



SDG BLOCKCHAIN ACCELERATOR

Technical Architecture Document – Template

1. Project Information

- **Project Name:** Unicorn
- **Challenge & UNDP Office:** UNDP OP CCIT
- **Document Version:** 1

2. Overview

(Provide a high-level description of the project, its purpose, and the problem it solves.)

This project allows users to make donations to projects on Giveth using the Cardano network.

3. System Architecture Diagram

Mermaid sequence diagram

It covers wallet connect → balances/prices → build/sign/submit → record donation → explorer link, with ADA vs token branches and common error paths.

```
sequenceDiagram
    autonumber
    actor U as User
    participant UI as Next.js UI
    participant MR as @meshsdk/react
    participant MC as @meshsdk/core
    participant W as Wallet Extension<br/>(Yoroi/Nami/Eternl)
    participant BF as Blockfrost API
    participant PX as Prices (MuesliSwap/CoinGecko)
    participant IG as Impact Graph (GraphQL)
    participant CN as Cardano Node
    participant CS as Cardanoscan (Explorer)

    %% Connect
    U->>UI: Click "Connect Wallet"
    UI->>MR: useWallet().connect()
    MR-->>W: request connect
```

```

W-->>MR: session + networkId (Mainnet/Preprod)
MR-->>UI: connected(wallet, networkId)

%% Load balances & token meta
UI->>MR: wallet.getBalance()
MR-->>UI: [{unit:"lovelace", qty}, {unit:policy+assetNameHex, qty}, ...]
UI->>BF: GET /assets/{unit} (metadata, decimals)
BF-->>UI: token metadata (name, decimals, policyId)
UI->>PX: price(token→ADA), price(ADA→USD)
PX-->>UI: price objects

%% Build donation
U->>UI: Select token + enter amount
UI->>UI: normalizeAmount + toUnits(amount, decimals)
UI->>MR: wallet.getNetworkId() (guard recipient addr)
alt ADA (unit == "lovelace")
    UI->>MC: new Transaction().sendLovelace(to, amountLovelace)
else Token (native asset)
    UI->>MC: new Transaction().sendAssets(to, [{unit, quantity}])
    note right of MC: Optionally add ~1-1.5 ADA to same output<br/>to
satisfy min-UTxO
end

%% Sign & submit
UI->>MC: tx.build()
MC-->>UI: unsignedTx (CBOR)
UI->>W: wallet.signTx(unsignedTx)
W-->>UI: signedTx
UI->>W: wallet.submitTx(signedTx)
W->>CN: submitTx
CN-->>UI: txHash

%% Persist donation off-chain
UI->>IG: mutation createCardanoDonation(txHash, amount, token, addresses,
usdValue)
IG-->>UI: donationId

%% Explorer link
UI->>CS: Open link (mainnet/preprod) with txHash
CS-->>U: Transaction details

%% Error paths
opt Error: Wrong Network

```

```
CN-->>UI: TxSubmitFail: WrongNetwork (Testnet/Mainnet)
UI-->>U: Show network mismatch message
end

opt Error: OutputTooSmallUTxO
  CN-->>UI: TxSubmitFail: BabbageOutputTooSmallUTxO
  UI-->>U: Suggest  $\geq 1$  ADA (token output needs min-ADA)
end

opt Error: BigInt conversion
  UI-->>U: "Invalid amount" (normalize + toUnits required)
end
```

4. Blockchain Design

4.1 Networks & Addressing

- Networks: Mainnet and Preprod Testnet.
- Detection: `wallet.getNetworkId()` \rightarrow 1 (Mainnet) or 0 (testnets).
- Recipient routing: Pick recipient address per network (mainnet `addr1...`, preprod `addr_test1...`).
- Wrong-network guard: Block submits if wallet network and recipient network differ (prevents `WrongNetwork` errors).

4.2 Asset Model & Units

- Accounting model: UTxO (unspent transaction outputs), multi-asset native support.
- Base currency: ADA; smallest unit lovelace (1 ADA = 1,000,000 lovelace).
- Native tokens: Identified on-chain by `policyId + assetName (hex)` \rightarrow concatenated unit.
- Quantities are integers (smallest units).
- Decimals are off-chain metadata and enforced in the UI only.
- Min-ADA requirement: Any output that carries tokens must include a minimum ADA “top-up” (protocol-dependent; we budget $\sim 1\text{--}1.5$ ADA).

4.3 Transaction Construction (no smart contracts)

- Wallet API: Browser wallet via CIP-30 (connect, sign, submit).
- Builder: Mesh SDK Transaction with the connected wallet as initiator.
- ADA donation: **`tx.sendLovelace(recipient, amountInLovelace.toString());`**
- Token donation:

`tx.sendAssets(recipient, [{ unit, quantity: tokenUnits.toString() }]);`

// include ADA top-up on same output when needed

`tx.sendLovelace(recipient, MIN_UTXO_ADA.toString());`

- Change & balancing: Wallet/provider selects UTxOs, computes fees, creates change outputs automatically when building.

4.4 Validation & Fees

- Protocol checks:
- Network discriminant (prevents mainnet↔testnet mismatch).
- Min-UTxO value on token outputs (avoids OutputTooSmallUTxO).
- Integer quantities only (avoid BigInt/decimal errors by converting with toUnits()).
- Fees: Automatically computed at build time; UI shows fee from wallet prompt.
- Pre-submit checks in UI:
- Amount normalization (0,25 → 0.25),
- toUnits(amount, decimals) → BigInt,
- Balance guard (donation + buffer ≤ balance).

4.5 Wallet Integration (CIP-30)

- Discovery & connect: @meshsdk/react (useWallet, useWalletList).
- Core calls:

- `getNetworkId()`, `getBalance()`, `getChangeAddress()`,
- `signTx(unsignedCbor)`, `submitTx(signedCbor)`.
- Security: Private keys never leave the wallet; dApp only receives signatures and a txHash after submit.

4.6 On-Chain vs Off-Chain Data

- On-chain: Transfer outputs (ADA and/or tokens), fee, change, tx hash.
- Optional: attach tx metadata (e.g., donation reference) if needed in later phases.
- Off-chain: Donation record (projectId, token symbol & decimals, USD value, anonymous flag) stored via GraphQL.
- Note: “Anonymous” is an off-chain presentation rule; on-chain transfers are public.

4.7 Error Handling (design-time assumptions)

- Wrong network: guard by checking `getNetworkId()` before building the tx.
- Min-UTxO / small output: add ADA top-up for token outputs; block ADA sends < 1 ADA where appropriate.
- Decimal \rightarrow integer: all user inputs normalized and converted to smallest units; never pass floats to `BigInt`.
- Provider/rate limits: cache token metadata; throttle price calls.

4.8 Security Considerations

- Key custody: Client-side only (wallet extension); no server signing.
- API auth: GraphQL calls carry a short-lived JWT; backend validates before persisting donation.
- Replay / integrity: Unique txHash stored; network ID tracked to avoid cross-network confusion.
- PII: Store only what's needed for donation analytics; honor “anonymous” in UI and API responses.

4.9 Extensibility

- Smart contracts (future): Can evolve from direct transfers to script-based donations (escrow, matching pools, recurring).
- Dynamic fees/min-UTxO: Swap fixed buffer for provider-driven estimation.
- Multi-wallet support: The same flow works with Yoroi, Nami, Eternl (CIP-30 compatible).

5. Data Flow & Transaction Lifecycle

5.1 User Input & Validation

1. User selects token & amount in the donation modal.
 - Input normalized (commas → dots).
 - `normalizeAmount()` ensures >0 .
 - Amount converted to smallest units with `toUnits(value, decimals) → BigInt`.
 2. Pre-flight checks:
 - `wallet.getNetworkId()` → must match project network (Mainnet or Preprod).
 - For ADA: must be ≥ 1 ADA (min-ADA requirement).
 - For tokens: add ADA top-up to prevent `OutputTooSmallUTxO`.
 - `hasSufficientBalance(amount, balance, buffer)` ensures balance covers amount + fee buffer.
-

5.2 Transaction Construction

1. Transaction builder (Mesh SDK):

- tx = new Transaction({ initiator: wallet })
- If ADA:

```
tx.sendLovelace(toAddr, donationLovelace.toString());
```

- If token:

```
tx.sendAssets(toAddr, [{ unit, quantity: tokenUnits.toString() }]);  
tx.sendLovelace(toAddr, MIN_UTXO_ADA.toString()); // ADA top-up
```

2. Change outputs: Wallet auto-balances inputs, computes fees, and adds change outputs.

5.3 Wallet Signing & Submission

1. Build unsigned CBOR tx: `const unsignedTx = await tx.build();`
 2. User signs via wallet (CIP-30):
 - Wallet prompts → user confirms donation.
 - `signedTx = await wallet.signTx(unsignedTx);`
 3. Submit to network:
 - `txHash = await wallet.submitTx(signedTx);`
 - Returns a unique transaction hash.
-

5.4 On-Chain Processing

- Node validates:
 - Correct network ID (prevents mainnet/preprod mismatch).
 - Min-UTxO satisfied.
 - Inputs unspent, fees covered.
 - Transaction included in block → permanently recorded.
-

5.5 Backend Persistence

1. GraphQL mutation:

After tx is submitted, frontend calls:

```

mutation CreateCardanoDonation(...) {
  createCardanoDonation(
    amount: $amount
    transactionId: $txHash
    projectId: $projectId
    fromWalletAddress: $fromAddr
    toWalletAddress: $toAddr
    token: $token
    tokenAddress: $tokenAddress
    anonymous: $anonymous
    valueUsd: $valueUsd
    priceUsd: $priceUsd
    userId: $userId
  )
}

```

txHash links the off-chain donation record to on-chain activity.

- anonymous hides donor identity in the UI, though tx remains public on-chain.
2. Backend stores donation record:
 - Tx hash
 - Token info (symbol, decimals, address)
 - USD value at donation time
 - Project ID, user ID (if not anonymous)

5.6 Lifecycle Summary

1. Input validation (amount, balance, network).
2. Build tx (ADA or token + min ADA).
3. Sign tx (wallet → user approves).
4. Submit tx → blockchain.
5. Record donation (GraphQL mutation).
6. UI feedback:
 - Pending → Success (tx hash link) → Confirmation.
 - Failures → error modals (Rejected, Failed, WrongNetwork).

5.7 Sequence Diagram (simplified)

```
sequenceDiagram
    participant User
    participant dApp
    participant Wallet
    participant Cardano
    participant Backend

    User->>dApp: Select token & amount
    dApp->>dApp: Normalize & validate
    dApp->>Wallet: Build + request signature
    Wallet->>User: Prompt confirm
    User->>Wallet: Approve
    Wallet->>Cardano: Submit signed tx
    Cardano->>Wallet: Return txHash
    Wallet->>dApp: txHash
    dApp->>Backend: createCardanoDonation(txHash, details)
    Backend->>dApp: Donation saved
    dApp->>User: Show success + explorer link
```

This ties directly to the errors you saw earlier (WrongNetwork, OutputTooSmallUTxO, decimal → BigInt), all are accounted for in this lifecycle.

6. Off-chain Components

While Cardano manages the immutable ledger of donations, several off-chain components complement the system to ensure usability, price tracking, and integration with the Giveth ecosystem. These off-chain modules are essential for bridging on-chain activity with user-facing features such as dashboards, donation histories, and reporting.

6.1 Frontend (dApp Interface)

- Built with Next.js + React.
- Integrates with MeshJS SDK for wallet connections (CIP-30 wallets such as Yoroi, Eternl, Nami).
- Handles:
- Token list display and user balances (wallet.getBalance()).

- Donation form logic (amount normalization, min-ADA check, fee buffer check).
 - Transaction construction via Transaction object (ADA or native tokens).
 - Submission of GraphQL mutations (createCardanoDonation) after successful on-chain confirmation.
 - Provides user feedback: pending state, success with block explorer link, or error modals (Rejected, Failed, Wrong Network, Insufficient Balance).
-

6.2 Backend (Giveth API / Impact Graph)

- GraphQL API (Node.js, Apollo) receives donation metadata after tx submission.
 - Responsibilities:
 - Persisting donations with associated txHash, projectId, userId (if not anonymous).
 - Storing price data (priceUsd, valueUsd) for reporting and eligibility checks (e.g., GIVbacks).
 - Linking on-chain tx data with off-chain project records.
 - Supporting queries for project dashboards, user donation history, and QF matching calculations.
-

6.3 Price & Token Data Services

- Blockfrost API:
 - Fetches token metadata (policyId, name, decimals).
 - Tracks balances and asset information.
- MuesliSwap API:

- Retrieves real-time token ↔ ADA conversion rates.
 - Coingecko API:
 - Fetches ADA ↔ USD exchange rate.
 - Combined, these services allow displaying token value in ADA and USD for user transparency.
-

6.4 Notifications & UX Enhancements

- Inline toasts and modals display donation status.
 - Planned integrations (future scope):
 - Email or Telegram notifications for donation confirmations.
 - Real-time updates via websockets or server-sent events.
-

6.5 Monitoring & Error Handling

- Frontend logs wallet errors (e.g., WrongNetwork, OutputTooSmallUTxO) for debugging.
 - Backend validates inputs to prevent injection or replay attacks.
 - Transaction failures are surfaced to users with contextual hints (e.g., “Switch to Preprod network” or “Increase amount to ≥ 1 ADA”).
-

In summary: Off-chain components act as the glue between Cardano on-chain logic and the Giveth donation ecosystem, ensuring usability, price accuracy, persistence, and donor feedback.

7. Sandbox/Testnet Results

To validate the donation flow, the system was tested extensively on Cardano Preprod Testnet using wallets such as Yoroi and Eternl. Below are the observed results:

7.1 Wallet Integration

- Wallets (Yoroi, Nami, Eternl) successfully connected via MeshJS (CIP-30).
- Users could view their test ADA & token balances.
- Issue noted: Yoroi UI does not always display test tokens even though they exist on-chain (balances confirmed via Blockfrost).

7.2 Transaction Lifecycle

- ADA Transfers: Transactions of ≥ 1 ADA were successfully built, signed, and submitted.
- Native Token Transfers (e.g., SHEN test token) executed correctly when proper policyId + assetName were supplied.
- Transaction hashes were returned and verifiable on Preprod Cardanoscan.
- Error encountered when sending amounts below 1 ADA → BabbageOutputTooSmallUTxO (expected behavior due to min-ADA requirement).
- Wrong Network Error → When wallet set to Mainnet but API/project setup expected Preprod (WrongNetwork Testnet error).

7.3 Off-Chain API (Impact Graph)

- GraphQL mutation createCardanoDonation successfully stored donation metadata (amount, txHash, projectId, userId).
- Donations appeared correctly in project dashboards with USD equivalents using Coingecko price feed.
- Anonymous donations required additional conditional handling (no sender wallet stored).

7.4 Edge Cases Tested

- Insufficient balance → properly blocked by helper (hasSufficientBalance).
- Decimal amounts entered as 0,05 or 0.05 → normalized before conversion to lovelace.
- Network switching → when user on Mainnet, dApp prompted to switch back to Preprod.

7.5 Overall Result

The sandbox implementation confirmed that:

- The core donation flow works end-to-end on testnet.
- All key errors are caught and surfaced to users.
- Off-chain persistence & price conversion logic function as expected.
- The system is ready for Mainnet rollout after further UI polishing and minor edge-case handling.

8. Tools and Environments Used

- Development Environment
- Next.js (v14+) for frontend dApp development.
- TypeScript & styled-components for strongly typed UI logic and styling.
- Node.js (v18+) for local development.
- Cardano SDKs & Libraries
- MeshJS (@meshsdk/core, @meshsdk/react) for wallet connection, transaction building, and signing.
- Blockfrost API for fetching token metadata, balances, and network info.
- Testing & Debugging Tools
- Postman for GraphQL API testing (e.g., createDonation mutations).
- Console logging in browser DevTools for tracing wallet balances, transaction hashes, and API responses.
- External APIs
- Coingecko API for fetching ADA → USD conversion rates.
- MuesliSwap API for Cardano token price discovery in ADA.
- Wallets
- Yoroi Wallet (extension) for transaction signing and testing ADA/native token transfers.
- Also tested with Nami/Eternl for broader compatibility.
- Networks
- Cardano Mainnet (for real token metadata and production-like environment).
- Cardano Preprod Testnet (for safe testing with test tokens).

9. Remaining Considerations / Next Steps

- *Transaction Fee Estimation*
- *Need a more reliable way to estimate Cardano network fees before submission.*
- *Current workaround uses buffer checks (hasSufficientBalance), but lacks precision.*
- *Anonymous Donations Handling*
- *UI condition for anonymous Cardano/EVM donations requires refinement (preventing transaction link exposure).*
- *Token Handling Improvements*
- *Standardize conversion functions for ADA (toUnits, normalizeAmount) and ensure consistent BigInt usage across ADA and native tokens.*
- *Verify decimals handling for all supported Cardano tokens.*
- *Wallet Detection & UX*
- *Improve error handling when no wallet is connected or when the user is on the wrong network (Mainnet vs Preprod).*
- *Add responsive UI feedback if wallet balance or selected token info fails to load.*
- *Testing Coverage*
- *Extend test cases for token transfers beyond ADA (e.g., SHEN, test tokens).*

- • *Validate donations with multiple Cardano wallets (Yoroi, Nami, Eternl).*
- • *API Integration*
- • *Ensure GraphQL mutation (createCardanoDonation) aligns with backend schema (types and variable mapping).*
- • *Implement proper error messaging for unauthorized or invalid transaction submissions.*
- • *Next Steps*
- • *Deploy fixes to staging for end-to-end donation flow testing.*
- • *Gather feedback from test users (using Yoroi/Nami wallets) on transaction UX.*
- • *Finalize minimum-ADA validation rules and sync with backend enforcement*
-