Hawamoni — Product Requirements Document (PRD)

One-line:

Hawamoni is a blockchain-native group treasury for student groups and small teams: deposits via Solana Pay, on-chain governed withdrawals that execute only after ≥80% owner approval, and SMS + in-app notifications for transparency and trust.

Purpose & Goals

Purpose:

Deliver a secure, auditable, easy-to-use group fund management platform for campus groups and small teams, where money is held in a program-owned treasury and withdrawals require supermajority approval.

Primary goals

- Enable fast deposits via Solana Pay (QR / link).

- Enforce withdrawal governance on-chain (≥80% approval).

- Provide real-time notifications (SMS + in-app).

- Keep UX simple for non-technical users (mobile-first).

- Produce a hackathon-ready MVP (end-to-end deposit → request → approve → execute on devnet).

Success metrics (MVP)

- Working end-to-end demo on devnet: deposit → request → reach 80% approvals → executed tx.

- Demo latency: deposit-to-credit ≤ 60 seconds (indexer/verification).

- SMS delivery success rate ≥ 90% for test numbers.

- Basic automated tests: Anchor unit tests + API integration tests pass in CI.

Target Users & Personas

- Coursemates / Student clubs: Need a transparent way to collect dues and pay vendors.

- Project teams / cofounders: Manage project funds and reimbursements.

- Campus merchants/vendors: Accept payments and receive verifiable receipts.

Persona specifics: mobile-first, limited crypto knowledge; expect wallet onboarding help.

Key Problems to Solve

- Lack of transparency and trust for group fund handling.

- Irregular contributions and delayed reconciliations.

- Manual approval processes that are fragile and opaque.

- Need for quick payments (low fees, fast confirmations).

- Need for notifications and auditable receipts.

Product Scope — MVP Features (Prioritized)

1. Wallet-based Authentication (nonce challenge)

Acceptance criteria:

- Users sign a server-issued nonce with their wallet; server verifies signature and issues a session token.

Demo checklist:

- Login flow showing signed challenge and authenticated UI.

2. Create / Join Group

Acceptance criteria:

- Creator can create a group with owners and approvals\_required computed as ceil(0.8 \* owners).

- Group is recorded on-chain (Group PDA) and in the database.

Demo checklist:

- Create group with 5 owners; UI shows 4 required approvals.

3. Treasury PDA & Deposit via Solana Pay

Acceptance criteria:

- Group treasury is a program-owned PDA.

- Backend can generate a Solana Pay URL/QR that pays to the treasury or an intermediary account and credits DB after verification.

Demo checklist:

- Generate QR, scan with Phantom, confirm transaction, DB shows credited deposit.

4. Create Withdrawal Request

Acceptance criteria:

- Requests are stored on-chain and mirrored in DB with Pending status, include amount, recipient, and reason (or off-chain reference).

Demo checklist:

- Create a request; it appears in approvals feed.

5. Approve / Reject Request

Acceptance criteria:

- Only group owners can approve or reject.

- Approvals are unique and tracked; once approvals >= approvals\_required, status becomes ReadyToExecute.

Demo checklist:

- Owners approve using wallet signatures; approval counter updates; threshold leads to executable status.

6. Execute Withdrawal

Acceptance criteria:

- Execute transfers funds from treasury PDA to recipient and stores transaction proof.

- Execution only allowed after required approvals and with sufficient treasury balance.

Demo checklist:

- Execute transaction; show tx on devnet explorer and updated balance.

7. Notifications (SMS + In-app)

Acceptance criteria:

- On request creation and execution, SMS and in-app notifications are enqueued and sent; delivery status recorded.

Demo checklist:

- Receive SMS for request creation and execution on test numbers; in-app notification appears.

8. Basic Audit Trail & History

Acceptance criteria:

- All deposits, requests, approvals, and executes are recorded in DB with on-chain transaction references; users can export CSV.

Demo checklist:

- Download CSV with tx signatures and timestamps.

Stretch / Future Features

- Tokenized shares and weighted approvals.

- Dispute resolution and arbitration UI.

- Scheduled payouts and batching.

- Gasless UX (relayer) and shielding of fees.

- Group budgets, alerts, and auto-approvals under thresholds.

- AI: fraud detection, anomalous request flagging, weekly spending insights.

- Multi-chain / SPL token support.

High-level System Architecture (text)

User devices (mobile/desktop) ↔ Next.js Frontend ↔ Java Backend (Spring Boot) ↔ Node Solana Service (@solana/pay, web3.js)

↘ Python AI (FastAPI) (async)

↘ Postgres (Supabase) + Redis queue + Vector DB

↘ SMS Provider (Africa's Talking / Twilio)

↘ Anchor Program on Solana devnet

Responsibilities

- Frontend: UI, wallet flows, QR scanning, approvals.

- Java Backend: auth, orchestration, API, DB, job enqueue, SMS job creation.

- Node Solana Service: create Solana Pay URLs, generate references, verify transactions, interact with on-chain program.

- Python AI: embeddings, fraud detection, insights (async jobs).

- Anchor (Rust): enforce group & request logic, move funds from treasury PDA.

Data Model & Database Summary

Core tables to store:

- users: wallet\_pubkey, phone, display\_name, created\_at

- groups: id, pda\_pubkey, name, description, treasury\_pubkey, approvals\_required, created\_by, created\_at

- group\_members: group\_id, user\_id, role, joined\_at

- requests: group\_id, request\_pda, requester, recipient\_pubkey, amount, token\_mint, reason or off-chain reference, status, approvals\_count, approvals\_required, executed\_tx, timestamps

- approvals: request\_id, approver, approved boolean, timestamp

- deposits: group\_id, tx\_sig, sender\_pubkey, amount, status, created\_at

- notifications: user\_id, event\_type, payload, provider\_message\_id, status, created\_at

On-chain Design (Anchor) — Overview

Accounts:

- Group PDA: owner (creator), owners list, approvals\_required, treasury\_pubkey.

- Treasury PDA: program-owned account that holds funds.

- Request PDA: group reference, requester, recipient, amount, approvals list, status, executed transaction reference.

Core instructions:

- initialize\_group: create Group PDA and set owners and approvals\_required.

- create\_request: create Request PDA with details and Pending status.

- approve\_request: record approver; if approvals meet threshold, set ReadyToExecute.

- execute\_request: transfer funds from treasury PDA to recipient and set Executed status.

- cancel\_request: allow cancellation under rules.

API Surface (Java Backend) — Key Endpoints

Auth:

- POST /auth/nonce — request a nonce to sign (wallet authentication).

- POST /auth/verify — verify signed nonce to create a session.

Groups:

- POST /groups — create a group with owners and details.

- GET /groups/{id} — retrieve group details.

Deposits:

- POST /groups/{id}/deposit — request creation of Solana Pay URL/QR and reference for deposit.

Requests:

- POST /groups/{id}/requests — create withdrawal request.

- POST /requests/{id}/approve — approve request (signed by owner).

- POST /requests/{id}/execute — execute withdrawal (after approvals met).

- GET /requests/{id} — get status and approval list.

Webhooks:

- POST /webhooks/solana — receive transaction events or confirmations.

- POST /webhooks/sms-delivery — receive SMS delivery callbacks.

Solana Microservice (Node) Responsibilities (overview)

- Generate Solana Pay URLs/QRs and reference keys.

- Watch for transactions referencing a payment reference.

- Validate transaction details (amount, recipient).

- Notify the Java backend when deposits are confirmed.

AI Microservice (Python) Responsibilities (overview)

- /summarize: generate concise spending summaries and suggestions.

- /flag: score suspicious requests using embeddings and models.

- /embed: compute embeddings for receipts and store them in a vector DB.

- Integrate with the job queue for heavy/async workloads.

Notifications & SMS System

Event triggers:

- withdraw\_request\_created: notify owners.

- approval\_received: notify requester (optional).

- withdrawal\_executed: notify group and requester.

- deposit\_confirmed: notify group or owner.

Templates (examples):

- Request created: 'Hawamoni: New withdrawal request [amount] by [requester] for [short reason]. Approve or view in app.'

- Approval confirmation: 'Hawamoni: [owner] approved your withdrawal request. Current approvals: x/y.'

- Execution notification: 'Hawamoni: Withdrawal of [amount] to [recipient] executed. Tx: [tx\_sig].'

Delivery:

- Use an abstracted notification worker that sends messages via provider API and logs delivery status.

- Preferred providers for Nigeria: Africa's Talking (local coverage) and Twilio as fallback.

Frontend — Key Pages & Flows

- Dashboard: groups, balances, pending actions.

- Group view: members, treasury balance, deposit QR, requests list.

- Create Request modal: amount, recipient, reason, attachments (off-chain).

- Approvals feed: pending requests with Approve action that triggers wallet signing.

- Deposit flow: merchant view generates QR; payer scans with Phantom.

- Settings: phone, language, notification preferences.

Team Roles & Responsibilities (3-person team)

Frontend Developer:

- Build Next.js TypeScript app, wallet integration, UI components, QR views, mobile UX.

- Example tasks: scaffold app and auth UI; implement Dashboard and Group views; build deposit QR page; implement create request modal; approvals feed; notifications UI; transaction history and exports.

Backend Developer (Java):

- Implement Spring Boot APIs, DB models, job queue, SMS enqueueing, security, and CI/CD.

- Example tasks: auth endpoints, groups CRUD, deposit endpoint, request handling, webhooks for Solana and SMS, notification job creator, CI pipeline.

AI Engineer (Python):

- Build FastAPI microservice for summarization, embedding, and fraud detection.

- Example tasks: FastAPI endpoints (/summarize, /embed, /flag), embedding storage in vector DB, prompt engineering for saving tips, async job worker for heavy processing.

72-Hour Hackathon Plan (summary)

Day 1 — Core flow:

- Frontend: scaffold app, wallet auth, basic dashboard.

- Backend: auth endpoints, group create, DB models, deposit endpoint (stub).

- Node Solana service: QR generation and devnet test.

Checkpoint: login, create group, produce Solana Pay QR.

Day 2 — Governance & verification:

- Backend: create\_request endpoint, store in DB, enqueue notification job.

- Frontend: create request UI and approvals feed.

- Node service: verification webhook to mark deposit confirmed.

Checkpoint: request created and visible; owners can approve.

Day 3 — Execute & polish:

- Anchor or simulation for approve → execute flow.

- Backend: execute endpoint wired to on-chain program or simulate transfer.

- Frontend: final polish, demo script, slides.

- AI: optional summarize endpoint for demo.

Deliverable: 3-minute demo, deployed frontend and backend, Anchor on devnet or simulated execution.

Testing Strategy (overview)

- Anchor unit tests for on-chain logic (approval thresholds, execution rules).

- Backend unit and integration tests for API and DB.

- End-to-end tests with devnet or mocks for wallet flows.

- SMS delivery test matrix across carriers.

Security & Compliance Checklist

- Use PDA for treasury; program-only fund movement.

- Store secrets in KMS; do not hardcode private keys.

- Wallet auth via signed nonce; no passwords.

- SMS opt-in and allow STOP; log consent and opt-outs.

- Avoid storing sensitive content on-chain; use IPFS or off-chain storage for attachments.

- Rate-limiting and input validation; audit before mainnet deployment.

Monitoring & Observability (key metrics)

- Transaction success/failure rate and confirmation latency.

- Approval latency (time from request to threshold).

- SMS delivery rate and failure reasons.

- Job queue length and AI job processing time.

Deployment & CI/CD (summary)

- GitHub Actions pipeline: lint, unit tests, build, Anchor tests, deploy to staging.

- Frontend: Vercel. Backend/Microservices: Render, Fly, or Cloud Run.

- Managed Postgres (Supabase/Neon), Redis for queue, Vector DB for embeddings.

- Secrets managed in environment manager and KMS.

Risks & Mitigations (top items)

- RPC/indexer rate limits: use webhooks from reliable providers.

- SMS delivery failures: use local provider with fallback.

- Small-group unanimous requirement: allow configurable threshold or minimum group size.

- Key compromise: KMS and audit trails.

- Replay/double-execute: idempotency checks and executed\_tx records.

- UX confusion with wallets: onboarding tutorial and fallback flows.

- LLM costs: async jobs and usage caps.

- On-chain bugs: audits before mainnet.

Acceptance Criteria (MVP)

- Wallet auth works with signed nonce.

- Group creation writes Group PDA and DB record.

- Solana Pay QR generation and deposit verification on devnet.

- Request creation, owner approvals, approvals threshold enforcement, and execution of withdrawal (on devnet or simulated) function end-to-end.

- SMS is sent for request creation and execution with delivery status logged.

- README includes run instructions and demo guide.

Deliverables & README Outline

- Project overview, architecture diagram, quickstart (env, devnet setup), run commands for frontend/backend/solana service/anchor, demo script and sample accounts, API docs, contract addresses, contributors list.

Next Actions (first 7 tasks)

1. Create repository and GitHub Actions skeleton.

2. Provision dev Postgres and Redis and store credentials securely.

3. Scaffold Next.js frontend with wallet-adapter.

4. Implement backend auth nonces and verification endpoint.

5. Create Node Solana service route for Solana Pay QR generation and test on devnet.

6. Scaffold Anchor program with Group and Request structures.

7. Configure SMS provider sandbox and test sending messages.

Document version: Generated 2025-09-20 12:09:42 UTC