SDG BLOCKCHAIN ACCELERATOR

Technical Architecture Document

Atlas Ledger

1. Project Information

- **Project Name:** Atlas Ledger
- **Challenge & UNDP Office:** Reforestation projects with UNDP Burkina Faso
- **Document Version:** V 1.0

2. Overview

Atlas Ledger is a blockchain-enabled climate impact platform that connects reforestation and climate-positive projects with eco-conscious donors, companies, and contributors. Built on the Cardano blockchain, it ensures that every contribution is transparent, accountable, and tied to measurable outcomes.

Projects receive milestone-based funding through smart contracts, with disbursements unlocked only when validators confirm progress against agreed targets. AI-powered assessments and dashboards allow projects to demonstrate environmental and social impact, while donors can track in real time how their contributions support reforestation, ecosystem restoration, and community resilience.

By making climate donations verifiable, accessible, and impactful, Atlas Ledger helps scale reforestation efforts, channel resources to local communities, and deliver tangible contributions to global sustainability goals.

Donor Incentive:

- **Trust & accountability**: Donors see exactly how their funds are used, with releases tied to verified progress
- **Risk mitigation:** Funds are only released when measurable milestones are achieved, protecting against project failures
- **Impact visibility:** Real-time tracking of reforestation progress with verifiable results
- **Smart donation splits:** Immediate gratification through instant partial payouts combined with milestone-locked long-term impact

- Transparency: On-chain verification provides immutable proof of fund usage and project outcomes
- **Community impact:** Collective funding power enables larger, more impactful reforestation initiatives

Project Benefits & Motivation:

- **Structured funding:** Predictable cash flow tied to achieving specific, measurable goals
- **Accountability framework:** Clear deliverables and timelines prevent scope creep and maintain focus
- **Credibility building:** Completing milestones builds trust and attracts additional funding
- **Progress incentives:** Payment structure rewards consistent progress rather than upfront commitments
- **Scalability:** Successful milestone completion enables access to larger funding pools
- **Professional validation:** Third-party milestone verification enhances project credibility

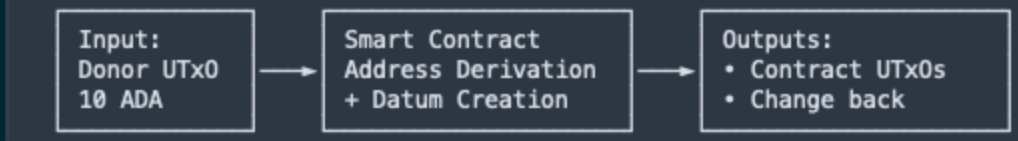
System Advantages:

Users can donate to climate initiatives with automated milestone-based payouts, ensuring funds are only released when project goals are achieved. The system solves the trust problem in climate financing by providing on-chain verification of project progress, automated transparent fund distribution, and aligned incentives that benefit both donors seeking impact and projects needing sustainable funding structures.

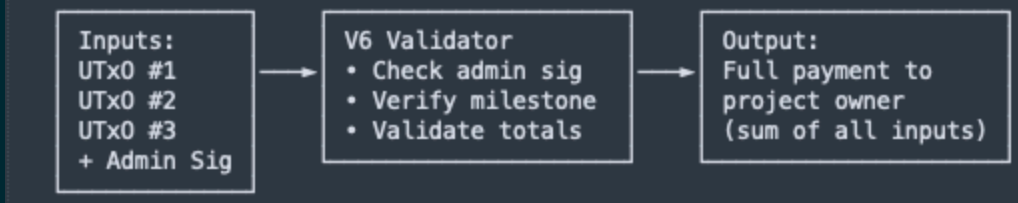
3. System Architecture Diagram

TRANSACTION FLOWS:

2. DONATION TRANSACTION (Donor → Contract)

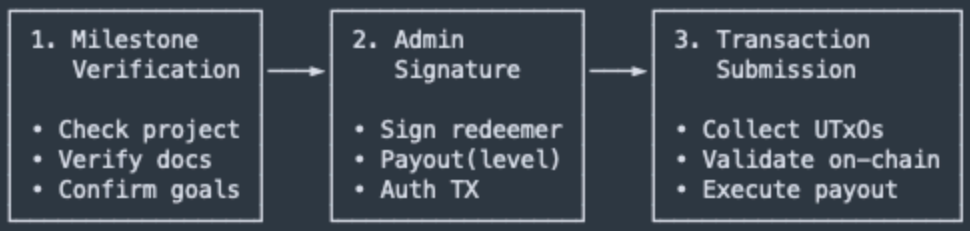


3. ADMIN PAYOUT TRANSACTION (Contract → Beneficiary)



ORACLE TRIGGER MECHANISM:

Admin Oracle Decision Process:



KEY SECURITY VALIDATIONS:

ON-CHAIN VALIDATOR CHECKS (Executed for every payout transaction)

```
1. has_admin_signature(tx, datum.admin_oracle_vkh)
2. redeemer.reached_up_to >= datum.milestone_ix
3. sum_script_inputs(tx) <= paid_total_to_beneficiary(tx)
4. no_change_back_to_script(tx.outputs, script_addr)
5. datum.beneficiary != script_address
6. multi_input_total_validation(all_utxos)
```

4. Blockchain Design

- **Smart Contract:**

  Validator script:

  - V6 Admin Validator Script (oracle_payout_simple_v6_admin)
  - Purpose: Milestone-based payout management with admin signature authorization and multi-input UTxO processing
  - Access Control: Admin oracle signature verification with milestone progression gating

  Complete Datum Structure (metadata, project ID, beneficiary info):

```
pub type DatumV6 {
  project_id: ByteArray,
// Project identifier (hex-encoded UUID for routing)
  beneficiary: Address,
// Project owner's payout address (metadata)
  admin_oracle_vkh: VerificationKeyHash,
// Admin signature authority (access control)
  milestone_ix: Int,
// Milestone index (0=immediate, 1,2,3...=locked milestones)
}

// Example datum instances:
// Milestone 0 (immediate): { project_id: #"abc123", beneficiary:
addr_test1..., admin_vkh: #"def456", milestone_ix: 0 }
// Milestone 1 (locked):    { project_id: #"abc123", beneficiary:
addr_test1..., admin_vkh: #"def456", milestone_ix: 1 }
// Milestone 2 (locked):    { project_id: #"abc123", beneficiary:
addr_test1..., admin_vkh: #"def456", milestone_ix: 2 }
```

Redeemer Structure (actions: payout with milestone validation):

```
pub type Redeemer {
  Payout(Int)
// reached_up_to: milestone completion level (action: validate and
release funds)
}

// Redeemer usage examples:
// Payout(0) - Release milestone 0 funds (immediate payout)
// Payout(1) - Release milestone 1 funds (first locked milestone
achieved)
// Payout(2) - Release milestones 0,1,2 funds (batch payout for
multiple milestones)
// Payout(3) - Release all milestones 0-3 (project completion payout)
```

Validator Script Logic:

```
// ---------- Helpers ----------

n paid_total_to(outs: List<Output>, addr: Address) -> Int {
  list.foldl(
    outs,
    0,
    fn(o: Output, acc: Int) -> Int {
      let add = if o.address == addr { lovelace_of(o.value) } else { 0
}
      acc + add
    }
  )
}

fn sum_script_inputs(inputs: List<Input>, script_addr: Address) -> Int
{
  list.foldl(
    inputs,
    0,
    fn(i: Input, acc: Int) -> Int {
      let add = if i.output.address == script_addr {
lovelace_of(i.output.value) } else { 0 }
      acc + add
    }
```

```
  )
}

fn no_change_back_to_script(outs: List<Output>, script_addr: Address)
-> Bool {
  let has_change = list.any(outs, fn(o: Output) -> Bool { o.address ==
script_addr })
  !has_change
}

fn has_admin_signature(tx: Transaction, admin_vkh:
VerificationKeyHash) -> Bool {
  list.has(tx.extra_signatories, admin_vkh)
}

// ---------- Validator ----------
validator oracle_payout_contract_simple_v6_admin {
  spend(
    d_opt: Option<DatumV6>,           // Aiken passes Option<DatumV6>
    r: Redeemer,
    own_ref: OutputReference,
    ctx: Transaction,
  ) {
    when d_opt is {
      None -> False
      Some(d) -> {
        when list.find(ctx.inputs, fn(i: Input) { i.output_reference
== own_ref }) is {
          None -> False
          Some(own_in) -> {
            // Redeemer gating by milestone
            let Payout(reached_up_to) = r
            let reached_ok   = reached_up_to >= d.milestone_ix

            let script_addr  = own_in.output.address
            let admin_ok     = has_admin_signature(ctx,
d.admin_oracle_vkh)

            // --- New: totals over the whole transaction ---
            let total_from_script = sum_script_inputs(ctx.inputs,
script_addr)
            let paid_total        = paid_total_to(ctx.outputs,
d.beneficiary)
```

```
            let pay_ok            = paid_total >= total_from_script

            // Keep forbidding any output back to the same script
address
            let change_ok   = no_change_back_to_script(ctx.outputs,
script_addr)

            // (Optional) guard against beneficiary == script
            let bene_not_script = d.beneficiary != script_addr

            reached_ok && admin_ok && pay_ok && change_ok &&
bene_not_script
          }
        }
      }
    }
  }

  // Disallow use in other contexts (withdrawal/mint/cert/etc.)
  else(_) { fail }
```

- **UTxO Model Usage:**
  - Input Handling: Collects one or multiple script UTxOs for batch processing
  - Output Requirements: Full payment to beneficiary, no change back to script
  - Multi-input Support: Validates total_from_script ≤ total_paid_to_beneficiary
  - Split Payments: Supports multiple outputs to same beneficiary address

Typical Transaction Flows:

- ➔ Deployment: Admin → Contract (creates milestone UTxOs with specific datum)
- ➔ Donation: Donor → Contract (funds existing milestone UTxOs)
- ➔ Payout: Admin signature + UTxO(s) → Beneficiary (milestone completion)
- ➔ Smart Donation: Automatic split between immediate payout and contract deposit

- **Token Management:**
    - Native ADA handling (no custom tokens)
    - UTxO-based fund management with datum metadata
    - Milestone tracking via datum milestone_ix field
    - Project identification through project_id in datum

- **Security Considerations:**
    - Admin Signature Verification: has_admin_signature(tx, admin_oracle_vkh)
    - Payment Validation: paid_total ≥ sum_script_inputs
    - Change Prevention: no_change_back_to_script(outputs, script_addr)
    - Milestone Gating: reached_up_to ≥ milestone_ix
    - Beneficiary Validation: beneficiary != script_addr (prevents impossible payouts)
    - Double-satisfaction Prevention: Total-sum validation across all script inputs

5. Data Flow & Transaction Lifecycle

- **Donation Transaction Lifecycle**:
    1. User selects project and donation amount in frontend
    2. Frontend calls backend API with donation parameters
    3. Backend calculates smart donation split (immediate vs. contract)
    4. Backend builds transaction using Lucid Evolution
    5. Transaction includes proper datum construction for milestone UTxOs
    6. Transaction submitted to Cardano Preprod testnet
    7. Backend monitors transaction confirmation via Blockfrost
    8. Database updated with transaction record and enrichment data
    9. Frontend displays updated project status and user dashboard

- **Payout Transaction Lifecycle:**
    1. Admin initiates payout from admin panel
    2. Backend queries available UTxOs for specific milestone
    3. Admin signature verification and milestone validation
    4. Multi-input transaction built to collect all relevant UTxOs
    5. Full payment sent to project beneficiary address
    6. Transaction submitted with admin signature authorization
    7. Blockchain validation via V6 smart contract logic
    8. Database synchronized with new transaction state
    9. Project dashboard updated with payout confirmation

6. Off-chain Components

- **Backend Services (NestJS):**
    - OracleContractsV6AdminService: Core contract interaction service
    - V6TransactionService: Transaction building and submission
    - V6BlockchainSyncService: Real-time blockchain synchronization
    - V6DatumService: Datum construction and validation
    - V6AddressService: Address derivation and validation
    - DonorContributionService: Donation tracking and management

- **Integration with Cardano:**
    - Blockfrost API: Blockchain queries with intelligent rate limiting
    - Lucid Evolution: Transaction building and signing
    - Preprod Testnet: Live deployment and testing environment
    - UTxO Monitoring: Real-time contract state synchronization

- **Dashboard & Reporting:**
    - Next.js Frontend: Real-time project and contract management
    - Admin Panel: Contract deployment and payout management
    - User Dashboard: Donation tracking and project progress
    - Project Analytics: Milestone completion and fund distribution metrics

- **Database Integration:**
    - PostgreSQL with TypeORM: Transaction and project data storage
    - Transaction Enrichment: Enhanced transaction metadata with user context
    - Real-time Updates: Blockchain sync with database consistency

7. Sandbox/Testnet Results

Preprod Test Results:

Transaction ID	Type	Status	CPU	Memory	Notes
83730d50102cb51a7b8a9cb2b0cd9a5e94cfdaf2992a303fc874bf83b0757949	Milestone 1 (index) pay out - 1 donor	Success	55M	127k	Validator passed all checks Admin signature and milestone index verified successfully
077ae2d79d1de8113400c619a8e16c351c9e47c24a84c5a4bba63a00c6e7d537	Milestone 2 (index) payout - 2 donors	Success	avg for every donor 67M	avg for every donor 155k	Validator passed all checks Admin signature and milestone index verified successfully
a79c2f27e3646d935741f8f80311fa213d1a5debfd9e62838359864386965926	Milestone 3 (index) payout - 3 donors	Success	avg for every donor 81M	avg for every donor 186k	Validator passed all checks Admin signature and milestone index verified successfully

Contract Addresses (HEX):

- 106c8c2497ab7173c16a389f9948aa798a22d2d476e1785b0706dfa7af534b62ad660488b3c51fd524faeed907f5612f6682b37f7ae48393d8 (used for the table)
- 108c22570eeb82089ada1539e4709841e3d3b740fde3577ad700080e90b19bd26b60b052cd99d01cac3cdbf3ed4531d6483d019c6e49998ffd

8. Tools and Environments Used

- **Smart Contract Development:**
    - Aiken CLI v1.1.19+e525483 (aiken build, aiken check, aiken test)
    - Plutus V3 compilation target

- **Backend Development:**
    - Lucid Evolution for transaction building
    - Blockfrost API for blockchain queries

- **Testing Infrastructure:**
    - Aiken built-in testing framework
    - Cardano Preprod testnet integration

9. Remaining Considerations / Next Steps

- **Issue**: Many-Donors Scaling

  Problem (in the validator):

  The current V6 validator executes once per donor UTxO and, on each run, recomputes:

  - the sum of all script inputs in ctx.inputs, and
  - the sum paid to the beneficiary in ctx.outputs.

  This makes every execution scan the whole transaction. As the number of donor inputs grows, CPU and memory per input increase, and total ex-units grow even faster. This is intrinsic to the “pay the total once” rule implemented on-chain.

  Consequence:

  Payout transactions with many donors can approach per-transaction CPU limits even when individual code paths are simple.

  Proposal to solve it:

  - Structured on-chain flow

    Add a consolidation path (new redeemer branch) that merges many donor UTxOs into one script UTxO for the same milestone; the regular Payout then spends just one input.

  - Off-chain mitigation

    Chunk payouts into batches (e.g., ≤ 20 donors/tx) and EvaluateTx each built transaction; shrink the batch if steps approach your safety threshold.

- **Performance Optimizations:**
  - Monitor real-world gas costs and execution efficiency on Preprod
  - Optimize batch transaction handling for large milestone collections
  - Implement UTxO consolidation strategies for better efficiency
  - Cache optimization for frequent blockchain queries

- **Security Enhancements:**
    - Conduct comprehensive security audit of V6 validator logic
    - Implement additional validation for edge cases in multi-input scenarios
    - Enhance admin signature verification with multi-sig support (future V7)
    - Strengthen input validation across all contract entry points

- **Mainnet Preparation:**
    - Stress testing with higher transaction volumes on Preprod
    - Documentation completion for production deployment procedures
    - Monitoring and alerting system setup for mainnet operations
    - Backup and disaster recovery procedures for contract management