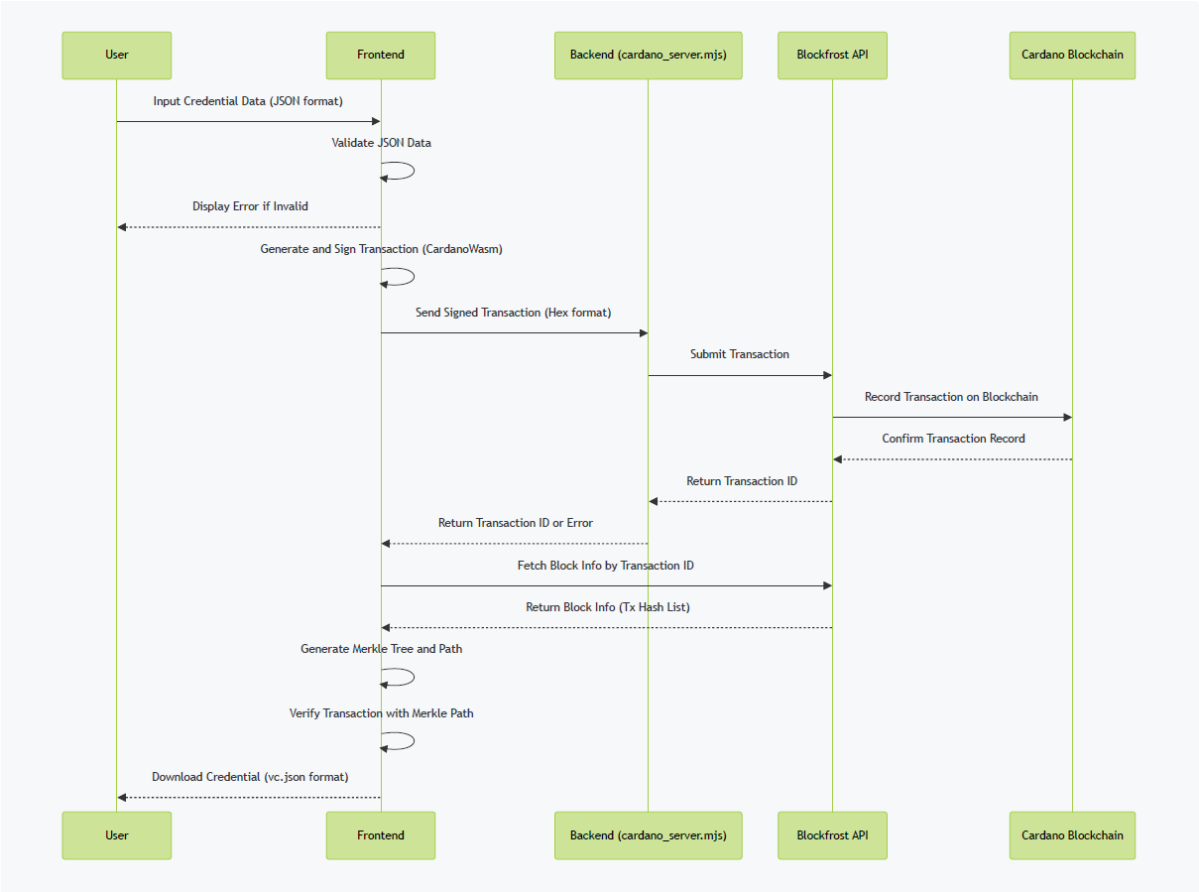
1.4 Data Flow and Algorithm Design

1.4.1 Data Flow and Algorithm

The certificate issuance tool provides an end-to-end process that seamlessly handles: User input、Generation of credential data、Transaction signing and submission、Recording on the blockchain、Retrieval and download of verification data. Below is the detailed data flow and algorithm for each step.

Data Flow Diagram



1.4.1.1 Credential Data Input

- **Processing Details**

  The user inputs credential data (e.g., name, role, skills, authorized regions) in JSON format. The user submits the data by clicking the submit button on the frontend interface.

- **Data Validation**

  The frontend verifies whether the input follows the correct JSON format. If the format is invalid, an error message is displayed. If the format is valid, the system proceeds to transaction generation.

1.4.1.2 Transaction Generation and Signing

- **Algorithm**

  Uses CardanoWasm（Cardano Web Assembly）library to encode credential data as Auxiliary Data and generate a transaction.

- **Detailed Steps**

  Address Generation

  Derives the public key from the user's private key. Retrieves the corresponding Cardano address.

  Fetching UTXO

  Retrieve the UTXOs (Unspent Transaction Outputs) associated with the user's address and select the largest UTXO for use.

  Transaction Generation

  Calculates transfer amount and transaction fees based on the selected UTXO. Constructs the transaction and adds the credential data as Auxiliary Data.

  Signing Process

  Signs the generated transaction using the user's private key. Produces a signed transaction in Hex format.

1.4.1.3 Transaction Submission

- **Backend Processing**

  The frontend sends a signed transaction to the /submit endpoint of cardano_server.mjs. The sendTransaction function forwards the transaction to the Blockfrost API, attempting to record it on the Cardano blockchain.

- **Data Validation**

  The frontend ensures that the transaction data is in correct JSON format before submission. If the format is invalid, an error message is displayed. If valid, the system proceeds with transaction submission.

- **On Success**

  The Blockfrost API returns a transaction ID. The backend sends this transaction ID back to the frontend for further processing.

- **On Failure (Error Handling)**

  If transaction submission fails, an error message is sent to the frontend. The user is prompted to retry the submission.

1.4.1.4 Retrieving Block Information and Verifying via Merkle Tree

- **Processing Details**

  Once a transaction is included in a block, the block information is retrieved from Blockfrost API using the transaction ID.

- **Algorithm: Merkle Tree Generation**

  1. Retrieve the list of transaction hashes in the target block from Blockfrost API.
  2. Pair and hash each transaction hash using SHA3-256, forming an upper-level hash.
  3. Repeat the process recursively until reaching a single root hash, constructing the Merkle tree.

- **Algorithm: Path Generation and Verification**

  1. **Generate the Merkle Path**

     The findPath function identifies the path from the transaction hash to the Merkle root.

  2. **Verify the Merkle Path**

     The verifyMerklePath function checks whether the specified transaction exists in the block using the Merkle tree.

1.4.1.5 Certificate Data Download

- **Processing Details**

  The credential certificate data is structured, including the transaction ID, block information, and Merkle tree path. The data is provided to the user in JSON format.

- **Download Format**

  The data can be downloaded as a JSON file named vc.json. The file can be used offline to verify credential information.

- **Purpose**

  Since the integrity of the certificate data is ensured, it can be used for credential verification in disaster relief scenarios and other applications.

1.4.2 Algorithm Overview

- **Transaction Generation and Signing**

  Uses the CardanoWasm library to encode user-input credential data and include it as part of the transaction.

  Calculates the transfer amount and transaction fee based on the largest UTXO. Signs the generated transaction with the user's private key, producing a signed transaction data.

- **Merkle Tree Generation and Path Verification**

  Retrieves the list of transaction ID hashes from the Blockfrost API to construct a Merkle tree.

  Generates the Merkle path from the specific transaction hash to the root hash. Uses this path to verify the data integrity, ensuring the transaction is correctly recorded on the blockchain.

1.4.3 Summary

- **Frontend**

  Handles credential data input and validation. Generates and signs transactions. Sends signed transactions to the backend.

- **Backend**

  Submits signed transactions to the blockchain. Retrieves the transaction ID. Manages error handling.

- **Blockfrost API**

  Records transactions on the blockchain. Retrieves block information. Performs data verification using the Merkle tree.

1.5 Data Design

The following details the data design of the certificate issuance tool. This data design specifies the role, format, and storage method of each data item used throughout the system, ensuring data consistency and reliability across all components.

Since the certificate issuance tool records user-inputted credential data on the blockchain and manages it in a verifiable format, various data items are handled across the frontend, backend, Blockfrost API, and blockchain. The following sections describe the role and design of each data item.

1.5.1 Frontend Data Items

1.5.1.1 **metadataInput**

- **Description**

  This is the input field where the user enters credential certificate data. The user provides information such as:

  Volunteer role、Skills、Authorized regions、Certification details

- **Format**

  JSON format

- **Data Example**

```
{
   "name": "John Doe",
   "role": "Heavy Machinery Operator",
   "skills": ["Excavator Operation", "Bulldozer Operation"],
   "authorizedRegions": ["Prefecture A, City B"]
}
```

1.5.1.2 signedTxHex

- **Description**

  A Hex-encoded signed transaction that includes credential certificate data. This signed transaction is sent from the frontend to the backend for submission to the Cardano blockchain. Once received by the backend, it is ready for blockchain recording.

- **Format**

  Hex-encoded string

- **Example Data**

  ```
  "abcdef1234567890abcdef1234567890"
  ```

1.5.1.3 transactionId

- **Description**

  A unique identifier returned when a transaction is successfully recorded on the blockchain. Used for credential data verification by retrieving block information and Merkle tree validation.

- **Format**

  String

- **Example Data**

  ```
  "abc123def456"
  ```

1.5.1.4 downloadableCertificate

- **Description**

  A JSON-formatted file that allows users to download credential certificate data recorded on the blockchain. Includes the transaction ID, Merkle tree information, and certificate details. Can be used for offline verification of credential authenticity.

- **Format**

  JSON format

- **Example Data**

```
{
   "transactionId": "abc123def456",
   "credentialSubject": { "name": "John Doe", "role": "Operator" },
   "merklePath": [ /* Merkleパス情報 */ ]
}
```

1.5.2 Backend Data Items

1.5.2.1 payload

- **Description**

  The signed transaction data sent from the frontend as a POST request. This data is passed to the Blockfrost API, where it is recorded on the blockchain.

- **Format**

  JSON format

- **Example Data**

```
{
   "payload": "abcdef1234567890abcdef1234567890"
}
```

1.5.2.2 Response Data from Blockfrost API

- **Description**

  The response data returned from the Blockfrost API after submitting a transaction.

  - On success: Includes the transaction ID.
  - On failure: Contains an error message.

- **Format**

  JSON format

- **Example Data(Successful Transaction Response)**

```
{
   "transactionId": "abc123def456"
}
```

- **Example Data(Failed Transaction Response)**

```
{
   "error": "Transaction submission failed"
}
```

1.5.3 Blockfrost API Related Data Items

1.5.3.1 transactionId

- **Description**

  A unique identifier returned by the Blockfrost API when a transaction is successfully recorded on the Cardano blockchain. Later used to retrieve block information and verification data.

- **Format**

  String

- **Example Data**

  `"abc123def456"`

1.5.3.2 blockHeight

- **Description**

  The block height where the transaction was recorded. Required for transaction verification using a Merkle tree.

- **Format**

  Numeric (Integer)

- **Example Data**

  `123456`

1.5.3.3 transactionHashes

- **Description**

  A list of all transaction ID hashes included in a specific block. Used for Merkle tree construction to verify transaction integrity.

- **Format**

  Array of strings

- **Example Data**

```
["hash1", "hash2", "hash3", "..."]
```

1.5.3.4 merklePath

- **Description**

  The Merkle path corresponding to the transaction ID of the credential certificate data. Used to verify data integrity by reconstructing the Merkle tree and confirming the transaction's validity in the block.

- **Format**

  Array of strings

- Example Data

```
[
   {"position": "left", "hash": "hash1"},
   {"position": "right", "hash": "hash2"}
]
```

1.5.4 Certificate Data Items

The credential certificate data structure follows the Verifiable Credential (VC) format, ensuring compliance with decentralized identity standards. It includes the following fields:

1.5.4.1 @context

- **Description**

  A schema URL that defines the data structure of the credential certificate. Helps ensure compatibility with Verifiable Credential standards.

- **Format**

  URL format

- **Example Data**

  ```
  "https://www.w3.org/2018/credentials/v1"
  ```

1.5.4.2 id

- **Description**

  A unique identifier for the credential certificate. Typically formatted as a URN (Uniform Resource Name) or UUID (Universally Unique Identifier).

- **Format**

  String (URN or UUID format)

- **Example Data**

  ```
  "urn:uuid:12345-67890"
  ```

1.5.4.3 type

- **Description**

  Specifies the type of certificate. Typically includes ”VerifiableCredential” along with a custom certificate type.

- **Format**

  Array of strings

- **Example Data**

```
["VerifiableCredential", "DisasterReliefVolunteerCredential"]
```

1.5.4.4 issuer

- **Description**

  Contains information about the entity that issued the credential certificate. This can be an organization, government entity, or decentralized identity (DID).

- **Format**

  Object

- **Example Data**

```
{
   "id": "https://disaster-response.example.org",
   "name": "Disaster Response Headquarters"
}
```

1.5.4.5 issuanceDate

- **Description**

  The date when the certificate was issued. Follows the ISO 8601 date format for standardization.

- **Format**

  ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)

- **Example Data**

  `"2024-11-03T00:00:00Z"`

1.5.4.6 credentialSubject

- **Description**

  Contains information about the subject (the individual or entity receiving the certificate). Includes name, role, skills, and authorized regions for validation.

- **Format**

  Object

- **Example Data**

```
{
   "id": "did:example:volunteer-001",
   "name": "John Doe",
   "role": "Operator",
   "skills": ["Excavator Operation"],
   "authorizedRegions": ["Prefecture A, City B"]
}
```

1.5.5 Summary

This data design ensures consistent data management across the system, maintaining the reliability and verifiability of credential certificate data.

- **Frontend**

  Receives user-inputted credential information in JSON format. Generates a signed transaction that includes the credential data.

- **Backend**

  Submits the signed transaction to the Cardano blockchain via Blockfrost API. Manages response data, including the transaction ID and error handling.

- Blockfrost API

  Provides block information and Merkle tree data based on the transaction ID. Supports credential certificate verification.

- **Credential Data**

  Follows a verifiable certificate format. Stored on the blockchain, ensuring it remains tamper-proof and secure.