1. Certificate issuing tool [1]

1.1 Function Description

This system is a web-based tool that utilizes the Cardano blockchain to issue and manage verifiable certificates for disaster relief volunteers. The certificates include detailed information such as volunteers' skills, roles, and authorized regions, enabling reliable and transparent qualification verification in disaster response situations. The main functions of the system are listed below.

1.1.1 Certificate Registration Function [1-a]

1.1.1.1 Creation and Issuance of Qualification Certificates

- Users enter volunteer qualification information (e.g., name, role, skills, available activity regions, etc.) through the frontend interface to create certificate data structured in JSON format.
- The certificates comply with the "Verifiable Credential" format, ensuring a reliable structure. This certificate data is recorded on the blockchain as verification information, allowing others to reference and verify it in the future.

1.1.1.2 Transaction Generation and Signing

- A transaction is generated using the CardanoWasm (Cardano Web Assembly) library, and credential data is set as auxiliary data. The credential data is serialized to be recorded on the blockchain and signed with the user's private key.
- The signature prevents tampering with the transaction and ensures the reliability of the credential. Through this process, the user's credential data is uniquely included in the transaction.

1.1.1.3 Transaction Submission and Recording on the Blockchain

- The signed transaction is submitted via the backend server and recorded on the Cardano blockchain. The backend server manages the transaction submission using the Blockfrost API, and upon successful recording, a transaction ID is returned.
- Using this transaction ID, it is possible to verify that the submitted credential data has been recorded in the block.

1.1.2 Function to Retrieve Transaction Information [1-b]

1.1.2.1 Retrieving Block Information and Verification via Merkle Tree

- After a transaction is included in a block, a Merkle tree is constructed based on the block information and transaction ID to verify the credential data.
- The hash of each transaction is stored in the Merkle tree, ensuring the validity and integrity of the transaction. This procedure proves that a specific credential is securely recorded in the correct block on the Cardano blockchain.

1.1.2.2 Downloading Credential Data

- Ultimately, the credential data, including the Merkle tree path and block information, can be downloaded in JSON format. This allows users to refer to and verify the credential data later, while also enabling third parties to verify the integrity of the credential data even in an offline environment.

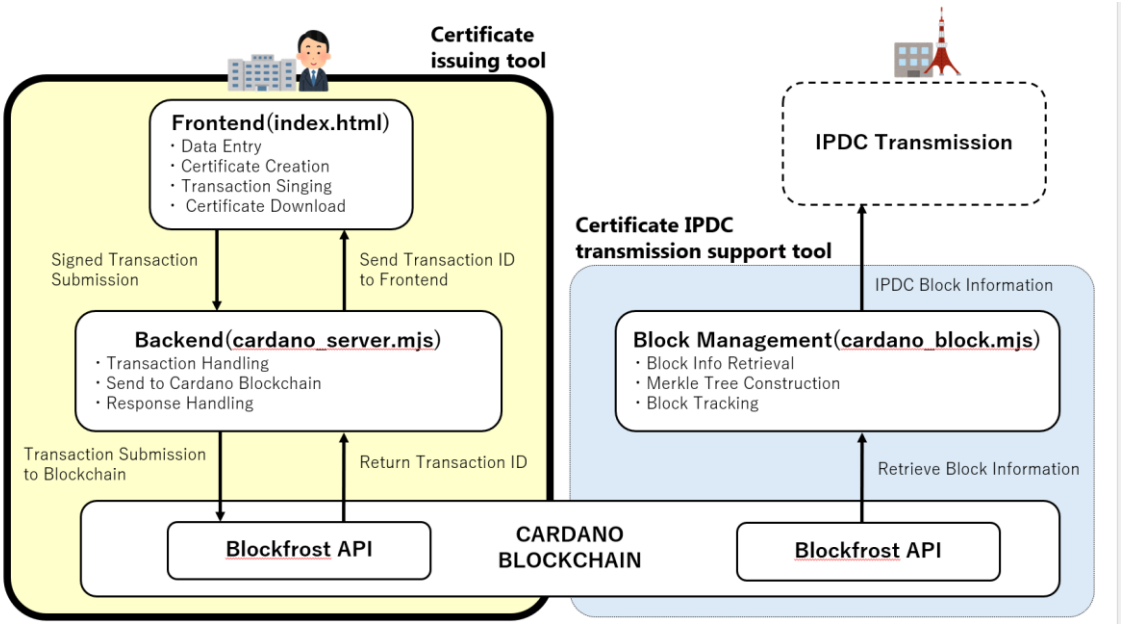
Through this system, volunteer credentials can be quickly and reliably verified in disaster relief situations. Additionally, blockchain technology ensures protection against credential forgery, enhancing the credibility of volunteer qualifications. Disaster response agencies can confirm volunteer credentials with confidence, even without scanning cards or online access, thereby improving the efficiency and safety of disaster response activities.

1.2 System Architecture

The certificate issuance tool integrates with the frontend, backend, and external APIs (Blockfrost API) to comprehensively handle the process from user input to certificate generation, signing, transaction submission, and retrieval of verification data. This architecture ensures the generation of credential data and guarantees its reliability on the blockchain.

1.2.1 System Architecture Diagram

This system consists of a blockchain information update and management module (cardano_block.mjs) and operates in coordination with the Cardano blockchain and the Blockfrost API. It periodically retrieves IPDC block information from the Cardano blockchain and provides the data to the broadcasting station for IPDC transmission.



1.2.2 System Components

1.2.2.1 Frontend

- **Role**

The frontend is responsible for inputting credential data, generating and signing transactions, and communicating with the backend.

- **Key Components**

1. **Data Input Form**

   A text area where users enter credential information in JSON format.

2. **Transaction Generation Function**

   Uses the CardanoWasm（Cardano Web Assembly）library to generate and sign a transaction based on user-inputted data, creating a signed transaction.

3. **Backend Communication Function**

   Sends the signed transaction to the backend's /submit endpoint, requesting transaction submission via the Blockfrost API.

4. **Download Function**

   Once the transaction is successfully recorded, users can download the credential data in JSON format.

1.2.2.2 Backend (cardano_server.mjs)

- **Role**

The backend is responsible for recording signed transactions received from the frontend onto the Cardano blockchain.

- **Key Components**

1. **Data Input Form**

   The backend is responsible for recording signed transactions received from the frontend onto the Cardano blockchain.

2. **Submit Endpoint**

   Receives POST requests from the frontend and passes the signed transaction data to the sendTransaction function.

3. **Transaction Submission Function**

   The sendTransaction function sends the transaction via the Blockfrost API, retrieves the transaction ID, and returns it to the frontend.

4. **Error Handling**

   If the transaction submission fails, an error response containing a message is sent to the frontend to prompt appropriate action.

1.2.2.3 Blockfrost API

- **Role**

The Blockfrost API receives transaction submissions from the backend and records them on the Cardano blockchain. It also provides functionality for retrieving and verifying block information.

- **Key Features**

1. **Transaction Submission (/tx/submit Endpoint)**

   Accepts signed transactions and records them on the blockchain. Upon success, it returns a transaction ID, which can be used for verification based on block information.

2. **Block Information Retrieval**

   Retrieves block information, hashes, and Merkle tree data based on the transaction ID, enabling the verification of credential data integrity.

1.2.3 System Data Flow

1. **Credential Data Input**

   The user enters credential information in the frontend.

2. **Transaction Generation and Signing**

   The frontend generates and signs a transaction.

3. **Transaction Submission**

   The signed transaction data is sent to the backend.

4. **Blockchain Recording via Blockfrost API**

   The backend submits the transaction via the Blockfrost API and retrieves the transaction ID.

5. **Block Information Retrieval and Verification**

   After the transaction is recorded, the user can download verification data.

1.2.4 System Architecture Features

1. Separation of Roles

The frontend handles user input and transaction generation, the backend manages transaction submission and error handling, and the Blockfrost API is responsible for blockchain recording. This modular approach ensures a consistent data flow with clear role distribution.

2. Security and Reliability

Credential data is signed and recorded on the blockchain, ensuring tamper resistance. Additionally, integration with the Blockfrost API enables easy retrieval of blockchain information and verification of transaction integrity.

3. Downloadable Data in a Verifiable Format

Data, including the transaction ID and block information, can be downloaded in JSON format, allowing for reliable data reference in disaster relief situations.

With this architecture, the certificate issuance tool ensures a seamless process from credential data input to transaction recording and verification, maintaining data security and reliability throughout.

1.3 Module Design

1.3.1 Frontend

The frontend provides an interface for creating and managing disaster relief volunteer certificates. It enables certificate data input, transaction generation and signing, submission to the backend server, retrieval of block information, verification using a Merkle tree, and certificate data download. The main functionalities and their roles are as follows:

1.3.1.1 Transaction Metadata Input

- **Elements**

  <textarea id=”metadataInput”>

- **Description**

  A text area for users to input volunteer credential information in JSON format. The JSON follows the ”Verifiable Credential” format, containing details such as the volunteer’s role, skills, and authorized regions.

- **Behavior**

  The inputted credential data is encoded as **Auxiliary Data** in the transaction. This data is later included in the transaction and sent to the blockchain for permanent storage.

1.3.1.2 Transaction Submission Button

- **Element**

  <button id="submitBtn">Submit Transaction</button>

- **Description**

  A button for submitting transactions. When clicked, it triggers transaction generation, signing, and submission to the backend.

- **Behavior**

  Clicking the button initiates the **transaction creation, signing, and submission process** with the following steps:

  1. **Address Generation**
     Uses a hardcoded private key to generate a public key and address.
  2. **Retrieve Maximum UTXO**
     Fetches Unspent Transaction Outputs (UTXOs) associated with the user's address and selects the largest UTXO.
  3. **Transaction Generation**
     Calculates the transfer amount and transaction fee based on the maximum UTXO. Constructs a transaction using the generated address.
  4. **Metadata Encoding**
     Encodes the inputted metadata from the frontend and adds it to the transaction as Auxiliary Data.
  5. **Transaction Signing**
     Signs the generated transaction using the private key. Produces a signed transaction in **Hex format.**
  6. **Submission to Backend**
     Sends the signed transaction data as a JSON object to the backend server. Receives a transaction ID from the backend upon successful submission.

1.3.1.3 Retrieving Block Information and Generating Merkle Tree

- **Functions**

  fetchBlockInfo()
  fetchBlockTransactions()
  buildMerkleTree()
  findPath()
  verifyMerklePath()

- **Description**

  Retrieves block information from the Blockfrost API based on the transaction ID.

  Generates a Merkle tree from the transaction hash and constructs a Merkle path for verification.

  Uses the verifyMerklePath function to check whether a specific transaction is legitimately recorded in the Merkle tree.

- **Purpose**

  Ensures that the transaction is correctly recorded on the blockchain. Guarantees the integrity and authenticity of the certificate data.

1.3.1.4 Certificate Data Download

- **Element**

  <button id=”vc_download” disabled>Download Certificate</button>

- **Description**

  A button for downloading the credential certificate data in JSON format. It remains disabled until the transaction is recorded and verified on the blockchain.

- **Behavior**

  Once the transaction and block information are retrieved, the certificate data (including certificate details, transaction Merkle path, etc.) is generated. The vc_download button becomes enabled. When the user clicks the button, the certificate data is downloaded as a JSON file (vc.json).

1.3.1.5 Main Data Flow

- **Data Flow ①**

  Certificate Data Input
  → **Transaction Generation & Signing**
  → **Submission**

  The user inputs credential data, and a signed transaction is generated. The signed transaction is sent to the backend, where it is recorded on the blockchain.

- **Data Flow ②**

  Transaction ID Reception
  → **Block Information Retrieval & Merkle Tree Generation**
  → **Certificate Data Download**

  Based on the received transaction ID, the system retrieves block information and generates the Merkle tree path. The verified certificate data is then made available to the user for download.

This frontend design enables users to seamlessly complete the entire process—from certificate data entry to blockchain recording, verification, and certificate download. Disaster relief organizations can easily and securely verify volunteer credentials, ensuring the trustworthiness of volunteer activities.

1.3.2 Backend

The backend server is built using the Express framework. It is responsible for receiving signed transactions from the frontend, submitting them to the blockchain, and managing transaction results. Additionally, it serves as an intermediary that interacts with the Blockfrost API to ensure proper transaction recording.

1.3.2.1 Server Configuration and Dependencies

- **Dependencies**
  - `express`: Web server framework.
  - `body-parser`: Parses request bodies.
  - `axios`: Sends HTTP requests to external APIs (e.g., Blockfrost API).
  - `dotenv`: Loads environment variables (e.g., API keys).
  - `log4js`: Manages logging for error messages and transaction processing.
- **Configuration**
  - Loads the Blockfrost API key from the .env file. Initializes logging when the server starts.
  - Supports logging configurations to record error messages and transaction processing statuses in both files and console output.

1.3.2.2 Logging Configuration

- **Log File**

  File Path: logs/cardano_server.log

- **Configuration**

  Uses log4js to set up both file logging and console logging. Specifies maximum log file size and number of backup files to maintain.

- **Purpose**

  Enhances tracking of transaction processing and error handling. Facilitates debugging by making it easier to identify transaction flows and pinpoint issues.

1.3.2.3 Endpoint

/submit (POST)

- **Description**

  Receives a signed transaction from the frontend and sends it to the Cardano network.

- **Processing Flow**

  1. **Request Handling**

     Extract the **signed transaction data (Hex format)** from the payload field.

  2. **Transaction Submission**

     Call the sendTransaction function to send the transaction to the Cardano blockchain using the Blockfrost API.

  3. **Response Return**

     Return the Blockfrost API response to the frontend:

     On success: Returns the transaction ID.

     On failure: Returns an error message.

/submit (GET)

- **Description**

  Receives the payload query parameter and executes the transaction submission process (for debugging and testing purposes).

- **Processing Flow**

  1. Retrieve the payload parameter (signed transaction) from the query parameters and call the sendTransaction function.
  2. Return the processing result as an HTTP response.

1.3.2.4 sendTransaction Function

- **Parameters**

  signedTxHex: The signed transaction data in Hex format.

- **Processing**

  Encode the signed transaction data in CBOR format. Send a POST request to the Blockfrost API's /tx/submit endpoint.

  Upon success, retrieve the transaction ID and return the response data for error handling.

- **Error Handling**

  If the transaction submission fails, return the error message and response code.

  Log the error details in the console and server logs. Send an appropriate error response to the frontend.

1.3.2.5 Error Handling

- **Configuration**

  If a network error or authentication error occurs, it is caught within the sendTransaction function. An HTTP 500 status code along with an error message is sent to the frontend.

- **Purpose**

  Ensures that accurate error messages are returned to the frontend. Allows users to be notified of error details, improving debugging and user experience.

1.3.2.6 CORS Configuration

- **Purpose**

  Allows cross-origin requests (CORS) from the frontend. Enables requests between different origins to ensure proper communication between the frontend and backend.

1.3.2.7 Server Listening

- **Configuration**

  The server **listens for requests on port 1337** using:
  app.listen(1337);
  This ensures the server is **actively running** and ready to handle incoming requests.

1.3.2.8 Backend Role and Data Flow

The backend server acts as an intermediary that processes transaction submission requests from the frontend and sends credential certificate data to the Cardano blockchain.

Data Flow

1. Receives a transaction submission request from the frontend.
2. Sends the signed transaction to the Cardano blockchain via the Blockfrost API.
3. On success, returns the transaction ID to the frontend.
4. The transaction ID is then used for block verification and Merkle tree validation.

By implementing robust error handling, the backend enhances system reliability and improves user experience.