
Check Balance	Login → Navigate to dashboard → Verify balance matches expected

Test Case	Steps
Request Gift Redemption	Login → Browse gifts → Select gift → Submit request → Verify pending status
View Transaction History	Login → Navigate to history → Verify past transactions displayed

Admin Journey

Test Case	Steps
Approve Redemption	Login as Admin → View pending → Approve → Verify tokens burned
Reject Redemption	Login as Admin → View pending → Reject with reason → Verify status
Manage Gift Catalog	Login as Admin → Add gift → Edit gift → Verify updates

Super Admin Journey

Test Case	Steps
Manage Admins	Login as Super Admin → Create admin → Modify permissions → Deactivate
View Audit Logs	Login as Super Admin → Navigate to audit → Filter by date/action

5.2.4. API Contract Testing

Scope

Validate API contracts and backward compatibility.

Tools

- **Contract Definition**: OpenAPI 3.0
- **Contract Testing**: Dredd, Pact

API Tests

Endpoint	Method	Validations
/auth/login	POST	Request/response schema, error codes
/users/me	GET	Authentication required, response structure
/balance	GET	Correct balance returned, caching behavior
/redemptions	POST	Validation errors, success response

5.2.5. Performance Testing

Tools

- **Load Testing**: k6, JMeter
- **Profiling**: clinic.js, 0x

Performance Test Scenarios

Scenario	Users	Duration	Success Criteria
Normal Load	100 concurrent	30 minutes	P95 < 500ms, 0% errors
Peak Load	500 concurrent	15 minutes	P95 < 1000ms, < 0.1% errors
Batch Processing	10,000 records	Single run	Complete < 4 hours
Stress Test	1000 concurrent	5 minutes	Graceful degradation

5.2.6. Security Testing

Tools

- **SAST**: SonarQube, Semgrep
- **DAST**: OWASP ZAP
- **Dependency Scan**: Snyk, npm audit

Security Test Areas

Area	Tests
Authentication	Brute force protection, session management, token validation
Authorization	Role enforcement, privilege escalation attempts
Input Validation	SQL injection, XSS, command injection
API Security	Rate limiting, CORS, security headers
Data Protection	Encryption validation, key management

5.2.7. Test Environment Strategy

Environment	Purpose	Data	Hedera Network
Local	Developer testing	Mock/seed data	Testnet or Mock
CI/CD	Automated tests	Generated test data	Testnet
Staging	Integration & E2E	Anonymized production	Testnet

Environment	Purpose	Data	Hedera Network
Pre-Production	Final validation	Production replica	Testnet

5.3. Testing Strategy

Section	Details
1. Test Strategy	1.1 Automation / Manual Test Strategy 1.2 Test Schedule 1.3 Resource Planning
2. Test Development	2.1 Test Plans 2.2 Test Scripts 2.3 Test Data
3. Test Execution	3.1 Defects 3.2 Test Reports 3.3 Test Metrics (Traceability Matrix)
4. Defect Management	4.1 Bug Tracking 4.2 Bug Fixing 4.3 Bug Verification
5. Delivery	5.1 UAT 5.2 Installation Testing Environment 5.3 Requirements Verification

Chapter 6. Risk & Technical Debt

6.1. Risks

The following risks represent potential future uncertainties that may impact the Enda Tamweel Loyalty Platform if not addressed:

ID	Severity	Description	Remediation	Status
R1	High	Hedera network outage or degradation could prevent token operations (minting, burning, balance queries)	Implement circuit breaker pattern; queue failed operations for retry; maintain cached balances in database; monitor Hedera network status	Monitoring in place
R2	High	Private key compromise could allow unauthorized token operations	Keys encrypted with AES-256 and stored in KMS; strict access controls; regular security audits; key rotation procedures	Mitigated
R3	Medium	Middle DB integration failure could halt daily points calculation	Implement retry logic; manual trigger capability; alerting for sync failures; fallback to previous day's data if critical	Monitoring in place
R4	Medium	Rapid user growth exceeds batch processing capacity	Horizontal scaling of worker pods; chunked processing; performance monitoring; capacity planning reviews	Planned
R5	Medium	Database discrepancy between application DB and on-chain token balances	Daily reconciliation job; automated alerts for discrepancies; manual review process; audit logging	Implemented
R6	Low	Users lose access to accounts and cannot recover	Standard password reset flow; admin-assisted account recovery; no user-held keys to lose (custodial model)	Mitigated by design
R7	Medium	HBAR price volatility affects operational costs	Budget buffer for fee fluctuations; batch operations to minimize transactions; monitor daily costs; alert on budget threshold	Monitoring in place

ID	Severity	Description	Remediation	Status
R8	High	Data breach exposing user personal information or encrypted keys	Encryption at rest and in transit; WAF and DDoS protection; incident response plan; regular penetration testing	Security measures active
R9	Medium	Regulatory changes affecting blockchain token operations in Tunisia	Monitor regulatory landscape; design for flexibility; maintain compliance documentation; legal review of operations	Under monitoring
R10	Low	Third-party dependency vulnerabilities (npm packages, SDKs)	Automated vulnerability scanning (Snyk); regular dependency updates; security-focused code reviews	CI/CD integrated

6.1.1. Risk Matrix

	Low Impact	Medium Impact	High Impact
High Likelihood		R4, R7	R1, R8
Medium Likelihood	R10	R3, R5, R9	R2
Low Likelihood	R6		

6.1.2. Risk Monitoring

Automated Monitoring

- Hedera network status integration
- Database reconciliation alerts
- Security vulnerability scanning
- Cost tracking and budget alerts

Regular Reviews

- Weekly: Operational risks review
- Monthly: Security posture assessment
- Quarterly: Full risk register review

6.2. Technical Debt

The following technical debt items represent known deficiencies in the current system implementation. They are acknowledged and planned for future resolution:

ID	Severity	Description	Remediation	Target
TD1	High	Hedera operator credentials stored in environment variables rather than secure vault	Migrate to HashiCorp Vault or cloud KMS for all sensitive credentials	Sprint 3
TD2	Medium	Limited automated test coverage for Hedera integration layer	Add comprehensive integration tests using Hedera Testnet; mock SDK for unit tests	Sprint 4
TD3	Medium	Points calculation rules hardcoded in service; no admin interface for rule management	Implement rule engine with database-driven configuration; add admin UI for rule management	Phase 2
TD4	Low	Batch processing does not support partial failure recovery from exact failure point	Implement fine-grained checkpointing; store progress per client; resume from exact failure point	Sprint 5
TD5	Medium	No rate limiting on internal service-to-service communication	Implement service mesh with rate limiting; add circuit breakers between services	Sprint 6
TD6	Low	Audit log query performance degrades with large datasets	Implement log partitioning; add time-based indexes; consider moving to time-series database	Phase 2
TD7	Medium	Mobile app stores JWT access token in secure storage instead of memory	Refactor mobile auth to use memory-only storage with background refresh	Sprint 4
TD8	Low	API documentation not automatically generated from code annotations	Integrate Swagger/OpenAPI generation from NestJS decorators; publish to developer portal	Sprint 3
TD9	Medium	No automated backup verification for encrypted private keys	Implement automated backup restore testing; verify key decryption from backups monthly	Sprint 5
TD10	Low	Frontend components not fully accessible (WCAG compliance)	Conduct accessibility audit; implement ARIA labels; keyboard navigation support	Phase 2

6.2.1. Technical Debt Tracking

Prioritization Criteria

- **Severity**: Impact on security, reliability, or user experience

- **Effort**: Development time required for resolution
- **Risk**: Likelihood of the debt causing issues
- **Business Value**: Benefit from resolution

Review Cadence

- **Weekly**: Review new debt items, update status
- **Sprint Planning**: Select debt items for inclusion
- **Monthly**: Technical debt burndown review
- **Quarterly**: Comprehensive debt assessment

6.2.2. Debt Reduction Strategy

Sprint Allocation	Approach
20% capacity	Dedicated time each sprint for debt reduction
Boy Scout Rule	Leave code better than you found it
Refactoring Stories	Major debt items tracked as user stories
Tech Spikes	Time-boxed investigation for complex debt

Chapter 7. Features Traceability Matrix (RTM)

This matrix traces platform features to their requirements, quality attributes, architecture decisions, and identified risks.

ID	Feature	Requirements	Quality Attributes	Arch. Decisions	Risks/Tech Debt
F1	**User Authentication**: Secure login with JWT tokens and refresh token rotation, protecting user sessions with HTTP-only cookies and in-memory access tokens.	R7 - JWT Auth	Q1 - Security	ADR-003 Auth Strategy	TD7 - Mobile Token Storage
F2	**User Registration**: New client onboarding with automatic Hedera wallet creation and secure key storage.	R1 - Wallet Creation	Q1 - Security, Q6 - Maintainability	ADR-004 Custodial Wallet	R2 - Key Compromise
F3	**Daily Points Calculation**: Automated daily extraction from Middle DB and rule-based points calculation.	R2 - Daily Extraction, R3 - Points Engine	Q2 - Reliability, Q4 - Scalability	ADR-005 Batch Processing	R3 - Middle DB Integration, TD3 - Hardcoded Rules
F4	**Token Minting**: Automatic minting of HTS loyalty tokens based on calculated points.	R4 - Auto Minting, IR2 - HTS Integration	Q2 - Reliability, Q7 - Auditability	ADR-001 HTS	R1 - Hedera Outage, R7 - HBAR Costs
F5	**Gift Redemption**: Client gift requests with admin approval workflow and token burning.	R5 - Redemption Workflow	Q7 - Auditability, Q8 - Data Integrity	ADR-001 HTS	R5 - Balance Discrepancy
F6	**Admin Management**: Super Admin panel for Admin account lifecycle management.	R6 - Super Admin Panel	Q1 - Security, Q7 - Auditability	ADR-003 Auth Strategy	N/A

ID	Feature	Requirements	Quality Attributes	Arch. Decisions	Risks/Tech Debt
F7	**Balance Reconciliation**: Regular comparison of database and on-chain token balances.	R8 - Reconciliation	Q8 - Data Integrity	ADR-001 HTS	R5 - Balance Discrepancy
F8	**Transaction History**: Storage and retrieval of historical point calculations and redemption logs.	R9 - History Storage	Q7 - Auditability	ADR-002 NestJS	TD6 - Audit Log Performance
F9	**Web Application**: React-based web interface for clients, admins, and super admins.	R10 - Web/Mobile Apps	Q3 - Performance, Q5 - Availability	ADR-002 NestJS	TD10 - Accessibility
F10	**Mobile Application**: React Native mobile app for iOS and Android.	R10 - Web/Mobile Apps	Q3 - Performance	ADR-003 Auth Strategy	TD7 - Mobile Token Storage

7.1. Coverage Summary

7.1.1. Requirements Coverage

Requirement	Covered By	Status
R1 - Wallet Creation	F2	✔
R2 - Daily Extraction	F3	✔
R3 - Points Engine	F3	✔
R4 - Auto Minting	F4	✔
R5 - Redemption Workflow	F5	✔
R6 - Super Admin Panel	F6	✔
R7 - JWT Auth	F1	✔
R8 - Reconciliation	F7	✔
R9 - History Storage	F8	✔

Requirement	Covered By	Status
R10 - Web/Mobile Apps	F9, F10	✔
IR1 - Middle DB Integration	F3	✔
IR2 - HTS Integration	F4, F5	✔
IR3 - Hedera Account Service	F2	✔

7.1.2. Quality Attributes Coverage

Quality Attribute	Covered By Features
Q1 - Security	F1, F2, F6
Q2 - Reliability	F3, F4
Q3 - Performance	F9, F10
Q4 - Scalability	F3
Q5 - Availability	F9
Q6 - Maintainability	F2
Q7 - Auditability	F4, F5, F6, F8
Q8 - Data Integrity	F5, F7

7.1.3. ADR Coverage

ADR	Related Features
ADR-001 HTS Token	F4, F5, F7
ADR-002 NestJS	F8, F9
ADR-003 Authentication	F1, F6, F10
ADR-004 Custodial Wallet	F2
ADR-005 Batch Processing	F3