Rewards SDK Test Plan

Author: Orlando Perez **Date**: 01-14-2026

Background:

Game developers want to reward users with Bitcoin for completing in-app achievements. Our Rewards SDK allows this without developers handling Lightning infrastructure.

Test Scope:

High Priority - Identify validation, Bitcoin payout speed <60 seconds, rate limiting 10 per hour per user, and anti-fraud blocking.
Deffered - Lower risk dashboard UI testing.

Test ID	Scenario	Expected Result	Priority	Risk Mitigated
01-Dev Funds Reward	Developer defines reward event payout 1-100,000 sats, pre-fund reward pool >100,000 sats; validates 2% service fee.	Valid reward events require a >= 100,000 sats pool balance. Apply a 2% service fee to the developer pool, and credit users the full payout amount for rewards 1-100,000 sats.	High	Prevents payouts without sufficient prefunding. Ensures correct fee accounting and avoild financial loss.
02-Happy Path	Valid user triggers a rewared event and paid out <60s, ex. (1,000 sat reward.)	User wallet credited 60s. API returns success; webhook sent; unique user's and developer dashboard total updates correctly.ZBD collects 2% fee, ex. (980 sats arrive in ZBD wallet).	High	Ensures core reward functionality and user trust. Enables fee revenue functions correctly.
03-Rate Limit	User attempts 11th reward within 60 minutes	Additional reward attempts are rate limited, no payouts occur and real time dashboard notifications are triggered.	High	Prevents reward spamming and uncontrolled cost.
04-Insufficient Funds	User triggers reward larger than pool minimum balance. Developer reward pool drops <100,000 sats.	Reward payout fails,user alerted "Reward Pending". System triggers "Low Balance" alert to developer, webhook sent, payout blocked until reward pool threshold maintained.	Critical	Prevents false fraud lockouts caused by developer funding issues. Maintains reputation and service agreement of developer and ZBD.
05-Multiple Account Fraud	Multiple accounts attempt reward farming via emulators/devices (including shared Wi-Fi tournament scenario)	System detects correlated suspicious behavior and blocks payouts without blocking legitimate tournament users solely due to shared IP	High	Prevents large scale reward farming and minimizing false positives.
06-Blocked Account	User triggers 3 user attributable failures within 10 minutes and attemps additional reward claim.	SDK handles retry or records payout as pending/unknown, preventing duplicate payouts and preserving reconciliation.	High	Prevents abuse and retry attacks while avoiding false user lockouts and protecting legitimate rewards.
06-Network Failure	Internet cut out during payout trigger.	SDK handles retry or logs failure and stops tracking.	Medium	Prevents lost or inconsistent payout state when network issues interrupt reward processing.
07-Testnet Mode	Developer execute payout with sandbox testnet bitcoin.	Transaction succeeds on testnet and no impact on mainnet developer balance.	Medim	Unintended real Bitcoin payouts during testing.
08-Webhook Down	Webhook endpoint is unavailable or network conditions degrade.	Webhooks are retried per policy; final payout state remains consistent; duplicate deliveries include unique event IDs.	High	Prevents loss of payont confirmation and downstream inconsistencies causing reconciliation issues for developer.

Test Environment Strategy

Testing is performed across isolated environments to validate payout behavior without risking real funds:
Local / Dev: Use Postman and API automation against testnet for rapid validation of core flows (happy path, invalid users, insufficient funds). A local tunnel is used to receive and inspect webhook callbacks during development.

Sandbox / Pre-Prod: A dedicated sandbox environment mirrors production configuration (rate limits, fraud thresholds, fees, webhook retry policies) while using testnet or fake sats. This environment is used for end-to-end testing, CI pipelines, and load validation.

Production: Limited to manual smoke tests only using very small payouts. All payout requests are safe to retry without risking duplicate payouts.Real time monitoring, no automated tests run against mainnet.

Key Risks & Mitigations

1. Risk: Double payouts due to retries or timeouts
   Mitigation: Enforce idempotency on payout requests, track transaction state, and reconcile mismatches.

2. Risk: Users blocked due to non-user faults (funding, network)
   Mitigation: Only count user-attributable failures toward fraud thresholds; classify funding/network errors separately.

3. Risk: Webhook delivery failures cause inconsistent downstream state
   Mitigation: At-least-once delivery with retries, unique event IDs, and a status lookup endpoint.

4. Risk: Reward pool depletion during traffic spikes
   Mitigation: Enforce minimum pool balance, block payouts deterministically, and surface low-balance alerts.