Project:

STUDY CONTROL SYSTEM FOR UNIVERSITIES
DIPLOMADA #1100266
MILESTONE 2 COMPLIANCE REPORT
Test report for each API Endpoint

1. GENERAL INFORMATION:
- Endpoint Name: LUCID Cardano Wallet NAMI
- HTTP Method: [GET/POST]
- Description: Web-based communication bridge that allows web pages (i.e. dApps) to interact with Cardano wallets. This is done through javascript code injected into web pages. This specification defines the way the website/dApp should access said code, as well as defines the API for the dApps to communicate with the user's wallet.
2. TESTS PERFORMED:
- Test 0: Communication from web pages to a user's Cardano wallet:
- Result: Success
- Observations: Communication with different types of Cardano wallets with different extensions is tested, cardano.{walletName}.enable():

cardano.{walletName}.enable({ extensions: Extension[] } = {}): Promise<API>
This is the entry point to initiate communication with the user's wallet. The wallet must request the user's permission to connect the web page to the user's wallet and if permission is granted, the entire API will be returned to the dApp for use. The wallet can choose to maintain a whitelist so as not to necessarily ask the user for permission every time access is requested, but this behavior depends on the wallet and should be transparent to web pages that use this API. If a wallet is already connected, this function should not request access a second time, but simply return the API object.
DApps are expected to use this endpoint to perform an initial handshake and ensure that the wallet supports all required functionality. Note that it is possible for two extensions to be mutually incompatible (because they provide two conflicting features). While we can try to avoid this as much as possible when designing CIP, it is also the responsibility of wallet providers to evaluate whether they can support a given combination of extensions or not. Wallets are therefore not expected to fail if they do not recognize or support a particular combination of extensions. Instead, they should decide what they allow and reflect their choice in the response to api.getExtensions() in the full API. As a result, dApps may crash and report to their users or may use a different, less efficient strategy to address the lack of functionality.

- Test 1: Checking the list of extensions enabled by the wallet:
- Result: Success
- Remarks: Retrieves the list of extensions enabled by the wallet, api.getExtensions():
api.getExtensions(): Promise<Extension[]>
Errors: APIError

- Test 2: Verification of the connected network:
- Result: Success

- Remarks: Returns the network ID of the currently connected account. 0 is testnet and 1 is mainnet, but wallets may return other networks, api.getNetworkId():
api.getNetworkId(): Promise<number>
Errors: APIError

- Test 3: Verification of unspent transaction outputs:
- Result: Success
- Remarks: Returns a list of all UTXOs (unspent transaction outputs) controlled by wallet.api.getUtxos():
api.getUtxos(amount: cbor<value> = undefined, paginate: Paginate = undefined): Promise<TransactionUnspentOutput[] | null>
Errors: APIError, PaginateError

- Test 4: Verification of total balance available in the wallet:
- Result: Success
- Observations: Returns the total available balance of the wallet. This is the same as adding the results of api.getUtxos().api.getBalance():
api.getUtxos(amount: cbor<value> = undefined, paginate: Paginate = undefined): api.getBalance(): Promise<cbor<value>>
Errors: APIError

- Test 5: Verification of list of addresses used by the wallet:
- Result: Success
- Remarks: Returns a list of all used addresses (included in some on-chain transactions) controlled by wallet.api.getUsedAddresses():
api.getUsedAddresses(paginate: Paginate = undefined): Promise<Address[]>
Errors: APIError