# Rewards Program System - Implementation Guide

This document provides comprehensive guidelines for building a rewards program system using Supabase and the existing KV store architecture.

## Table of Contents

1. [System Overview](#system-overview)

2. [Data Models](#data-models)

3. [KV Store Schema](#kv-store-schema)

4. [Server API Endpoints](#server-api-endpoints)

5. [Frontend Integration](#frontend-integration)

6. [Authentication & Security](#authentication--security)

7. [Business Logic](#business-logic)

8. [Implementation Checklist](#implementation-checklist)

## System Overview

The rewards program consists of four main components:

- \*\*User Rewards\*\*: Points balance, level progression, and statistics

- \*\*Activities\*\*: Ways users can earn points (daily, weekly, one-time, repeatable)

- \*\*Rewards Catalog\*\*: Items users can redeem with points

- \*\*Transaction History\*\*: Record of all point earnings and redemptions

`

## Data Models

### UserRewards

```typescript

interface UserRewards {

userId: string;

currentPoints: number;

totalEarned: number;

level: number;

nextLevelPoints: number;

currentLevelPoints: number;

createdAt: string;

updatedAt: string;

}

```

### Activity

```typescript

interface Activity {

id: string;

title: string;

description: string;

points: number;

type: 'daily' | 'weekly' | 'one-time' | 'repeatable';

icon: string;

active: boolean;

maxProgress?: number;

createdAt: string;

updatedAt: string;

}

```

### UserActivity

```typescript

interface UserActivity {

userId: string;

activityId: string;

completed: boolean;

progress: number;

lastCompletedAt?: string;

completionCount: number; // For repeatable activities

createdAt: string;

updatedAt: string;

}

```

### Reward

```typescript

interface Reward {

id: string;

title: string;

description: string;

pointsCost: number;

category: string;

imageUrl: string;

available: boolean;

estimatedDelivery?: string;

featured: boolean;

maxRedemptions?: number;

currentRedemptions: number;

createdAt: string;

updatedAt: string;

}

```

### Transaction

```typescript

interface Transaction {

id: string;

userId: string;

type: 'earned' | 'redeemed';

description: string;

points: number;

category: string;

status: 'completed' | 'pending' | 'cancelled';

metadata?: Record<string, any>;

createdAt: string;

updatedAt: string;

}

```

### UserRedemption

```typescript

interface UserRedemption {

id: string;

userId: string;

rewardId: string;

transactionId: string;

status: 'pending' | 'processing' | 'fulfilled' | 'cancelled';

fulfillmentData?: Record<string, any>;

createdAt: string;

updatedAt: string;

}

```

## KV Store Schema

Use the following key patterns for storing data in the KV store:

### Key Patterns

```typescript

// User data

`user:${userId}:rewards` → UserRewards

`user:${userId}:activities` → UserActivity[]

`user:${userId}:transactions` → Transaction[]

`user:${userId}:redemptions` → UserRedemption[]

// Global data

`activities:all` → Activity[]

`rewards:all` → Reward[]

`rewards:featured` → Reward[]

`rewards:category:${category}` → Reward[]

// Daily/Weekly tracking

`user:${userId}:daily:${date}` → string[] (completed activity IDs)

`user:${userId}:weekly:${weekKey}` → { activityId: completionCount }

// Leaderboards (optional)

`leaderboard:points:top100` → { userId: string, points: number, level: number }[]

`leaderboard:level:top100` → { userId: string, level: number, points: number }[]

```

## Server API Endpoints

Create the following endpoints in `/supabase/functions/server/index.tsx`:

### User Rewards Endpoints

```typescript

// GET /make-server-64cb7b77/user/rewards

// Returns user's current rewards status

app.get('/make-server-64cb7b77/user/rewards', async (c) => {

// Verify auth, get user rewards from KV store

});

// POST /make-server-64cb7b77/user/rewards/initialize

// Initialize rewards for new user

app.post('/make-server-64cb7b77/user/rewards/initialize', async (c) => {

// Create initial user rewards record

});

```

### Activities Endpoints

```typescript

// GET /make-server-64cb7b77/activities

// Returns all available activities with user progress

app.get('/make-server-64cb7b77/activities', async (c) => {

// Get activities and merge with user progress

});

// POST /make-server-64cb7b77/activities/:id/complete

// Complete an activity and award points

app.post('/make-server-64cb7b77/activities/:id/complete', async (c) => {

// Validate activity completion, award points, update level

});

// GET /make-server-64cb7b77/activities/daily-reset

// Reset daily activities (called by cron or user login)

app.get('/make-server-64cb7b77/activities/daily-reset', async (c) => {

// Reset daily activities for user

});

```

### Rewards Endpoints

```typescript

// GET /make-server-64cb7b77/rewards

// Returns available rewards catalog

app.get('/make-server-64cb7b77/rewards', async (c) => {

// Get rewards catalog with availability

});

// POST /make-server-64cb7b77/rewards/:id/redeem

// Redeem a reward

app.post('/make-server-64cb7b77/rewards/:id/redeem', async (c) => {

// Validate points, deduct points, create redemption record

});

// GET /make-server-64cb7b77/rewards/categories

// Returns reward categories

app.get('/make-server-64cb7b77/rewards/categories', async (c) => {

// Get unique reward categories

});

```

### Transaction Endpoints

```typescript

// GET /make-server-64cb7b77/transactions

// Returns user's transaction history

app.get('/make-server-64cb7b77/transactions', async (c) => {

// Get paginated transaction history

});

// GET /make-server-64cb7b77/transactions/stats

// Returns transaction statistics

app.get('/make-server-64cb7b77/transactions/stats', async (c) => {

// Calculate earnings/spending stats

});

```

## Frontend Integration

### API Client Setup

```typescript

// utils/api/rewards.ts

import { projectId, publicAnonKey } from '../supabase/info';

const API\_BASE = `https://${projectId}.supabase.co/functions/v1/make-server-64cb7b77`;

export class RewardsAPI {

private async request(endpoint: string, options: RequestInit = {}) {

const response = await fetch(`${API\_BASE}${endpoint}`, {

headers: {

'Authorization': `Bearer ${publicAnonKey}`,

'Content-Type': 'application/json',

...options.headers,

},

...options,

});

if (!response.ok) {

throw new Error(`API Error: ${response.statusText}`);

}

return response.json();

}

async getUserRewards() {

return this.request('/user/rewards');

}

async getActivities() {

return this.request('/activities');

}

async completeActivity(activityId: string) {

return this.request(`/activities/${activityId}/complete`, {

method: 'POST',

});

}

async getRewards() {

return this.request('/rewards');

}

async redeemReward(rewardId: string) {

return this.request(`/rewards/${rewardId}/redeem`, {

method: 'POST',

});

}

async getTransactions(page = 1, limit = 20) {

return this.request(`/transactions?page=${page}&limit=${limit}`);

}

}

```

### Component Integration Pattern

```typescript

// hooks/useRewards.ts

import { useState, useEffect } from 'react';

import { RewardsAPI } from '../utils/api/rewards';

export function useRewards() {

const [userRewards, setUserRewards] = useState(null);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

const api = new RewardsAPI();

useEffect(() => {

loadUserRewards();

}, []);

const loadUserRewards = async () => {

try {

const rewards = await api.getUserRewards();

setUserRewards(rewards);

} catch (err) {

setError(err.message);

} finally {

setLoading(false);

}

};

const completeActivity = async (activityId: string) => {

try {

const result = await api.completeActivity(activityId);

await loadUserRewards(); // Refresh rewards

return result;

} catch (err) {

throw err;

}

};

return {

userRewards,

loading,

error,

completeActivity,

refresh: loadUserRewards,

};

}

```

## Authentication & Security

### Required Auth Setup

```typescript

// Server-side auth verification

const verifyAuth = async (request: Request) => {

const accessToken = request.headers.get('Authorization')?.split(' ')[1];

const { data: { user }, error } = await supabase.auth.getUser(accessToken);

if (!user?.id) {

throw new Error('Unauthorized');

}

return user.id;

};

// Use in protected routes

app.post('/make-server-64cb7b77/activities/:id/complete', async (c) => {

try {

const userId = await verifyAuth(c.req.raw);

// ... rest of the logic

} catch (error) {

return c.json({ error: 'Unauthorized' }, 401);

}

});

```

### Frontend Auth Integration

```typescript

// Use Supabase auth in frontend

import { createClient } from '@supabase/supabase-js';

const supabase = createClient(

process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!

);

// Get auth token for API calls

const getAuthToken = async () => {

const { data: { session } } = await supabase.auth.getSession();

return session?.access\_token;

};

```

## Business Logic

### Level Progression Algorithm

```typescript

const calculateLevel = (totalPoints: number): { level: number, currentLevelPoints: number, nextLevelPoints: number } => {

// Each level requires increasingly more points

// Level 1: 0-499, Level 2: 500-1499, Level 3: 1500-2999, etc.

let level = 1;

let currentLevelPoints = 0;

let accumulatedPoints = 0;

while (accumulatedPoints + (level \* 500) <= totalPoints) {

currentLevelPoints = accumulatedPoints + (level \* 500);

accumulatedPoints += (level \* 500);

level++;

}

const nextLevelPoints = accumulatedPoints + (level \* 500);

return { level, currentLevelPoints, nextLevelPoints };

};

```

### Activity Reset Logic

```typescript

const resetDailyActivities = async (userId: string) => {

const today = new Date().toISOString().split('T')[0];

const userActivities = await kv.get(`user:${userId}:activities`) || [];

const updatedActivities = userActivities.map(activity => {

if (activity.type === 'daily') {

return { ...activity, completed: false, progress: 0 };

}

return activity;

});

await kv.set(`user:${userId}:activities`, updatedActivities);

await kv.del(`user:${userId}:daily:${today}`);

};

```

### Points Transaction System

```typescript

const awardPoints = async (userId: string, points: number, description: string, category: string) => {

// Get current rewards

const userRewards = await kv.get(`user:${userId}:rewards`);

// Calculate new totals and level

const newCurrentPoints = userRewards.currentPoints + points;

const newTotalEarned = userRewards.totalEarned + points;

const levelInfo = calculateLevel(newTotalEarned);

// Create transaction

const transaction = {

id: crypto.randomUUID(),

userId,

type: 'earned',

description,

points,

category,

status: 'completed',

createdAt: new Date().toISOString(),

updatedAt: new Date().toISOString(),

};

// Update rewards

const updatedRewards = {

...userRewards,

currentPoints: newCurrentPoints,

totalEarned: newTotalEarned,

...levelInfo,

updatedAt: new Date().toISOString(),

};

// Save to KV store

await kv.set(`user:${userId}:rewards`, updatedRewards);

const transactions = await kv.get(`user:${userId}:transactions`) || [];

transactions.unshift(transaction);

await kv.set(`user:${userId}:transactions`, transactions.slice(0, 100)); // Keep last 100

return { updatedRewards, transaction, levelUp: levelInfo.level > userRewards.level };

};

```

## Implementation Checklist

### Backend Setup

- [ ] Create all server endpoints in `/supabase/functions/server/index.tsx`

- [ ] Implement authentication middleware

- [ ] Set up KV store data patterns

- [ ] Create utility functions for level calculation

- [ ] Implement activity reset logic

- [ ] Add error handling and logging

- [ ] Set up CORS headers

### Frontend Integration

- [ ] Create API client utility

- [ ] Build React hooks for data fetching

- [ ] Update existing components to use real API

- [ ] Add loading states and error handling

- [ ] Implement optimistic updates for better UX

- [ ] Add real-time updates (optional)

### Data Management

- [ ] Create initial activities data

- [ ] Set up rewards catalog

- [ ] Implement data seeding scripts

- [ ] Add data validation

- [ ] Set up backup/export functionality

### Testing & Deployment

- [ ] Test all API endpoints

- [ ] Test authentication flow

- [ ] Test level progression logic

- [ ] Test reward redemption flow

- [ ] Performance testing with large datasets

- [ ] Deploy to production environment

### Optional Enhancements

- [ ] Leaderboards

- [ ] Social sharing

- [ ] Push notifications

- [ ] Analytics tracking

- [ ] A/B testing framework

- [ ] Referral system

- [ ] Seasonal campaigns

## Key Considerations

1. \*\*Data Consistency\*\*: Use transactions when updating multiple KV entries

2. \*\*Performance\*\*: Implement caching for frequently accessed data

3. \*\*Scalability\*\*: Consider pagination for large datasets

4. \*\*Security\*\*: Always validate user permissions and input data

5. \*\*User Experience\*\*: Provide immediate feedback and optimistic updates

6. \*\*Monitoring\*\*: Log important events and track key metrics

This guide provides the foundation for building a robust rewards program. Adapt the schema and business logic to match your specific requirements and use cases.