Wallet as a Service

Vasile Vlad-Andrei
Rotaru Petru-Alexandru

Project Link:
https://github.com/alexRot622/WaaS

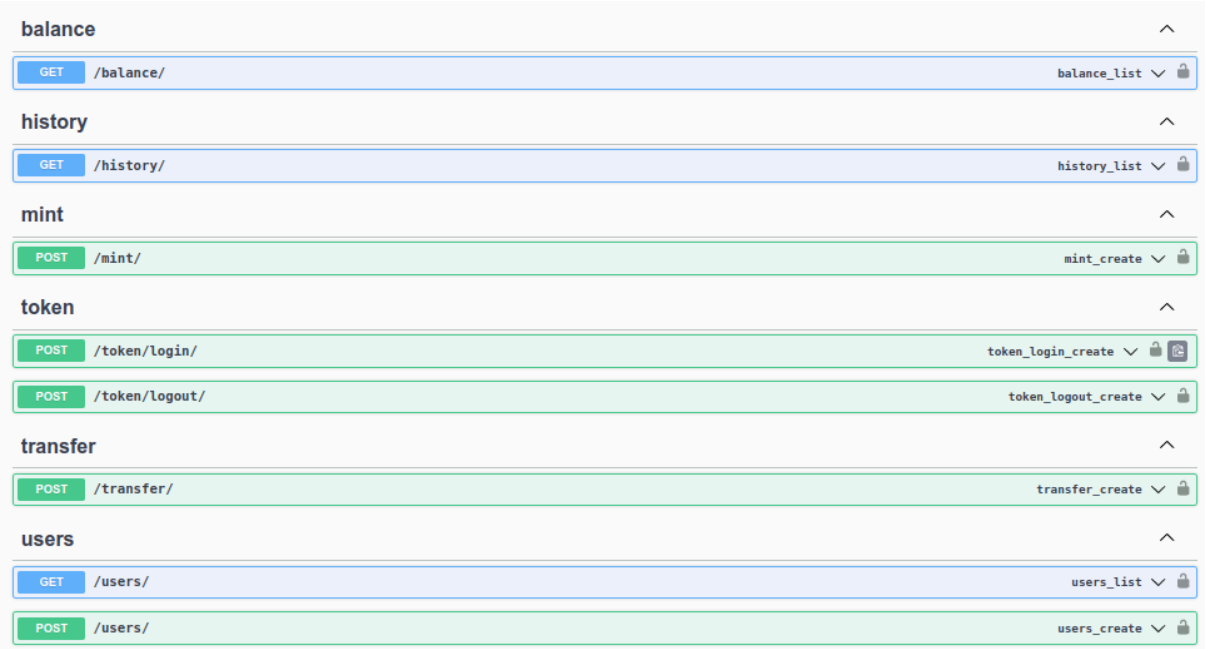
Description
This project implements a “wallet as a service” application, which is capable of creating a MultiversX wallet for each user and interact with a smart contract in order to implement transactions using its own token between users. The users also have access to their transaction history. This wallet service can be easily integrated with other applications through its REST API.

REST API
The REST API was written in Python using Django Rest Framework with a Swagger for ease of access.

In order to use the API, the user will first create an account. When creating an account, the application will also generate a PEM key pair representing that user’s MultiversX wallet, and store it in its database. After logging in, the user will have access to endpoints that interact with the blockchain using their wallet.

The API exposes the following endpoints:

- balance (GET) - view user’s balance
- history (GET) - view user’s transaction history
- login / logout (POST) - logging in and out of an account
- users (POST) - creating an account
- transfer (POST) - transfer a given amount from the user’s account to another user



Application Swagger API

Smart Contract

The smart contract is written in Rust, using the MultiversX SDK for smart contracts. The source code is found in sc/src/waas.rs, where the WalletContract trait is defined. This trait defines a smart contract for the wallet system, and supports the basic functionality necessary for the service, such as minting new tokens, transferring funds, and querying account information. Each wallet may correspond to multiple accounts on this platform.

The following storage mappers are used to represent the smart contract state:

- account_balance: MapMapper<ManagedBuffer, BigUint> - the balance of the account, initially 0.
- history: VecMapper<(BigUint, ManagedBuffer, ManagedBuffer)> - store transactions recorded on the contract, with the format (amount, from, to)

The following endpoints are defined in the smart contract:

- init(): deploys the contract
- upgrade(): upgrades the contract
- add_account(account: ManagedBuffer): creates a new account with the provided account name, with an initial balance of zero.
- mint(account: ManagedBuffer, amount: BigUint): mints new tokens to the specified account (accessible only to the contract owner, which is the service administrator). This action is added to the history, with the recipient being the provided account and the source being “Mint”.
- transfer(from: ManagedBuffer, to: ManagedBuffer, amount: BigUint): transfers tokens from one account to another. In case the transaction is successful, it is added to the history.

- get_account_history(account: ManagedBuffer): view that retrieves the transaction history for a specific account.
- get_account_balance(account: ManagedBuffer): view that retrieves the balance of a specific account.

References:

- MultiversX Rust Developer reference: https://docs.multiversx.com/developers/developer-reference/sc-annotations
- Django reference: https://docs.djangoproject.com/en/5.0/
- Swagger (API documentation): https://django-rest-swagger.readthedocs.io/en/latest/
- Djoser (user management): https://djoser.readthedocs.io/en/latest/getting_started.html