Exploring

Self-Sovereign Identity

Coding Standards

1. **Limited use of global variables:**

    - Only use global variables where completely necessary and when implementation of a local variable would be more complicated or impossible.

2. **Standard headers for different modules:**

Headers for each code module should follow the format set out below:

- Name of the module
- Date of module creation
- Author of the module
- Modification history
- Synopsis of the module about what the module does
- Different functions supported in the module along with their input output parameters
- Global variables accessed or modified by the module

3. **Naming conventions for local variables, global variables, constants, and functions:**
    i. Meaningful and understandable variables names must be used
    ii. Local variables should be named using camel case lettering starting with small letter (i.e., localVar)
    iii. Global variables names should start with a capital letter (i.e., GlobalVar).
    iv. Constant names should be formed using capital letters only (i.e., CONSVAR).
    v. Avoid the use of digits in variable names.
    vi. The names of the function should be written in camel case starting with small letters.

vii. The name of the function must describe the reason of using the function clearly and briefly.

4. **Indentation:**

The following indentation conventions must be followed to ensure readability and standardization:

i. There must be a space after giving a comma between two function arguments.
ii. Each nested block should be properly indented and spaced.
iii. Proper Indentation must be used at the beginning and end of each code block.
iv. All braces must start from a new line and the code segments following the end of braces should also start from a new line.

5. **Error return values and exception handling conventions:**

i. Functions encountering an error condition should either return a 0 or 1, or an error message for interpretation when debugging.

Code Quality Metrics (Back-end API)

Metric	Description	Value
Maintainability Index	Measures ease of code maintenance. Higher values are better.	104
Depth of Inheritance	Measures length of object inheritance hierarchy. Lower values are better.	3
Class Coupling	Measures number of classes that are referenced.	175

Github File Structure

Each component of the system was branched off the “develop branch” and committed to by those working on the respective sections of the project. The individual sub-branches of develop were merged to the main develop branch, which was then merged to the master branch for the final demo.

Existing Branches in their hierarchy

This is a list of the branches contributing to actual production, not branches used for testing purposes or those that had no effect on the final develop branch. For an exhaustive list see repository: https://github.com/COS301-SE-2022/Exploring-Self-Sovereign-Identity

Active Branches:

- develop

Branched from develop / branches based on develop

- develop_Avatar_integrations (Avatar system integration)
- develop_api_auth (Authorization & Authentication)
- develop_smartcontract (Smart Contracts)
- develop_api_refactor (Changes on the API)
- develop_scAuth (Smart Contract Authorization & Authentication)
- develop_marketplace (Data Marketplace)
- develop_api (API)
- develop-vuejs-material (Frontend UI)
- develop_SSI_Avatars (Avatar System Creation)
- develop_SSIAvatars (Avatar System Refactor)
- develop-vuejs-client- (Frontend UI)
- develop_api_neth (Netherium)
- develop_api_netherium (Netherium)
- develop_api_blockchain (Blockchain)

These branches were merged to develop as they were finalized.

develop was merged to main for the final demo.