Documentation for Cardano Implementation in hummingbot gateway

Table of Contents

cardano.ts

Overview

The Cardano class is a TypeScript implementation designed to interface with the Cardano blockchain. It provides utilities for initializing blockchain connections, managing wallets, retrieving

balances, and interacting with tokens. The class leverages the Lucid library for Cardano and Blockfrost APIs for network communication.

Key Components

1. CardanoTokenInfo

A type definition for representing information about Cardano tokens:

- policyId: The unique policy identifier for the token.
- assetName: The token's asset name.
- decimals: The number of decimal places the token supports.
- name: The token's full name.
- symbol: The token's symbol.
- logoURI: A URI pointing to the token's logo.

2. Cardano Class

Static Properties

- _instances: A dictionary of Cardano instances for different networks.
- lucidInstance: A shared instance of Lucid for interacting with the blockchain.

Instance Properties

- tokenList: An array of CardanoTokenInfo objects.
- _tokenMap: A map of token symbols to CardanoTokenInfo objects.
- _tokenListSource: The source URL or file path for token information.
- _tokenListType: The type of token list (e.g., static, dynamic).
- network: The blockchain network name (e.g., preprod, mainnet).
- allowedSlippage: The allowed slippage for transactions.
- blockfrostProjectId: The API key for Blockfrost.
- ttl: The time-to-live for transactions.
- _chain: The blockchain name (cardano).
- _ready: A boolean indicating whether the instance is initialized.
- apiURL: The API URL for network communication.
- defaultPoolId: The default pool ID for staking.
- nativeTokenSymbol: The symbol for the native currency (e.g., ADA).
- controller: Reference to the CardanoController class.

Constructor

- Cardano(network: string): Initializes a new Cardano instance for the specified network.

Static Methods

- getInstance(network: string): Cardano: Retrieves or creates a Cardano instance for the specified network.

- getConnectedInstances(): { [name: string]: Cardano }: Returns all connected instances.

Public Methods

- ready(): boolean: Returns whether the instance is initialized.
- init(): Promise<void>: Initializes the instance and loads tokens.
- getWalletFromPrivateKey(privateKey: string): Promise<{ address: string }>: Retrieves the wallet address associated with the provided private key.
- getWalletFromAddress(address: string): Promise<{ privateKey: string }>: Retrieves the private key associated with a wallet address.
- getNativeBalance(privateKey: string): Promise<string>: Returns the native ADA balance of the wallet.
- getAssetBalance(privateKey: string, token: string): Promise<string>: Returns the balance of a specific token in the wallet.
- getCurrentBlockNumber(): Promise<number>: Fetches the latest block number from the blockchain.
- getTransaction(txHash: string): Promise<Object>: Fetches transaction details by hash.
- getTokenForSymbol(symbol: string): CardanoTokenInfo[] | undefined: Returns token information for a specific symbol.
- getTokenAddress(symbol: string): string: Returns the token address (policy ID + asset name).
- storedTokenList: CardanoTokenInfo[]: Returns the stored token list.

Private Methods

- getLucid(): Lucid: Retrieves the Lucid instance. Throws an error if not initialized.
- encrypt(secret: string, password: string): Promise<string>: Encrypts a secret using AES-256-CTR.
- decrypt(encryptedSecret: string, password: string): Promise<string>: Decrypts an encrypted secret.
- loadTokens(tokenListSource: string): Promise<void>: Loads tokens from the specified source.
- getTokenList(tokenListSource: string): Promise<CardanoTokenInfo[]>: Fetches and parses the token list from the source.

cardano.controller.ts

CardanoController class defines methods to handle requests related to balances, tokens, and transaction polling for the Cardano blockchain within a Hummingbot client.

Import Statements

```
import { Cardano, CardanoTokenInfo } from './cardano';
import { BalanceRequest } from '../../network/network.requests';
```

```
import { validateCardanoBalanceRequest, validateAssetsRequest, validateCardanoPollRequest } from
'./cardano.validators';
import { PollRequest, getNetworkId } from './cardano.requests';
import { TokensRequest } from '../../network/network.requests';
import { TokenInfo } from '../../services/base';
```

These imports bring in required classes, interfaces, and utility functions for working with Cardano blockchain operations, request validation, and token information handling.

Method: poll

Definition:

```
static async poll(cardano: Cardano, req: PollRequest): Promise<Object | undefined>
```

Description:

Fetches transaction details for a given transaction hash.

Parameters:

- cardano: Instance of the Cardano class to interact with the blockchain.
- req: An instance of PollRequest containing the transaction hash (txHash) to fetch details.

Returns:

- A Promise resolving to an Object with transaction details or undefined if the transaction is not found.

Implementation:

1. Validates the incoming PollRequest using validateCardanoPollRequest.
2. Calls cardano.getTransaction(req.txHash) to retrieve transaction details.

Method: balances

Definition:

```
static async balances(chain: Cardano, request: BalanceRequest): Promise<{ balances: Record<string, string> }>
```

Description:

Calculates the balances of native tokens (ADA) and custom tokens based on the user's wallet.

Parameters:

- chain: An instance of the Cardano class to interact with the blockchain.
- request: An instance of BalanceRequest containing:
    - address: The wallet address to fetch balances for.
    - tokenSymbols: An array of token symbols to fetch balances.

Returns:

- A Promise resolving to an object of type:

{ balances: Record<string, string> }

where keys are token symbols and values are balances.

Implementation:

1. Validates the request using validateCardanoBalanceRequest.
2. Decrypts the wallet private key from the provided address using chain.getWalletFromAddress.
3. Initializes a balances object.
4. Fetches native ADA balance using chain.getNativeBalance.
5. Iterates through custom tokens (tokenSymbols) and retrieves balances via chain.getAssetBalance.
6. Returns the compiled balances as an object.

Method: getTokens

Definition:

static async getTokens(cardano: Cardano, request: TokensRequest): Promise<{ tokens: TokenInfo[] }>

Description:

Fetches a list of token information based on the requested token symbols.

Parameters:

- cardano: Instance of the Cardano class to interact with the blockchain.
- request: An instance of TokensRequest containing:
    - tokenSymbols: (Optional) An array of token symbols to filter the results.
    - network: The network identifier for the Cardano blockchain.

Returns:

- A Promise resolving to an object of type:

{ tokens: TokenInfo[] }

where each TokenInfo contains:

- address: The concatenated policyId and assetName.
- chainId: The Cardano network identifier.
- decimals: Fixed at 6.
- name: The token's name.
- symbol: The token's symbol.

Implementation:

1. Validates the request using validateAssetsRequest.
2. If no tokenSymbols are provided, retrieves all tokens from cardano.storedTokenList.
3. If tokenSymbols are specified, filters tokens using cardano.getTokenForSymbol.
4. Maps each CardanoTokenInfo into a TokenInfo object.

5. Returns the list of TokenInfo.

Validators and Utility Functions

- validateCardanoPollRequest(req): Ensures the validity of a poll request.
- validateCardanoBalanceRequest(request): Ensures the validity of a balance request.
- validateAssetsRequest(request): Ensures the validity of a token assets request.
- getNetworkId(network): Converts the network identifier to the corresponding chainId.

cardano.requests.ts

cardano.requests.ts file, is used for handling Cardano-related requests and network operations.

1. Types

AssetsRequest

```
export type AssetsRequest = {
  network?: string;
  tokenSymbols?: string[];
};
```

Description:

Represents a request for querying Cardano assets.

Fields:

- network? *(optional, string)*: Specifies the Cardano network (e.g., mainnet, preprod).
- tokenSymbols? *(optional, string[])*: Array of token symbols to filter the assets.

AssetsResponse

```
export type AssetsResponse = {
  assets: CardanoAsset[];
};
```

Description:

Defines the structure of the response for an asset query.

Fields:

- assets *(CardanoAsset[])*: An array of CardanoAsset objects containing asset details.

2. Interfaces

CardanoAsset

```
export interface CardanoAsset {
  symbol: string;
  assetId: number;
  decimals: number;
}
```

Description:

Defines the structure of a Cardano asset.

Fields:

- symbol *(string)*: The symbol of the asset (e.g., ADA).
- assetId *(number)*: A unique identifier for the asset.
- decimals *(number)*: Number of decimal places used by the asset.

PollRequest

```
export interface PollRequest {
  network: string;
  txHash: string;
}
```

Description:

Represents a request to poll transaction details.

Fields:

- network *(string)*: The Cardano network (e.g., mainnet, preprod).
- txHash *(string)*: The hash of the transaction to poll.

3. Enums

CardanoNetworkID

```
export enum CardanoNetworkID {
  MAINNET = 764824073,
  PREPROD = 1,
}
```

Description:

Defines constants representing Cardano network IDs.

Values:

- MAINNET: 764824073 — The network ID for the Cardano mainnet.
- PREPROD: 1 — The network ID for the Cardano pre-production environment.

4. Functions

getNetworkId

```
export function getNetworkId(network: string = ''): number {
  switch (network) {
    case 'mainnet':
      return CardanoNetworkID.MAINNET;

    case 'preprod':
      return CardanoNetworkID.PREPROD;
```

```
      default:
        return 0;
    }
}
```

Description:

Maps a network name to its corresponding network ID.

Parameters:

- network *(string, default = '')*: The name of the network (e.g., mainnet, preprod).

Returns:

- *(number)*: The network ID:
    - 764824073 for mainnet.
    - 1 for preprod.
    - 0 for unknown or unspecified networks.

Implementation:

- Uses a switch statement to map network names to their corresponding IDs.
- Defaults to 0 if the provided network is not recognized.

cardano.validators.ts

cardano.validators.ts ensure that incoming requests conform to the expected structure and constraints, safeguarding the integrity of Cardano-related operations.

1. Imports

```
import {
  invalidTokenSymbolsError,
  mkRequestValidator,
  mkValidator,
  RequestValidator,
  validateTokenSymbols,
  Validator,
} from '../../services/validators';
```

These imports include utility functions, error messages, and types for creating and managing validators:

- mkValidator: Creates a single field validator.
- mkRequestValidator: Creates a request-level validator composed of multiple field validators.
- Validator: Type for single field validators.
- RequestValidator: Type for request-level validators.

2. Error Messages

```
const invalidChainError: string = 'The chain param is not a string.';
const invalidNetworkError: string = 'The network param is not a string.';
const invalidCardanoAddressError = "Invalid Cardano address.";
const invalidTxHashError: string = 'The txHash param must be a string.';
```

Description:

Predefined error messages used in validators to indicate why a field failed validation:

- invalidChainError: Indicates that the chain parameter is missing or invalid.
- invalidNetworkError: Indicates that the network parameter is missing or invalid.
- invalidCardanoAddressError: Indicates that the provided Cardano address is invalid.
- invalidTxHashError: Indicates that the txHash parameter must be a string.

3. Validators

3.1. validateCardanoChain

```
const validateCardanoChain: Validator = mkValidator(
  'chain',
  invalidChainError,
  (val) => typeof val === 'string' && val === 'cardano'
);
```

Description:

Validates that the chain parameter is a string and equals cardano.

Parameters:

- val: The value to validate.

Returns:

- *(boolean)*: true if the chain is valid, otherwise false.

3.2. validateNetwork

```
export const validateNetwork: Validator = mkValidator(
  'network',
  invalidNetworkError,
  (val) => typeof val === 'string'
);
```

Description:

Validates that the network parameter is a string.

Parameters:

- val: The value to validate.

Returns:

- *(boolean)*: true if the network is valid, otherwise false.

3.3. validateTxHash

```
const validateTxHash: Validator = mkValidator(
  'txHash',
  invalidTxHashError,
  (val) => typeof val === 'string'
);
```

Description:

Validates that the txHash parameter is a string.

Parameters:

- val: The value to validate.

Returns:

- *(boolean)*: true if the txHash is valid, otherwise false.

3.4. validateCardanoAddress

```
const cardanoAddressRegex = /^(addr|addr_test)[0-9a-zA-Z]{1,}$/;

export const validateCardanoAddress: Validator = mkValidator(
  'address',
  invalidCardanoAddressError,
  (val) => typeof val === 'string' && cardanoAddressRegex.test(val)
);
```

Description:

Validates that the address parameter is a valid Cardano address.

Parameters:

- val: The value to validate.

Returns:

- *(boolean)*: true if the address is valid, otherwise false.

3.5. validateAssetSymbols

```
export const validateAssetSymbols: Validator = (req: any) => {
  const errors: Array<string> = [];
  if (req.assetSymbols) {
    if (Array.isArray(req.assetSymbols)) {
      req.tokenSymbols.forEach((symbol: any) => {
```

```
            if (typeof symbol !== 'string') {
              errors.push(invalidTokenSymbolsError);
            }
          });
        } else if (typeof req.assetSymbols !== 'string') {
          errors.push(invalidTokenSymbolsError);
        }
      }
      return errors;
};
```

Description:

Validates the assetSymbols parameter. Ensures it is either a string or an array of strings.

Parameters:

- req *(any)*: The request object containing assetSymbols.

Returns:

- *(Array<string>)*: An array of error messages, or an empty array if validation succeeds.

4. Request Validators

4.1. validateAssetsRequest

```
export const validateAssetsRequest: RequestValidator = mkRequestValidator([
  validateNetwork,
  validateAssetSymbols,
]);
```

Description:

Validates an assets request by ensuring:

- The network field is valid.
- The assetSymbols field is valid.

4.2. validateCardanoPollRequest

```
export const validateCardanoPollRequest: RequestValidator = mkRequestValidator(
  [validateNetwork, validateTxHash]
);
```

Description:

Validates a Cardano transaction polling request by ensuring:

- The network field is valid.
- The txHash field is a string.

4.3. validateCardanoBalanceRequest

```
export const validateCardanoBalanceRequest: RequestValidator =
  mkRequestValidator([
    validateCardanoChain,
```

```
    validateNetwork,
    validateCardanoAddress,
    validateTokenSymbols,
]);
```

cardano.config.ts

It defines the structure of Cardano-specific configurations and fetches configuration values from the ConfigManagerV2 service.

NetworkConfig Interface

Defines the configuration parameters for a Cardano network.

- **Properties**:
    - name (string): The name of the network (e.g., mainnet, testnet).
    - apiurl (string): The API URL for accessing Cardano blockchain data.

Config Interface

Describes the overall configuration structure for the Cardano chain.

- **Properties**:
    - network (NetworkConfig): Contains network-specific configurations.
    - allowedSlippage (string): Defines the allowed slippage for transactions (e.g., 0.5%).
    - blockfrostProjectId (string): Project ID for accessing Blockfrost API.
    - preprodBlockfrostProjectId (string): Project ID for accessing Blockfrost API in the pre-production environment.
    - ttl (string): Time-to-live for transactions.
    - defaultPoolId (string): Default pool ID for staking or other operations.
    - defaultAddress (string): Default wallet address for the chain.
    - nativeCurrencySymbol (string): Symbol for the native currency (e.g., ₳ for ADA).
    - tokenListType (string): Type of the token list. Default is FILE.
    - tokenListSource (string): Path to the token list file. Default is src/chains/cardano/cardano_tokens.json.

Function: getCardanoConfig

Fetches the configuration for the Cardano blockchain based on the chain name and network name.

- **Parameters**:
    - chainName (string): The name of the blockchain (e.g., cardano).
    - networkName (string): The specific network for which the configuration is required (e.g., mainnet, preprod).
- **Returns**:

- A Config object containing the Cardano configuration for the specified chain and network.

- **Implementation Details**:
  - **Network Configuration**:
    - network.name: The name of the network.
    - network.apiurl: API URL fetched from ConfigManagerV2.
  - **Other Configurations**:
    - allowedSlippage: Fetched from ConfigManagerV2 using the key <chainName>.allowedSlippage.
    - blockfrostProjectId: Fetched using <chainName>.blockfrostProjectId.
    - preprodBlockfrostProjectId: Fetched using <chainName>.preprodBlockfrostProjectId.
    - defaultPoolId: Fetched from <chainName>.defaultPoolId.<networkName>.poolId.
    - defaultAddress: Fetched from <chainName>.defaultAddress.
    - ttl: Fetched from <chainName>.ttl.
    - nativeCurrencySymbol: Fetched from <chainName>.networks.<networkName>.nativeCurrencySymbol.
    - tokenListType: Fetched from <chainName>.networks.<networkName>.tokenListType.
    - tokenListSource: Fetched from <chainName>.networks.<networkName>.tokenListSource.

Dependencies

This module relies on the ConfigManagerV2 service to fetch configuration values dynamically.

- ConfigManagerV2.getInstance():
  - Provides access to the singleton instance of the configuration manager.
  - Uses key-based queries to fetch configuration values.

Changes Made in network.controllers.ts

Overview: The changes made to the network.controller.ts file involve adding support for the Cardano blockchain. Specifically, the connections to Cardano instances are now retrieved and merged into the overall list of blockchain connections. This was achieved by incorporating the Cardano.getConnectedInstances() function and concatenating the result with other blockchain connections.

Code Changes:

1. **Importing Cardano Class:** At the top of the file, the Cardano class from the ../chains/cardano/cardano module is imported:

```
import { Cardano } from '../chains/cardano/cardano';
```

2. **Retrieving Cardano Connections:** Inside the getStatus function, Cardano connections are retrieved similarly to other blockchains. The Cardano.getConnectedInstances() method is called to fetch the connected Cardano instances:

```
const cardanoConnections = Cardano.getConnectedInstances();
```

3. **Merging Cardano Connections:** The retrieved Cardano connections are concatenated with the other blockchain connections, ensuring that all connected instances across different blockchains are included in the connections array:

```
connections = connections.concat(
  cardanoConnections ? Object.values(cardanoConnections) : []
);
```

Changes Made in wallet.controllers.ts

Overview:

The changes introduced in the wallet.controller.ts file add support for Cardano wallet handling. Specifically, these changes allow the retrieval and encryption of a Cardano wallet from a private key, as well as storing it securely. The modifications have been made in the addWallet function.

Key Changes:

1. **Import of Cardano Module:**
    - The Cardano class is imported from the ../../chains/cardano/cardano module:

```
import { Cardano } from '../../chains/cardano/cardano';
```

2. **Addition of Cardano Wallet Handling in** addWallet **Function:**
    - A new else if block has been added to handle the Cardano chain. This block is executed when the connection is an instance of Cardano.
    - The process retrieves the wallet from the private key and extracts the wallet's address and encrypted private key.

```
else if (connection instanceof Cardano) {
  const wallet = await connection.getWalletFromPrivateKey(req.privateKey);
  address = wallet.address;
  encryptedPrivateKey = await connection.encrypt(req.privateKey, passphrase);
}
```

In this change:

- getWalletFromPrivateKey: Retrieves the wallet associated with the provided private key.
- address: Stores the wallet's address.
- encryptedPrivateKey: Encrypts the private key using the provided passphrase for secure storage.

Changes Made in `wallet.validators.ts`

This document outlines the changes made to the `wallet.validator.ts` file, specifically the additions and modifications related to the validation of Cardano private keys. The changes are designed to ensure that Cardano private keys are correctly validated based on the expected format and requirements.

Changes:

1. **Added** `invalidCardanoPrivateKeyError` **Constant**: A new error message was introduced to handle invalid Cardano private keys:

```
export const invalidCardanoPrivateKeyError: string =
  'The privateKey param is not a valid Cardano private key.';
```

2. **Implemented** `isCardanoPrivateKey` **Validation Function**: A new function `isCardanoPrivateKey` was added to validate if a given string matches the format of a valid Cardano private key. The function checks for the following:
    - The string must start with the prefix `ed25519_sk`, which is specific to Cardano's Bech32 private key.
    - The string should contain valid Bech32 characters, which include alphanumeric characters excluding '1', 'b', 'i', or 'o'.
    - The length of the key must be between 57 and 128 characters (inclusive).

The implementation uses a regular expression to validate the key format:

```
export const isCardanoPrivateKey = (str: string): boolean => {
  try {
    return /^ed25519_sk[0-9a-zA-HJ-NP-Z]{53,120}$/.test(str);
  } catch {
    return false;
  }
};
```

3. **Modified** `validatePrivateKey` **to Include Cardano**: The `validatePrivateKey` function was updated to include validation for the `cardano` chain. The new validation function uses the previously defined `isCardanoPrivateKey` to ensure that the `privateKey` field is valid for the Cardano blockchain. The updated validator includes the following check:

```
cardano: mkValidator(
  'privateKey',
  invalidCardanoPrivateKeyError,
```

```
  (val) => typeof val === 'string' && isCardanoPrivateKey(val),
),
```

4. **Updated** `validateChain` **to Include Cardano**: The `validateChain` validator was modified to include `cardano` as a valid chain. The updated validation ensures that the chain can either be one of several predefined chains, including `cardano`:

```
export const validateChain: Validator = mkValidator(
  'chain',
  invalidChainError,
  (val) =>
    typeof val === 'string' &&
    (val === 'algorand' ||
        ...
      val === 'cardano'),
);
```

`cardano.yml`

This configuration file is used for setting up and managing Cardano network connections and trading parameters.

Key Configurations

1. **Allowed Slippage**
    - `allowedSlippage`: Defines the maximum permissible unfavorable movement in execution price during a trade.
    - Example: `'1/100'` indicates a 1% slippage tolerance.
2. **Blockfrost API**
    - `blockfrostProjectId`: Project ID for connecting to the Cardano mainnet.
    - `preprodBlockfrostProjectId`: Project ID for connecting to the Cardano pre-production network.
3. **Trade Time-to-Live (TTL)**
    - `ttl`: Specifies how long a trade remains valid, in seconds.
    - Default: `300` seconds (5 minutes).
4. **Default Pool IDs**
    - `defaultPoolId`: Specifies the default liquidity pool for mainnet and pre-production networks.
        - Example Mainnet Pool ID: `6aa2153e1...6a2`
        - Example Preprod Pool ID: `3bb0079...d0d`
5. **Contract Addresses**
    - `contractAddresses`: API URLs for connecting to the Blockfrost API for mainnet and pre-production.
        - Mainnet: `https://cardano-mainnet.blockfrost.io/api/v0`
        - Preprod: `https://cardano-preprod.blockfrost.io/api/v0`
6. **Networks**

- `networks`: Defines settings for the mainnet and pre-production networks:
  - `tokenListType`: Source type for token lists (e.g., "FILE").
  - `tokenListSource`: Path to token list configuration files.
  - `nativeCurrencySymbol`: Symbol for the native currency (e.g., `'ADA'`).

cardano-schema.json

This JSON schema defines the structure and validation rules for a Cardano.yml file. It ensures that all required properties are present and adhere to specific data types and structures.

Properties

1. **allowedSlippage**
   - Type: `string`
   - Example: `'1/100'`
2. **blockfrostProjectId**
   - Type: `string`
   - Mainnet Blockfrost API Project ID.
3. **preprodBlockfrostProjectId**
   - Type: `string`
   - Pre-production Blockfrost API Project ID.
4. **nativeCurrencySymbol**
   - Type: `string`
   - Example: `'ADA'`
5. **ttl**
   - Type: `integer`
   - Trade validity in seconds (e.g., `300`).
6. **defaultPoolId**
   - Type: `object`
   - Contains `poolId` for each environment (e.g., `mainnet`, `preprod`).
   - Each pool has:
     - `poolId`: Type `string`
7. **contractAddresses**
   - Type: `object`
   - Contains `apiurl` for each environment (e.g., `mainnet`, `preprod`).
   - Each address has:
     - `apiurl`: Type `string`
8. **networks**

- Type: `object`
- Contains settings for `mainnet` and `preprod`:
    - **Properties**:
        - `tokenListType`: Type `string` (e.g., `"FILE"`)
        - `tokenListSource`: Type `string` (Path to token list file)
        - `nativeCurrencySymbol`: Type `string` (e.g., `'ADA'`)

Required Properties

The following are mandatory:

- `allowedSlippage`
- `blockfrostProjectId`
- `preprodBlockfrostProjectId`
- `ttl`
- `defaultPoolId`
- `contractAddresses`
- `networks`

cardano_mainnet_tokens.json & cardano_preprod_tokens.json

These files provides a structured list of tokens on the Cardano blockchain, including their metadata for use in decentralized applications.

Fields:

1. `name`:
    - The name of the token list.
    - Example: `"Cardano Tokenlist"`
2. `logoURI`:
    - URL of the logo representing the token list.
    - Example: `"https://ibb.co/N1Jmswk"`
3. `keywords`:
    - An array of keywords associated with the token list.
    - Example: `["cardano", "tokens", "MIN"]`
4. `timestamp`:
    - ISO 8601 timestamp indicating the last update.
    - Example: `"2024-12-12T12:00:00+00:00"`
5. `tokens`:
    - An array of token objects, each representing a specific token with its metadata.

Token Object Structure:

- `policyId`: Unique identifier for the token's minting policy.
  - Example: `"29d222ce763455e3d7a09a665ce554f00ac89d2e99a1a83d267170c6"`
  - Empty for native Cardano tokens like ADA.
- `assetName`: Hexadecimal representation of the token's name.
  - Example: `"4d494e"` (represents "MIN").
- `decimals`: Number of decimal places used by the token.
  - Example: `6`
- `name`: Readable name of the token.
  - Example: `"Minswap Token"` or `"Cardano Native Token"`
- `symbol`: Abbreviation of the token's name.
  - Example: `"MIN"` or `"ADA"`
- `logoURI`: URL to the token's logo.
  - Example: "https://ibb.co/L1Tp0rQ"

minswap.ts

The `MinSwap` class manages the singleton instances for a given network and provides methods to initialize and check readiness.

Properties:

- `_instances: { [name: string]: MinSwap }`
  A static property to store instances of `MinSwap` for different networks.

Constructor:

- `constructor(network: string)`
  Initializes a new instance of the `MinSwap` class for the provided network. Logs the network information using the `logger`.

Static Methods:

- `getInstance(network: string): MinSwap`
  Returns the singleton instance of `MinSwap` for the specified network. If the instance does not exist, a new one is created and returned.

Instance Methods:

- `async init()`
  An asynchronous method intended for initialization (currently empty).

- `ready(): boolean`
  Returns `true`, indicating the instance is ready (a placeholder method for future logic).

`minswap.controller.ts`

Overview

The `minswap.controller.ts` file contains the implementation of the MinSwap connector for interacting with the Cardano blockchain and the MinSwap decentralized exchange (DEX). It provides methods for handling price queries, trade execution, and pool price fetching, along with utility functions for initializing requests and committing transactions.

Imports

External Libraries

- `Cardano`: Represents Cardano blockchain-related functionalities.
- `MinSwap`: Provides interaction methods for MinSwap DEX.
- `ConfigManagerV2`: Manages configuration values.
- **Decimal.js-light**: Handles decimal arithmetic for precise calculations.
- `BlockfrostAPI`: Blockfrost API client for querying Cardano data.
- `lucid-cardano`: Provides Cardano-related utilities such as transaction building and data manipulation.

Interfaces

- `PriceRequest`, `PriceResponse`: Define the structure for price-related requests and responses.
- `TradeRequest`, `TradeResponse`: Define the structure for trade-related requests and responses.
- `PoolPriceRequest`, `PoolPriceResponse`: Define the structure for pool price-related requests and responses.
- `AddLiquidityRequest`, `AddLiquidityResponse`: Define the structure for adding liquidity operations.
- `RemoveLiquidityRequest`, `RemoveLiquidityResponse`: Define the structure for removing liquidity operations.

Interfaces and Utility Functions

Interfaces

`InitialResponse`

Structure for storing the initialized values required for trade execution.

Properties:

- `network`: The Cardano network (e.g., Mainnet or Preprod).
- `ttl`: Time-to-live for the transaction.

- `slippage`: Allowed slippage for the trade.
- `blockfrostUrl`: Blockfrost API URL.
- `blockfrostProjectId`: Blockfrost Project ID.
- `poolId`: Identifier of the liquidity pool.
- `blockfrostAdapterInstance`: Instance of BlockfrostAdapter.
- `address`: Wallet address for the transaction.
- `lucid`: Lucid instance for transaction creation.

`InitialPriceResponse`

Structure for storing the initialized values required for fetching price data.

Properties:

- Same as `InitialResponse` (excluding `address` and `lucid`).

Utility Functions

`initializePrice`

Initializes data required to handle a price request.

Parameters:

- `req`: PriceRequest object containing network and pool information.
- `cardanoish`: Cardano instance containing blockchain configuration.

Returns: Promise of `InitialPriceResponse`.

`initializeTrade`

Initializes data required to execute a trade request.

Parameters:

- `req`: TradeRequest object containing trade details.
- `cardanoish`: Cardano instance containing blockchain configuration.

Returns: Promise of `InitialResponse`.

`commitTransaction`

Signs and submits a transaction.

Parameters:

- `txCompleteReq`: A complete transaction object ready for signing.

Returns: Promise of the transaction ID (`txId`).

Main Functions

`price`

Handles price queries on the MinSwap DEX.

Parameters:

- `cardanoish`: Cardano instance.
- `minSwap`: MinSwap instance.
- `req`: PriceRequest object.

Process:

1. Initializes required data using `initializePrice`.
2. Fetches the pool and calculates the price based on the request type (BUY/SELL).
3. Constructs and returns a `PriceResponse` object.

Returns: Promise of `PriceResponse`.

`trade`

Executes a trade on the MinSwap DEX.

Parameters:

- `cardanoish`: Cardano instance.
- `minSwap`: MinSwap instance.
- `req`: TradeRequest object.

Process:

1. Initializes trade data using `initializeTrade`.
2. Fetches UTXOs for the user's wallet.
3. Depending on the `side` (BUY/SELL), creates a transaction (ExactIn or ExactOut).
4. Returns a `TradeResponse` object containing the trade details and transaction hash.

Returns: Promise of `TradeResponse`.

`poolPrice`

Fetches pool-specific price details.

Parameters:

- `cardano`: Cardano instance.
- `req`: PoolPriceRequest object.

Process:

1. Initializes pool data.
2. Fetches pool-specific prices using the Blockfrost adapter.
3. Constructs and returns a `PoolPriceResponse` object.

Returns: Promise of `PoolPriceResponse`.

`addLiquidity`

Adds liquidity to a specified liquidity pool on the Cardano blockchain using MinSwap.

Parameters:

- `cardano`: Cardano instance.
- `req`**:** AddLiquidityRequest object.

Process:

1. Initialize network and wallet (Blockfrost, Lucid).
2. Fetch pool details and calculate deposit parameters.
3. Build and submit the transaction.
4. Return transaction details as the response.

Returns: Promise of `AddLiquidityResponse`**.**

`removeLiquidity`

Removes liquidity from a specified liquidity pool on the Cardano blockchain using MinSwap.

Parameters:

- `cardano`: Cardano instance.
- `req`**:** RemoveLiquidityRequest object.

Returns: Promise of Remove`LiquidityResponse`**.**

Error Handling

Common exceptions include:

- `HttpException` for unsupported tokens or invalid inputs.
- Custom errors for invalid network or pool identifiers.

Dependencies

External

- `@minswap/sdk`: Provides core MinSwap DEX functionalities such as pool and token management.
- `@blockfrost/blockfrost-js`: Client library for Blockfrost API interactions.
- `lucid-cardano`: Provides utilities for Cardano transactions and data handling.
- `Decimal.js-light`: For precise decimal calculations.

Internal

- `cardano/cardano.ts`: Cardano-specific utilities and configuration.
- `services/config-manager-v2.ts`: Configuration manager for fetching network settings.
- `amm/amm.requests.ts`: Request and response interfaces for AMM operations.
- `services/error-handler.ts`: Provides error handling utilities.