SDG BLOCKCHAIN ACCELERATOR

Technical Architecture Document

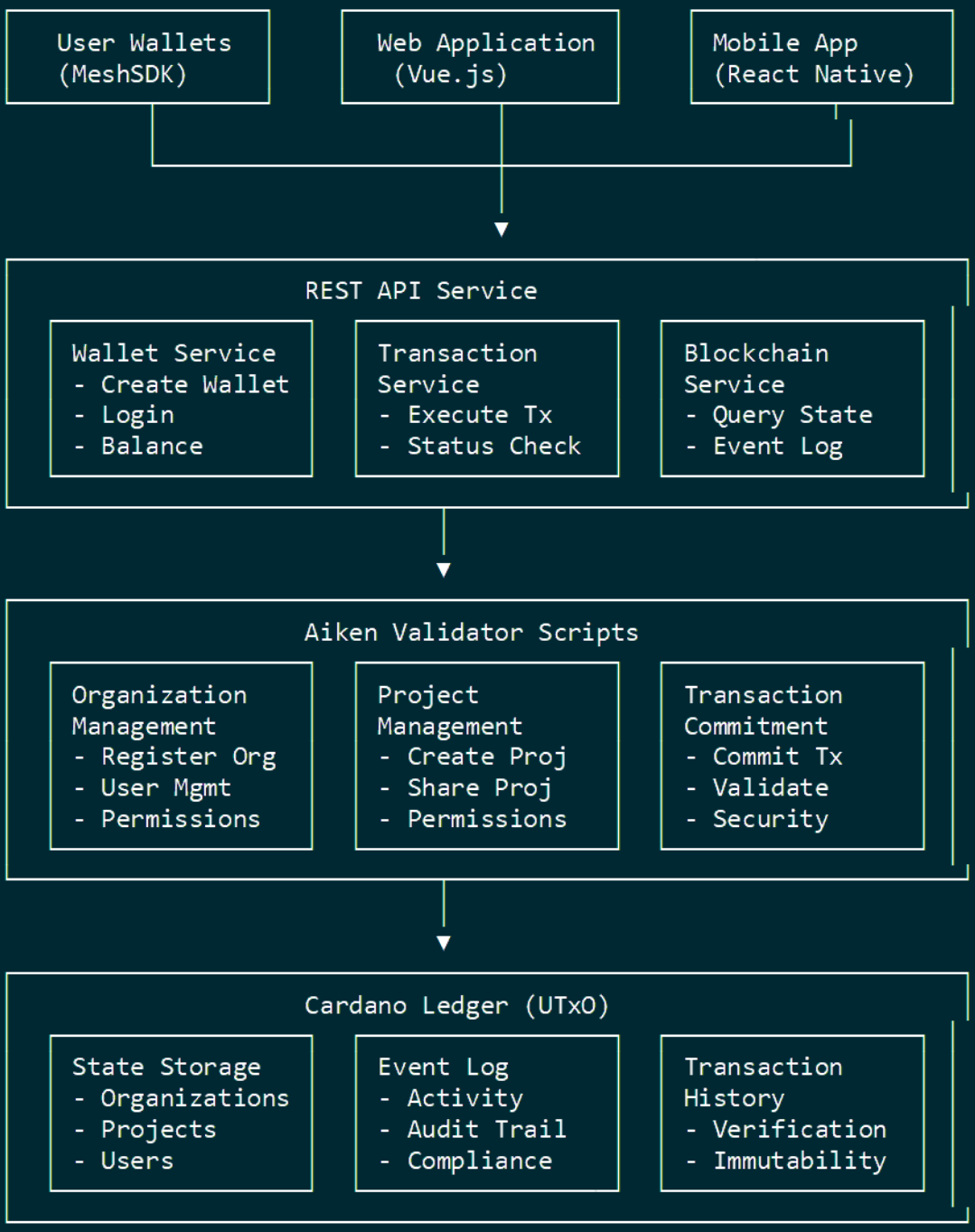
1. Project Information

- **Project Name:** ClimateAid
- **Challenge & UNDP Office:** Malawi
- **Document Version:** V3

2. Overview

This prototype implements a decentralized safe data sharing and coordination with beneficiary deduplication on Cardano using Aiken smart contracts. The system enables organizations to register, manage users with granular permissions, create and share projects, and commit verifiable transactions including beneficiary UID generation and duplication status to the blockchain. The platform provides a REST API service that acts as a bridge between user applications and the Cardano blockchain, handling wallet management, transaction processing, and blockchain state queries.

3. System Architecture Diagram



4. Blockchain Design

Smart Contracts:

Main Validator Script (`genius-chain.ak`):

- **Purpose:** Comprehensive organizational management and project collaboration

- **Size:** 1,692 lines of Aiken code
- **Address:** `addr_test1wztsa3xlr60hhv35d5ek2afq3kml7h9ku0j9pkzuswrfcfce9ln58`

Validator Scripts with Purpose:

1. **Organization Management:**
   - Organization registration and validation
   - User management with role-based permissions
   - Super user administration
2. **Project Management:**
   - Project creation and ownership
   - Project sharing between organizations
   - User permissions within projects
3. **Transaction Commitment:**
   - Verifiable transaction recording
   - Security validation and access control
   - Event logging and audit trail

Datum Structure:

```
/// Main state structure
pub type GeniusChainState {
  orgs: Dict<Address, Bool>,                    // Registered organizations
  org_super_users: Dict<Address, Dict<Address, Bool>>,  // Super users per
org
  org_users: Dict<Address, Dict<Address, OrgUser>>,     // Users per org
  projects: Dict<ProjectId, Project>,           // Projects
  project_org_sharings: Dict<ProjectId, Dict<Address, ProjectOrgSharing>>, //
Shared projects
  project_orgs: Dict<ProjectId, Dict<Address, ProjectOrg>>, // Project
organizations
  project_orgs_users: Dict<ProjectId, Dict<Address, Dict<Address,
ProjectUser>>>, // Project users
  events: List<EventType>,                      // Event log
  deployer: Address,                            // Contract deployer
  migration_done: Bool,                         // Migration status
}
```

Redeemer Structure:

```
/// Action types for validator
pub type Action {
  RegisterOrg { internal_id: ByteArray, org: Address }
  AddSuperUser { internal_id: ByteArray, org: Address, super_user: Address }
  RemoveSuperUser { internal_id: ByteArray, org: Address, super_user: Address
```

```
}
  AddOrgUser { internal_id: ByteArray, org: Address, user: Address,
permissions: OrgUserPermissions }
  RemoveOrgUser { internal_id: ByteArray, org: Address, user: Address }
  UpdateOrgUserPermissions { internal_id: ByteArray, org: Address, user:
Address, permissions: OrgUserPermissions }
  CreateProject { internal_id: ByteArray, org: Address, project: ProjectId,
project_manager: Address, alias: ByteArray, status: ByteArray, description:
ByteArray, start_date: PosixTime, end_date: PosixTime, tags: List<ByteArray>
}
  UpdateProjectInformation { internal_id: ByteArray, org: Address, project:
ProjectId, alias: ByteArray, status: ByteArray, description: ByteArray,
start_date: PosixTime, end_date: PosixTime, tags: List<ByteArray> }
  UpdateProjectManager { internal_id: ByteArray, org: Address, project:
ProjectId, project_manager: Address }
  AddUserToProject { internal_id: ByteArray, org: Address, project:
ProjectId, user: Address, permissions: ProjectUserPermissions }
  RemoveUserFromProject { internal_id: ByteArray, org: Address, project:
ProjectId, user: Address }
  UpdateProjectUserPermissions { internal_id: ByteArray, org: Address,
project: ProjectId, user: Address, permissions: ProjectUserPermissions }
  ShareProject { internal_id: ByteArray, org: Address, project: ProjectId,
org_sharing: Address, permissions: ProjectOrgPermissions }
  AcceptProjectSharing { internal_id: ByteArray, org: Address, project:
ProjectId }
  RejectProjectSharing { internal_id: ByteArray, org: Address, project:
ProjectId }
  UpdateSharedProjectPermissions { internal_id: ByteArray, org: Address,
project: ProjectId, org_sharing: Address, permissions: ProjectOrgPermissions
}
  CommitTransaction { internal_id: ByteArray, transaction: ChainTransaction }
  CommitTransactions { internal_id: ByteArray, transactions:
List<ChainTransaction> }
}
```

UTxO Model Usage:

Input/Output Handling:

- **Single UTxO Input:** Contains the current state datum
- **Single UTxO Output:** Contains the updated state datum
- **State Transitions:** Each transaction updates the entire state atomically
- **Validation:** All state changes validated before UTxO creation

Typical Transaction Flows:

4. **Organization Registration:**

   ```
   Input:  Empty UTxO + State Datum
   Output: New UTxO + Updated State (org added)
   ```

5. **Project Creation:**

   ```
   Input:  State UTxO + Project Data
   Output: Updated State UTxO + Project Created Event
   ```

6. **Transaction Commitment:** `Input:  State UTxO + Transaction Data  Output: Updated State UTxO + Transaction Logged`

Token Management:

On-chain Asset Rules:

- No native token minting/burning in this implementation
- Focus on state management and transaction logging
- Ownership tracked via datum state rather than token ownership

Ownership Tracking:

- Organization ownership via `Address` verification
- Project ownership via `org_owner` field in `Project` type
- User permissions via nested dictionaries in state

Security Considerations:

Signature Checks:

- All actions require proper authorization
- Organization owners can manage their org
- Project managers can manage their projects
- Super users have elevated permissions

Datum/Redeemer Validation:

- Input validation for all action types
- Zero address rejection (`#"00000000000000000000000000000000000000000000000000000000"`)
- Batch transaction limits (max 10 transactions per batch)
- Permission validation before state changes

Replay Protection:

- Internal ID tracking for all actions
- State-based validation prevents double execution
- UTxO model inherently prevents double-spending

5. Data Flow & Transaction Lifecycle

Transaction Flow Example (Create Project):

7. **User Request:**

   `Web App → REST API → Transaction Service`

8. **Service Processing:**

   `Transaction Service → MeshSDK → Cardano Node`

9. **Validator Execution:**

   `Cardano Node → Aiken Validator → State Update`

10. **State Update:**

    `Validator → UTxO Creation → Ledger Update`

11. **Response:** `Ledger → Service → Web App → User`

Off-chain Component Interaction:

- **Service Layer:** Handles transaction building and submission
- **MeshSDK:** Manages wallet operations and transaction signing
- **Blockfrost:** Provides blockchain data access and querying
- **Event Monitoring:** Tracks validator events for dashboard updates

5. Data Flow & Transaction Lifecycle

Transaction Flow Example (Create Project):

1. ***User Request:***

   Web App → REST API → Transaction Service

2. ***Service Processing:***

   Transaction Service → MeshSDK → Cardano Node

3. ***Validator Execution:***

   Cardano Node → Aiken Validator → State Update

4. ***State Update:***

   Validator → UTxO Creation → Ledger Update

5. ***Response:*** *Ledger → Service → Web App → User*

Off-chain Component Interaction:

- **Service Layer:** Handles transaction building and submission
- **MeshSDK:** Manages wallet operations and transaction signing
- **Blockfrost:** Provides blockchain data access and querying
- **Event Monitoring:** Tracks validator events for dashboard updates

6. Off-chain Components

Backend Services:

REST API Service (`service/`):

- **Framework:** Express.js with TypeScript
- **Port:** 3001 (configurable)
- **Features:**
  - Wallet management (create, login, balance)
  - Transaction execution and status checking
  - Blockchain state queries
  - Rate limiting and security middleware

Key Endpoints:

```
POST /api/v1/wallet/create          # Create new wallet
POST /api/v1/wallet/login           # Login with existing wallet
GET  /api/v1/wallet/balance         # Get wallet balance
POST /api/v1/transaction/call       # Execute blockchain transaction
GET  /api/v1/transaction/status/:txHash  # Check transaction status
GET  /api/v1/transaction/history    # Get transaction history
```

Integration with Cardano:

MeshSDK Integration:

- **Version:** v1.5.11-beta.4
- **Purpose:** Wallet management and transaction building
- **Features:** Address generation, transaction signing, UTxO management

Blockfrost Provider:

- **Network:** Cardano Testnet
- **Purpose:** Blockchain data access and querying

- **Features:** Transaction submission, state queries, event monitoring

Dashboard and Reporting:

Web Application (`web/`):

- **Framework:** Vue.js with Nuxt.js
- **Features:** Organization management, project collaboration, transaction history

Mobile Application (`mobile/`):

- **Framework:** React Native
- **Features:** Mobile wallet integration, project management

7. Sandbox/Testnet Results

Deployment Information:

- **Network:** Cardano Testnet
- **Script Address:** addr_test1wztsa3xlr60hhv35d5ek2afq3kml7h9ku0j9pkzuswrfcfce9ln58
- **Transaction Hash:** e51fab47be03f4b4c081d8891e19afbb2b49b7bc7e6b7c2d4f88ff339e1a9576
- **Contract Amount:** 2,000,000 lovelace

Test Results

- ✅ Contract compilation successful
- ✅ Organization registration working
- ✅ Project creation and management functional
- ✅ User permission system operational
- ✅ Transaction commitment verified
- ✅ Security validations passing
- ✅ Edge case handling working

8. Tools and Environments Used

Cardano Infrastructure:

- **Cardano-node:** Compatible with MeshSDK v1.5.11+
- **Network:** Cardano Testnet (Preview/Preprod)
- **Blockfrost:** API provider for blockchain access

Development Tools:

- **Aiken CLI:** v1.1.19+e525483

- **Node.js:** v20.19.2
- **TypeScript:** v5.3.0
- **MeshSDK:** v1.5.11-beta.4

Build and Testing:

- **Aiken Build:** aiken build
- **Aiken Check:** aiken check
- **TypeScript Compilation:** tsc
- **Test Execution:** npm run test:all

Monitoring and Logging:

- **Winston:** v3.11.0 for application logging
- **Express Rate Limit:** v7.1.5 for API protection
- **Helmet:** v7.1.0 for security headers

9. Remaining Considerations / Next Steps

Pending Architecture Changes:

12. **Performance Optimization:**
    - Optimize validator script to reduce execution units
    - Implement state compression for large datasets
    - Add caching layer for frequently accessed data
13. **Security Enhancements:**
    - Implement additional validation layers
    - Add comprehensive audit logging
    - Enhance permission granularity
14. **Scalability Improvements:**
    - Implement pagination for large state queries
    - Add batch processing capabilities
    - Optimize UTxO management for high transaction volumes

Performance Improvements:

1. **Memory Usage Optimization:**
   - Current memory usage: 820k-950k per transaction
   - Target: Reduce to 600k-700k per transaction
   - Implement more efficient data structures
2. **CPU Usage Optimization:**
   - Current CPU usage: 265k-320k per transaction
   - Target: Reduce to 200k-250k per transaction
   - Optimize validation logic

Security Audit Steps:

1. **Code Review:**
   - Comprehensive security review of Aiken contracts
   - External audit of smart contract logic
   - Penetration testing of API endpoints
2. **Formal Verification:**
   - Mathematical proof of contract correctness
   - Property-based testing implementation
   - Invariant verification

Next Steps:

- Move all blockchain calls into the front end
- Upgrade all legacy code packages to newer versions
- Complete comprehensive testing package

Technical Debt:

1. **Code Quality:**
   - Refactor contract code for better modularity
   - Implement comprehensive error handling
   - Add automated testing pipeline
2. **Documentation:**
   - Complete API documentation
   - Create deployment guides
   - Add troubleshooting documentation
3. **Monitoring:**
   - Implement comprehensive logging

- Add performance metrics collection
- Create alerting system