📘 Implementation Guide for SDG-Focused Blockchain Projects

EMURGO x UNDP Blockchain Accelerator
Version: April 2025

Content

🎯 Purpose

This guide provides SDG teams with a comprehensive, step-by-step process to design and deploy blockchain-powered Proofs-of-Concept (PoCs) using the Cardano ecosystem. It draws from real challenges submitted by UNDP country offices and maps each challenge to sample verticals: **Climate Action**, **Digital Identity**, **Financial Inclusion** and **Supply Chain & Product Traceability.**

🔧 Core Design Considerations

Layer	Component	Purpose	Tools on Cardano
Identity	DID / VC Issuance	Attest roles, eligibility, impact	**anoncreds-cardano**, **Veridian SDKs**
Wallet	Custody & Signing	Secure access to funds & credentials	**Lace**, **Nami**, **Eternl**, **Cardano Identity wallet**
Smart Contracts	Rules and automation	Loan logic, payout conditions, traceability	**Plutus**, **Marlowe**, **Aiken**
Data & Metadata	Track assets and flows	Represent impact, VCs, token properties	**CIP-68**, **IPFS**, **Koios**, **Blockfrost**, **GraphQL**
UI & APIs	Human interaction layer	Access, dashboards, verification	**Lucid SDK**, **Mesh**, **React**, **GraphQL APIs**

🌱 Vertical 1: Climate Action

🧠 Use Case

Design a transparent, blockchain-based Payment for Ecosystem Services (PES) program.

The goal is to reward verified ecological actions (like tree planting or soil restoration) with programmable payments in ADA or stablecoins using Verifiable Credentials (VCs), smart contracts, and metadata NFTs.

1. Functional Flow

1. A farmer plants trees or performs conservation activity.
2. A verifier (e.g., NGO officer or local cooperative) inspects or validates the activity.

3. A Verifiable Credential (VC) is issued, confirming the ecosystem action.

4. A CIP-68 NFT is minted, embedding metadata like tree species, timestamp, GPS location, and VC hash.

5. A smart contract verifies the NFT/VC and disburses funds (ADA or USDA) to the farmer's wallet.

2. Architecture (Simplified View)

```
[Farmer] → [Verifier Mobile App] → [VC Issued]
           ↓
      [NFT Minted w/ Metadata (CIP-68)]
           ↓
 [Smart Contract Validates → Disburses Funds]
           ↓
    [Farmer Wallet (Lace, Eternl, Custodial)]
```

3. Tools & Libraries

Layer	Tool	Description
Identity	Veridian / anoncreds	VC issuance for impact verification
Metadata	CIP-68 + IPFS	Mint on-chain NFT referencing off-chain impact proof
Contracts	Aiken / Plutus	Verify VC/NFT and release programmable payouts

UI	Mesh SDK / Lucid	Wallet connect, mint, and verifier dashboard
Storage	IPFS or Arweave	Store images, documents, or GPS logs from fieldwork

4. Credential Format (VC Example)

File: `vc-schema/climate-activity.json`

```
json
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "id": "vc:climate:12345",
  "type": ["VerifiableCredential", "ClimateActivity"],
  "issuer": "did:cardano:verifier123",
  "issuanceDate": "2025-07-03T00:00:00Z",
  "credentialSubject": {
    "id": "did:cardano:farmer987",
    "activityType": "TreePlanting",
    "treeSpecies": "Acacia Senegal",
    "location": {
      "lat": "15.8921",
      "lng": "35.1831"
    },
    "timestamp": "2025-07-01T13:00:00Z"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2025-07-03T01:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:cardano:verifier123#keys-1",
    "jws": "eyJhbGciOiJFZERTQSJ9.."
  }
}
```

5. CIP-68 NFT Metadata Example (Off-chain Pointer)

```
json
{
  "name": "TreePlanting #12345",
  "description": "Verified climate action - Acacia planting",
  "image": "ipfs://QmExampleHash/photo.jpg",
  "vc_hash": "0x9f2a38...d21b7a",
  "lat": "15.8921",
  "lng": "35.1831",
  "tree_species": "Acacia Senegal",
  "verified_by": "did:cardano:verifier123",
  "timestamp": "2025-07-01T13:00:00Z"
}
```

6. Smart Contract Logic (Aiken Sample)

`contracts/payout.aiken`

```
fn payout(vc_hash: ByteArray, submitted_hash: ByteArray) -> Bool {
  assert(vc_hash == submitted_hash, "Credential mismatch");
  true
}
```

Smart contract logic may use a mapping of verified hashes, require an authorized issuer signature, or even fetch from an oracle if ZK or external proofs are integrated later.

7. Frontend Features (Verifier & Farmer UI)

- Verifier Login
- VC Issuance Form (inputs: lat/lng, tree species, photos)
- NFT Mint Flow (attached metadata + VC hash)

- Farmer Wallet View (shows NFT + received funds)
- Admin Dashboard (shows total impact, disbursed funds, pending reviews)

8. Deployment Simulation

- Use testnet wallets (Lace or Eternl)
- Mint tokens using CIP-68 + Mesh UI components
- Trigger the Aiken payout contract manually via test dApp
- Record all steps in GitHub with signed TX hashes and metadata

9. Output Expectations

Deliverable	Details
`contracts/payout.aiken`	VC-linked smart contract logic
`vc-schema/climate-activity.json`	Sample schema + JSON credential
`mint-nft.tsx`	NFT mint UI using Lucid or Mesh
`demo/funding-flow.mp4`	Demo of full flow from VC → NFT → fund disbursement
`docs/dashboard-mockup.png` *(optional)*	UI concept for verifier dashboard

10. Best Practices for Climate Action PoCs

✅ Start Small, Then Expand

- Begin with one verifier (NGO) and a handful of farmers.
- Use manual VC issuance first, then consider sensors or satellite proof.

✅ Design Realistic, Flexible Schemas

- Use field names like `activityType`, `treeSpecies`, `timestamp`.
- Include a proper `@context` and `issuer` DID in the VC.

✅ Separate On-Chain vs. Off-Chain Data

- Use IPFS for large files (photos, logs).
- Only store hashes or summarized impact on-chain.

✅ Fund Governance

- Use a multisig wallet or DAO for funding the payout contract.
- Consider who has authority to revoke a VC or pause disbursements.

✅ Traceable Reporting

- Log VC issuance → NFT mint → fund release in one explorer-visible chain.
- Optionally publish these logs via Koios/GraphQL dashboard.

✅ Multilingual & Offline Capable

- Prepare the UI with translation tags.
- Cache VC verification logic locally if field agents operate without internet.

✅ Document Everything

- Save signed TXs, screenshots, credential metadata, and test results in your GitHub repo.
- Include an audit log of who issued which VC, and when funds were released.

🧾 Vertical 2: Digital Identity

🧠 Use Case

Develop a decentralized ID system to support equitable access to services.

Many people in developing regions lack formal identity documentation. This vertical focuses on creating decentralized identity systems that can issue and verify credentials for employment, education, healthcare, or social programs — using open standards and self-sovereign identity principles.

1. Functional Flow

1. **Citizen or beneficiary** accesses an onboarding interface (mobile, kiosk, or agent-assisted).
2. A **trusted verifier** (NGO, government office, school) issues a **Verifiable Credential** (VC) with the citizen's identity or eligibility info.
3. VC is stored in the user's digital wallet (e.g., Lace, Eternl, or a local mobile app).
4. When needed, the **user presents the VC** to an employer or service provider.
5. The verifier confirms VC integrity (signature, issuer DID, expiration).
6. Based on the VC, access is granted to a job, subsidy, health service, or school enrollment.

2. Architecture (Functional Diagram)

```
[Citizen Onboards] → [Credential Issuer (e.g., NGO)]
     ↓                             ↓
[DID + VC Created]          [Veridian / anoncreds]
     ↓
[Stored in Wallet] ──┐
                    ↓
[Verifier (Employer or School) Requests Proof]
```

```
                     ↓
[User Presents VC → Verifier Validates → Access Granted]
```

Optional Enhancements:

- VC registry or revocation list
- Zero-knowledge proof (ZK-PoP) for privacy-preserving verification
- GLEIF-backed issuer IDs for institutional credibility

3. VC Schema Example

File: `vc-schema/id-basic.json`

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": ["VerifiableCredential", "IdentityCredential"],
  "issuer": "did:cardano:ngoid123",
  "issuanceDate": "2025-06-12T00:00:00Z",
  "expirationDate": "2026-06-12T00:00:00Z",
  "credentialSubject": {
    "id": "did:cardano:user456",
    "firstName": "Awa",
    "lastName": "Diop",
    "birthYear": "1996",
    "nationality": "Senegalese",
    "verifiedBy": "Red Cross Dakar Chapter"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2025-06-12T01:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:cardano:ngoid123#keys-1",
    "jws": "eyJhbGciOiJFZERTQSJ9.."
  }
}
```

Alternate schemas can support education, disability status, refugee ID, etc.

4. Tools & Libraries

Layer	Tool	Description
DID & VC	Veridian / anoncreds	Open source SDKs for DID generation and credential issuance
Wallet	Lace / Eternl / WatrID	Custodial or self-custodied credential storage
Credential UI	Mesh SDK / React	Mobile or browser UI for issuance, storage, and proof presentation
Verifier	Veridian Verifier Tool	Validates signature, schema, issuer, and status
Registry	JSON-LD Schema + CIP-68	For standardizing and referencing credential formats

5. Optional Smart Contract (Access Control)

Some services may require smart contract logic to enforce access or payments based on VC possession.

Example (Aiken pseudocode):

```
fn is_verified(vc_hash: ByteArray, required: ByteArray) -> Bool {
  vc_hash == required
}
```

This validator can be used in dApps offering access to discounts, services, or contracts gated by proof of identity or eligibility.

6. Frontend Features (Onboarding & Verification App)

- **Issuer Admin Panel:** Issue, view, revoke credentials
- **User Wallet Screen:** View VCs, export/share proofs
- **Verifier App:** QR scanner → verify → see result
- **Access Control Demo:** Try entering a “service” with a valid VC

UI should also include:

- Language toggle
- Offline mode for VC display
- Visual trust cues (verified badge, expiration status)

7. Deployment Simulation

- Set up DID method (`did:cardano:user123`)
- Issue credential using Veridian backend
- Store it in Lace / Eternl testnet wallet or mock vault
- Build simple dApp to simulate an employer verifying VC
- Record testnet TXs and store public schema

8. Output Expectations

Deliverable	Details
`vc-schema/id-basic.json`	VC schema with identity fields
`vc-issuer/index.ts`	VC issuance script via Veridian or anoncreds SDK
`demo/presentation.tsx`	React screen to present and verify credentials
`contracts/gate_access.aiken`	Optional VC-gated smart contract
`demo/id-flow.mp4`	Full flow: issue → store → present → verify

9. Best Practices for Digital Identity PoCs

✅ Start with Low-Stakes Credentials

- Begin with employment, training, or age credentials

- Avoid storing sensitive health or biometric data at PoC stage

✅ Align with Open Standards

- Use W3C Verifiable Credential Data Model and JSON-LD
- Follow schema.org or DIF vocabularies for fields (e.g., birthdate, nationality)

✅ Offer Both Custodial and Self-Custodial Options

- Allow NGO to hold VC on behalf of user (custodial mode)
- Also support wallets like Lace for user-owned identities

✅ Plan for Revocation and Reissuance

- Use a revocation registry or timestamp-based expiry
- Allow the issuer to revoke lost or misused credentials

✅ Design for Low Connectivity

- Verifier UI should cache credential schemas and verification logic offline
- Offer QR download / printed credential with QR link to on-chain hash

✅ Multi-Language and Inclusive Design

- Use simple language, icons, and offline walkthroughs
- Translate UI and onboarding to regional dialects where needed

✅ Governance and Trust

- Define who can issue what type of VC
- Document their DID, public key, and signature policy
- If government-backed, register GLEIF identifier or other legal proof

💸 Vertical 3: Financial Inclusion & Payments

🧠 Use Case::

“Build a blockchain-based microloan platform for underserved small businesses.”

Access to credit remains a critical barrier for smallholder farmers, women-led cooperatives, and micro-entrepreneurs across Africa, Latin America, and Asia. This vertical provides a full blueprint for a decentralized loan disbursement and repayment system, based on Verifiable Credentials and smart contracts on Cardano.

1. Functional Flow

1. MSME (e.g., smallholder farmer or vendor) signs up through a mobile app or with agent support.
2. A **credential** is issued based on verified data: ID, income estimate, business license.
3. A **credit score** is assigned via rules or off-chain oracle.
4. The user requests a loan in the app; a **smart contract** is deployed for loan logic.
5. Funds are disbursed in **ADA** or **stablecoin** (e.g., USDA).
6. **Repayments** are tracked via on-chain interactions (voluntary or auto-scheduled).
7. NGOs or impact funders monitor **repayment events**, default rates, and impact data.

2. Architecture Flow (Functional View)

```
[Borrower Onboards]
     ↓
[VC Issued: KYC + Income]
     ↓
```

```
[Credit Risk Score Calculated]
     ↓
[Loan Request Submitted]
     ↓
[Smart Contract Deployed for Loan]
     ↓
[Funds Released to Borrower Wallet]
     ↓
[Repayment Events Logged On-Chain]
```

Advanced additions:

- Time-locked repayments (e.g., every 30 days)
- Collateral via NFT or VC pledge
- Default notification and alerting system
- Group lending logic (shared responsibility VC)

3. Credential Schema Example

`vc-schema/msme-loan-eligibility.json`

```
json
{
  "@context": ["https://www.w3.org/2018/credentials/v1"],
  "type": ["VerifiableCredential", "MSMEEligibilityCredential"],
  "issuer": "did:cardano:ngo123",
  "issuanceDate": "2025-07-15T00:00:00Z",
  "credentialSubject": {
    "id": "did:cardano:user789",
    "businessName": "Ama's Farm Produce",
    "sector": "Agriculture",
    "incomeRange": "USD 200-400/month",
    "location": "Bauchi, Nigeria",
    "verifiedBy": "Farmers' Union Local Chapter"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2025-07-15T01:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:cardano:ngo123#key-1",
    "jws": "eyJhbGciOiJFZERTQSJ9.."
  }
}
```

Additional attributes (optional): gender, years in business, GPS zone, group affiliation.

4. Smart Contract Logic

You can build the loan contract using Aiken for readability and lifecycle control. Here's a basic lifecycle pattern:

States:

- `LoanRequested`
- `Funded`
- `InRepayment`
- `Repaid`
- `Defaulted`

Triggers:

- On-chain funding received
- Due date passed
- Scheduled repayment made
- Admin override or NGO callback

Aiken Snippet (Repayment Check)

```
rust
```

```
fn validate_payment(due: Int, paid: Int, today: Int) -> Bool {
  today <= due && paid >= min_payment
}
```

5. Tools & Libraries

Layer	Tool/Library	Description
Identity	Veridian / anoncreds	Issue borrower eligibility VCs
Contracts	Aiken or Plutus	Build time-locked loan logic
Wallet	Lace / Eternl	Receive and repay from browser or mobile
UI	Lucid SDK + React	Borrower interface to view terms, repay
Dashboard	GraphQL, Koios	NGO interface to monitor repayments
Off-chain	Oracle Hook / API	Calculate credit score / income bracket

6. Frontend Components

- **Borrower Dashboard:**
  View VC status, request loan, track repayments, submit support request.
- **Lender/NGO Dashboard:**
  Track repayment history, segment borrowers, send incentives or reminders.
- **Repayment Flow:**
  UI that lets user confirm amount, sign wallet transaction, receive receipt.

7. Deployment & Demo Suggestions

- Issue real VCs using anoncreds or Veridian CLI
- Deploy Aiken smart contract to preprod testnet
- Simulate 3 borrowers: one repays, one defaults, one in progress
- Record full transaction hashes, wallet screenshots, and dashboard view

8. Output Expectations

Deliverable	Description
`vc-schema/msme-loan-eligibility.json`	Verifiable Credential JSON
`contracts/msme-loan.aiken`	Smart contract for disbursement + tracking
`frontend/borrower.tsx`	Form + repayment screen
`dashboard/ngoview.tsx`	Chart and status table

`demo/repayment-flow.mp4`	Demo video of full process

9. Best Practices

✅ **Use Tiered Credit Limits**
Start with small amounts tied to VC level (e.g., $10 → $50), increasing as repayments are made or more VCs are added.

✅ **Prioritize Low-Risk Credentials First**
Begin with identity + income level VCs, then expand to behavioral VCs (e.g., on-time repayments, co-signer logic).

✅ **Handle Defaults Gracefully**
Don’t penalize wallets permanently. Add a "reapply after cooldown" policy, with reason codes for refusal.

✅ **Enable NGO Overrides**
Allow NGOs to act as admins with override rights (e.g., cancel debt, extend deadline).

✅ **Track Impact, Not Just Payment**
Log secondary benefits (e.g., VC showing "business expanded" or "hired 1 employee") as new credentials.

✅ **Simulate Real Constraints**
Deploy dApp on poor internet and mobile devices. Test SMS-based access to wallet notifications or deadlines.

✅ **Reward Success with New VC**
After repayment, issue "Loan Completed Successfully" VC for better creditworthiness in future PoCs.

📦 Vertical 4: Supply Chain & Product Traceability

🧠 Use Case:

“Create a blockchain-based traceability system for agricultural exports (e.g., coffee, cocoa), ensuring product origin, sustainability claims, and fair trade certifications are verifiable across the supply chain.”

Traceability is essential for combating fraud, proving sustainability, and ensuring fair trade in commodity markets. This vertical shows how to tokenize batch-level product data, anchor credentials, and verify integrity at each supply chain step.

1. Functional Flow

1. **Farmer or Cooperative Onboards:** Receives a Verifiable Credential (VC) certifying organic/fair trade status.

2. **Batch is Minted:** Product batch is represented as a native NFT with embedded metadata (origin, date, cert link).

3. **Supply Chain Transfers:** As the batch moves (Processor → Exporter → Importer → Retailer), each actor signs a transfer.

4. **QR Verification:** Retailer attaches QR code to product; buyer scans and views traceability data from chain.

2. Architecture Flow

```
[Farmer Registration]
     ↓
[VC Issued: Organic / Fair Trade]
     ↓
[Product Batch NFT Minted]
     ↓
[Signed Transfer Events (On-Chain)]
     ↓
[Retail Display → QR Scan → Verification Page]
```

Actors:

- Farmer
- Verifier / Certifier
- Supply Chain Actors (Processor, Exporter, Retailer)
- Consumer / Buyer

3. Credential Schema Example

`vc-schema/fair-trade-certification.json`

```
json
{
  "@context": ["https://www.w3.org/2018/credentials/v1"],
  "type": ["VerifiableCredential", "FairTradeCertification"],
  "issuer": "did:cardano:certifierXYZ",
  "issuanceDate": "2025-06-12T00:00:00Z",
  "credentialSubject": {
    "id": "did:cardano:farmer123",
    "farmName": "Alto Sierra Coop",
    "certification": "Fair Trade Certified",
    "validUntil": "2026-06-12",
    "region": "Antioquia, Colombia",
    "product": "Coffee Arabica"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2025-06-12T12:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:cardano:certifierXYZ#key-1",
    "jws": "eyJhbGciOiJFZERTQSJ9..."
  }
}
```

4. Batch NFT Metadata (CIP-68 Format)

`nft-metadata/coffee-batch-234.json`

```
json

{
  "721": {
    "batch234policyid": {
      "CoffeeBatch#234": {
        "product": "Coffee",
        "harvestDate": "2025-07-15",
        "farm": "Alto Sierra Coop",
        "origin": {
          "lat": 6.25184,
          "lon": -75.56359,
          "country": "Colombia"
        },
        "cert_vc_hash": "c6f0d3a1...b0f2",
        "supplyChain": [
          {
            "actor": "ProcessorCo",
            "timestamp": "2025-08-01T12:00:00Z",
            "txHash": "abcd123..."
          }
        ],
        "qrUrl": "https://trace.app/coffee/234"
      }
    }
  }
}
```

- Use IPFS hash for `cert_vc_hash`
- QR links to the NFT metadata explorer or dApp

5. Smart Contract for Transfer Authorization (Optional)

Basic Aiken validator to verify:

- Transfer must be signed by previous holder
- Must match predefined supply chain order

```
rust
fn validate_transfer(current_holder: PubKeyHash, signer: PubKeyHash) ->
Bool {
  signer == current_holder
}
```

More advanced: Only allow specific PubKeyHashes in a preset order (enforced via datum).

6. Tools & Libraries

Layer	Tool / Library	Description
Identity	Veridian / anoncreds	Issue credentials for certification
Token	CIP-68 / Native NFT	Represent batch with metadata
Wallet	Lace / Eternl	For all actors to sign transfers
Contract	Aiken / Lucid	Validate batch transfers
UI	Lucid SDK + QR Tool	Explorer and scan interface
Storage	IPFS / Arweave	Store VC proof and batch images

7. Demo Suggestions

- Create 1 VC (e.g. “Organic Certified”)
- Mint 1 NFT batch linked to that VC
- Simulate 3 supply chain hops with wallet signatures
- Create QR for final product — viewable in Lucid + IPFS explorer

8. Output Expectations

Deliverable	Description
`vc-schema/fair-trade-cert.json`	Credential file for batch producer
`nft-metadata/coffee-batch.json`	CIP-68 metadata with full trace
`contracts/verify-transfer.aiken`	Validator enforcing legit hops
`demo/scan-qr.mp4`	Full scan-to-trace demo
`frontend/trace-ui.tsx`	Interface to render batch history

9. Best Practices for Supply Chain PoCs

✅ **Mint at Batch Level, Not Unit Level**
NFT per harvest lot = better performance, lower fees.

✅ **Use Trusted VC Issuers**
Certifications must be issued by recognized authorities (e.g., Rainforest Alliance) and include expiry.

✅ **Sign Transfers On-Chain**
Require each actor to sign handoff for full audit trail.

✅ **Avoid Fake Proofs**
Use CIP-68 + IPFS hash of credential to lock in verification.

✅ **Enable QR-based Discovery**
Each NFT should resolve via QR to a readable dApp view of full chain-of-custody.

✅ **Simulate All Edges**
Demo scenarios: missing hop, invalid cert, expired VC → all should return warnings.

✅ **Add Consumer VC Loop**
Optionally issue "Verified Buyer" VC to end-user, enabling return/rewards flow or proof of ethical sourcing.

🆘 Vertical 5: Humanitarian Aid Delivery & Crisis Response

🧠 Use Case

Design a blockchain-based aid delivery platform that ensures traceability, eligibility verification, and fast disbursement in crisis zones (e.g., refugee camps, natural disaster areas). The goal is to reduce fraud, deliver aid faster, and maintain real-time accountability for donors and agencies.

1. Functional Flow

1. **Beneficiary Onboarding**: NGO or aid organization creates a digital ID and issues a Verifiable Credential (VC) for each verified beneficiary.

2. **Aid Eligibility Verification**: The VC includes metadata such as family size, location, or medical need, signed by a known issuer.

3. **Aid Distribution Logic**: A smart contract checks the VC and disburses a stablecoin or tokenized voucher.

4. **Receipt and Audit Trail**: Every disbursement is recorded on-chain with metadata linking to the VC hash, location, and timestamp.

5. **Post-Crisis Access**: The beneficiary keeps their DID/VC to access services or reapply for aid in future contexts.

2. Architecture

```
[NGO Verifier]
     ↓
[VC: Eligible for Aid]
     ↓
[Beneficiary Wallet (Lace / Mobile)]
     ↓
[Smart Contract]
     ↓
[ADA / USDA Disbursed + Metadata Logged]
     ↓
[Explorer / Dashboard for Public Audit]
```

3. Tools & Libraries

Layer	Tool	Description
Identity	Veridian / anoncreds	VC creation for aid eligibility
Wallet	Lace, Eternl, PWA	Lightweight identity & fund receipt for low-connectivity
Contract	Marlowe or Aiken	Escrow + conditional payout logic
Metadata	CIP-68 / IPFS	Track purpose, location, VC reference
Frontend	React + Lucid SDK	Aid worker dashboard and beneficiary portal

API	GraphQL / Koios	Enable reporting, transparency feeds

4. Code Snippets

VC Example (JSON-LD format):

```
json

{
  "@context": ["https://www.w3.org/2018/credentials/v1"],
  "id": "urn:uuid:beneficiary123",
  "type": ["VerifiableCredential", "AidEligibilityCredential"],
  "issuer": "did:cardano:ngo123",
  "issuanceDate": "2025-05-01T00:00:00Z",
  "credentialSubject": {
    "id": "did:cardano:beneficiary123",
    "eligibility": "YES",
    "location": "Zaatari Camp, Jordan",
    "familySize": 6,
    "aidType": "Shelter & Food"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2025-05-01T00:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:cardano:ngo123#key-1",
    "jws": "..."
  }
}
```

Aiken Contract Logic (simplified):

```
rust

fn validate(vc_hash: ByteArray, allowed_hash: ByteArray) -> Bool {
  vc_hash == allowed_hash
}
```

5. Output Expectations

- GitHub repo with Marlowe or Aiken contract for aid disbursement
- Sample VC JSON for a beneficiary
- Screenshots or mock UI of aid worker interface for issuing VCs
- On-chain proof of disbursement with metadata (CIP-68 or log in IPFS)
- Optional: QR verification demo for public aid proof

6. Best Practices

✅ Identity Onboarding

- Use face-to-face verification with digital document scanning (passport, national ID).
- Keep VC schemas simple (avoid over-personalization to reduce stigma).
- Make wallet flows compatible with offline usage or SMS interaction if needed.

✅ Disbursement Logic

- Hardcode the disbursement window and cap per beneficiary in the smart contract.
- Use fixed price stablecoins (e.g., USDA) to avoid volatility during crisis.
- Design contracts to allow *NGO override or freeze* in case of fraud or emergency halt.

✅ Transparency & Auditing

- Log VC hash and GPS to chain for transparency, not the full VC (preserve privacy).
- Use open dashboards for donors and regulators (GraphQL feeds).
- Allow revocation of VCs and reissuance through versioning.

✅ Post-Aid Access

- Encourage users to retain wallet access post-crisis: this enables long-term benefits like employment ID, financial onboarding, and healthcare follow-ups.

- Consider biometric link for continuity in identity recovery (e.g., ZK facehash or fingerprint backup).

✅ Community Integration

- Train field officers in VC issuance tools like Veridian's offline kits.

- Partner with local mobile agents for onboarding in low-digital areas.

- Local NGOs can act as VC sub-issuers if delegated securely.

✅ General Implementation Checklist

Task	Tools / Outcome
Define actors, flow, and logic	Use PoC design canvas
Select wallet strategy	Lace, Eternl, or backend module
Choose VC tool	Veridian / anoncreds / JSON-LD
Fork from open source or build from scratch	Marlowe, Mesh
Mint tokens / NFTs if needed	CIP-68 + Mesh
Store off-chain data	IPFS, Arweave
Build front-end	Lucid SDK, React, Mesh
Document everything	GitHub with README + architecture.md