📘 Accelerator Program Curriculum

EMURGO x UNDP Blockchain Accelerator
Version: May 2025

Contents

🎯 Purpose

This curriculum is designed to equip SDG-focused innovators with practical skills to deploy blockchain-based solutions using the Cardano stack. It combines theory, case studies, and hands-on development using existing Cardano dev tools and open source projects.

📘 How to Use This Curriculum

Audience: Teams | Mentors | Evaluators
This section explains how to engage with the EMURGO x UNDP Blockchain Accelerator Curriculum.

For Accelerator Teams (Founders / Builders)

This curriculum is your **step-by-step guide** to building a real-world, SDG-aligned blockchain Proof-of-Concept (PoC).
Each module includes:

- **Learning outcomes** → what you'll understand by the end
- **Conceptual explanations** → tailored to SDG contexts
- **Tooling walkthroughs** → using Cardano, Aiken, Marlowe, VC/ID stacks
- **Exercises** → designed to produce tangible outputs for Demo Day
- **Sample outputs** → showing what good looks like

▶️ How to progress

Go through the modules in order. Complete each set of exercises in:

- Your GitHub repo (`/contracts`, `/vc-schema`, `/docs`, `/demo`)
- Your pitch deck or GTM canvas (for Module 2)
- Video demos or screenshots (where applicable)

You will use this curriculum **during mentorship**, **for your PoC build**, and as your guide through technical review.

For Mentors

Use the exercises, outputs, and best practices in each module to:

- Track team progress
- Flag missing elements (e.g., no VC schema? Missing on-chain logic?)

- Suggest better tools or design patterns

Focus especially on:

- VC / ID design quality (Modules 3, 6, 7)
- Smart contract structure (Module 3)
- Clarity and realism of implementation (Module 6)
- Adoption readiness (Module 2)

For Evaluators

Each module provides evidence of:

- Real-world applicability to SDG challenges
- Technical feasibility (with code, schema, and simulation support)
- Clarity of design and impact
- Reusability and ecosystem alignment with Cardano

*This curriculum was rewritten in textbook-style detail **to meet all reviewer feedback** from the last round, including:*

- *Fully detailed modules*
- *Best practices for every module*
- *Deep implementation integration*
- *Case study application*
- *Full traceability from learning to deliverables*

📊 Rubric Mapping: Curriculum Alignment to Reviewer Criteria

To ensure full alignment with reviewer expectations and scoring metrics, this curriculum has been explicitly mapped to the official Accelerator Evaluation Rubric. Each module delivers content and exercises that prepare teams to meet or exceed expectations in every category.

Rubric Category	Covered in Curriculum Module(s)	How It's Addressed
Blockchain Integration & Technical Readiness	Modules 3, 4, 6, 7	Participants write and deploy smart contracts (Plutus, Aiken, Marlowe), use SDKs, build frontends, and design full-stack architectures. All PoCs are required to be testnet-deployable with GitHub documentation.
Curriculum Completion & Learning Outcomes	All Modules	Each module has clear outcomes, exercises, and outputs. Sample outputs are required per team. "Mentorship Tracker" ensures accountability.
Documentation Quality & Developer Usability	Modules 4, 6	Projects must submit: GitHub repo, README, architecture.md, contract files, VC schemas, and a visual flow diagram. Mentors will review code structure, naming, and deployment clarity.
SDG Fit and Impact Alignment	Modules 1, 2, 5, 7	Teams must map their use case to an SDG. Case studies and integration pathways help refine impact framing. PoCs must document how their work addresses real-world problems and vulnerable populations.
Demo & Functionality	Module 6 + Demo Week	Final deliverable includes testnet demo (video or live), functional wallet flow, and GitHub repo. Participants simulate full use cases with contract interaction, credential usage, and token flows.
Mentor Engagement & Feedback Loops	Module 2	Weekly mentor check-ins are tracked via Notion templates. Pitch and architecture are revised iteratively. Scorecards are reviewed before Demo Day.
Open-Source Reusability	Modules 3–6	Projects must push modular, documented contracts and schemas that can be reused across other SDG verticals. Licensing, folders, and standards (e.g., CIP-68, JSON-LD) are enforced.

Post-Program Continuation Path	Module 2 (GTM Strategy)	Participants define pilot plans, funding needs, and partner pathways. The GTM Canvas aligns PoCs with grants, ministries, NGOs, or commercial partners.

📚 Detailed Modules

📦 Module 1: Blockchain Fundamentals in the SDG Context (Enhanced)

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Explain what a blockchain is and how it differs from centralized or traditional data systems
- Compare public vs. private blockchains and choose the right model for SDG use cases
- Identify SDG-aligned use cases where blockchain adds **value through transparency, traceability, or trust minimization**
- Describe how Cardano differs from Ethereum or Bitcoin in its design principles and implications for development

📖 Section 1: What Is Blockchain and Why It Matters

A **blockchain** is a distributed ledger maintained by a network of independent nodes. Unlike traditional databases, where a single entity controls data input and storage, blockchains rely on **consensus protocols** to achieve data integrity and eliminate single points of failure.

In SDG contexts, the benefits include:

- **Immutable records**: Farmers can't have their subsidy logs altered by corrupt intermediaries
- **Decentralized governance**: Multiple stakeholders can share control of disbursement contracts
- **Auditable processes**: Anyone can verify that aid was disbursed as promised

📖 Section 2: Public vs. Private Blockchains

Feature	Public Blockchain	Private Blockchain
Access	Open to all	Restricted to select nodes
Governance	Community-driven	Controlled by institution
Transparency	Fully auditable	Partially visible
Best Use	Public goods, donations	Internal logistics, compliance

Example:

- Use a **public chain** for citizen registries or subsidy disbursement
- Use a **private chain** for an internal NGO inventory system

📖 Section 3: Blockchain + SDGs – Why It's a Fit

Blockchain supports SDG efforts by addressing common gaps:

SDG Challenge	Blockchain Enabler	Example Tool
No identity → no service access	Decentralized ID + VC	JSON-LD schemas

Funds disappear in bureaucracy	Smart contract-based aid disbursement	Marlowe, Aiken
No proof of ecosystem work	GPS + sensor metadata + immutable logs	CIP-68, IPFS
No access to financial markets	Tokenized credit, P2P lending	Lucid, Aiken

📖 Section 4: Why Cardano?

Cardano was **built from the ground up** for sustainability, peer-reviewed governance, and **support for low-infrastructure environments**, which makes it a great fit for SDG implementations.

3 key innovations:

- **Proof-of-Stake (Ouroboros)**
    - Energy-efficient, no need for mining hardware
    - Ideal for projects in low-energy regions
- **eUTXO Model**
    - Every transaction is explicitly validated by conditions
    - Safer, parallelized, and easier to audit
    - Prevents double-spending without global state (unlike Ethereum)
- **Native Assets**
    - Mint tokens *without smart contracts*
    - Simpler for representing crops, housing NFTs, credits

🛠️ Practical Exercise

1. SDG Challenge Mapping

Task: Pick one SDG challenge (e.g., land title fraud, school attendance verification, farmer subsidy leakage)
Deliverable: Fill this table

SDG Problem	Blockchain Tool	Rationale
e.g. Land fraud in rural Kenya	Native token NFT + VC	Verifiable proof of land title on-chain
e.g. Training not recognized	Verifiable Credential	Proof of program completion stored in wallet

2. Cardano Explorer Task

Task: Visit Cardanoscan and paste a token policy ID
Reflect:

- Who minted it?
- What metadata is attached?
- What is the transaction size?

3. Technical Concept Comparison

Task: Write a 1-paragraph comparison of Ethereum vs. Cardano transaction models
Resources:

- Cardano Docs - eUTXO
- Aiken Quickstart

🧪 Sample Outputs

File Name / Output	Description
`sdg-mapping.md`	Table of SDG challenges and mapped blockchain tools
`cardano-explorer-task.md`	Cardanoscan screenshot + explanation
`eth-vs-eutxo.md`	Reflection paragraph comparing Ethereum vs. Cardano

🎤 Module 2: Ongoing Mentorship, Pitch Practice & Go-to-Market Strategy

This is a continuous module, running in parallel to all technical learning. It bridges product development with real-world deployment.

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Receive and act on structured feedback from mentors to improve their PoC.
- Develop a compelling, SDG-aligned pitch with clear impact and technical framing.
- Build a Go-to-Market (GTM) strategy tailored for grant programs, pilots, or real-world launches.

🧭 Why This Module Matters

Many promising blockchain solutions fail not due to weak tech, but poor framing, lack of stakeholder buy-in, or uncertain adoption paths. In SDG contexts, you must communicate

clearly with NGOs, funders, and local partners who may not be technical. This module ensures your PoC is not only well-built but clearly understood, investible, and deployable.

Mentorship begins in **Week 2** of the program and runs until **Demo Day**. You will work with technical and strategic mentors weekly.

🧑‍🏫 Section 1: Technical Mentorship

🔍 Topics Covered

- **Smart Contract Architecture**
  Plutus, Aiken, or Marlowe — depending on your project. Review validation logic, VC triggers, and payout conditions.

- **DID/VC Integration**
  Use JSON-LD VC schemas with off-chain verification. If modeling issuer portals, reference W3C-compliant stacks like Veramo or custom backend flows. Note: On-chain VC signature validation is not currently supported.

- **Wallet & Frontend Flows**
  Connect with Lucid, Mesh SDK, or backend signing APIs. Clarify what your users do and how they interact with contracts.

- **GitHub Project Structure**
  Ensure your repo is clean, documented, and logically organized. Mentors may review commits, file structure, and test coverage.

👥 Format

- Weekly 1:1 or 1:small-group mentor check-ins
- Async feedback on GitHub PRs or Notion
- Written scorecards with structured commentary
- Team action items tracked for accountability

✅ Expectations

You Should...	Mentors Will...
Bring live demo or blocker weekly	Ask clarifying questions and give actionable advice
Maintain repo + documentation	Review GitHub for clarity and logic
Implement suggested changes	Help you prepare for Demo Day readiness

📣 Section 2: Pitch Practice

🎯 Why It Matters

You are not only building a product — you are **telling a story**. Your pitch must explain the **real-world problem**, how your **solution solves it**, and why your approach is **technically sound and credible**.

🧱 Structure of a Strong Pitch

1. **Title + SDG Link**
   "Decentralized Identity for Refugee Education Access (SDG 4)"

2. **Problem Statement**
   Define the challenge with data or lived experience:
   "Refugee students lack verifiable education records. 62% drop out due to lack of eligibility verification."

3. **Solution Overview**
   One-liner of what your project does:
   "We issue Verifiable Credentials of school attendance via NGOs that can be verified by any institution."

4. **How It Works** (Technical)
   Smart contract triggers, credential issuance, wallet usage, etc.

5. **Impact & Benefits**
   Link to the SDG, end-users, and what changes.

6. **Go-to-Market Strategy**
   How you plan to deploy it — partners, pilots, or grants.

7. **Ask / Call to Action**
   "Looking for $25K in grant funding and an NGO pilot partner."

🔁 Format

- Pitch workshops every 2–3 weeks
- Asynchronous peer reviews and mentor scoring
- One live pitch simulation (Demo Day format: 5 min pitch + 5 min Q&A)

📦 Tools

- Canva / Pitch / Google Slides for deck
- Loom or Zoom for async video feedback
- Notion template to track revisions

📈 Section 3: Go-To-Market Strategy (GTM)

🔍 What Is GTM in an SDG Context?

It's your plan for how real users (farmers, NGOs, migrants, teachers...) will **find, trust, and use** your solution — and how you'll sustain or scale it.

Topics Covered

Area	Questions to Answer
Target Users	Who are they? Where do they live/work?
Value Proposition	What changes for them when they use your tool?
Distribution	Through what channel: app, SMS, local partner?
Partners	Who can help deploy this? NGOs? Ministries?
Financial Model	Will you rely on grants, subsidies, or fees?
Pilot Plan	Where, when, and how will the first deployment happen?

🧰 **Tools**

- **GTM Canvas (Notion Template)**

```
## Go-to-Market Canvas
- Target User(s): e.g., smallholder farmers in rural Niger
- Value Prop: Subsidy access without bureaucracy
- Distribution: NGO-led onboarding, SMS alerts
- Partners: VC Issuer, Koios (data infra), Ministry of Ag
- Pilot Plan: 3 regions, 2,000 farmers, 1 NGO verifier
```

- **Stakeholder Mapping Matrix**
  Identify all actors, their incentives, and blockers.

🛠️ Practical Assignments

1. **Mentor Interaction Tracker**
    - Log weekly mentor check-ins in Notion
    - Track feedback, code updates, pitch revisions
2. **Pitch Iteration Tracker**
    - Keep versions of pitch decks
    - Document evolution of story + visuals
3. **GTM Strategic One-Pager**
    - Submit draft roadmap + key partners
    - Format: Markdown or slide

🧪 Sample Outputs

File	Description
`notion/pitch-tracker.md`	Mentor feedback log + links to updated repo/deck
`slides/final-pitch-v4.pdf`	Final 5-minute Demo Day-ready pitch
`gtm/roadmap.md`	One-pager with next steps for pilot/deployment

`demo/demo-day-practice.mp4`	Screen recording of 5-min pitch rehearsal

🔐 Module 3: Smart Contracts on Cardano – Plutus, Aiken & Marlowe

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Explain the role of smart contracts in decentralized applications on Cardano.
- Distinguish between the Plutus, Aiken, and Marlowe frameworks.
- Choose the most appropriate contract framework based on their use case.
- Design and test basic contracts using Cardano tooling and playgrounds.
- Implement credential checks, milestone conditions, and fund release logic.

🧭 Module Overview

Smart contracts define the logic of decentralized systems. In Cardano, they are executed using the Extended UTXO (eUTXO) model, which provides higher security and determinism compared to traditional account-based models. This module gives you hands-on experience with:

- **Plutus** – Cardano's original smart contract language built on Haskell.
- **Marlowe** – A visual/DSL framework for financial contracts.
- **Aiken** – A modern, developer-friendly language compiling to Plutus Core.

Each approach is designed to serve different technical levels and use cases.

📘 Section 1: What is a Smart Contract?

Smart contracts are self-executing programs deployed on a blockchain. They are triggered by transactions and control the flow of assets or data.

On Cardano, smart contracts use the **eUTXO model**, where logic is attached to outputs. A smart contract consists of:

- **Datum** – Data carried by the output (e.g., a beneficiary's wallet or credential hash).
- **Redeemer** – Data supplied by the user attempting to spend the output.
- **Validator** – The logic that determines whether the transaction is allowed.

💡 Unlike Ethereum, Cardano contracts are deterministic, meaning their behavior is predictable and guaranteed to complete or fail cleanly.

🧪 Section 2: Plutus – For Custom Logic and Full Control

What is it?
Plutus is Cardano's native contract language, built in Haskell. It allows you to define powerful and expressive logic for any application.

When to use:

- You need fine-grained control over contract behavior.
- You are building something that Marlowe or Aiken cannot support (e.g., multi-asset auctions, permissionless protocols).
- Your team is comfortable with functional programming.

Use Cases:

- Role-based payments
- Custom access logic (e.g., VC or signature required)
- Auctions or lending logic

Example:

```
Haskell

{-# INLINABLE mkValidator #-}
mkValidator :: Datum -> Redeemer -> ScriptContext -> Bool
mkValidator datum redeemer ctx = traceIfFalse "Invalid redeemer" (redeemer
== expected)
```

⚖️ Section 3: Marlowe – No-Code Financial Contracts

What is it?
Marlowe is a domain-specific language (DSL) for financial smart contracts. It is designed for **non-programmers**, allowing logic to be defined through forms, drag-and-drop interfaces, or JSON.

When to use:

- You're building milestone-triggered payments, grants, escrow, or conditional transfers.
- You want to focus on flows rather than low-level syntax.

Use Cases:

- NGO grants tied to verified activity
- Microinsurance payouts
- Conditional voucher systems

Tools:

- Marlowe Playground
- Marlowe CLI (for testing & deployment)
- JSON contract export/import

Example Workflow:

1. NGO commits 100 ADA
2. Farmer submits VC ("Training Completed")
3. Funds are released automatically

⚡ Section 4: Aiken – Modern, Fast, and Developer-Friendly

What is it?
Aiken is a new open-source smart contract language designed for rapid prototyping and improved developer ergonomics. It compiles to Plutus Core but uses a **Rust-like syntax**, ideal for teams familiar with modern languages.

When to use:

- You need readable, efficient, and testable smart contracts
- You want fast feedback loops
- You're building validators that check VCs, role-based logic, or gated access

Tools:

- Aiken Lang Docs
- Aiken CLI
- VSCode Plugin (syntax highlighting, linting)

Example Validator:

```
rust

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

📊 Tool Comparison

Feature	Plutus	Marlowe	Aiken
Language	Haskell	DSL / JSON	Rust-style DSL
Best For	Custom logic	Milestone payouts	Fast MVPs, VC checks
Learning Curve	High	Low	Moderate
Tools	Playground, CLI	Playground, JSON	CLI, VSCode plugin

🛠️ Practical Exercises

✅ Exercise 1: Marlowe Grant Flow

- Open Marlowe Playground
- Define contract where `Payer = NGO` and `Payee = Farmer`
- Use a role token or VC proof as release condition
- Export JSON and simulate payout

✅ Exercise 2: Aiken Validator

- Install Aiken CLI
- Scaffold a new contract

```
bash
cargo install aiken
aiken new aid-payout
cd aid-payout
```

- Write a validator to check that a submitted VC hash matches allowed values
- Test it using local inputs

✅ **Exercise 3: Plutus Escrow Fork**

- Clone the Plutus escrow example
- Modify to include a time-based condition (e.g., must claim within 3 days)
- Document changes and push to GitHub

🧪 Sample Outputs

File	Description
`contracts/release.aiken`	Aiken validator checking VC hash
`contracts/milestone-grant.json`	Marlowe JSON for staged grant
`contracts/escrow-plutus.hs`	Modified Plutus contract with deadline logic
`README.md`	Step-by-step instructions

`demo/simulation.mp4`	Recorded testnet transaction walkthrough

Best Practices: Smart Contracts on Cardano

🔒 Security and Validation

- **Always perform type-checking and bounds enforcement** in your validator logic (e.g., ensuring token amounts, validating role-based access).
- **Test against malicious input scenarios** using emulator or CLI-based test scripts (e.g., empty datum, incorrect redeemer types).
- **Use on-chain hashes** of sensitive off-chain data (like credential or metadata verification) to preserve privacy while ensuring integrity.

🧪 Testing Strategy

- **Plutus**: Use emulator trace in Haskell to simulate contract interactions and assert success/fail conditions.
- **Aiken**: Write unit tests directly in `test/` folder using built-in Aiken test CLI. Validate all edge cases.
- **Marlowe**: Use Marlowe Playground's simulator to visualize contract execution across all paths (happy path + timeout + failure).

🔧 Optimization

- Avoid deeply nested logic in Plutus validators — it increases execution cost and reduces auditability.
- Keep datum and redeemer structures minimal to save script size and reduce on-chain data fees.
- Use **Aiken's functional clarity** to separate concerns: validate, transform, emit — each in its own function block.

📁 **Contract Design**

- **Document every contract with expected input/output format**, fallback cases, and assumptions in `README.md` and `architecture.md`.
- **Define clear triggers**: When should the contract execute? Who signs the transaction?
- **Deploy contracts to preprod/testnet first** and include signed sample transactions for mentors to validate.

Simulation, Testing & Deployment of Smart Contracts on Cardano

🎯 Purpose

This section ensures participants produce **verifiable, on-chain-valid simulations** of their smart contracts using current Cardano tools.

🧪 A. Aiken: Writing and Testing Credential-Verified Contracts

Aiken is a modern, Plutus-Core-compatible smart contract language designed for simplicity and speed.

🔧 Step 1: Write Your Validator (contracts/aid_release.ak)

```
validator {
  fn main(vc_hash: ByteArray, submitted_hash: ByteArray) -> Bool {
    vc_hash == submitted_hash
  }
}
```

This contract releases ADA if the hash of a presented credential matches an expected value.

🧪 Step 2: Write Unit Tests (test/aid_release_test.ak)

```
test "VC hash matches" {
  let expected = b"0xdeadbeef"
  let submitted = b"0xdeadbeef"
  assert main(expected, submitted)
}

test "VC hash mismatch" {
  let expected = b"0xdeadbeef"
  let submitted = b"0xcafebabe"
  assert !main(expected, submitted)
}
```

▶️ Step 3: Run the Tests

```
aiken test
```

📁 **Output**: Save output to `test/test_results.log`

Example output:

```
PASS test VC hash matches
PASS test VC hash mismatch
All 2 tests passed.
```

📌 **Best Practice**: Include negative test cases to validate failure behavior.

🏗️ Step 4: Compile Contract

```
aiken build
```

📁 This generates the compiled `.plutus` file in `contracts/aid_release.plutus`

🧪 B. Marlowe: Milestone-Based Smart Contracts

Marlowe is a DSL for modeling conditional disbursements like grants or milestone payments. It's ideal for non-developers or financial use cases.

Option 1: Use Marlowe Playground (https://marlowe.iohk.io)

1. Go to **Marlowe Playground**
2. Select **New Contract > Escrow**
3. Configure:
    - **Payer**: NGO
    - **Payee**: Farmer
    - **Condition**: Pay if VC proof submitted
    - **Amount**: 10 ADA
    - **Timeout**: 3 days
4. Run simulation in 3 modes:
    - ✅ Success: VC provided
    - ❌ Failure: VC not submitted
    - ⏰ Timeout: No action → refund NGO
5. Export:
    - JSON Contract: `contracts/milestone-grant.json`
    - Screenshots or video: `demo/marlowe-simulation.mp4`

Option 2: Marlowe CLI Simulation

Required for reviewer traceability if Playground is unavailable.

1. Save this contract JSON to `contracts/milestone-grant.json`:

```
json
{
  "when": [
    {
      "case": {
        "party": { "role_token": "Farmer" },
        "deposits": 0,
        "of_token": {
          "currency_symbol": "",
          "token_name": "lovelace"
        },
        "into_account": { "role_token": "NGO" }
      },
      "then": {
        "pay": {
          "amount": 10000000,
          "to": { "party": { "role_token": "Farmer" } }
        },
        "token": {
          "currency_symbol": "",
          "token_name": "lovelace"
        },
        "then": "close"
      }
    }
  ],
  "timeout": 172800,
  "timeout_continuation": "close"
}
```

2. Install CLI:

```
git clone https://github.com/input-output-hk/marlowe-cardano
cd marlowe-cardano
cabal build
```

3. Run the simulation:

```
marlowe-cli run \
  --contract-file contracts/milestone-grant.json \
  --print-stats
```

📁 Save CLI output in `logs/marlowe_simulation.log`

🧪 C. Realistic VC Hash Testing on Testnet (Optional but Highly Recommended)

Simulate **wallet + contract + VC hash interaction** using MeshJS on Preprod:

1. Hash a Sample Credential

File: `utils/hash.ts`

```
ts
import { sha256 } from "js-sha256";

export function hashVC(vc: object): string {
  const raw = JSON.stringify(vc);
  return sha256(raw); // returns hex string
}
```

2. Submit a Transaction with VC Hash

File: `frontend/send_tx.tsx`

```
tsx
import { useWallet, Transaction } from '@meshsdk/core';

const SendVCProof = () => {
  const { wallet, connected } = useWallet();

  const sendTx = async () => {
```

```
    const hash = "0x3b7c..."; // Insert hash from hashVC()

    const tx = new Transaction({ initiator: wallet })
      .sendAssets("addr_test1...your_script_address", [
        { unit: "lovelace", quantity: "2000000" },
      ])
      .attachSpendingValidator({
        type: "PlutusV2",
        script: "<compiled_script.cbor>", // From Aiken output
      })
      .redeemWith("PlutusData::new_constr(0,
[PlutusData::new_bytes(hash)])");

    const unsignedTx = await tx.build();
    const signedTx = await wallet.signTx(unsignedTx);
    const txHash = await wallet.submitTx(signedTx);

    alert(`Transaction submitted: ${txHash}`);
  };

  return connected ? (
    <button onClick={sendTx}>Claim Grant</button>
  ) : (
    <div>Please connect your wallet</div>
  );
};
```

3. Record Demo Flow

Record:

- Wallet connect
- Button click
- Transaction hash popup
- Preprod explorer link

📁 Save video: `demo/sdk_flow.mp4`
📁 Save script: `frontend/send_tx.tsx`

🧩 Final Submission Checklist

File	Description
`contracts/aid_release.ak`	Aiken source
`contracts/aid_release.plutus`	Compiled contract
`test/aid_release_test.ak`	Aiken tests
`logs/test_results.log`	Aiken test results
`contracts/milestone-grant.json`	Marlowe JSON
`logs/marlowe_simulation.log`	CLI simulation
`demo/marlowe-simulation.mp4`	Playground simulation
`frontend/send_tx.tsx`	Frontend submission logic

`utils/hash.ts`	VC hash function
`demo/sdk_flow.mp4`	Frontend interaction video
`README.md`	Steps to reproduce locally

🔁 Mini Project Challenge: Hash-Gated Training Subsidy Flow

🧩 **Scenario:**
An NGO wants to issue subsidies to farmers who completed a training course. Each participant receives a **Verifiable Credential (VC)** off-chain. Only wallets that can prove possession of the VC via a known hash will receive funds.

🎯 Your Mission

Design an **Aiken smart contract** that:

- Accepts a hash (representing a credential)
- Compares it to an expected value embedded in the validator
- Releases ADA if it matches

This bypasses the cryptographic verification limitation, using hash-based inclusion instead.

⚠️ Technical Note: Identity Limitations on Cardano

> 🔒 On-chain verification of identity **signatures** is **not currently possible** with Veridian, anoncreds, or other DID stacks, due to the lack of cryptographic primitives in Plutus.
>
> ✅ This PoC uses **hash-based VC matching** to simulate VC verification in a minimal, feasible way.
>
> ❗ Do not model or imply signature verification unless you are using a ZK or external oracle-based system.

✅ Deliverables (Testnet Ready)

Path	What
`contracts/training_subsidy.aiken`	Aiken smart contract that checks a submitted credential hash
`vc-schema/training_completed.json`	The off-chain credential schema in JSON-LD
`demo/claim_flow.mp4`	Screen recording or screenshot of testnet interaction
`README.md`	Step-by-step test instructions for the PoC

🧪 Step-by-Step Instructions

1. Define the VC Schema (off-chain)

In `vc-schema/training_completed.json`:

```
json
{
  "@context": ["https://www.w3.org/2018/credentials/v1"],
  "type": ["VerifiableCredential", "TrainingCompleted"],
  "credentialSubject": {
    "id": "did:web:amina.farm",
    "name": "Amina Mbaye",
    "trainingId": "AGRO-101",
    "dateCompleted": "2025-05-01"
  },
  "issuer": "did:web:ngo.org",
  "issuanceDate": "2025-05-01"
```

```
}
```

2. Hash the Credential Off-Chain

```
bash
cat training_completed.json | sha256sum
# Output: e.g., 3b7c5d2f...abcdef
```

Use the hash of the **claim body**, not the full VC.

3. Write the Aiken Validator

Install Aiken:

```
bash
cargo install aiken
aiken new training_subsidy
cd training_subsidy
```

In `src/validator.ak`:

```
rust
fn main(submitted_hash: ByteArray) -> Bool {
  let expected_hash = 0x3b7c5d2fabcd1234deadbeef...
  submitted_hash == expected_hash
}
```

4. Simulate Testnet Flow (Client Side)

Use **MeshJS** or **Lucid-Evolution** (not deprecated Lucid) to:

- Lock 10 ADA in the script
- Submit a transaction with the hash of the user's VC

- Validator checks it, and if it matches, releases ADA

You must have a script address, valid UTXO, and hash input.

5. Record Demo

Record the wallet interaction (e.g., Eternl or Mesh-based dApp), transaction submission, and testnet explorer confirmation.

📁 Folder Structure

```
/contracts/
  training_subsidy.aiken
/vc-schema/
  training_completed.json
/demo/
  claim_flow.mp4
README.md
```

🔍 Evaluation Criteria

Area	What to Look For
✅ Validator Logic	Must check hash match only, no signature logic
✅ No Cryptographic Claims	No mention of on-chain VC verification
✅ Clean Folder Structure	Files in `/contracts`, `/vc-schema`, `/demo`
✅ Walkthrough Clarity	README must explain every step, no gaps

🟨 Optional Bonus (For Advanced Teams)

- Accept multiple valid hashes from a **hardcoded list**
- Add a **time check**: must claim before subsidy expires
- Log hash to a dummy CIP-68 metadata token for transparency

⚙️ Module 4: AI for Blockchain, SDKs & APIs

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Integrate smart contracts into frontends using modern Cardano SDKs (MeshJS, Lucid Evolution)
- Build browser-based and mobile-compatible dApps
- Query and analyze on-chain data using Koios, Blockfrost, or GraphQL
- Utilize AI tools (e.g., Copilot, GPT) to accelerate dApp prototyping and schema generation

🧭 Module Overview

Developing smart contracts is just one aspect of blockchain development. Most users interact through wallets, websites, and dashboards. This module introduces:

- SDKs for interacting with Cardano smart contracts
- APIs for querying on-chain data
- AI tools to expedite development processes

🧰 Section 1: SDKs – Connecting to Cardano from Your App

1.1 MeshJS (Recommended)

MeshJS is a modular and beginner-friendly SDK for Cardano, offering:

- Wallet integration (Eternl, Nami, Lace)
- Transaction construction and submission
- Native token minting
- Smart contract interactions

Installation:

```
npm install @meshsdk/core @meshsdk/react
```

Example: Send ADA to a wallet

```
typescript
import { Transaction } from '@meshsdk/core';

const tx = new Transaction({ initiator: wallet })
  .sendAssets("addr1...", [{ unit: "lovelace", quantity: "2000000" }])
  .complete();
```

Use Cases:

- Trigger contract logic from dApp frontend
- User signature verification
- Display wallet balances and token metadata

1.2 Lucid Evolution (Advanced Teams)

Lucid Evolution is a modern, actively maintained fork of the original Lucid SDK, designed for scalability and production readiness. It offers:

- Enhanced transaction building capabilities
- Support for multiple providers (e.g., Koios, Blockfrost)
- Improved developer experience with TypeScript support

Installation:

```
npm install @lucid-evolution/lucid
```

Installing the lucid package will automatically export all other Lucid Evolution packages as well. For more information on more granular package definitions for lightweight development, please refer to the Evolution library installation guide.

Example:

```
typescript
import { Lucid, Koios } from "@lucid-evolution/lucid";

const lucid = await Lucid(new Koios("https://preprod.koios.rest/api/v1"),
"Preprod");
```

Use Cases:

- Advanced dApp development
- Custom transaction workflows
- Integration with various Cardano tools and services

🔌 Section 2: Blockchain APIs – Querying Cardano Data

2.1 Koios

Koios is a REST-based API optimized for querying blockchain state.

Endpoints:

- `/account_info`
- `/asset_info`
- `/tx_info`
- `/utxo_by_address`

Example: Get wallet UTxOs

```
curl -X POST https://api.koios.rest/api/v0/address_info \
  -H "Content-Type: application/json" \
  -d '["addr1..."]'
```

Use Cases:

- Validate if a user has received a token
- Build NGO dashboards (e.g., disbursed aid, claim counts)
- Generate visual reports from on-chain metadata

2.2 Blockfrost

Blockfrost offers indexed access to Cardano data with usage quotas.

Setup:

- Create an API key at blockfrost.io

- Access via REST or client libraries

JS Fetch Example:

```
typescript
const res = await
fetch("https://cardano-preprod.blockfrost.io/api/v0/assets", {
  headers: { "project_id": "<your-key>" }
});
```

Use Cases:

- Simplified integration with frontend apps
- Enhanced metadata indexing for tokens
- Ideal for educational dashboards and partner pilots

2.3 GraphQL (Advanced)

For custom queries, such as "all VC issuances by NGO X," you can set up a GraphQL node from db-sync.

Requirements:

- Cardano-db-sync setup
- GraphQL server configuration

Use Cases:

- Exploratory queries
- Custom NGO monitoring dashboards
- Data warehousing or off-chain analytics

🤖 Section 3: AI Tools for Blockchain Dev Acceleration

Artificial Intelligence can significantly reduce development time by assisting with:

- UI scaffolding
- Code auto-completion
- VC schema generation
- Error debugging

3.1 ChatGPT / GPT-4

Use prompts to:

- Generate MeshJS functions (e.g., transaction creation)
- Convert JSON schemas into types
- Explain unfamiliar smart contract logic

Prompt Examples:

- "Write a MeshJS function that checks for token ownership before submitting a transaction"
- "Draft a JSON-LD schema for a Verifiable Credential proving school attendance"

3.2 GitHub Copilot

Integrates directly into VSCode to suggest:

- Contract logic (Plutus/Aiken)
- UI states for wallet flows
- Form validation and conditional flows

3.3 0dev.app (or similar)

AI-powered frontend generators that build full UIs with wallet connect, token tables, and input validation from a prompt.

🛠️ Practical Exercises

✅ Exercise 1: Connect a Wallet and Send ADA (MeshJS)

- Install MeshJS
- Connect Eternl or Lace
- Build and submit a test transaction
- Track the TX on Cardanoscan

✅ Exercise 2: Mint a Token Using MeshJS

- Install MeshJS
- Use TokenMinter component
- Set:
    - Token name = "UNDPaid"
    - Metadata = `{ goal: "SDG 2", expiry: "2026" }`
- Submit on Preprod
- Share token metadata via Blockfrost

✅ Exercise 3: Query Token Stats with Koios

- Use `/asset_info` endpoint
- Parse JSON response
- Display info in a frontend or dashboard

✅ Bonus: AI Prompt Challenge

Write a prompt to generate:

- A credential schema: "Farmer Credential with name, region, subsidy eligibility, and signature"
- A MeshJS transaction flow for issuing a VC-gated payment

🧪 Sample Outputs

File	Description
`wallet-connect.tsx`	MeshJS integration to connect/test wallet
`token-mint.tsx`	MeshJS UI for testnet token minting
`query-token.js`	Koios-based fetch for token metadata
`vc-schema/farmer.json`	AI-generated VC credential schema
`demo/wallet-flow.mp4`	Screen recording of wallet → mint → send flow

Best Practices: SDKs, APIs & AI-Enhanced Prototyping

🧩 SDK Integration

- Use MeshJS for composability: Break frontend logic into reusable functions (e.g., `buildTx`, `signTx`, `submitTx`) to ensure maintainability and reduce bugs in dApp flows.

- Always check wallet connection status before transaction signing. Provide fallback UI if the user has no funds or the wrong network selected.

- Store user session locally (e.g., with `localStorage` or secure cookies) to reduce re-authentication steps across sessions.

🔌 API Usage & Reliability

- Use Koios for reliable UTxO and address data — it's rate-limited but extremely stable for open-access projects.

- For production-level integrations or multi-user dashboards, use Blockfrost and handle API key throttling gracefully (add retries and backoff logic).

- Normalize timestamps, slot numbers, and transaction hashes before saving into your backend or visualizing on the frontend.

⚙️ Smart Contract Interaction

- Always validate minimum ADA requirements when sending assets or minting tokens.

- Use CIP-68 metadata schema standards for NFT or token minting — it increases compatibility across explorers and wallets.

- When wrapping smart contract logic in SDKs, log full errors and transaction status to help mentors debug user issues.

🤖 AI-Enabled Development

- When using 0dev or similar AI tools, treat output as scaffolding — not production code. Review, test, and refactor everything manually.

- Use AI to generate placeholder VC schemas, then validate them against JSON-LD schema validators.
- For non-coders on your team, AI tools like Copilot or GPT-based prompt templates can be used to auto-generate `README.md` content, license files, or boilerplate React components.

🧪 Testing & MVP Delivery

Before demo day, simulate full flows:

- Connect wallet
- Mint or send token
- Verify contract interaction on testnet explorer
- Include screenshots of each UI state (loading, success, error) in your demo folder to document user experience expectations.

Mini Project Challenge – SDK Integration & Testnet Contract Trigger

🧩 Scenario:
You've already written a smart contract in **Module 3** that validates a hash of a Verifiable Credential (VC). Now, you'll create a frontend that:

- Connects to a user's wallet
- Accepts or hardcodes a VC hash
- Submits a transaction to the contract
- Shows the result (success/failure) on Cardano **Preprod**

🎯 Goal:

Use **MeshJS** to build a simple frontend that connects to the wallet and interacts with your validator smart contract on the Cardano testnet.

✅ Deliverables:

File	Description
`frontend/send_tx.tsx`	React component that builds and submits the transaction
`utils/hash.ts`	Script that computes the hash of the VC
`demo/sdk_flow.mp4`	Screen recording of wallet → submit → transaction flow
`README.md`	Instructions to run frontend and simulate contract interaction

🛠️ Step-by-Step Instructions

1. Set Up MeshJS

```
npm install @meshsdk/core @meshsdk/react
```

2. Create the Submission Component

In `frontend/send_tx.tsx`:

```
tsx
import {
  useWallet,
  Transaction,
} from '@meshsdk/core';

const SendVCProof = () => {
  const { wallet, connected } = useWallet();

  const sendTx = async () => {
    const hash = "0x3b7c..."; // Replace with your VC hash (hex string)
```

```
    const tx = new Transaction({ initiator: wallet })
      .sendAssets("addr_test1...your_script_address", [
        { unit: "lovelace", quantity: "2000000" },
      ])
      .attachSpendingValidator({
        type: "PlutusV2",
        script: "<compiled_script.cbor>", // Path or string of compiled
contract
      })
      .redeemWith("PlutusData::new_constr(0,
[PlutusData::new_bytes(hash)])");

    const unsignedTx = await tx.build();
    const signedTx = await wallet.signTx(unsignedTx);
    const txHash = await wallet.submitTx(signedTx);
    alert(`Transaction submitted: ${txHash}`);
  };

  return connected ? (
    <button onClick={sendTx}>Claim Training Grant</button>
  ) : (
    <div>Please connect your wallet</div>
  );
};
```

3. Compute the VC Hash (Off-Chain)

In `utils/hash.ts`:

```
ts
import { sha256 } from "js-sha256";

export function hashVC(vc: object): string {
  const raw = JSON.stringify(vc);
  return sha256(raw); // Returns a hex string
}
```

You can hardcode the hash in the frontend or compute it dynamically if needed.

4. Run Frontend and Submit

- Ensure your smart contract is deployed and funded on Preprod.
- Run the frontend locally (e.g., using `vite`, `next`, or `react-scripts`).
- Connect wallet (Nami, Eternl, Lace).
- Click the button to trigger transaction.
- Verify on https://preprod.cardanoscan.io.

5. Record Demo

Record a screen capture of:

- Connecting wallet
- Submitting the transaction
- Viewing transaction confirmation
- Verifying funds released (on explorer)

Save as `demo/sdk_flow.mp4`.

📁 Recommended Folder Structure

```
/frontend/
  send_tx.tsx
  utils/hash.ts
/contracts/
  training_subsidy.aiken
/demo/
  sdk_flow.mp4
README.md
```

🔍 Evaluation Criteria

Area	Expectation
✅ Correct SDK Used	MeshJS only (Lucid is deprecated)
✅ VC Hash Injection	Provided or computed off-chain, submitted to validator
✅ Testnet Transaction	Funds released or rejected based on hash match
✅ Demo Proof	Screen recording or screenshots with testnet explorer link
✅ Documentation	`README.md` explains how to run and test the frontend

🌍 Module 5: Case Studies – Blockchain + SDG in the Field

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Analyze real blockchain implementations deployed in SDG-related sectors
- Identify the architecture patterns, tooling, and integration techniques used
- Recognize key challenges, such as trust-building, UX adaptation, or low-connectivity constraints
- Translate lessons from the field into tangible strategies for their own PoCs

🧭 Module Overview

In development environments, everything works. In the field, reality hits harder: users forget passwords, wallets aren't installed, and trust is a scarce currency.

This module explores real-world blockchain projects — in Africa, Latin America, and beyond — where solutions were deployed for **identity**, **payments**, **property rights**, and **climate accountability**. Each case study is broken down by:

- SDG relevance
- Technical stack
- Architecture pattern
- What worked vs. what failed
- Best practices for replication

You'll be asked to critically evaluate each example, identify architecture that maps to your use case, and write your own lessons learned.

📦 Case Study 1: Veridian – Digital Identity on Cardano

SDG Focus: SDG 16 (Peace, Justice, and Strong Institutions)
Region: Global (pilots in Africa and Latin America)
Problem: No unified, verifiable identity for talent, suppliers, or communities receiving aid.

Solution:
Veridian built a **decentralized identity platform** where NGOs, employers, and training institutions can issue **Verifiable Credentials (VCs)** to individuals — tied to a DID (Decentralized Identifier).

These VCs can be presented for aid eligibility, job access, or program verification.

Key Technologies:

- `did:cardano` and `did:web` methods
- Verifiable Credentials in JSON-LD
- Lace wallet integration

- Revocation registry via smart contracts
- VC request & presentation over QR

Architecture Summary:

```
css
[Issuer (NGO)] → Issues VC → [User Wallet]
[Verifier (Gov/Employer)] ← Verifies VC ← [User Presents]
```

What Worked:

- VCs created accountability and audit trails without exposing user data
- DID + wallet pairing worked well in self-custody
- VC schema reuse helped standardize identity types (e.g., “Farmer Verified”, “Attended Training”)

What Struggled:

- Training required for issuers to use credential tooling
- Some users lacked access to compatible wallets
- Device sharing in households introduced VC sync complexity

Best Practice Tip:
Start with a custodial wallet option and migrate to self-custody gradually. Include both JSON download and QR-based VC options.

💳 Case Study 2: Afriex – Crypto Remittance in West Africa

SDG Focus: SDG 1 (No Poverty), SDG 10 (Reduced Inequalities)
Region: Nigeria, Ghana, Ethiopia, Cameroon
Problem: Costly and slow cross-border payments for immigrant families

Solution:
Afriex allows users in the U.S. and Europe to send stablecoins to African wallets, where they can be withdrawn in local fiat or held in-app.

Key Technologies:

- Mobile-first custodial wallet
- Stablecoins (e.g., USDC, cUSD)
- Fiat on/off-ramps in local banks
- KYC via mobile number and selfie verification

Architecture Summary:

```
css
[Sender Wallet] → [Stablecoin Transfer] → [Afriex Custody Wallet] →
[Local Bank Withdrawal]
```

What Worked:

- Seamless UI reduced the mental friction around “crypto”
- Fast finality enabled 2-minute remittance confirmation
- Used blockchain as infrastructure, not brand — users didn’t even know they were using crypto

What Struggled:

- Regulatory shifts affected fiat ramps
- Education gap for “how to cash out”
- Low financial literacy among new users

Best Practice Tip:
Abstract crypto terminology in user-facing apps. UX should be local-language, with familiar metaphors (e.g., “mobile airtime” vs. “wallet balance”).

🏠 Case Study 3: HouseAfrica – NFT-Backed Property Registry

SDG Focus: SDG 11 (Sustainable Cities and Communities)
Region: Nigeria
Problem: Fraudulent land titles and opaque property ownership

Solution:
HouseAfrica provides a **verifiable property registry** on Cardano, where verified plots are represented by **CIP-68 NFTs**. Buyers can scan QR codes to check validity.

Key Technologies:

- CIP-68 NFTs with geodata, parcel ID, legal hash
- Web-based minting portal for verified surveyors
- QR-linking for resale or mortgage verification
- Community co-signing via VC system (optional)

Architecture Summary:

```
[Surveyor Verified Title]
  ↓
[NFT Minted on Cardano]
  ↓
[Buyer / Bank Scans NFT → Views Proof]
```

What Worked:

- Reduced title fraud for off-plan buyers
- Created asset history log (e.g., resale, mortgage use)
- Local governments showed interest for district land mapping

What Struggled:

- NFT metadata required standardization across projects

- Some banks hesitant to accept “digital land”
- Token transfer required legal agreements (off-chain)

Best Practice Tip:
Use tokens as pointers — not legal records. Pair every NFT with a credential or scanned deed stored off-chain.

🌱 Case Study 4: Changeblock – Tokenized Carbon Credits

SDG Focus: SDG 13 (Climate Action), SDG 12 (Responsible Consumption)
Region: Africa (multiple pilot regions)
Problem: Small carbon projects lack access to global carbon markets

Solution:
Changeblock helps verifiers issue carbon credit VCs which are then used to mint **verified NFTs**. These NFTs can be sold, audited, and tracked across time.

Key Technologies:

- VCs: “Offset Verified: 1 Ton CO_2”
- Smart contract-based royalty split (project owner gets a share on every resale)
- CIP-68 token standard
- On-chain payment logs for ESG reporting

Architecture Summary:

```
[Verifier] → Issues VC → [Token Minted]
       ↓
[Buyer Purchases NFT] → [Funds Disbursed to Project + Verifier]
```

What Worked:

- Small projects got global access to buyers
- On-chain audit trail reduced greenwashing

- Metadata-driven tokens enabled clear project info (location, verifier, timestamp)

What Struggled:

- Buyers had questions about VC authenticity
- Price discovery was difficult for new projects
- Off-chain sensor integration was incomplete

Best Practice Tip:
Bundle VC + token. On-chain alone is not enough — trust must be rooted in who issued the credential.

🛠️ Practical Exercises

1. Select One Case Study

Analyze it using the following lens:

- SDG relevance
- Key architecture flow
- Tech stack breakdown
- Real-world barriers and UX patterns

2. Map Learnings to Your Own Project

Answer:

- What tooling could I re-use from this case?
- What challenges will I face in my geography or user segment?
- How could I improve the model with AI, identity, or wallet strategy?

3. Write “What Worked / What Could Improve”

Two structured paragraphs:

```
✅ What Worked: [e.g., clear metadata, fast onboarding, VC reuse]
🧱 What Could Improve: [e.g., offline sync, KYC method, transparency gap]
```

🧪 Sample Outputs

Output	Description
`case-study-analysis.md`	A markdown summary of all 4 projects with tech + SDG notes
`my-project-mapping.md`	Description of which architecture pieces apply to your PoC
`slide-case-study-learnings.pdf`	One-slide visual summary of case study insights applied to your work

Best Practices: Applying SDG Case Study Insights

🧭 Technical Architecture Mapping

- **Extract and document system architecture** from each case study (e.g., Veridian, Afriex). Diagram how DIDs, VCs, contracts, tokens, and frontends interact.
- **Identify shared patterns** across use cases:
    - DID issuance → VC storage → verification before service
    - Token issuance tied to metadata (CIP-68, IPFS)
    - Role-based access enforced via credentials or contracts
- Reuse modular components (e.g., VC verification contract, Lucid SDK integration) instead of building from scratch.

🪪 Identity & Credential Learnings

- Projects like Veridian demonstrate the importance of **trusted credential issuers**. Establish real-world equivalence: who plays that role in your context (NGO, gov, coop)?

- Ensure that **VC schemas match real use cases** — don't over-abstract. A farmer's "Training Completed" VC needs only name, date, GPS, and hash.

💸 Payments & Wallet UX

- From Afriex: prioritize **mobile-first wallet design**, especially if your user base is in areas with low bandwidth or old devices.

- Consider **custodial vs. non-custodial tradeoffs** for users with limited tech exposure. Afriex succeeded partly because it abstracted private key management early on.

🌍 Traceability & Certification

- From Changeblock and HouseAfrica: tokens are not trusted unless the **source data is verifiable and auditable**.

- Always link tokens (NFTs or fungible) to:
    - VC hash of verified input ("Certified Project")
    - Timestamp, location, issuing entity

- Use **QR codes + explorer previews** to ensure transparency is human-readable, not just on-chain.

🧠 Reflection & Adaptation

- For each case you analyze, ask:
    - What assumptions did they make about users?
    - What tech did they not need at MVP stage?
    - What partnerships were critical to deployment?

- Then clearly document **how your use case differs**: new geography, user base, tech stack, or goal.

📁 Documentation & Sharing

Include a dedicated folder in your repo:

```
/case-study-analysis
  └── <project-name>.md
```

-
- Use this to summarize:
    - Case architecture
    - Reusable tools
    - What your team is adopting or changing
- This will also help with mentor review and judging alignment with real-world impact.

🧩 Module 6: Implementation Design & Architecture

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Design full-stack blockchain applications tailored to SDG-specific PoCs
- Translate use cases into modular technical architectures (contracts, wallets, credentials)
- Structure and document repositories for handoff, review, and scale-up
- Apply implementation best practices for open-source, testnet-deployable solutions

🧭 Module Overview

Even great ideas collapse without technical clarity. This module teaches how to **translate your PoC idea into a complete and professional blockchain system**, with:

- Defined actors, logic, and flows
- Clear separation of on-chain vs. off-chain components
- Credential schemas and token standards
- Smart contract triggers and fail conditions
- Clean, testable GitHub repositories

The goal is not just to build — it's to build in a way that mentors, funders, and partners can understand, test, and extend.

🔧 Section 1: Define Actors & Roles

Start with a **human-readable model**. Identify all the stakeholders in your system and define their roles, permissions, and trust relationships.

Example: Fertilizer Aid PoC

Actor	Role Description
Beneficiary	Farmer receiving aid; holds wallet and credentials
NGO	Issues eligibility credentials; funds the smart contract
Oracle	Provides ADA/USD price feed (optional)
Smart Contract	Releases funds based on VC and time logic

📌 **Best Practice:** Roles should align with real-life responsibilities (e.g., NGO signs VCs, not the farmer).

🔁 Section 2: Define Data Flows and Token Logic

Next, define how **data and tokens move** between those actors.

Break it down into:

- **Credential Flow**: Who issues the VC? Who holds it? Who verifies it?
- **Smart Contract Flow**: When does it get triggered? What inputs are needed?
- **Wallet Flow**: Who signs transactions? Is the wallet custodial or self-managed?

Visual Flow (Fertilizer Aid Example):

```
[NGO] --issues--> [VC: "Farmer Verified"]
          ↓
[Farmer Wallet] holds VC
          ↓
[Smart Contract] checks VC hash
          ↓
[Payment Released] to farmer in ADA
```

🛠️ Tools: Draw.io, Lucidchart, Whimsical, or Mermaid.js

🔐 Section 3: Smart Contract Lifecycle Planning

For each smart contract, document the **full lifecycle**:

Attribute	Example
Trigger	Submission of valid VC hash and signature
Validator	Check that the hash matches a known “Verified Farmer” VC
Redeemer	VC hash + timestamp
Output	Send ADA to farmer wallet
Rejection	If VC hash not in registry or expired

📌 **Pro Tip:** Describe both happy path and fail path. Include expiry rules or revocation options.

📁 Section 4: GitHub Repo Structure

A clean, well-documented repo is part of the PoC output. Use a modular folder system:

```
/contracts          → Aiken, Marlowe, or Plutus scripts
/frontend           → React or Next.js UI (with Lucid or Mesh)
/vc-schema          → JSON schemas for issued credentials
/docs               → architecture.md, sequence diagrams, VC templates
/demo               → screenshots, screen recordings, pitch video
README.md           → How to run the PoC
architecture.md     → System overview and flow explanation
.gitignore, LICENSE → Basic open-source hygiene
```

📌 **Minimum Requirements:**

- A working contract file (e.g., `contracts/aid_release.aiken`)
- A test credential in `/vc-schema`
- A clear walkthrough in `README.md`
- A diagram in `/docs` showing actors + flows

🧱 Section 5: Writing architecture.md

This is your single source of truth. It should cover:

1. **PoC Description**
    - What problem are you solving?
    - Who is the user?
    - What SDG does it align with?

2. **System Overview**
    - High-level diagram (e.g., [issuer] → [wallet] → [smart contract] → [payment])
    - On-chain vs. off-chain components
    - Credential, token, or NFT logic
3. **Smart Contracts**
    - What triggers them?
    - What do they validate?
    - How do they fail gracefully?
4. **Credential Schema**
    - VC schema JSON
    - Fields: subject, issuer, issuance date, claim type
5. **Deployment Setup**
    - Is this deployed on testnet?
    - Which wallet to use?
    - How to mint or simulate?

🛠️ Practical Exercises

Exercise 1: Actor & Flow Canvas

Use this markdown template:

```
## Actors
- NGO: Issues "Training Completed" VC
- Beneficiary: Holds wallet, receives aid
- Smart Contract: Releases ADA based on VC
```

```
## Data Flow
[NGO] → [VC Issued] → [Farmer Wallet]
↓
[Smart Contract Triggered] → [Payout Released]

## Smart Contract Logic
Trigger: VC hash + signature
Validator: VC hash must match registry
Output: Send 10 ADA
Failure: Reject if hash mismatch or expired
```

Exercise 2: Create Repo Skeleton

Push folders with dummy content for:

- `contracts/`
- `vc-schema/`
- `docs/architecture.md`
- `README.md`

📌 Use this as your weekly submission base.

Exercise 3: Diagram It

Create a flow diagram using draw.io or Mermaid.js:

- Actors (boxes)
- Flows (arrows)
- Smart contracts (special nodes)
- Labels like “VC issued”, “Signature verified”

🧪 Sample Outputs

File	Description
`architecture.md`	Text-based system blueprint
`contracts/aid_release.aiken`	Credential-based payout logic
`vc-schema/farmer.json`	JSON-LD VC schema used in demo
`docs/diagram.svg`	Visual of actors and contract flow
`README.md`	Setup instructions and test wallet flow
`demo/video.mp4`	Recorded walkthrough of PoC from wallet to contract

Best Practices: Full-Stack SDG Blockchain Implementation

🔧 1. Start from Roles and Flows – Not Tech

- Clearly define **who does what**: beneficiary, issuer, verifier, smart contract, oracle, frontend.
- Use a **real-world narrative** (“a woman farmer in Uganda verifies training → unlocks voucher”) to guide data and contract flows.
- Avoid tech-first designs — SDG PoCs fail when they’re disconnected from user reality.

🧱 2. Build Modular and Reusable Components

Separate your codebase into:

```
/contracts
/vc-schema
/frontend
/utils
```

- Keep your **VC logic reusable**: define a common JSON-LD schema and avoid hardcoding values.
- Make contracts configurable via parameters (e.g., payout amount, eligible role).

🧪 3. Test Smart Contracts and Credentials in Parallel

- Don't wait to “finish” contracts before testing VCs.
  Instead:
    - Simulate VC creation and verification
    - Use hardcoded hashes in test contracts
    - Validate flows before building full UI
- Use **testnets and tools like Marlowe CLI or Aiken Emulator** early.

📦 4. Simplify Credential Flow for MVP

- MVP Rule: **One VC, One Use Case**.
  Example: “Farmer Trained” = access to subsidy.
- Use mock issuers for testing (e.g., your own DID).
- Only store VC reference hash on-chain to keep costs low.

📁 5. GitHub Repo Best Practices

Structure folders clearly:

```
/contracts
/vc-schema
/frontend
/docs
/demo
```

- Include:
    - `README.md`: what, why, how to run
    - `architecture.md`: flows, diagrams, assumptions
    - `schema/`: all credential JSON schemas
    - Demo folder with a walkthrough video or screenshots

📊 6. Document Known Limitations

- Reviewers expect honesty. In `architecture.md`, add:
    - “Assumes manual VC issuance for now”
    - “Future: replace oracle with sensor feed”
    - “Current wallet UX not optimized for mobile”
- Mark `TODO:` areas in code for improvement roadmap.

👩‍💻 7. Mentor-Ready and Reviewer-Friendly

- Every PoC should be:
    - Forkable and runnable with clear instructions
    - Mapped to SDG goal(s) and real-world actors

- Traceable from user interaction to smart contract trigger

🧩 Module 7: Integration Pathways for Identity, Payments & Humanitarian Aid Transparency

🎓 Learning Outcomes

By the end of this module, participants will be able to:

- Integrate Decentralized Identity (DID) and Verifiable Credentials (VCs) into real-world service flows
- Design programmable aid disbursement mechanisms using stablecoins or ADA
- Implement on-chain traceability for humanitarian funding, beneficiaries, and outcomes
- Use schema-driven, privacy-conscious identity models compatible with Cardano infrastructure

🧭 Module Overview

Humanitarian and development programs face recurring challenges around trust, transparency, targeting, and traceability. This module focuses on **real-world integration pathways** for:

1. Decentralized identity issuance and verification
2. Conditional payments using Cardano smart contracts
3. Transparent, auditable records of aid impact

It builds on prior modules by stitching together key building blocks (VCs, contracts, wallets, metadata) into **full humanitarian-grade PoCs**.

🪪 Section 1: Identity Management Integration

Why Identity?

In SDG-aligned programs, the first question is often: *“Is this person eligible?”*

Whether for refugees, smallholder farmers, or women-led businesses, **identity is the gateway to service**. But many vulnerable populations lack a verifiable digital footprint.

That's where **self-sovereign identity (SSI)** and **VCs** come in.

Identity Stack (Cardano-based)

Layer	Tool	Role
DID Method	`did:cardano`, `did:peer`	Unique, resolvable identifier
VC Issuance	JSON-LD schema issuers (e.g., Veramo, custom issuer backend)	Issue credentials for attributes
Wallet	Lace, Eternl, custom PWA	Stores DIDs and credentials
Verification	VC Verifier Toolkit	Validates and parses proofs

Typical Humanitarian VC Claims

Credential Claim	Description
Refugee Status	Verified by UNHCR or local agency
Smallholder Farmer	Issued by cooperative or NGO
Women-Led Enterprise	Confirmed via survey + validator attestation
School Attendance	Credential issued by school or Ministry

Integration Flow

```
[Trusted Issuer (NGO)]
    → Issues VC: "School Attendance Confirmed"
    → Delivered to [Beneficiary Wallet]
    → Shared with [Smart Contract or Verifier]
```

📌 **Pro Tip:** If sensitive information must be protected, use **off-chain selective disclosure patterns** (e.g., JSON-LD with zero-knowledge proofs or encrypted payloads). These can be verified client-side, but **are not currently verifiable on-chain in Cardano** without external ZK integrations.

💸 Section 2: Conditional Payments Integration

Why Conditional Payments?

Direct payments are powerful, but in development settings, **you need guarantees**:

- Only verified people get paid
- Funds are used for intended purposes
- Disbursement is traceable and auditable

Cardano offers multiple tools for programmable disbursement:

Payment Type	Tool	Use Case
One-off trigger	Aiken, Plutus	Disburse if VC hash is valid
Milestone-based	Marlowe	Funds released in stages upon proof
Escrow with timeout	Plutus	Funds revert if not claimed within X days

Example: Fertilizer Subsidy via VC Verification

1. **Credential**: “Farmer Registered” VC issued

2. **Condition**: VC hash + wallet signature
3. **Action**: 20 ADA sent from NGO wallet to farmer

Sample Smart Contract Logic (Aiken)

```
fn validator(vc_hash: ByteArray, submitted: ByteArray) -> Bool {
  vc_hash == submitted
}
```

🔄 Integrate with frontends using Lucid SDK or Mesh to:

- Connect wallet
- Present VC or proof
- Submit to contract

📌 **Pro Tip:** Add time-based windows using Marlowe for safer rollouts.

🌐 Section 3: Transparency in Aid Flows

The Problem

Donors, governments, and auditors want visibility:

- Who received aid?
- When and where?
- Was it used correctly?

Traditional reporting is **centralized and prone to fraud**. On-chain, every transfer is **verifiable, timestamped, and immutable**.

Integration Stack

Component	Tool / Standard	Role
Aid Disbursement	Marlowe, Aiken	Automate and log fund transfers
Identity	VCs + DIDs	Link to verified recipients
Metadata	CIP-68 NFTs / IPFS	Embed purpose, GPS, proof docs
Dashboard	GraphQL, Blockfrost	Query and visualize outcomes

Sample Transparency Token Metadata

```
json
{
  "name": "Payment Record - Fertilizer",
  "purpose": "Fertilizer Aid Disbursement",
  "recipient_did": "did:cardano:xyz123",
  "vc_hash": "0xabc...",
  "timestamp": "2025-07-14T08:00Z",
  "gps": "12.5243, -1.5932"
}
```

Practical Exercise

🧩 Build a Minimal, Testable VC Integration Flow for Aid Disbursement

This exercise walks you through issuing a **Verifiable Credential off-chain**, verifying it via **hash** on-chain, and linking the event to **on-chain metadata** for transparency.

🪪 Part 1: Issue and Hash a Verifiable Credential (Off-Chain)

1. **Define VC schema** in `vc-schema/farmer.json` using JSON-LD.

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1"],
  "type": ["VerifiableCredential", "FarmerRegistered"],
  "credentialSubject": {
    "id": "did:web:amina.farm",
    "name": "Amina Mbaye",
    "farmId": "AGRO-4582",
    "region": "Kolda",
    "cropType": "Millet"
  },
  "issuer": "did:web:ngo.org",
  "issuanceDate": "2025-06-01"
}
```

2. **Hash the credential content** using SHA-256:

```
cat vc-schema/farmer.json | sha256sum
```

Save this hash. It will be embedded in your validator.

🔒 Part 2: Implement a Conditional Aiken Contract

In `contracts/aid_disburse.aiken`, write a validator that checks if the hash presented in the transaction matches the one from your credential:

```
validator {
  fn main(submitted_hash: ByteArray) -> Bool {
    let expected_hash = 0x3b7c5d2fabcd1234deadbeef...
    submitted_hash == expected_hash
  }
}
```

Replace `expected_hash` with your credential hash from Part 1.

💸 Part 3: Create a Frontend Flow

In `frontend/payment.ts`, write a simple wallet interaction that:

- Connects to Nami or Eternl
- Builds a transaction that:
    - Sends ADA to the user if validator accepts the hash
    - Attaches the validator script

Use **MeshJS** or **Lucid Evolution**.

🌐 Part 4: Log the Event for Transparency

1. **Mint a CIP-68 NFT or token** that references the aid disbursement:

```
{
  "name": "Aid Transfer - Farmer",
  "purpose": "Training Subsidy",
  "recipient_did": "did:web:amina.farm",
  "vc_hash": "0x3b7c5d2f...",
  "timestamp": "2025-06-01T10:00:00Z",
  "region": "Kolda"
}
```

2. **Pin proof documents (if any)** like photos or certificates to IPFS.
3. **Query the transaction using Koios or GraphQL** to simulate dashboard integration.

🧪 Sample Outputs

Output File	Description
`vc-schema/farmer.json`	JSON-LD credential schema used off-chain
`contracts/aid_disburse.aiken`	Aiken validator that compares submitted hash

`frontend/payment.ts`	Wallet connection and submission logic using MeshJS
`metadata/tx.json`	CIP-68 metadata with aid details and recipient DID
`dashboard/query.graphql`	Query to fetch aid disbursement records by district
`demo/aid-flow.mp4`	Screen recording of full wallet → contract → explorer flow

🛠️ Best Practices: Integration Pathways for Humanitarian-Grade Identity, Payments, and Transparency

🪪 Identity Integration

- **Minimize Friction in Onboarding**
    - Use SMS-based flows or kiosk agents where mobile penetration is low.
    - Use trusted local NGOs or UNDP field partners as VC issuers to bootstrap trust.
    - Create fallback pathways for users without smartphones (e.g., paper QR + PIN-based recovery).
- **Credential Lifecycle Management**
    - Define expiration, renewal, and revocation strategy for every VC type.
    - Use lightweight wallet interfaces (PWA preferred) with offline credential caching.
    - Anchor VC hashes on-chain only when needed (e.g., for payout triggers or auditing).
- **Schema Interoperability**
    - Use **JSON-LD** and W3C DID-compliant formats to future-proof your VCs.

    - Register schemas on IPFS and version them clearly (`vc-schema/farmer-v1.json`).
    - Map each VC schema to SDG indicators (e.g., VC: “School Attendance” → SDG 4.1.1).

💸 Payments Architecture

- **Contract Selection per Use Case**
    - Use **Marlowe** for conditional aid payouts and multi-party financial flows.
    - Use **Aiken** for credential-gated access, on-chain logic triggers, or localized rules.
- **On/Off-Ramp Design**
    - In MVPs, simulate stablecoin payouts with ADA testnet tokens.
    - For pilots, pre-select off-ramp partners (e.g., mobile money aggregators, banks).
    - Use Oracle fallback strategies: if USD/ADA feed fails, cap payout with default.
- **Trigger Conditions Must Be Clear**
    - Examples:
        - VC exists + signed by X = release funds
        - Contract time window expired = auto-expire claim
        - Oracle confirms location = approve remittance
    - Always include “rejection” logic in contracts for failed validation.

🌐 Transparency & Auditability

- **Trace Every Transfer with Metadata**
    - Use **CIP-68** NFTs or native tokens with embedded metadata for:
        - Program ID

        - VC reference hash
        - Beneficiary pseudonym (or DID)
        - Location + timestamp

- **Build Public Dashboards (Optional for Pilots)**
    - Use **Koios or GraphQL** to surface real-time statistics:
        - Funds disbursed per region
        - Credential issuers by type
        - Transaction audit trails

- **Balance Privacy and Public Trust**
    - Use pseudonymous DIDs and hash VC payloads
    - Never expose names, phone numbers, or geolocation without obfuscation
    - Allow selective disclosure of VC fields via frontend (ZK-ready patterns)

🛠️ Deployment & MVP Readiness

- Use **Marlowe Playground** or **Aiken CLI** to simulate flow before mainnet deployment.
- Push all flows to GitHub with:
    - `contracts/aid_release.aiken`
    - `vc-schema/farmer_verified.json`
    - `demo/funds_claimed.mp4`
- Link testnet addresses and block explorer proofs in your `architecture.md`.

✅ Participant Submission Checklist

For Demo Day & Final Evaluation

Every team must submit a **complete and well-documented Proof-of-Concept (PoC)**. The checklist below outlines the required artifacts for final evaluation by EMURGO, UNDP, and mentors.

All files should be organized in a **public GitHub repository** and follow the recommended folder structure.

📁 Core Repository Structure

Folder	Description
`/contracts/`	Aiken, Marlowe, or Plutus contracts used in the PoC
`/vc-schema/`	JSON or JSON-LD Verifiable Credential schemas
`/frontend/`	Codebase for Lucid/Mesh-based dApp or UI demo
`/docs/`	Architecture diagram, VC templates, assumptions
`/demo/`	Screenshots or walkthrough video (mp4 or GIF)
`README.md`	Install, run, and testnet usage instructions
`architecture.md`	Full system overview: actors, flows, triggers

📦 Required Deliverables

✅ Item	Description
✅ Smart Contract File	One or more working contracts for VC-gated flow, token payout, traceability, or other SDG logic
✅ Verifiable Credential Schema(s)	At least one VC in JSON-LD format aligned to SDG use case (e.g. "Farmer Verified", "School Attendance")
✅ GitHub Repo	Public, documented repo with PoC and README
✅ Architecture Diagram	System flow with actors, credentials, smart contract logic, and interaction flows
✅ Demo Video or Screenshots	1–3 min walkthrough of contract interaction or UI demo (hosted in `/demo/` folder)
✅ GTM Plan (1-pager)	Pilot roadmap and distribution strategy
✅ Pitch Deck (PDF)	5-minute pitch deck (aligned with Module 2 structure)
✅ Case Study Mapping	Short write-up mapping lessons learned from at least one case study (Module 5)
✅ Mentor Tracker	Log of mentor meetings, action items, feedback, and implemented changes

📝 Optional (Bonus)

🟡 Bonus Item	Description
🟡 ZK-based VC flow	If applicable, use anoncreds or selective disclosure
🟡 Real testnet transactions	Links to testnet contracts or NFT mints
🟡 Dashboard UI	Koios/GraphQL-based dashboard for NGO or verifier use
🟡 Local language UI	UI localized for specific region or country
🟡 Interoperable VC schema	Registered VC schema compatible with W3C standards and IPFS-hosted

🗓️ Accelerator Program Timeline (May 2025 - Feb 2026)

Module / Event	May 25	Jun 25	Jul 25	Aug 25	Sep 25	Oct 25	Nov 25	Dec 25	Jan 26	Feb 26
Cohort 1	**Start**									
Module 1: Blockchain Fundamentals										
Module 2: Mentorship, Pitch & Go-to-Market										
Module 3: Smart Contracts (Plutus/Marlowe/Aiken)										
Module 4: SDKs, APIs & AI Tools										
Module 5: Case Studies (Blockchain + SDG in the Field)										
Module 6: Implementation Design & Architecture										
Module 7: Integration Pathways (ID, Payments, Aid)										
RPPS - PoC Sprint (Cohort 1)										
Final Demos + Evaluation (Cohort 1)					**End**					
Cohort 2						**Start**				
Module 1: Blockchain Fundamentals										
Module 2: Mentorship, Pitch & Go-to-Market										
Module 3: Smart Contracts (Plutus/Marlowe/Aiken)										
Module 4: SDKs, APIs & AI Tools										
Module 5: Case Studies (Blockchain + SDG in the Field)										
Module 6: Implementation Design & Architecture										
Module 7: Integration Pathways (ID, Payments, Aid)										
RPPS - PoC Sprint (Cohort 2)										
Final Demos + Evaluation (Cohort 2)										**End**

Sample 3 weeks for Cohort 1

Week	Live Session Title	Day	Time (UTC+4)	Presenter	Position
1	Welcome & Orientation + Blockchain for SDG Impact	Monday	10:00–11:00	Ahmed M. Amer	CEO, EMURGO Labs
	Immutable Ledgers & Distributed Trust (Theory + Use Cases)	Wednesday	14:00–15:30	Ahmed Hadded	Product Manager, EMURGO Labs
2	Public vs. Private Blockchains + UNDP Use Cases	Monday	10:00–11:00	Yosuke Yoshida	CEO, EMURGO Africa
	How Cardano is Different – eUTXO + Sustainability	Wednesday	14:00–15:30	Tasos Valinos	CTO, EMURGO Labs
3	Native Assets + Mapping SDG Challenges to Blockchain Tools	Monday	10:00–11:30	Tasos Valinos	CTO, EMURGO Labs
	Workshop: Drafting Your “Why Blockchain, Why Cardano” 1-Pager	Wednesday	14:00–15:00	Ahmed Hadded	Product Manager, EMURGO Labs

*Mentorship, pitch and go-to market will take place on a weekly basis after mentors are matched with founders 2-4 weeks into the program

**To account for different time zones, some sessions will be live, while others will be pre-recorded

Community Mentors and Coaches

Name	Position	Organization
Zushan Hashmi	Founder	Tokeo
Yoram Ben Zvi	Strategy and BD	Connectality, ELKsconnect.com, Andamio
Chetan Padindala	Strategy Consultant and Advisor	-
Nathaniel Minton	CEO	Flux Point Studios, Inc.
Jon Kravetz	CEO	CSWAP Dex
Jo Allum	Founder Director	Venture Centre
Apex	CEO	Titans
Alex Pestchanker	Co Founder, CTO	Token Allies
Ivica Zafirovski	COO	Farmroll.io
Alex Maaza	Sustainability Lead	Cardano Foundation

Other Potential Mentors and Coaches

Name	Position	Organization
Ismael Belkhayat	CEO	Chari
Djamel Mohand	Former COO	Foodics
George Payne	Head of Accelerator	Adaverse
Vincent Li	Founding Partner	Adaverse
Driss Temsamani	Head of Digital	Citi
An Luu	Director	Ginar Solution
Ilan Benhalim	Co-Founder & Partner	VeePee
Basmah Alsinaidi	Managing Partner	Impact46
Waleed A. Alballaa	Investor	Sukna Ventures
Serena Sebastiani	Director - Financial Services Advisory	PwC
Sam Corcoran	Co-Founder	cander
Dave Parker	Managing Partner	DKParker LLC
Pavel Kaminsky	Advisory Board Member	Merchant Payments Ecosystem
Abrar Khan	CEO	Rockville Technologies