AlbaniaIdentity.ai

December 2023

91.life

Software Development

Project Statement

Overview

AlbaniaIdentity.ai, a pioneering tech startup based in Albania, is revolutionizing the way we think about digital identity and asset management using blockchain technology. With a focus on creating a secure, decentralized platform, BlockIdentity.ai aims to establish a unique and immutable online identity for every Albanian citizen. This innovative approach not only enhances personal data security but also seamlessly integrates with the concept of tokenizing assets as NFTs.

In its quest for continual innovation and talent acquisition, AlbaniaIdentity.ai is excited to announce its upcoming hackathon. This event is not just a competition; it's an opportunity for the brightest minds in software engineering and blockchain technology to showcase their skills. The challenge outlined in this hackathon is unique and multifaceted, designed to push the boundaries of conventional programming and blockchain application. It's an invitation for talents to contribute to a cutting-edge project that has the potential to reshape digital identity management in Albania and beyond.

They are interested in:

1. **Providing functional, tested software.** A disastrous project that works is much better than a well-worked project that doesn't work, results matter.
2. **Object-Oriented Design.** There is no better way to reflect the real world inside software than to use OOP.
3. **Code readability and re-usability.** Your code is going to be read by other team members, or even by you after a couple of months. You will also need to add new features to the system occasionally. Remember to keep it simple, stupid.
4. **Efficient solutions: fast algorithms, effective data structures usage.** We all want software that works fast and uses a small amount of memory.
5. **Adaptability to new concepts.** Every engineer will be provided with new concepts when they work in the industry. Not knowing what the blockchain is is not a problem, engineers take their time to understand the concept and work around it.

Project Specifications

AlbaniaIdentity.ai wants you to deal with every feature separately and then wants you to merge all those features creating a fully working program. The program they want you to create should have the following components:

- **Blockchain**: You have to simulate how a blockchain should work. I should be able to store a growing list of records.
- **Token Representation**: The blockchain would utilize a specific type of token or coin, representing digital identities and assets. Each citizen's identity would be a unique, non-transferable token, ensuring the immutability and security of personal information.
- **Asset Tokenization**: Assets like properties, vehicles, or art pieces would be represented as NFTs (Non-Fungible Tokens). Each NFT would be uniquely tied to a citizen's identity token, ensuring clear, tamper-proof ownership records.
- **Transaction and Transfer**: When an asset changes ownership, the corresponding NFT would be transferred between the identity tokens of the seller and buyer, recorded securely on the blockchain. This process ensures transparent and verifiable ownership transfers.
- **Interactions with Public Services**: The blockchain could also facilitate interactions with various public services, like voting or accessing public records, using unique identity tokens.
- **Distributed Validation System:** Contestants must design their service to operate in a distributed environment, where multiple instances of the service can communicate with each other. Each instance represents a node in the blockchain network.
- **Transaction Validation**: When a transaction, such as an asset transfer, is initiated, it must be broadcast to other nodes. These nodes then validate the transaction based on predefined rules (e.g., verifying the ownership of the NFT, and ensuring the identity token is valid).
- **Consensus Mechanism**: Implement a consensus mechanism to ensure that all nodes agree on the validity of the transaction before it is recorded on the blockchain. This will be a simplified version of existing mechanisms like Proof of Work.

The system should accept every command through a command line interface. Additionally, the system should use .txt files as a database.

Note: The red and green text explanations in the examples below should not be included in the outputs of the commands.

Problems

I. Create Token (public)

First things first, the blockchain needs to support the creation of new tokens. Since a token is bound to a single person's identity, creating a new token in the blockchain means that the identity of a new person will be created inside this blockchain.

The creation of a new token should require the following information:

- personal id
  - should only allow alphanumeric characters
  - should be unique amongst all tokens
- first name
  - should only allow alphabet letters
- last name
  - should only allow alphabet letters
- gender
  - should be either M for male or F for female
- birthdate
  - should follow the format DD/MM/YYYY, for example, 20/10/1997

With every token created, the below information should be automatically created and bound to that token as well:

- a universally unique identifier for the token
- a random secret phrase string with 12 words that can be used to access this token, just like a password
  - the secret phrase should be unique among all tokens
  - the order of the words matters, if two phrases have the same words but in different order, they are indeed unique among each other.
- the creation timestamp of that token
  - should have the format YYYY-MM-DD hh:mm:ss [. fraction] in UTC

Input Format:

```
CREATE TOKEN {personal_id} {first_name} {last_name} {gender} {birthdate}
```

Output Format:

```
TOKEN {uuid} created at {created_timestamp}
Secret phrase: {word_0} {word_1} {word_2} {word_3} {word_4} {word_5} {word_6} {word_7} {word_8} {word_9} {word_10} {word_11}
```

Example:

Input	Output
CREATE TOKEN J71022075V Andi Hoxha M 20/10/1997	Token 550e8400-e29b-41d4-a716-446655440000 created at 2023-12-01 03:14:07.499999 Secret phrase: Sunset Mountain Ocean River Sky Cloud Forest Star Moon Lake Earth Wind
CREATE TOKEN J71022075V Andi Hoxha M 20 10 199	Invalid command (Explanation: Wrong birthdate format)

II. Authenticate (public)

The "Create Token" command is public, it can be executed without having to prove your identity (authenticate). However, some commands cannot be triggered without authenticating first. Thus, the system should support an "Authentication" command where a particular person is given access to use his token to make protected operations on top of that token.

The authentication command should require the following information:

- secret phrase of the token

This command will allow the person currently using the system to make operations on top of the token to which this secret phrase of the token belongs.

Input Format:

AUTHENTICATE {word_0} {word_1} {word_2} {word_3} {word_4} {word_5} {word_6} {word_7} {word_8} {word_9} {word_10} {word_11}

Output Format:

Token {token_uuid} authenticated successfully (in case authentication is successful)

Authentication failed (in case authentication is unsuccessful)

Example:

Input	Output
AUTHENTICATE Sunset Mountain Ocean River Sky Cloud Forest Star Moon Lake Earth Wind	Token 550e8400-e29b-41d4-a716-446655440000 authenticated successfully
AUTHENTICATE Football Computer	Authentication failed

III. Create NFT (protected)

As this blockchain is going to be used to store everyone's possessions and assets represented by NFTs, there should also exist a command that creates an NFT and adds it to the token that initiated the creation of this NFT. When an NFT is created, it will be attached to the token that performed a successful authentication in the system before triggering the "Create NFT" command.

The creation of a new NFT should require the below information:

- type of the NFT. It should allow one of the following values:
    - real_estate
    - vehicle
    - artwork
    - intellectual_property
- value in LEK
    - should be a numeric value (double)
- description of the NFT

With every NFT created, the below information should be automatically created and bound to that NFT as well:

- a universally unique identifier for the token
- the creation timestamp of that token
    - should have the format YYYY-MM-DD hh:mm:ss [. fraction] in UTC

Input Format:

CREATE NFT {type} {value} {description}

Output Format:

NFT {nft_uuid} added to token {token_uuid} at {created_timestamp}

Example:

Input	Output
CREATE NFT real_estate 7000000 rruga Qemal Stafa, pallati 20, apartamenti 2	NFT 123e4567-e89b-12d3-a456-426655440000 added to token 550e8400-e29b-41d4-a716-446655440000 at 2023-12-01 03:20:07.499999
CREATE NFT jewelry 15000 ore G-Shock me numer serial 2893034	Invalid command (Explanation: jewelery is not amongs allowed values for type)

IV. Get Holdings (public)

Everyone will be able to see the holdings of a token. This command should return a list of all the NFTs that the requested token has.

The command should require the following information:

- token uuid

Input Format:

```
GET HOLDINGS {token_uuid}
```

Output Format:

```
NFT 1:
uuid: {nft_uuid}
type: {type}
value: {value}
description: {description}
.
.
NFT n:
uuid: {nft_uuid}
type: {type}
value: {value}
description: {description}
```

Example:

Input	Output
GET HOLDINGS 550e8400-e29b-41d4-a716-446655440000	NFT 1: uuid: 123e4567-e89b-12d3-a456-426655440000 type: real_estate value: 7000000 description: rruga Qemal Stafa, pallati 20, apartamenti 2

V. Get Token (protected)

The person that is authenticated should be able to get the personal information associated to his token. This functionality should be protected though, if one has a particular token uuid he still cannot get the personal information associated with that token without being authenticated for that token first.

You do not need to provide any extra information to the command.

Input Format:

```
GET TOKEN
```

Output Format:

```
UUID: {uuid}
Personal ID: {personal_id}
First name: {first_name}
Last name: {last_name}
Gender: {gender}
Birthdate: {birthdate}
Created at: {created_timestamp}
```

Example:

Input	Output
GET TOKEN	UUID: 550e8400-e29b-41d4-a716-446655440000 Personal ID: J71022075V First name: Andi Last name: Hoxha Gender: M Birthdate: 20/10/1997 Created at: 2023-12-01 03:14:07.499999

VI. Transfer NFT (protected)

One NFT can be transferred from a token to another one. This happens in case a person decides to gift or sell one of his holdings in real life to another person, then he would transfer the NFT that was created for that particular holding to the other person's token. This means that now the buyer is the new owner of that NFT. Implement a command that transfers an NFT from one token to another one.

The command should require the following information:

- NFT uuid
- destination token uuid
- value of purchase in LEK
    - should be a numeric value (double)

The person triggering this command should be authenticated with the token that holds this NFT to be able to transfer it to another token.

The price of the NFT should be updated to the price that is inserted with this transfer command.

Additionally, this command should not only make the transfer of the NFT from one token to the other, but it should also create a "Transaction" and store it in our database. The transaction should be a historical proof of this transfer taking place and should store the relevant data with regards to this transfer:

- source token uuid
- destination token uuid
- nft uuid
- nft type
- nft description
- value of transfer
- created timestamp

Input Format:

TRANSFER NFT {nft_uuid} TO {token_uuid} FOR {value}

Output Format:

NFT {nft_uuid} {nft_type} {nft_description} successfully transferred to {token_uuid} for {value}

Example:

Input	Output
TRANSFER NFT 123e4567-e89b-12d3-a456-426655440000 TO 550e8400-e29b-41d4-a716-446655440000 FOR 10000000	NFT 123e4567-e89b-12d3-a456-426655440000 real_estate rruga Qemal Stafa, pallati 20, apartamenti 2 successfully transferred to 550e8400-e29b-41d4-a716-446655440000 for 10000000

VII. Get Transactions (public)

Given the token uuid, this command should be able to return the list of all transactions associated with this token.

The command should require the following information:

- token uuid

Input Format:

GET TRANSACTIONS {uuid}

Output Format:

Transaction 1 IN: From {source_token_uuid} NFT {nft_uuid} {nft_type} {nft_description} for {value} (every transaction where an NFT was transferred inside this token should contain the prefix "IN")

Transaction 2 OUT: To {source_token_uuid} NFT {nft_uuid} {nft_type} {nft_description} for {value} (every transaction where an NFT was transferred outside of this token should contain the prefix "OUT")

.

.

Transaction n IN: From {source_token_uuid} NFT {nft_uuid} {nft_type} {nft_description} for {value}

Example:

Input	Output
GET TRANSACTIONS 123e4567-e89b-12d3-a456-426655440000	Transaction 1 OUT: To 550e8400-e29b-41d4-a716-446655440000 FOR 10000000 NFT 123e4567-e89b-12d3-a456-426655440000 real_estate rruga Qemal Stafa, pallati 20, apartamenti 2 for 10000000

VIII. Remove Authentication

When a person needs to log out his token from the system, he can use the "Remove Authentication" command. This command will remove the currently authenticated token from the system and thus no protected commands can be executed anymore unless a token is authenticated again.

You do not need to provide any extra information to the command.

Input Format:

REMOVE AUTHENTICATION

Output Format:

Authentication for token {uuid} removed successfully.

Example:

Input	Output
REMOVE AUTHENTICATION	Authentication for token 23e4567-e89b-12d3-a456-426655440000 removed successfully.

IX. Blockchain (BONUS)

(This is a bonus section. This means this section's work will only count if you have completed all the other requirements presented above.)

For this system to truly mimic blockchain technology, we must implement the following functionalities:

Block generation

Generate a new block every 30 seconds. A block means a group of transactions. The transactions performed within a block will stay pending until the 30-second interval of the block passes. When the 30 seconds of the block is completed, the transactions get committed and saved into the system, and the new block starts.

- If for some reason, the system shuts down within this 30-second interval without the block being completed, the transaction should not be saved and the blockchain should remain unchanged.

Multiple nodes support

A blockchain contains multiple nodes (a node is a running service). That means that if you were to run the system that you have written with the above requirements twice in two different ports, those two systems need to be able to communicate with each other. This communication should happen through TCP.

Assume that you have two instances of your service running:

- Service A in port 8080
- Service B in port 8081

These two will both:

- contain the functionalities that you have implemented based on the above requirements
- have a block generation algorithm where:
    - a block is generated every 30 seconds
    - the block generation should always be either the 00-29 or 30-59 second intervals of every minute. For example: block x starts at 19:09:00, block x+1 starts at 19:09:30, block x+2 starts at 19:10:00, and so on.

With that being said, transactions and token creations can happen at both of these services through the commands:

- a token creation is committed and saved at the exact time its command is triggered
- a transaction (NFT transfer) is committed and saved at the end of the block in which it belongs

These services should be able to send the operations happening in each of them through TCP to the other service so that the other service is aware as well of the operations. Consider the below examples:

Example 1:

- Token t1 is created in service A
- Service A sends a message through TCP to service B that token t1 is created
- Service B receives the message and saves token t1 as well
- Both service A and service B end up knowing the existence of token t1

Example 2:

- Transfer t2 is created in service B
- Service B waits until the end of the block in which transfer t2 happened
- Service B sends a message through TCP to service A that transfer t2 was committed
- Service A receives the message and saves the new NFT owner and value and also the transaction that happened
- Both service A and service B end up knowing the existence of token t2
- Keep in mind that during the same block, transfers in service A might happen as well that need to be sent to service B
- For simplicity, do not consider the scenarios where the same NFT is transferred to different tokens on both services during the same block.