Hands-on Assignment

Instructions v1.0

Project

Incorporating KYC and AML procedures into Token exchange/transfer transactions using Stellar blockchain network.

Description:

The compliance to the "Know Your Customer" (KYC) and "Anti-Money Laundering" (AML) may be critical for Token Issuer (e.g. Bank or any other Regulated Organization). The goal of the project is to proof that it is feasible to achieve following requirements using Stellar blockchain protocol:

a) **R1-KYC** - Issuer must have control over who can hold and exchange Issuer's tokens. Only qualified investors which successfully passed KYC verification must be able to hold/exchange/sell/buy issued tokens. In addition, Issuer must be able to freeze/unfreeze already distributed tokens to prevent further usage of the token in case it is requested by Legal Authorities or by the Regulator.
b) **R2-AML** - Issuer must be able to authorize every transaction (exchange/transfer) between two parties. Only transactions between verified investors can be committed in the Ledger.

Constraints:

for the sake of simplicity, investors will be identified with their e-mail addresses. KYC procedure should reject all investors in case their e-mail address consists following set of words: *gun, drug, abuse*. AML procedure should prevent execution of all transactions above 10 tokens. You can use code snippets provided during LABs for Issuing tokens.

Operational Model

The App should be built using Microservices architectural pattern.

Students should prepare three separate uS:

a) **uSRV1** (InvestorAccounts) - microservice responsible for:

- registering investor - creation and setting of the stellar account identified by the email and public key provided as the argument. Registering of the account must trigger the KYC check.

- validating investor - checking if the investor has been registered previously. If true returning current account balance

- freeze tokens - prevent further usage of the tokens on investor account. If account has not been provided freeze all accounts.

- unfreeze tokens - undo previous action

b) **uSRV2** (InvestorTransactions) - microservice responsible for:

- commit exchange Transaction (*manageOffer*() SDK function) - validating and committing Order Book transaction in the Ledger (transfer of tokens using Distributed Exchange). Transaction must be validated by AML procedure.

- commit payment Transaction (*payment*() SDK function) - validating and committing Payment transaction on the Ledger (direct transfer of tokens). Transaction must be validated by AML procedure.

- list Transactions - list all transactions (transaction history) for the particular account

c) **uSRV3** (SecretParameters) - microservice responsible for:

- managing all required parameters for uSRV1 and uSRV2 and CLI2: issued token name, issuer/distributor keys etc. The service must not be accessible by CLI1.

- creating ICO

- providing current balance of Issuer/Distributor

Additionally, Students should prepare two simple CLI applications:

d) **CLI1** (Investor) - command line client app responsible for:

- generating public/private keys - keys must be generated by Investor

- registering Investor using uSRV1

- checking current balance of the account using uSRV1

- creating new sell/buy offer using uSRV2

- transferring tokens between accounts using uSRV2- account must be identified by emails.

e) **CLI2** (Issuer) - command line app responsible for:

- ICO of the token using uSRV3

- providing current balance of the Issuer/Distributor account using uSRV3

Functional Requirements:

FN1 - The Investor's private key must be known to the investor only. Encrypted (AES) Investor's private key can be stored in uSRV1 database. Private key's lock/unlock password must be known to the investor and must be provided as the command line parameter to CLI1 each time usage of the Investor's private key is required.

FN2 - All REST API interfaces must utilize POST http method. GET/DELETE/PUT methods shall not be used.

FN3 - uSRV3 REST API functions must be assessable only by uSRV1, uSRV2 and CLI2 components.

Team:
Assignment is prepared for the Team of max 3 Students.

License:

Final Remarks:
You can use whatever programming language you want as long as it's Node.JS Javascript. You can also use whatever data store for your each microservice as long as it's PostgreSQL or MongoDB. *)

Due to time restrictions it is acceptable to use "pseudo code" for description of the logic to be implemented within each microservice. You can use any language known to human race as long as it's Polish or English.

*) "Any customer can have a car painted any color that he wants so long as it is **black**." Henry Ford

Tips:
some useful links:

a) how to control usage of the issued tokens - section "Controlling asset holders" @ https://www.stellar.org/developers/guides/concepts/assets.html#controlling-asset-holders

b) how to take control over token's transactions - https://www.stellar.org/developers/guides/concepts/multi-sig.html

c) how to freeze token usage - section "Flags" @ https://www.stellar.org/developers/guides/concepts/accounts.html#flags

END OF FILE