University at Buffalo

CSE 526 Blockchain Development

First Name	Abhijeet	Kamal
Last Name	Sugam	Tekwani
Email ID	asugam@buffalo.edu	kamaltek@buffalo.edu
UB Person No.	50468253	50468254

Project Title: Credentials Identification System (The CredAuthentiFi App)

YouTube Link:

1. https://youtu.be/Xby80YHwoTs
2. https://www.youtube.com/watch?v=Avx_NbvhZGs [1]

Dapp url: https://credauthentifi.herokuapp.com [2] **(Removed from Heroku as it was getting charged on weekly basis but the Demo Video is on the Heroku Deployed Application. Suggested by Mr.Sean Sanders on Piazza.)**

Issue(s) Addressed:

One of the significant problems in today's world is the increased population. With the population increase, there is a growth in cybercrimes, which are occurring in a considerable number. One of the major superpowers of cyber criminals is swiping data and using it for personal means. According to the survey, these fraudulent activities are performed by gaining access or pretending to be someone else by hiding their identity or tampering with credentials.

Name of the Aligned Token: CREDCOIN

Token Symbol: CCN

Abstract:

In this modern era of technology, everyone is learning new tech and digital skills. One usually goes for certification to that specific skill by an institution or organization. So, it can make a count in their portfolio. But the story does not end here; this also raises the fraudulent activities related to the credentials acquired by the individual. The solution to the problem is if the credential issuing authority generates a piece of identity information into a block, then, automatically that can be calculated as a transaction and help authenticate credentials through smart contracts. Apart from these functions, we will try to implement some customized tokens as a payment against authentication charges and cashback programs.

Background and Significance:

The most advanced blockchain project idea is a credential authentication system. But unfortunately, as we all know nowadays, fraudulent activities are increasing exponentially due to fake certificates or credentials. Moreover, even verifying each certification by the traditional method is very time-consuming and delays the overall process because verifying credential is one of the crucial phases. So, we have a blockchain-based solution to eliminate all this. A unique decentralized address associated with the certificate is present on the blockchain so that it can be verified by any organization or vendor worldwide. It will compare the decentralized address against the entries in the Blockchain database. If the address matches, it will give details of the validity; otherwise, it will notify that the credential has been tampered with by some fake information.

Currently, Credential verification companies run on a subscription model, and it has monthly and yearly plans. The human resource team, background verification vendors, and everyday users must invest and get a subscription plan. It is a loss for customers as it is not used frequently. In our model, which is pay per transaction model, customers have to pay only for the number of transactions they request. This saves money for customers, and it also opens doors for one-time users because of the pay-per-transaction model, which uplifts Credential verification companies reach as more and more users can interact according to their convenience.

Planning and Implementation:

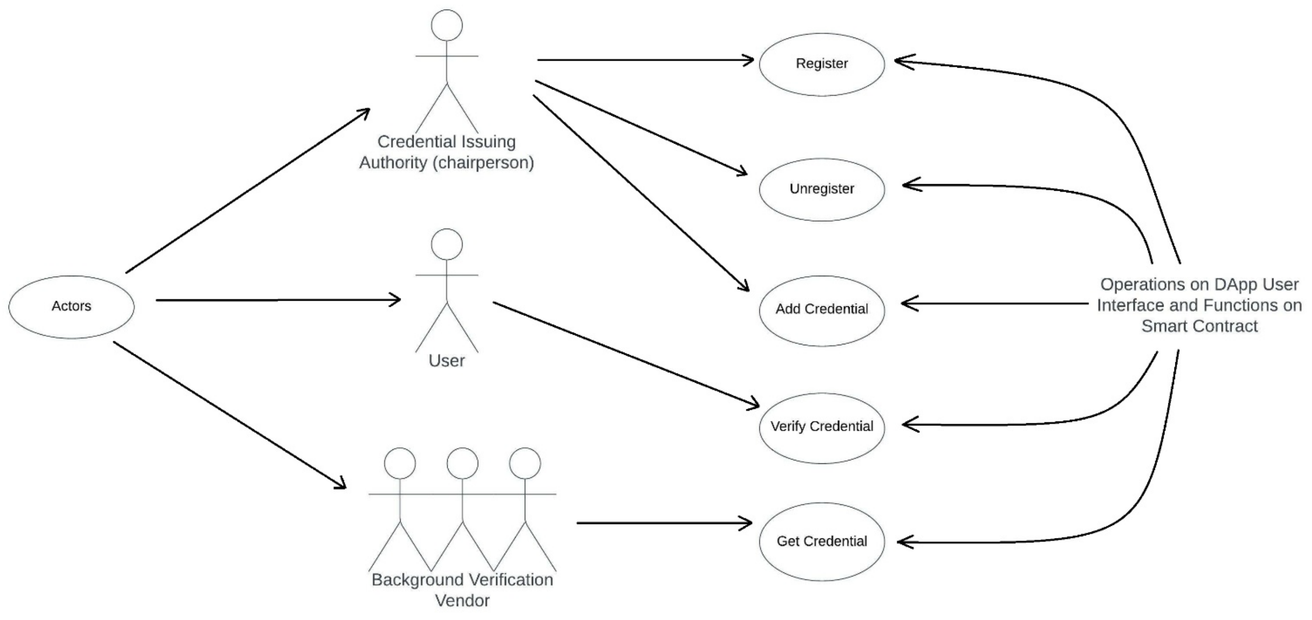
Unlike a standard database, Blockchain has a non-destructive (immutable) way of tracking data changes over time. This means that data is not editable. Instead, a new block is added to the "blockchain" whenever updates are made. This helps track historical data (authenticity and ownership) of a credential. Given the amount of data in credentials to be dealt with (many certificates being issued every day), if we must keep track of all of them, it is better to have a decentralized and distributed network of nodes so that no entity can tamper with the credential data. We also obtain zero latency in accessing the validity and authenticating them. The transparent nature of the blockchain helps avoid parallel trade. Using Blockchain, authenticity and ownership can be checked even if the credential has expired

Instruction to run the Project:

(1) Open URL: https://remix.ethereum.org/ in the chrome browser for the development of smart contract.
(2) Create a new solidity file named as 'CredAuth.sol' in the contracts folder and develop the smart contract by following proper design patterns.
(3) Compile the smart contract by selecting the compatible compiler.
(4) Now, our smart contract is ready for deployment, but there are some configuration changes we need to perform such as choosing the current environment to Injected Provider – Metamask and selecting the account from which the deployment needs to be performed in account section and finally click on the deploy button.
(5) After pressing deploy, a Meta mask Wallet is popped-up which have all the details of operation performed on the Meta mask. This popup reflects the gas fees charges and ask for user decision to go ahead with transaction or reject it.

(6) After successful deployment of smart contract, a unique address of smart contract has been generated at the blockchain and it will be visible in the 'Deployed Contracts' section on 'Remix IDE'.
(7) Copy the address of the deployed smart contract and paste it in the address attribute of App Configuration, which is present in 'app.js'.
(8) Run the front-end application also known as Decentralized Application (DApp) by running the below mentioned commands and try to perform some actions to ensure that the Application is up and working fine.

Use Case Diagram:


Credential Issuing Authority (chairperson)
Actors
User
Background Verification Vendor
Register
Unregister
Add Credential
Verify Credential
Get Credential
Operations on DApp User Interface and Functions on Smart Contract

This system has three entities: credentials issuing authority, user, and background verification vendor. The credential issuing authority acts as a chairperson, which has the privilege to register a user and a vendor and add a credential to the Blockchain. The user is an individual owning the certification and can check the validity of the certificate. Finally, the Background verification vendor fetches all the relevant details of the credentials.

Contract Diagram :

CredAuth.sol

address chairperson;
 	
struct credential { string status, string university, string zip_code, string issue_date, string expiry_date, string credential_id , string credential_title, string issued_to)
 	
mapping(address => credential) cred;
 	
IERC20 private _token;
 	
address[] credential_list;

mapping (address => uint) public registered;

modifier onlyChairperson()
modifier onlyUser()
modifier onlyVendor()

constructor (IERC20 token) payable public

function registerUser(address user) onlyChairperson public
function registerVendor(address vendor) onlyChairperson public
function unregister(address member) onlyChairperson public

function addCredential(address _address,string _status, string _university,string _zip_code, string _issue_date,string _expiry_date, string _credential_id,string _credential_title, string _issued_to) onlyChairperson public

function getCredentialList(address _address) public view onlyVendor
function validateCredential(address _address) public view onlyUser

function confirmVendorPayment() onlyVendor payable public
function confirmUserPayment() onlyUser payable public

CredToken.sol

string public constant name

string public constant symbol

uint8 public constant decimals = 2;

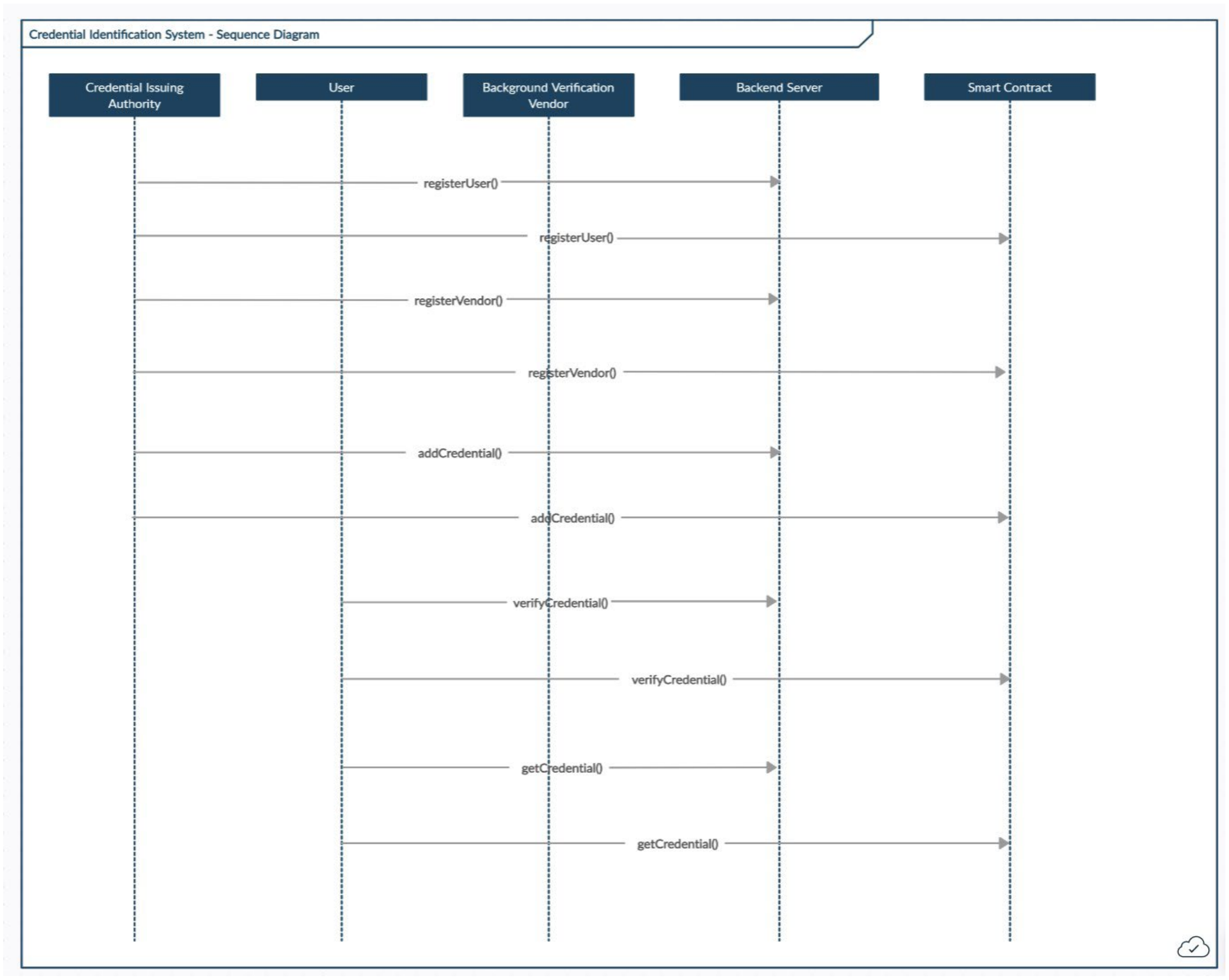
address ERCowner;

mapping(address => uint256) balances;

uint256 totalSupply_;

function totalSupply() public view returns (uint256)
function balanceOf(address tokenOwner) public view returns (uint)
function transfer(address receiver, uint numTokens) payable public returns (bool)
function transferFrom(address owner, address buyer, uint numTokens) public returns (bool)
function close() public
function sub(uint256 a, uint256 b) internal pure returns (uint256)
function add(uint256 a, uint256 b) internal pure returns (uint256)

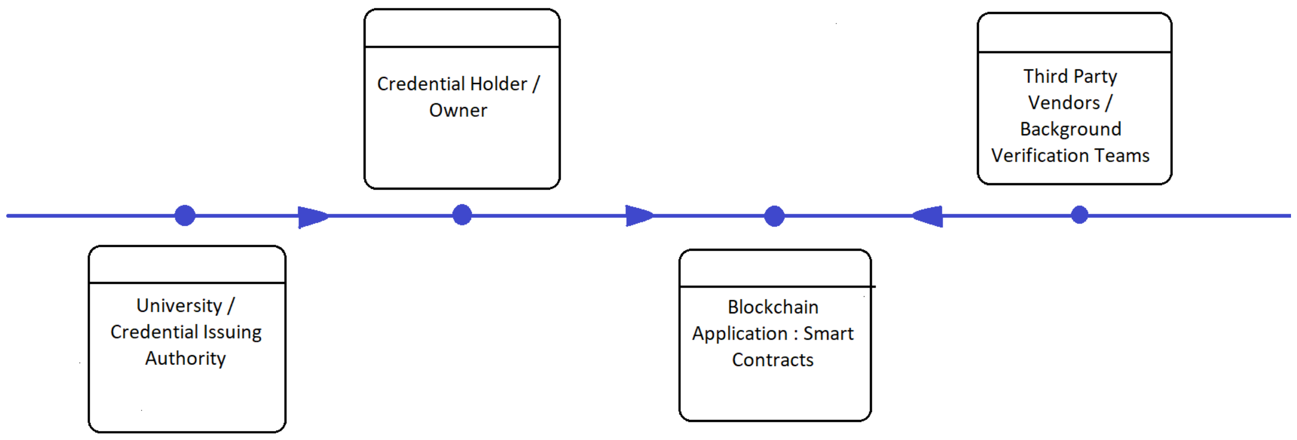
Sequence Diagram:



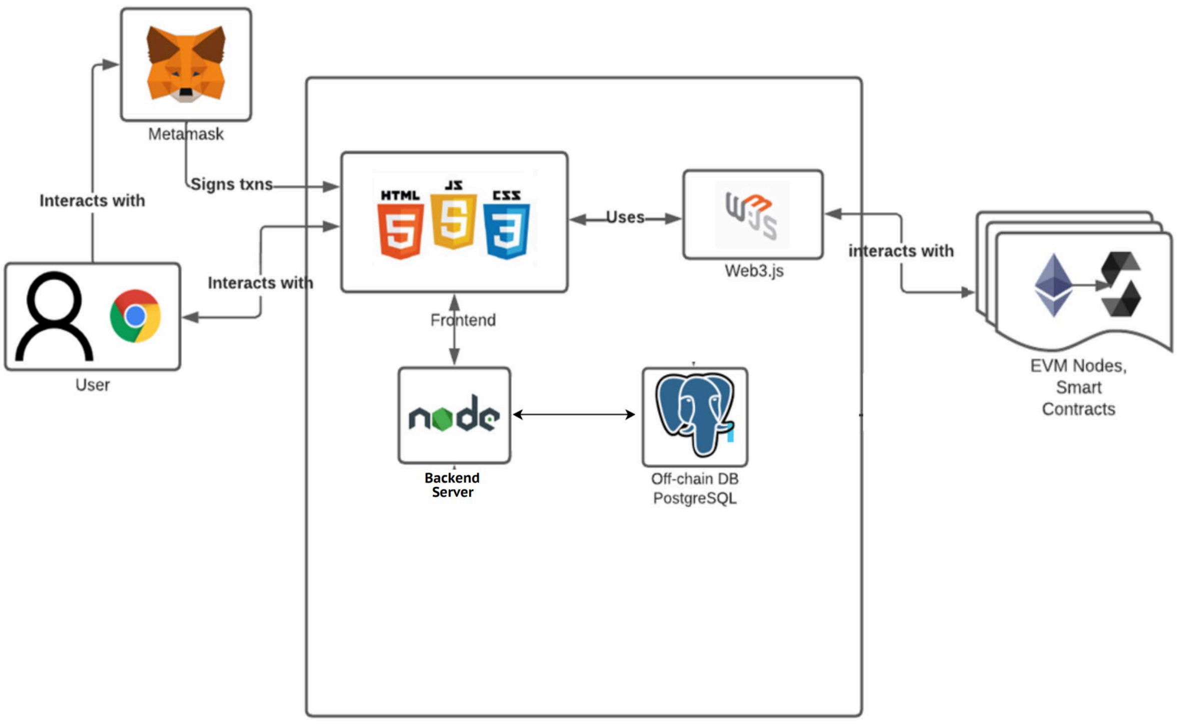
Quad Chart Diagram:

Use case: Credentials Identification System (CIS) **Problem Statement: A** decentralized blockchain-based system for verification and authentication of credentials.	**Issues with existing centralized model:** 1. Inadequate and inefficient response to users. 2. Inefficient verification and higher costs to users and background verification team. 3. Compulsory membership plan for verification. 4. No model for payment settlement for single transactions. **5.** Frauds by exploiting databases by tampering the credentials.
Proposed blockchain-based solution: 1. Verification can be done by users and background verification system. 2. Paying for verification and saving on subscription model. 3. Create a decentralized ticketing system which will have lower fees.	**Benefits:** 1. Saves time, effort, and cost for the users. 2. Better customer experience. 3. Efficient use of Verification system. 4. This idea will help users to save on unwanted payments of monthly or yearly subscription. 5. This idea will also increase the revenue of service providers since a substantial number of users will be willing to pay on a pay per verification basis. 6. More User Interaction.

Data Flow Diagram:


Credential Holder / Owner
Third Party Vendors / Background Verification Teams
University / Credential Issuing Authority
Blockchain Application : Smart Contracts

High Level Architecture Diagram :



Token Design:

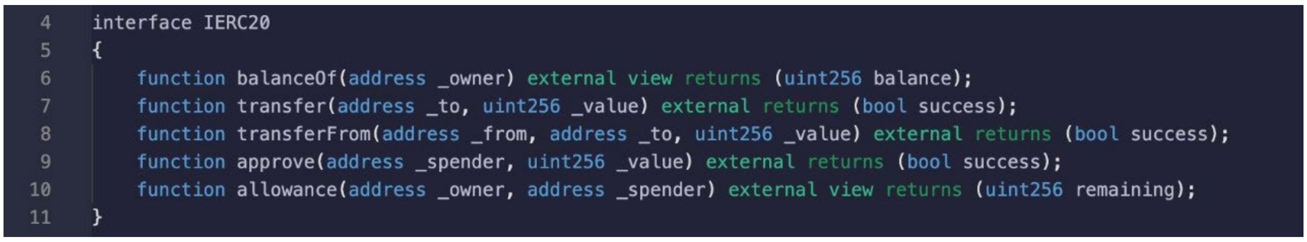
We have implemented ERC20 token for each credential transaction that a user or vendor fetches. The token is fungible. We named it as ***CredCoin*** as it goes with word credential.

ERC20 Token Implementation:

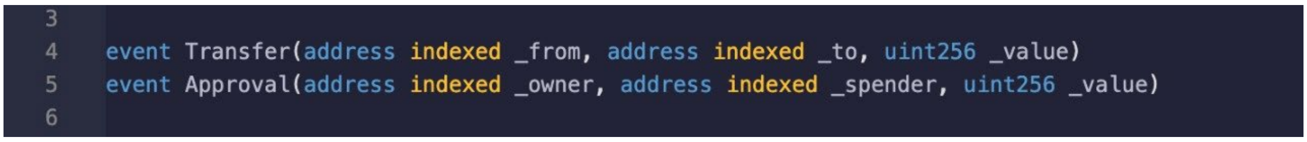
The technical specification for fungible tokens produced on the Ethereum network is called ERC-20. A fungible token is one that can be exchanged for another token in situations when the well-known non-fungible tokens (NFTs) cannot. Different tokens with smart-contract functionality can be exchanged thanks to ERC-20. Tokens are a representation of something that is not unique in and of itself but can be transferred, such as an asset, right, ownership, access, or cryptocurrency. The standard permits the exchange of tokens that represent one of these elements for another element as well as the use of smart contracts. Smart contracts are coding conditions that carry out several facets of a transaction between parties.

It basically is a smart contract that follows some sort standard which is recognized by the Ethereum virtual machine (EVM).

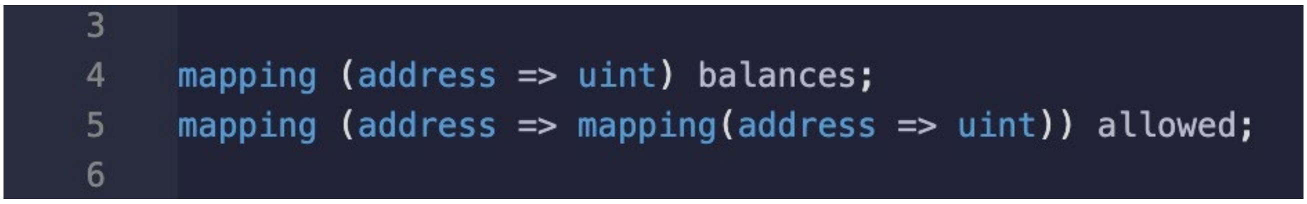
These are some of the mandatory functions that you need include into smart contract to be recognized as erc20 token. All These functions are wrapped as interface. Using interface keyword instead of contract makes it abstract it simple means that all your functions visibility is supposed to be external or else the compiler will throw an error.



```
interface IERC20
{
    function balanceOf(address _owner) external view returns (uint256 balance);
    function transfer(address _to, uint256 _value) external returns (bool success);
    function transferFrom(address _from, address _to, uint256 _value) external returns (bool success);
    function approve(address _spender, uint256 _value) external returns (bool success);
    function allowance(address _owner, address _spender) external view returns (uint256 remaining);
}
```

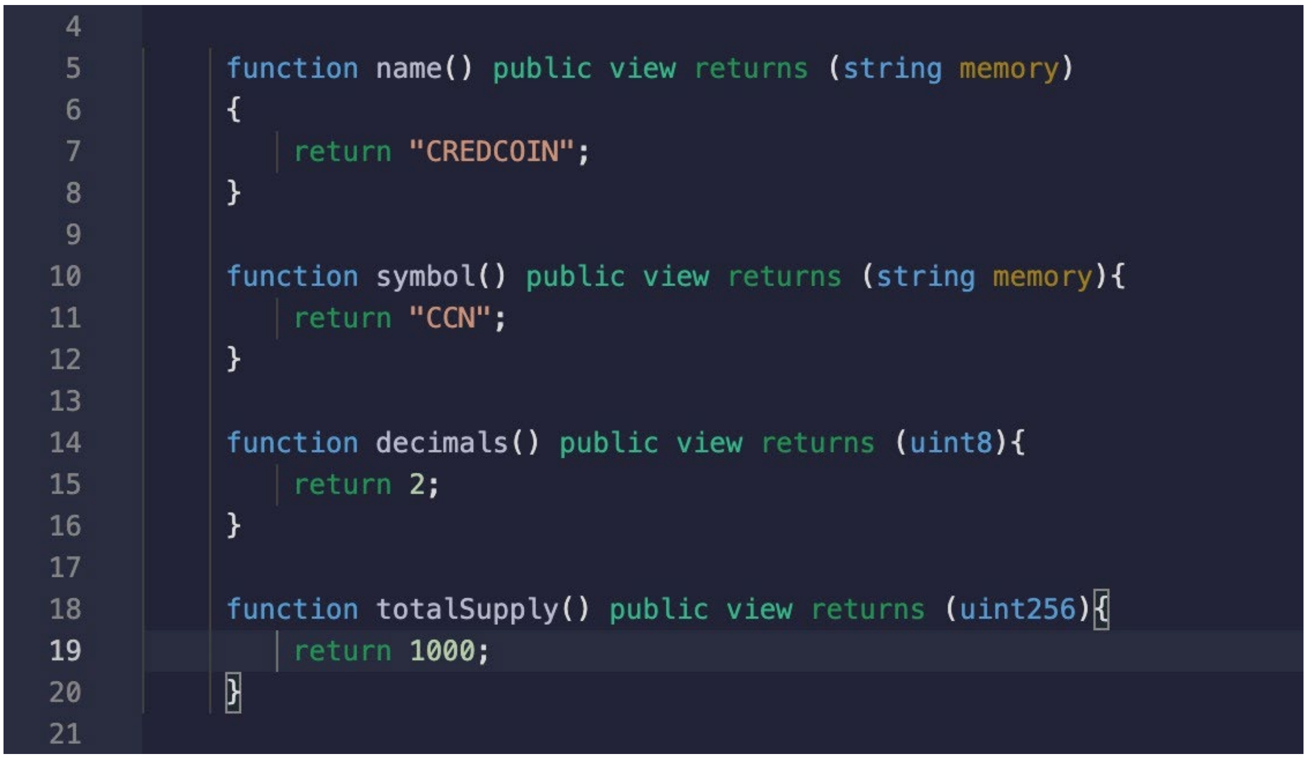


```

event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)

```

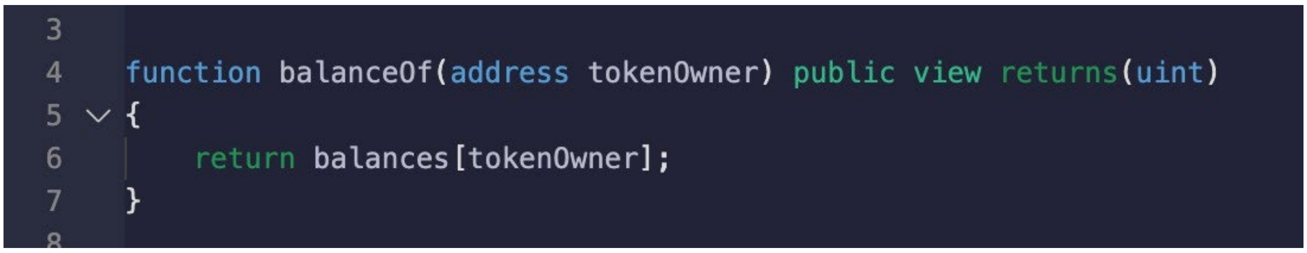


```

mapping (address => uint) balances;
mapping (address => mapping(address => uint)) allowed;

```

We have two different mapping types; the first, called balances, will store the token balance of each owner's account, and the second, called allowed, is a nested mapping that contains all the accounts that have been given permission to withdraw money from a particular account, along with the maximum withdrawal amount permitted for each.

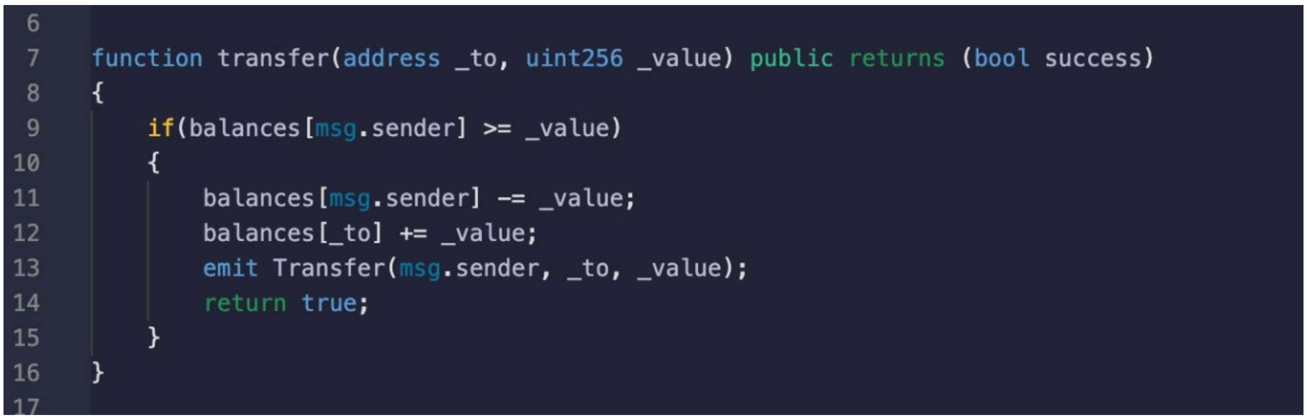


```

    function name() public view returns (string memory)
    {
        return "CREDCOIN";
    }

    function symbol() public view returns (string memory){
        return "CCN";
    }

    function decimals() public view returns (uint8){
        return 2;
    }

    function totalSupply() public view returns (uint256){
        return 1000;
    }

```



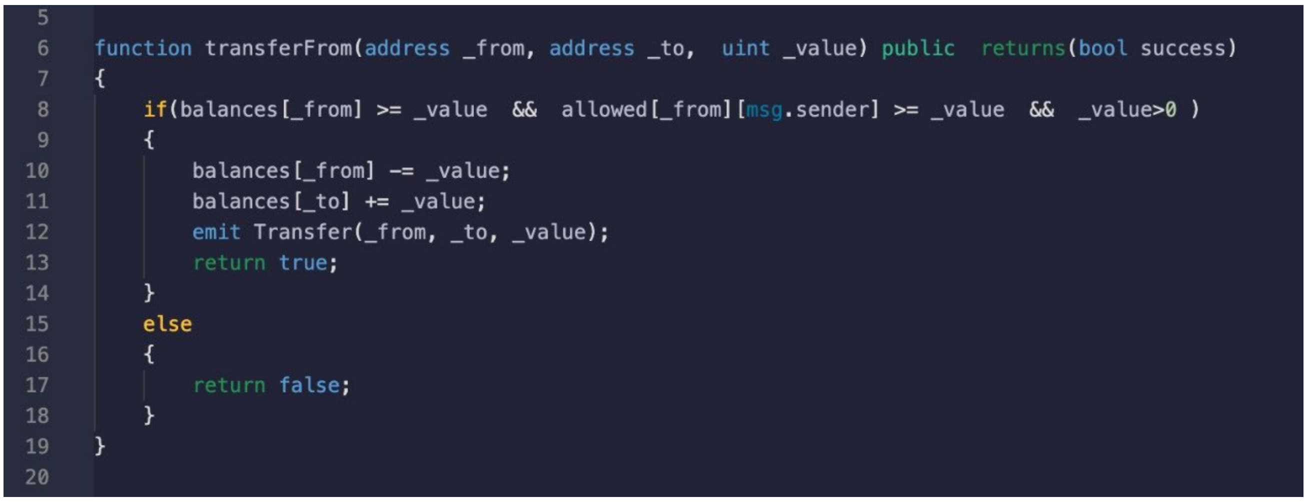
```
function balanceOf(address tokenOwner) public view returns(uint)
{
    return balances[tokenOwner];
}
```

balanceOf() function takes one parameter tokenOwner and returns the balance of the owner.



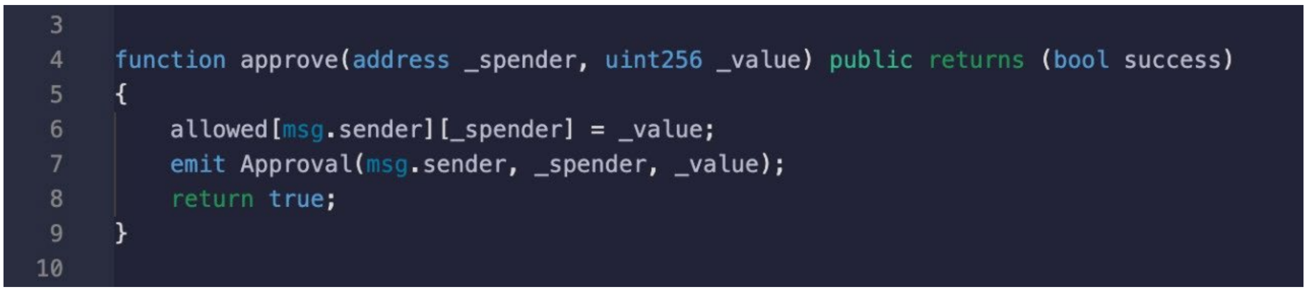
```
function transfer(address _to, uint256 _value) public returns (bool success)
{
    if(balances[msg.sender] >= _value)
    {
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
        return true;
    }
}
```

If the sender's balance exceeds the amount we want to send, transfer() transfers the values. This requirement is crucial because the sender's account must have more money in it than he intends to spend for the transaction to succeed. In this function, we have also fired an event emit.



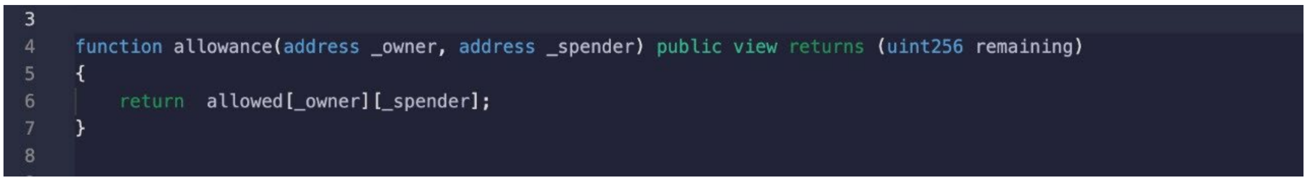
```
function transferFrom(address _from, address _to,  uint _value) public  returns(bool success)
{
    if(balances[_from] >= _value  &&  allowed[_from][msg.sender] >= _value  &&  _value>0 )
    {
        balances[_from] -= _value;
        balances[_to] += _value;
        emit Transfer(_from, _to, _value);
        return true;
    }
    else
    {
        return false;
    }
}
```

Contracts can transmit tokens on your behalf thanks to the transferFrom() method. It verifies that the amount to be spent is not equal to zero and that both the account being used for the transaction and the balance of the spender's account are more than zero. The transfer event is also fired.



```
function approve(address _spender, uint256 _value) public returns (bool success)
{
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

The authorize () method determines if the specified address is permitted to use contract funds or not. Additionally, we have activated the "Approval" event to communicate with the front end.



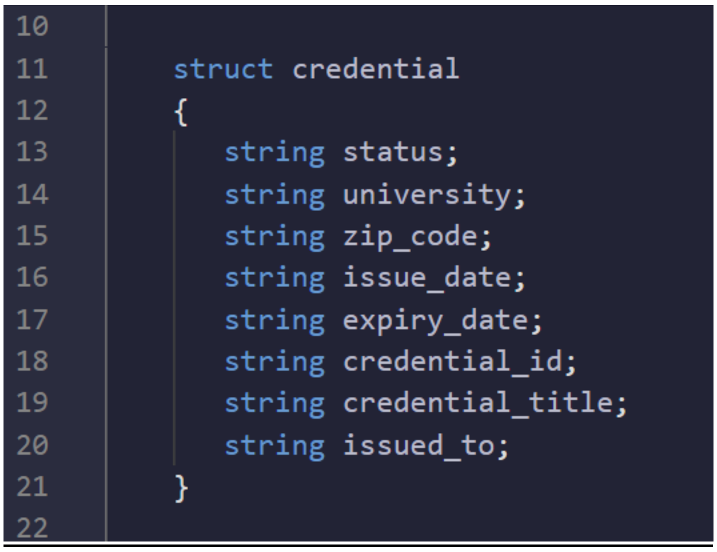
```
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
{
    return  allowed[_owner][_spender];
}
```

This function allowance determines whether the address is authorized to make purchases on the owner's behalf. In this case, the owner has the power to authorize someone else to spend his ether on his behalf and to control how that person spends.

DESCRIPTIONS:

(1) Used Data Structures:

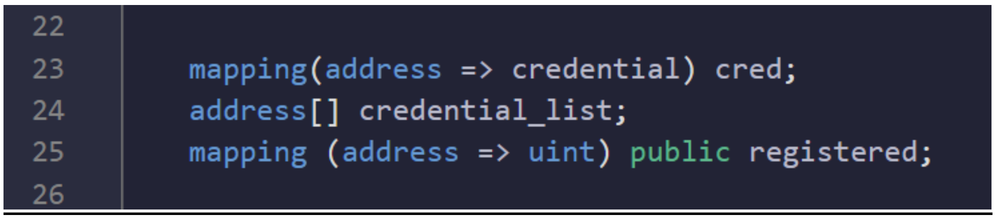
struct datatype :



```
struct credential
{
    string status;
    string university;
    string zip_code;
    string issue_date;
    string expiry_date;
    string credential_id;
    string credential_title;
    string issued_to;
}
```

An object representing a record is a structure type. The keyword "struct" creates a brand-new data type with many members. Here, we have declared struct datatype with attributes such university, status, issue date, issued to etc.

mappings :

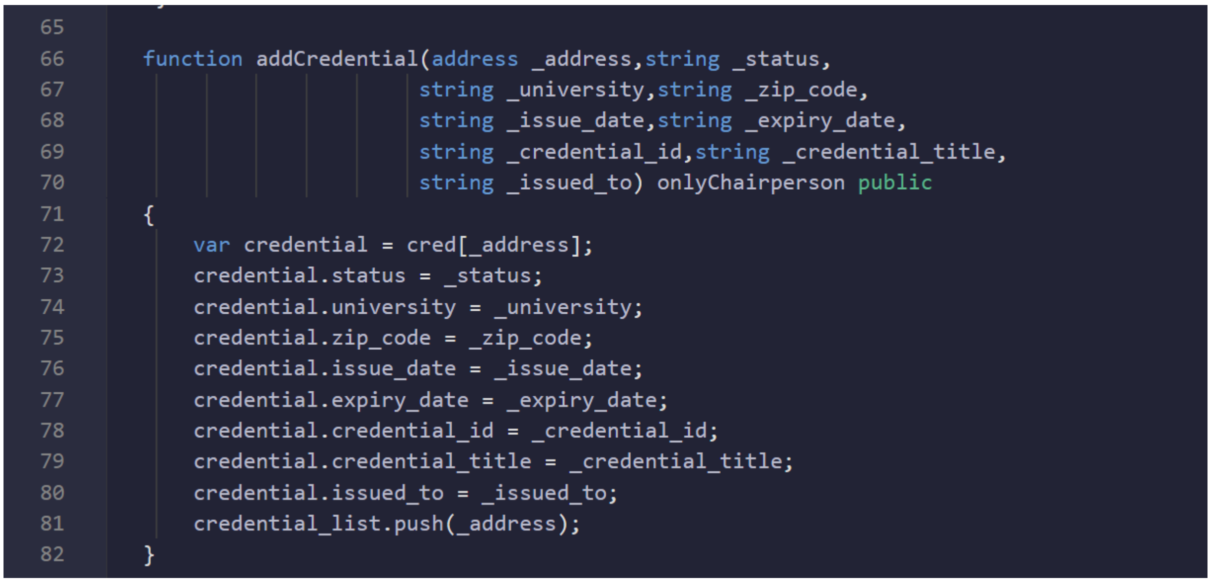


```
mapping(address => credential) cred;
address[] credential_list;
mapping (address => uint) public registered;
```

We have used an array data structure in order to store a fixed-size sequential collection of credentials and also described Mapping as a reference type as structs.

(2) Smart Contract Functions:

(a) Method: addCredential()



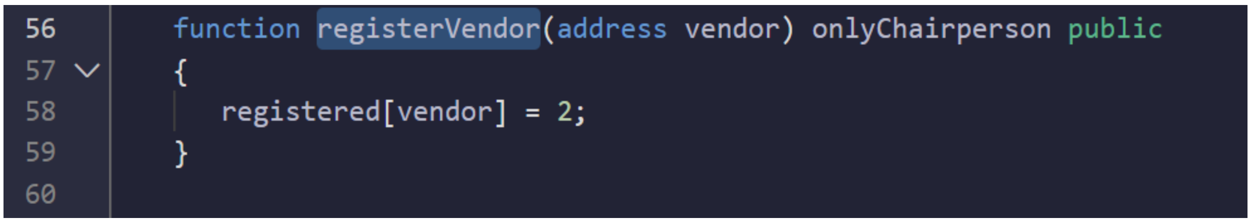
```
function addCredential(address _address,string _status,
                        string _university,string _zip_code,
                        string _issue_date,string _expiry_date,
                        string _credential_id,string _credential_title,
                        string _issued_to) onlyChairperson public
{
    var credential = cred[_address];
    credential.status = _status;
    credential.university = _university;
    credential.zip_code = _zip_code;
    credential.issue_date = _issue_date;
    credential.expiry_date = _expiry_date;
    credential.credential_id = _credential_id;
    credential.credential_title = _credential_title;
    credential.issued_to = _issued_to;
    credential_list.push(_address);
}
```

This method is used to capture all the details of the credentials. Then, all the captured details are encapsulated into a single variable of address type, which is mapped to the hash table. In the smart contract, we have defined the array data structure, which contains all the credentials on the blockchain. This method can only be invoked by Credential Issuing Authority (CIA), as this will be the entry point of credentials on the blockchain data. So, it can be verified by other users and other background verification vendors.

Input Parameters:

Address:	Decentralized Address of the credential.
Status:	Certificate is valid or invalid.
University:	University which issued the certificate.
Zip code:	Zip code of the university.
Issue date:	Issue date of the Certificate
Expiry date:	Expiry date of the certificate
Credential id:	ID of the credential
Credential Title:	Title of the Certificate issued.
Issued to:	The person under whose name the certificate is issued.

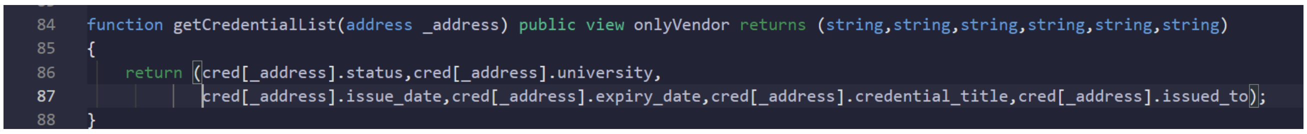
(b) Method : registerVendor() :



```
function registerVendor(address vendor) onlyChairperson public
{
    registered[vendor] = 2;
}
```

Only the chairperson can call the function registerVendor(), which takes as an input argument the decentralized address of the vendor or background check team.

(c) Method : getCredentialList ()



```
function getCredentialList(address _address) public view onlyVendor returns (string,string,string,string,string,string)
{
    return (cred[_address].status,cred[_address].university,
            cred[_address].issue_date,cred[_address].expiry_date,cred[_address].credential_title,cred[_address].issued_to);
}
```

Based on the address that the background verification team has provided, this approach is utilized to retrieve all the key information about the credential. The structure datatype "credential" stores information about the certificates. It is retrieved from the type array data structure for the specific address that the vendor entered. The address has been turned into a key so that it may be used to search for credentials using a key-value pair that contains information about the certificate, such as the name of the university that issued the credential, the date it was issued, when it expires, who the recipient is, etc. Since it will be used to retrieve the relevant details about the credential from the blockchain, this approach can only be utilized by the vendor providing background checks.

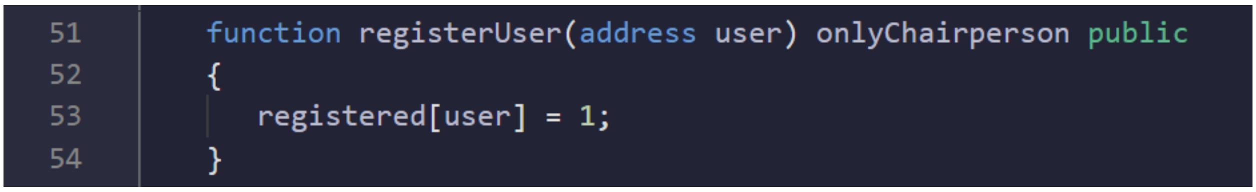
Input Parameters:

Address: Decentralized Address of the credential.

Output Parameters:

University:	University which issued the certificate.
Issue date:	Issue date of the Certificate
Expiry date:	Expiry date of the certificate
Status:	Certificate is valid or invalid.
Credential Title:	Title of the Certificate issued.
Issued to :	The person under whose name the certificate is issued.

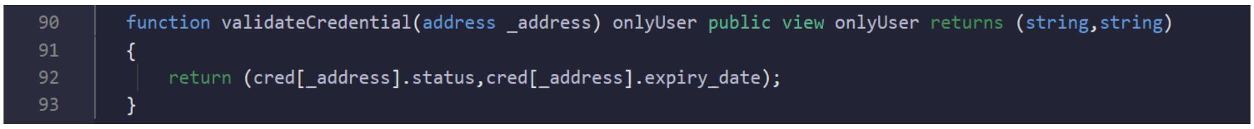
(d) Method: registerUser()



```
function registerUser(address user) onlyChairperson public
{
    registered[user] = 1;
}
```

Only the chairperson can call the function registerUser(), which accepts a decentralized address of the user as an input parameter.

(e) Method : validateCredential()



```
function validateCredential(address _address) onlyUser public view onlyUser returns (string,string)
{
    return (cred[_address].status,cred[_address].expiry_date);
}
```

This procedure is used to check the legitimacy of the credential given to the individual. The structure datatype "credential" stores information about the certificates. It is retrieved from the type array data structure for the specific address that the vendor entered. We have assigned the address a key so that it can be used to search the credential based on the key-value combination that contains the status, issue date, and expiration date of the certificate. Because it will be used to retrieve the pertinent information about the credential from the blockchain, this method can only be used by the user who owns the credential.

<u>Input Parameters:</u>

<u>Address:</u> Decentralized Address of the credential.

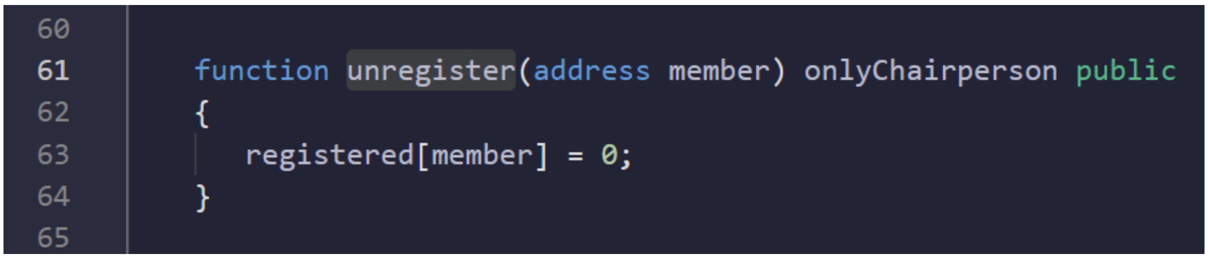
<u>Output Parameters:</u>

<u>Status:</u> Certificate is valid or invalid.

<u>Issue date:</u> Issue date of the Certificate

<u>Expiry date:</u> Expiry date of the certificate

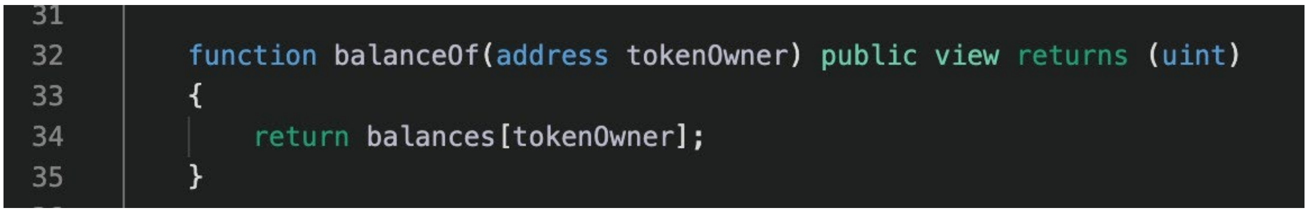
(f) Method : unregister()



```

function unregister(address member) onlyChairperson public
{
    registered[member] = 0;
}

```

Unregister() is a function that can only be used by the chairperson and takes a user's decentralized address as an input parameter. It takes away each entity's rights within the smart contract.

(g) Method : balanceOf()



```
    function balanceOf(address tokenOwner) public view returns (uint)
    {
        return balances[tokenOwner];
    }
```

balanceOf() is a function in CredToken.sol that fetches the available balance in terms of cred coin of a specific user or vendor. *Refer Fig 4 in Application Screenshot Section*

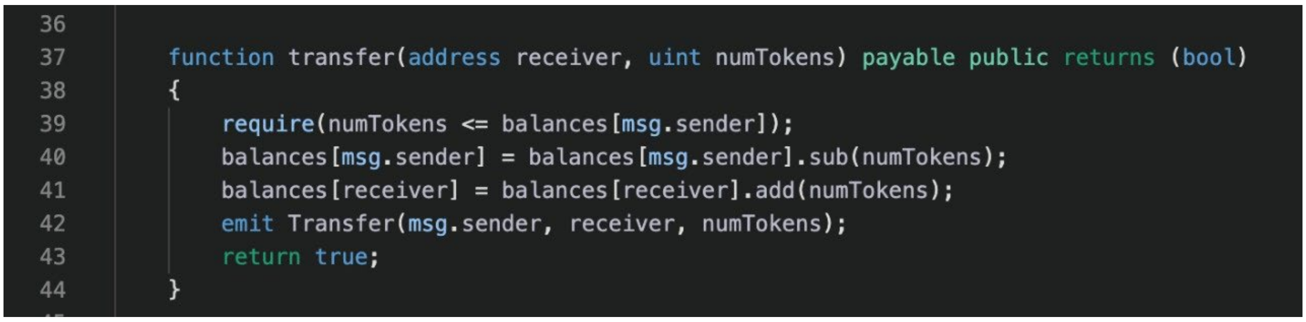
<u>Input Parameters:</u>

<u>Address:</u> Decentralized Address of an entity (User, Vendor, Credential Issuing Authority).

<u>Output Parameters:</u>

<u>Balance:</u> Available Balance.

(h) Method: transfer()



```
    function transfer(address receiver, uint numTokens) payable public returns (bool)
    {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }
```

transfer() is a function in CredToken.sol that transfer the requested amount from source account to the beneficiary account address. To process this transaction, requested amount should be less than or equal to the available balance in the source account. This method is used explicitly to transfer the cred tokens (CredCoin - CCN) between the entities. *Refer Fig 5 in Application Screenshot Section*

<u>Input Parameters:</u>

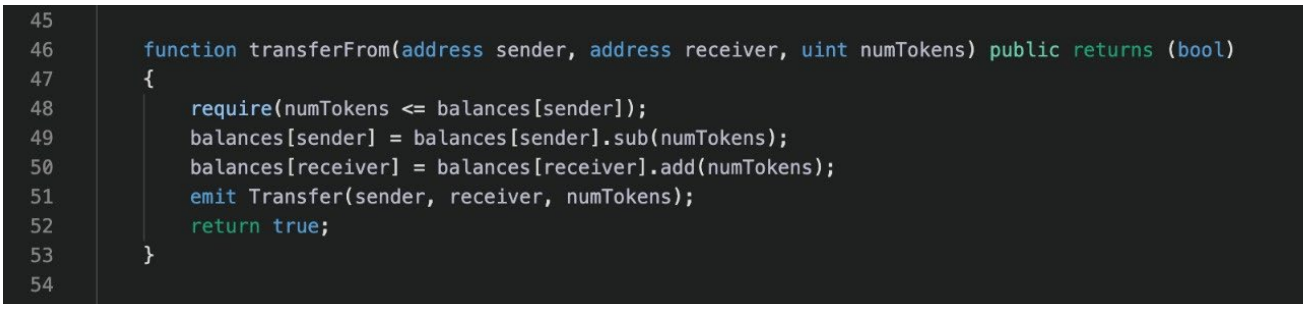
<u>Address:</u> Decentralized Address of the receiver entity (User, Vendor, Credential Issuing Authority).

<u>numTokens:</u> Number of tokens to be transferred.

Output Parameters:

Boolean: Returns true in case of transaction successful.

(i) Method: transferFrom()



```

    function transferFrom(address sender, address receiver, uint numTokens) public returns (bool)
    {
        require(numTokens <= balances[sender]);
        balances[sender] = balances[sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(sender, receiver, numTokens);
        return true;
    }

```

transferFrom() is a function in CredToken.sol that transfer the requested amount from source account to the beneficiary account address. This method is invoked when user or vendor initiates payment against fetching or validating a specific credential. To process this transaction, payment amount should be less than or equal to the available balance in the user or vendor account. This method is used explicitly to transfer the cred tokens (CredCoin - CCN) from vendor or user to the Credential Issuing Authority.

Input Parameters:

Address (Sender): Decentralized Address of the vendor or user.

Address (Receiver): Decentralized Address of the Credential Issuing Authority.

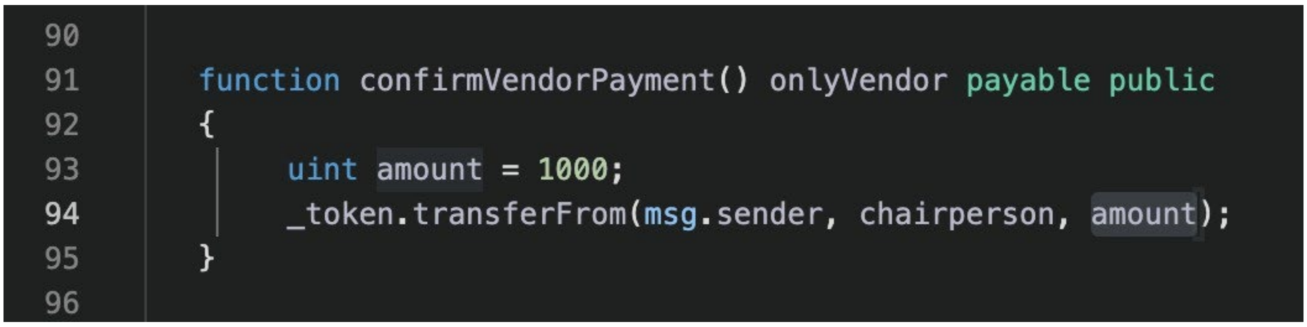
numTokens: Payment amount (token) to be transferred.

numTokens: Number of tokens to be transferred.

Output Parameters:

Boolean: Returns true in case of transaction successful.

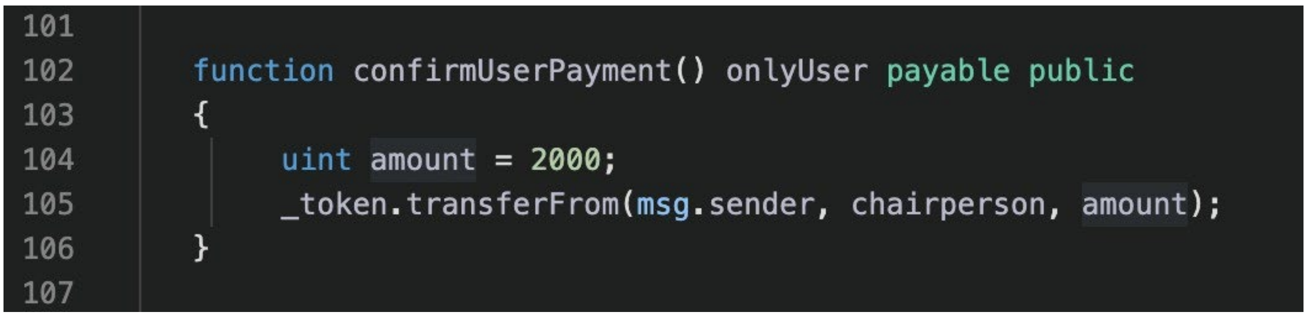
(j) Method: confirmVendorPayment()



```

function confirmVendorPayment() onlyVendor payable public
{
    uint amount = 1000;
    _token.transferFrom(msg.sender, chairperson, amount);
}

```

confirmVendorPayment () is a function in CredAuth.sol that will initiate the payment for Vendor in order to fetch the credential. *Refer Fig 8 in Application Screenshot Section*

(k) Method: confirmUserPayment ()

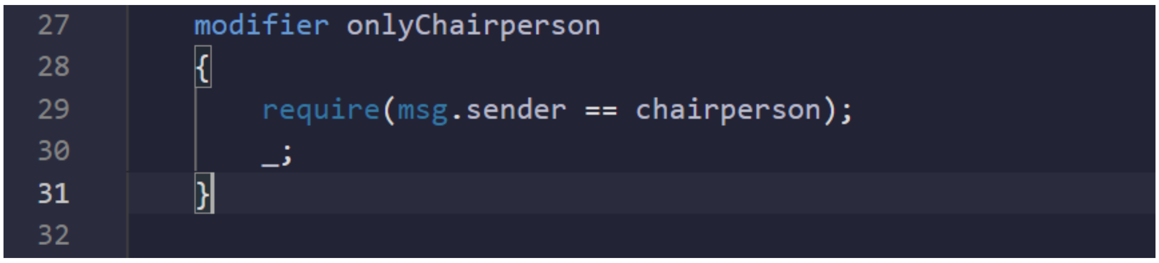


```

function confirmUserPayment() onlyUser payable public
{
    uint amount = 2000;
    _token.transferFrom(msg.sender, chairperson, amount);
}

```

confirmUserPayment () is a function in CredAuth.sol that will initiate the payment for user in order to validate the credential. *Refer Fig 7 in Application Screenshot Section*

(3) Modifiers:

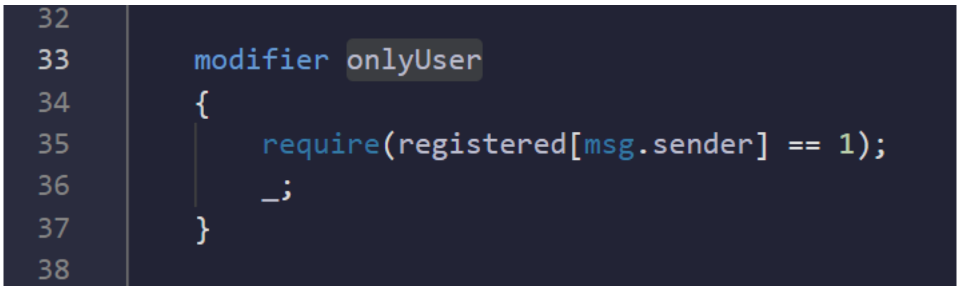
(1) onlyChairperson :



```
    modifier onlyChairperson
    {
        require(msg.sender == chairperson);
        _;
    }
```

The credential is issued to the recipient or an individual by the credential issuing authority, which is referred to as the chairperson. This modifier is restricting other users or actors to access a particular method on which it is applied to.

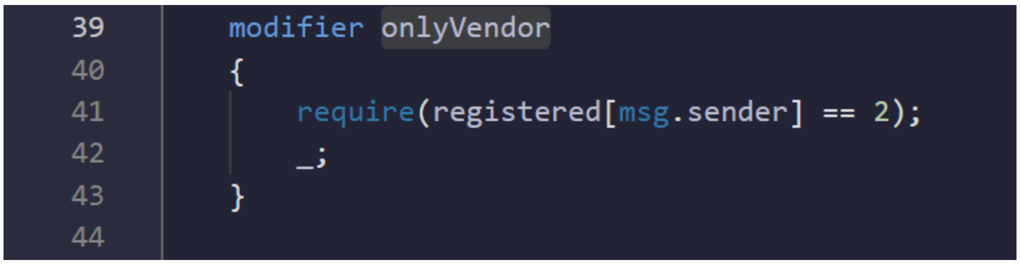
(2) onlyUser:



```
    modifier onlyUser
    {
        require(registered[msg.sender] == 1);
        _;
    }
```

An individual who owns the credential is referred to as the user by an entity. This modifier limits access for other users and entities to the specific method to which it is applied.

(3) onlyVendor:

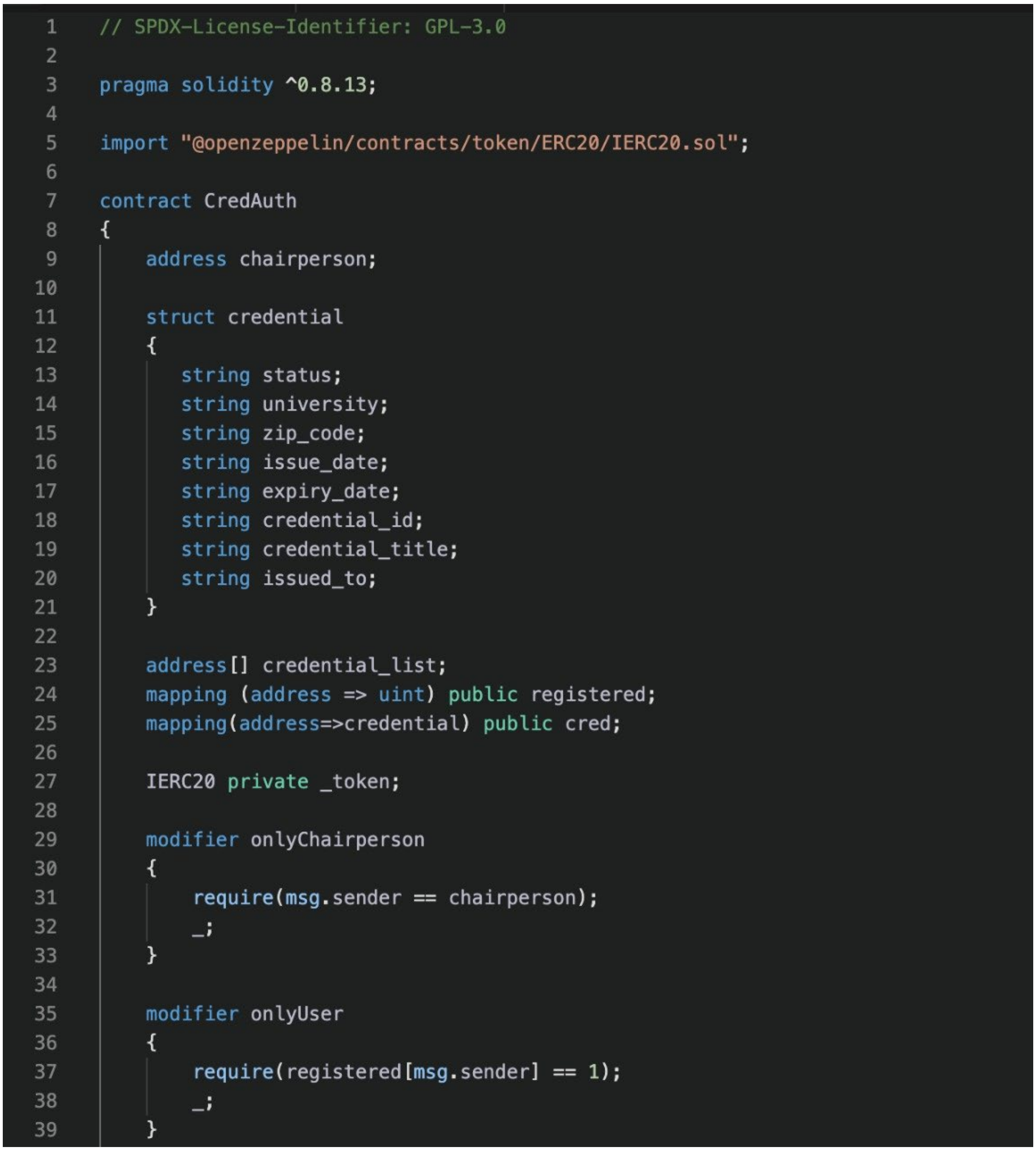


```
    modifier onlyVendor
    {
        require(registered[msg.sender] == 2);
        _;
    }
```

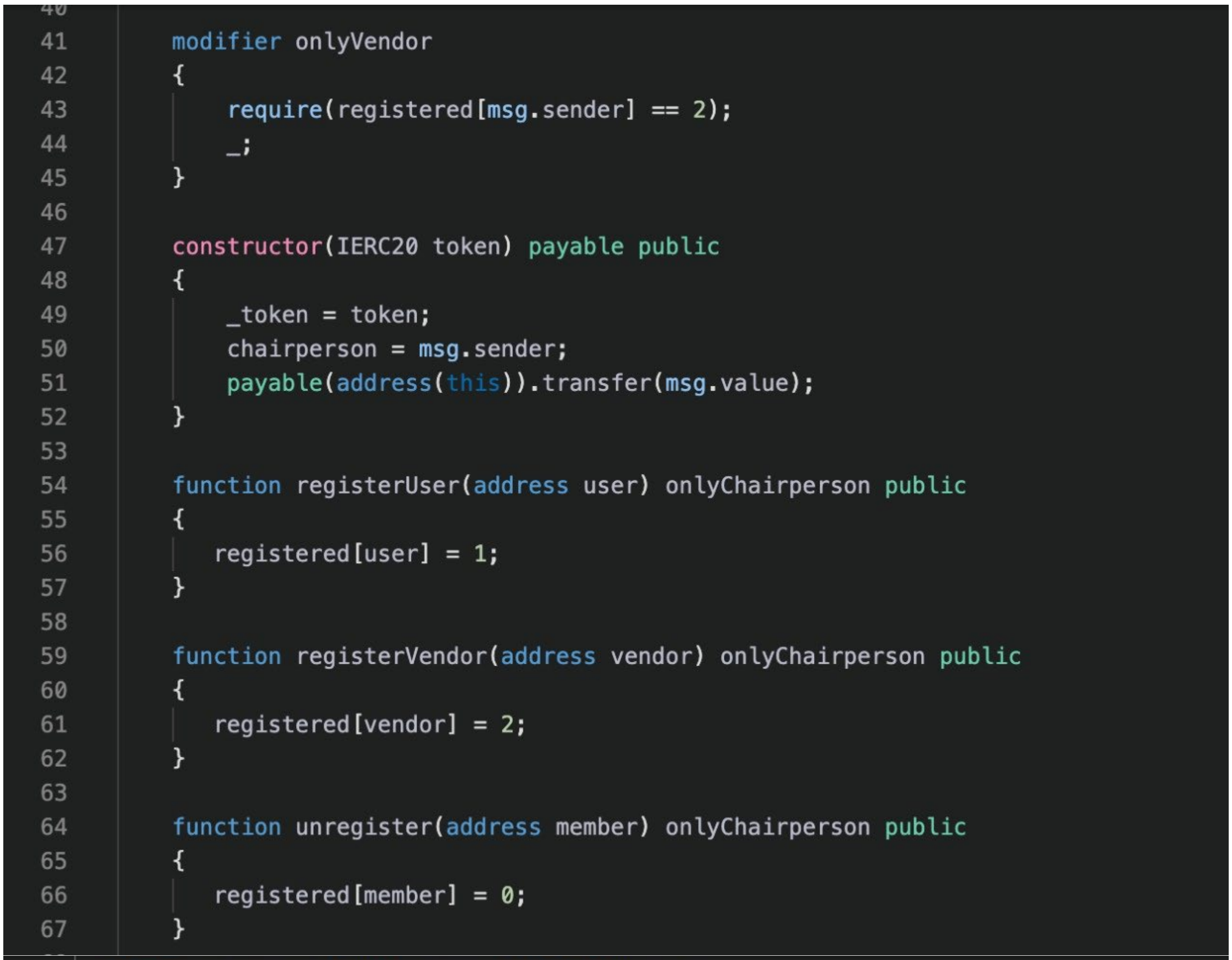
An organization that validates the integrity of the credential that users claim to possess, is referred to as the vendor.

Solidity Code:

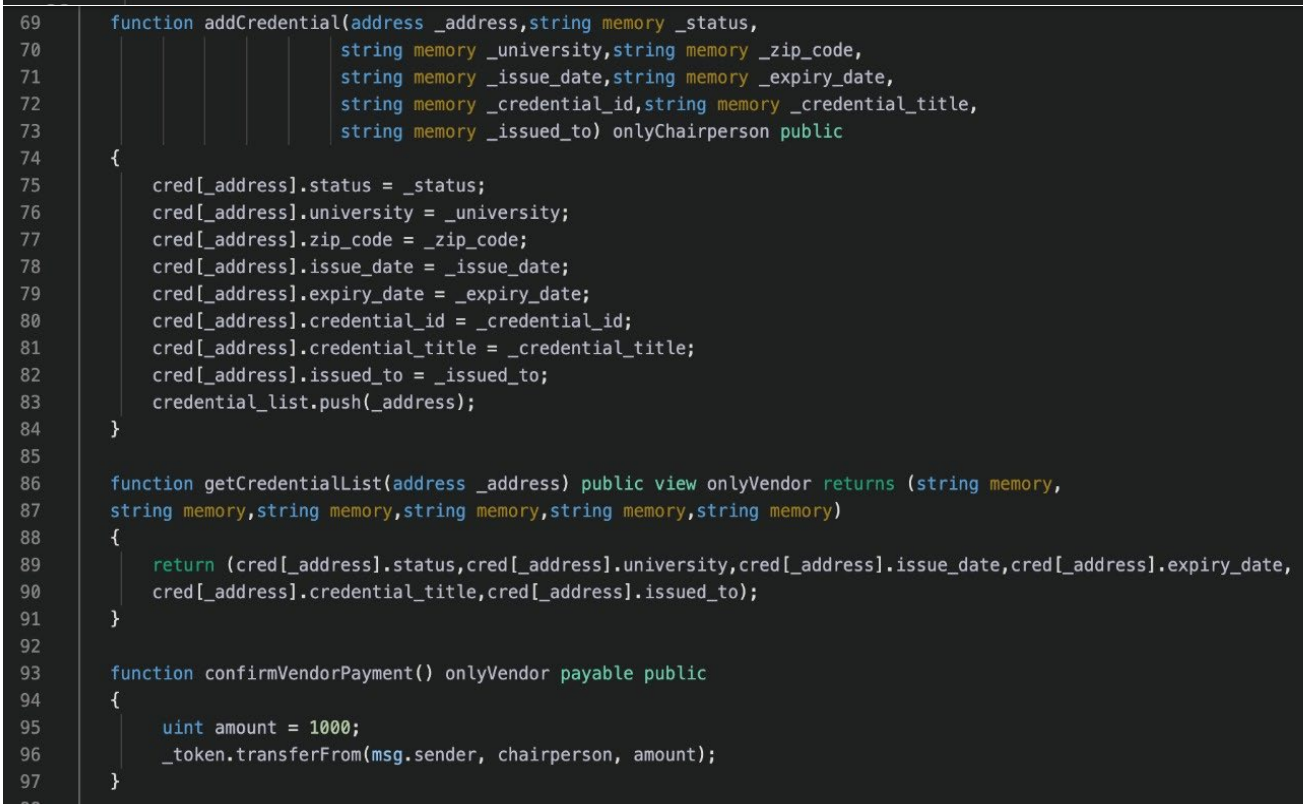
CredAuth.sol



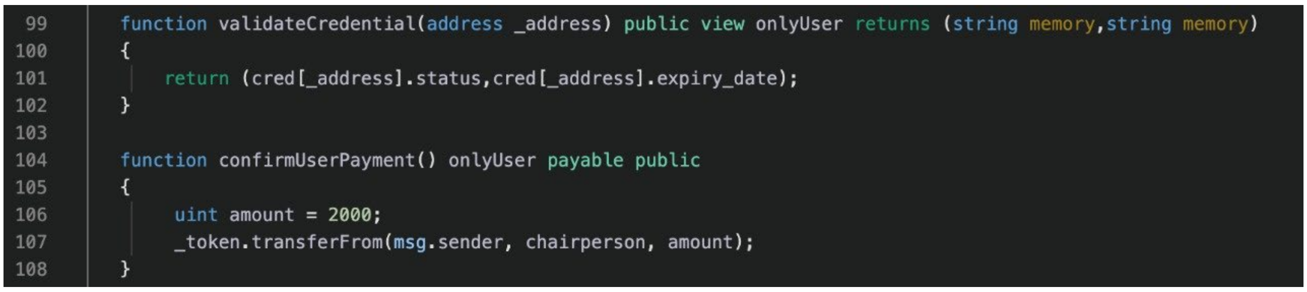
```solidity
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.13;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract CredAuth
{
    address chairperson;

    struct credential
    {
        string status;
        string university;
        string zip_code;
        string issue_date;
        string expiry_date;
        string credential_id;
        string credential_title;
        string issued_to;
    }

    address[] credential_list;
    mapping (address => uint) public registered;
    mapping(address=>credential) public cred;

    IERC20 private _token;

    modifier onlyChairperson
    {
        require(msg.sender == chairperson);
        _;
    }

    modifier onlyUser
    {
        require(registered[msg.sender] == 1);
        _;
    }
```



```
    modifier onlyVendor
    {
        require(registered[msg.sender] == 2);
        _;
    }

    constructor(IERC20 token) payable public
    {
        _token = token;
        chairperson = msg.sender;
        payable(address(this)).transfer(msg.value);
    }

    function registerUser(address user) onlyChairperson public
    {
        registered[user] = 1;
    }

    function registerVendor(address vendor) onlyChairperson public
    {
        registered[vendor] = 2;
    }

    function unregister(address member) onlyChairperson public
    {
        registered[member] = 0;
    }
```

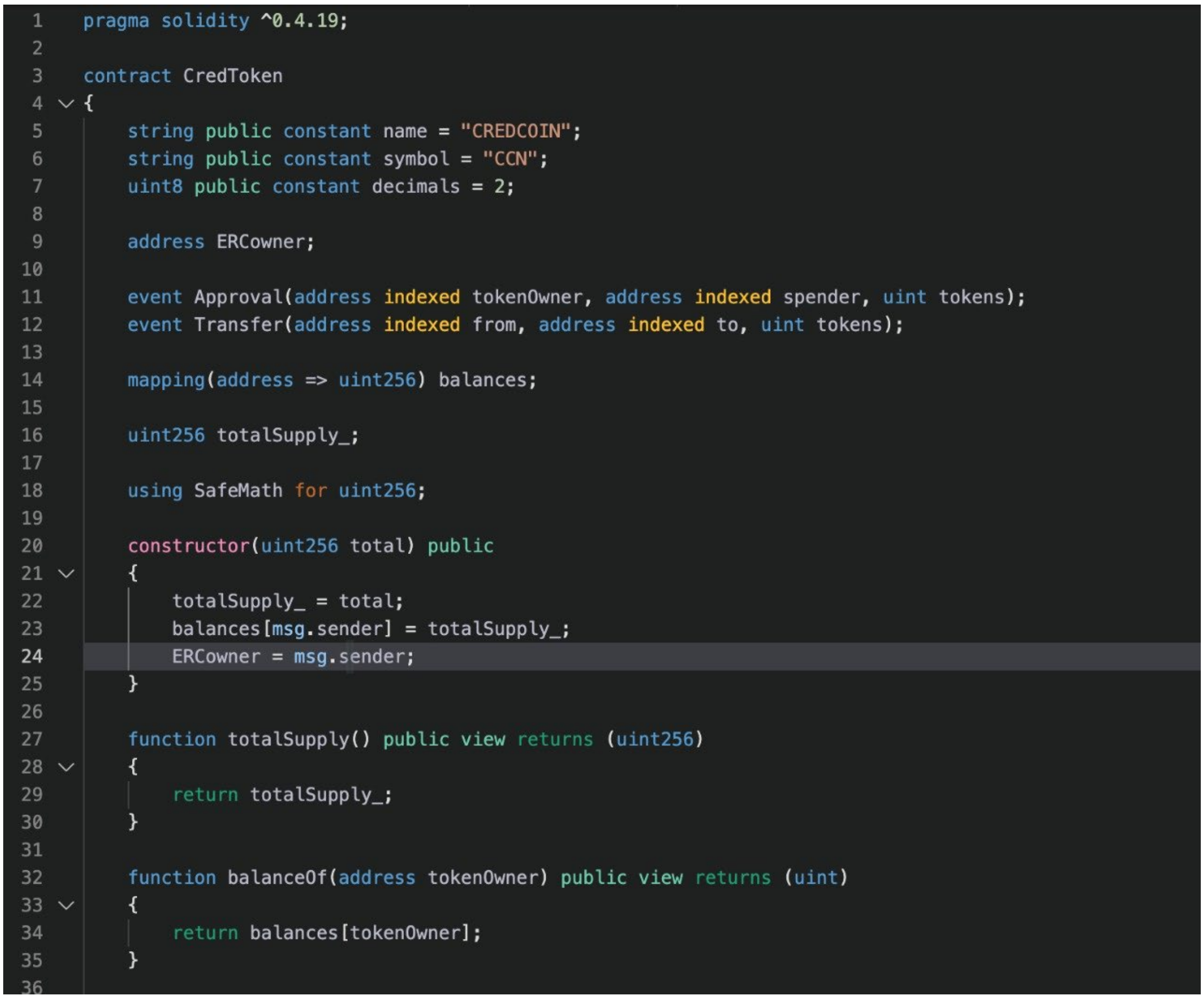


```
    function addCredential(address _address,string memory _status,
                           string memory _university,string memory _zip_code,
                           string memory _issue_date,string memory _expiry_date,
                           string memory _credential_id,string memory _credential_title,
                           string memory _issued_to) onlyChairperson public
    {
        cred[_address].status = _status;
        cred[_address].university = _university;
        cred[_address].zip_code = _zip_code;
        cred[_address].issue_date = _issue_date;
        cred[_address].expiry_date = _expiry_date;
        cred[_address].credential_id = _credential_id;
        cred[_address].credential_title = _credential_title;
        cred[_address].issued_to = _issued_to;
        credential_list.push(_address);
    }

    function getCredentialList(address _address) public view onlyVendor returns (string memory,
    string memory,string memory,string memory,string memory,string memory)
    {
        return (cred[_address].status,cred[_address].university,cred[_address].issue_date,cred[_address].expiry_date,
        cred[_address].credential_title,cred[_address].issued_to);
    }

    function confirmVendorPayment() onlyVendor payable public
    {
         uint amount = 1000;
         _token.transferFrom(msg.sender, chairperson, amount);
    }
```

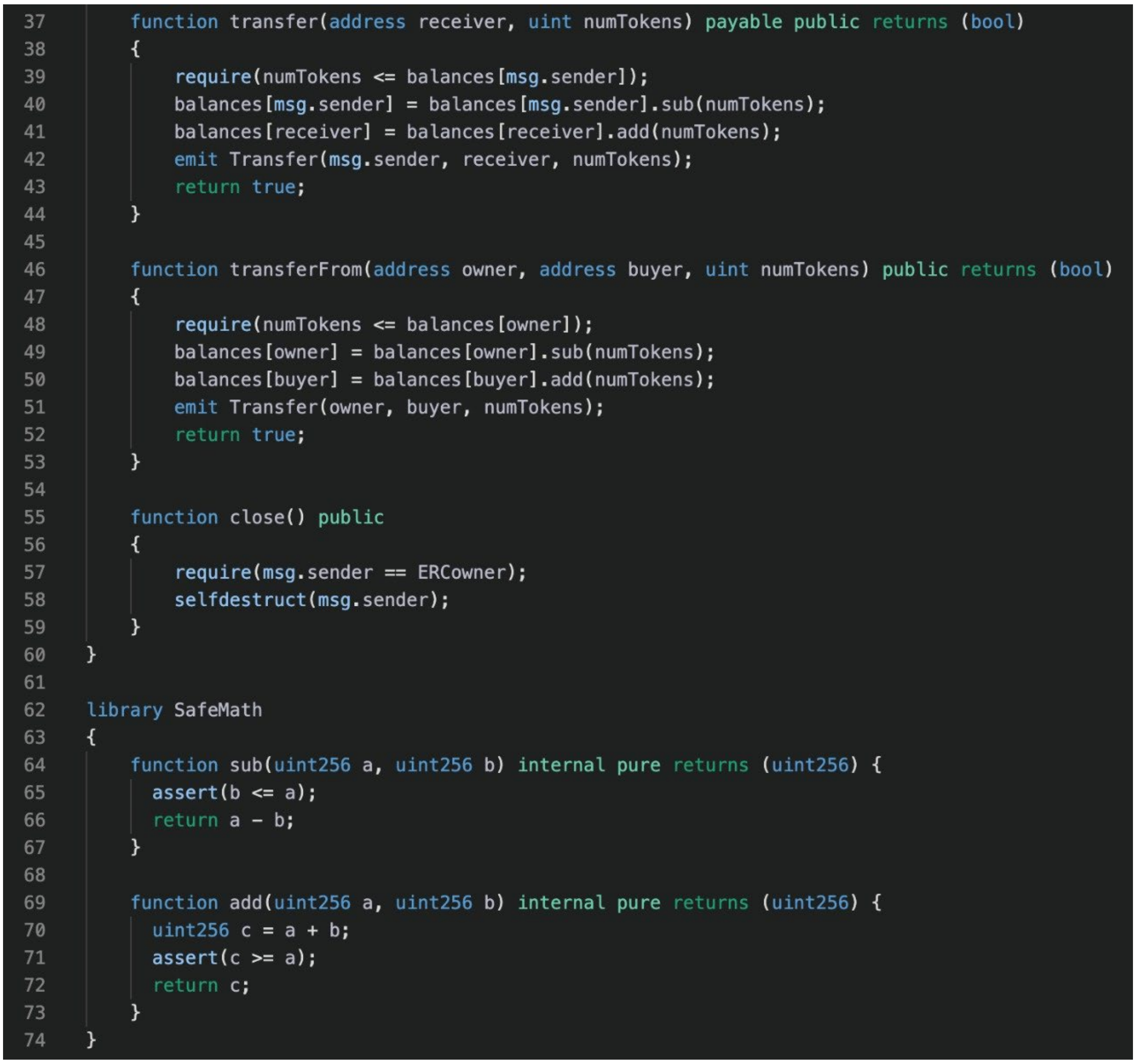


```
    function validateCredential(address _address) public view onlyUser returns (string memory,string memory)
    {
        return (cred[_address].status,cred[_address].expiry_date);
    }

    function confirmUserPayment() onlyUser payable public
    {
        uint amount = 2000;
        _token.transferFrom(msg.sender, chairperson, amount);
    }
```

CredToken.sol



```
pragma solidity ^0.4.19;

contract CredToken
{
    string public constant name = "CREDCOIN";
    string public constant symbol = "CCN";
    uint8 public constant decimals = 2;

    address ERCowner;

    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
    event Transfer(address indexed from, address indexed to, uint tokens);

    mapping(address => uint256) balances;

    uint256 totalSupply_;

    using SafeMath for uint256;

    constructor(uint256 total) public
    {
        totalSupply_ = total;
        balances[msg.sender] = totalSupply_;
        ERCowner = msg.sender;
    }

    function totalSupply() public view returns (uint256)
    {
        return totalSupply_;
    }

    function balanceOf(address tokenOwner) public view returns (uint)
    {
        return balances[tokenOwner];
    }

```



```
    function transfer(address receiver, uint numTokens) payable public returns (bool)
    {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }

    function transferFrom(address owner, address buyer, uint numTokens) public returns (bool)
    {
        require(numTokens <= balances[owner]);
        balances[owner] = balances[owner].sub(numTokens);
        balances[buyer] = balances[buyer].add(numTokens);
        emit Transfer(owner, buyer, numTokens);
        return true;
    }

    function close() public
    {
        require(msg.sender == ERCowner);
        selfdestruct(msg.sender);
    }
}

library SafeMath
{
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
      assert(b <= a);
      return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
      uint256 c = a + b;
      assert(c >= a);
      return c;
    }
}
```

Application Screenshot:

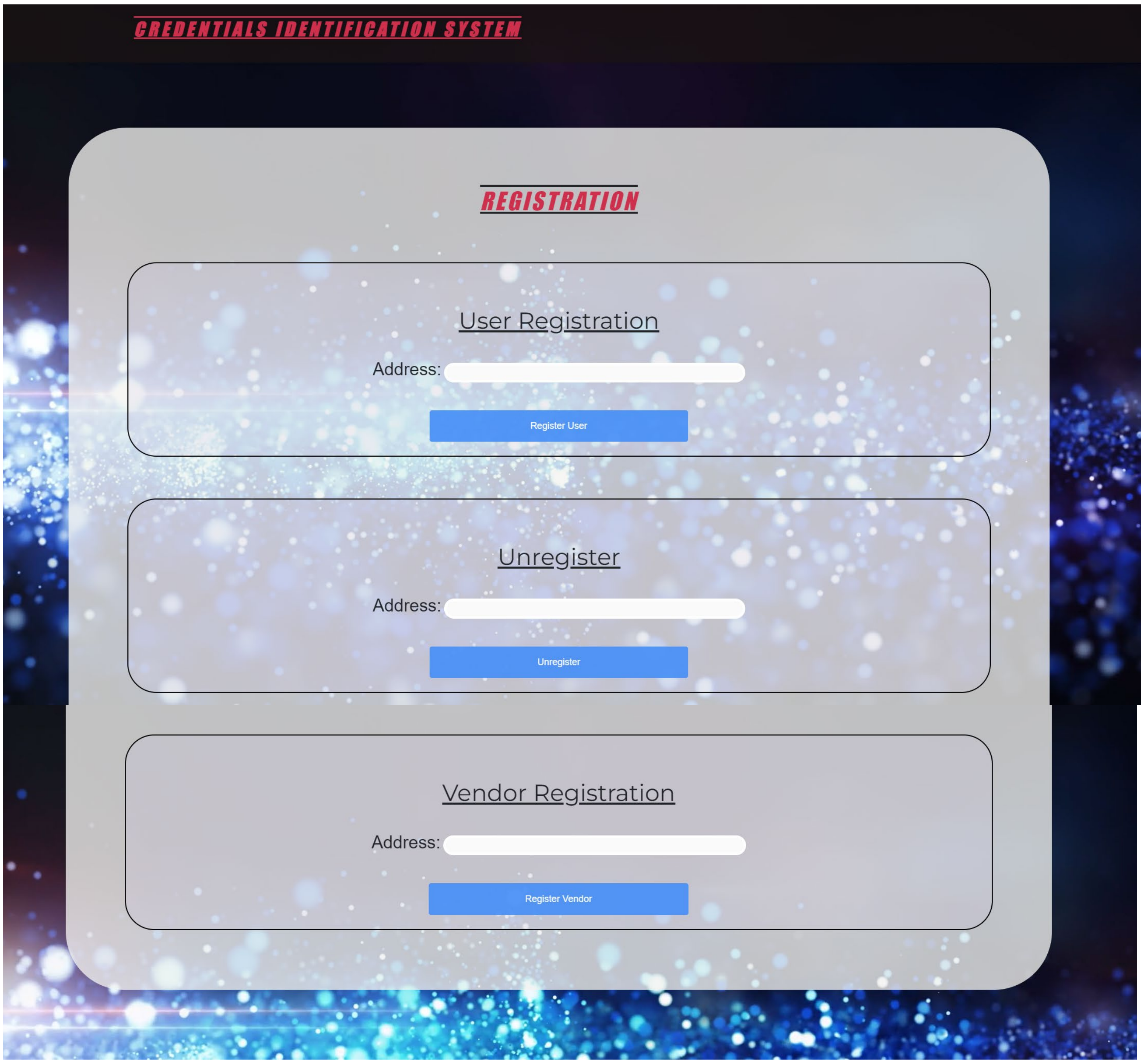


Fig 1: Registration

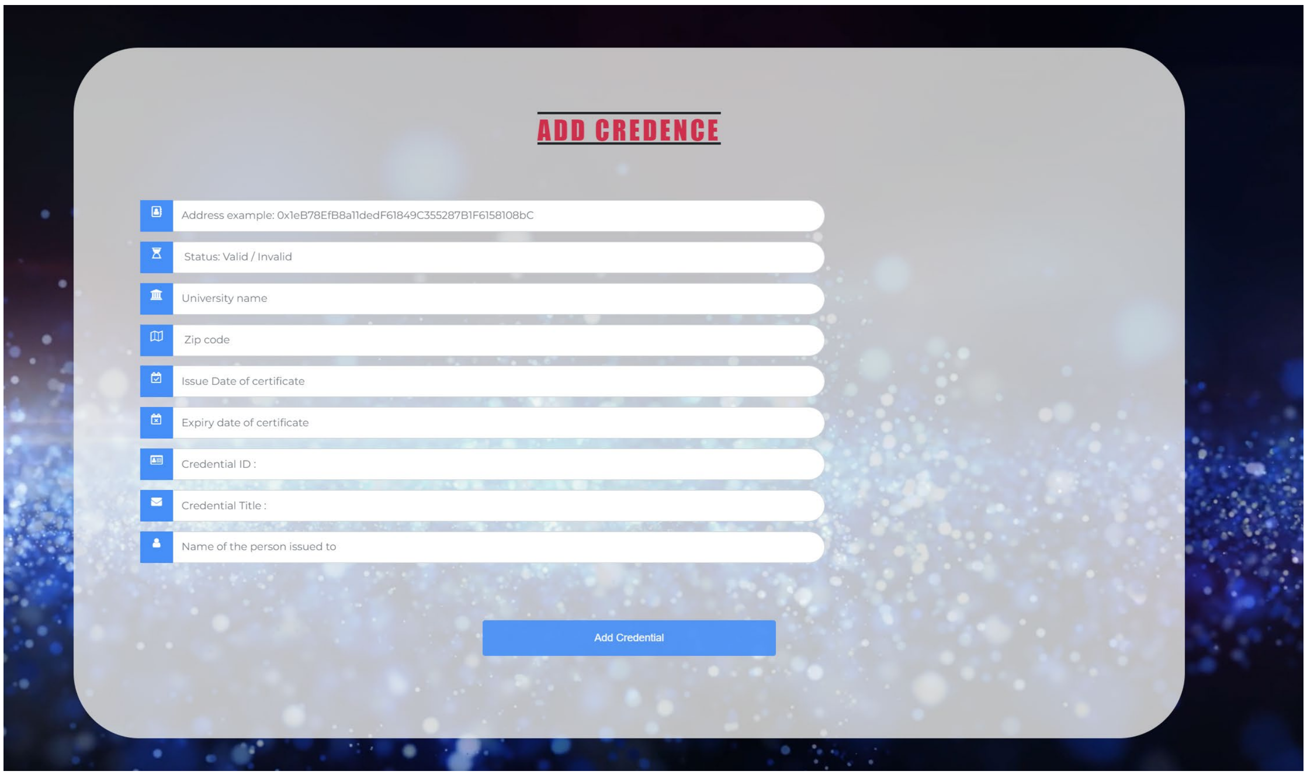

ADD CREDENCE
Address example: 0x1eB78EfB8a11dedF61849C355287B1F6158108bC
Status: Valid / Invalid
University name
Zip code
Issue Date of certificate
Expiry date of certificate
Credential ID :
Credential Title :
Name of the person issued to
Add Credential

Fig 2: Add credence

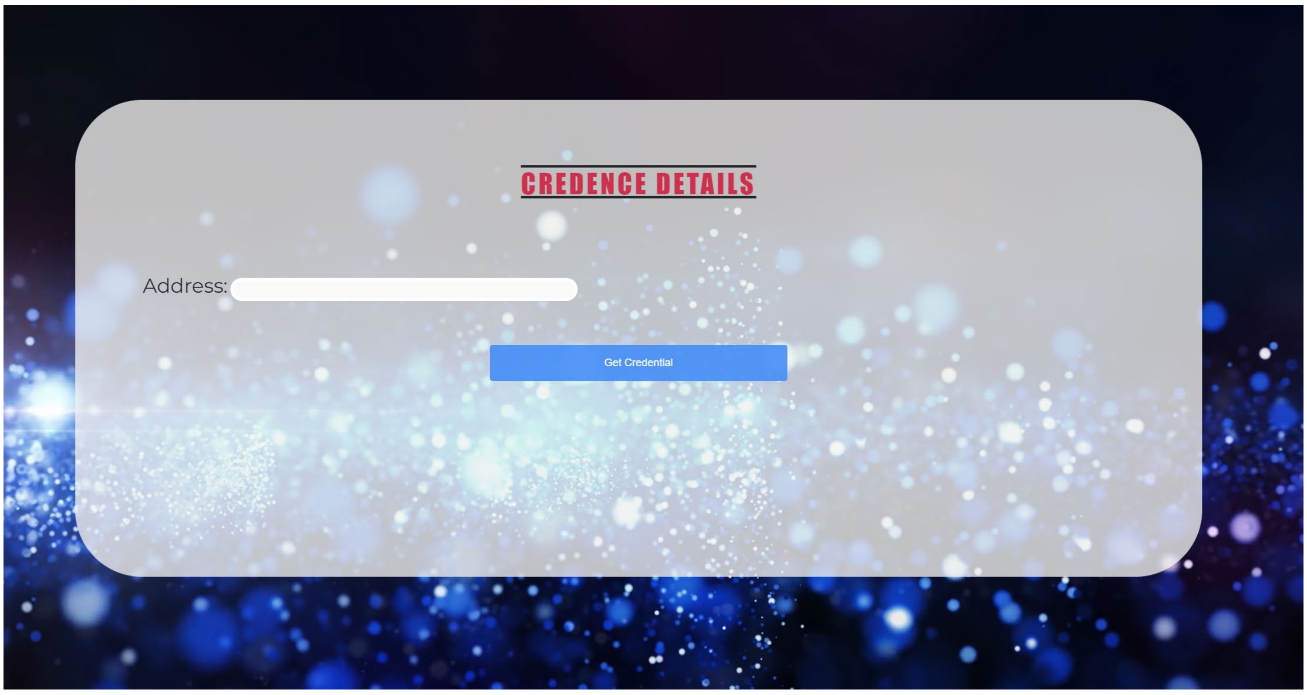


Fig 3: Credence Details

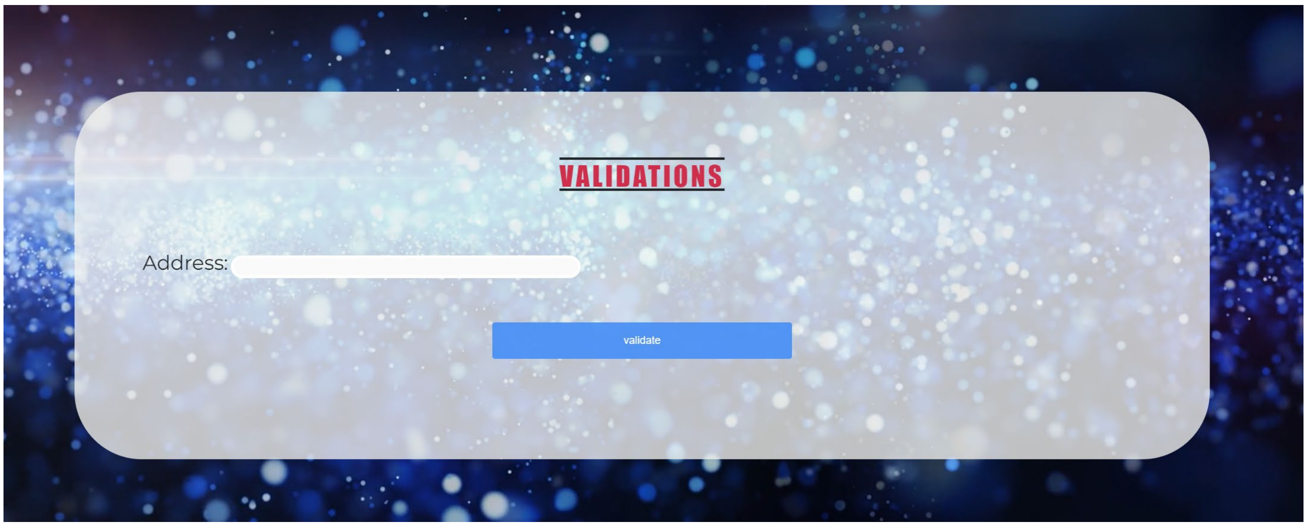


Fig 4: Validations

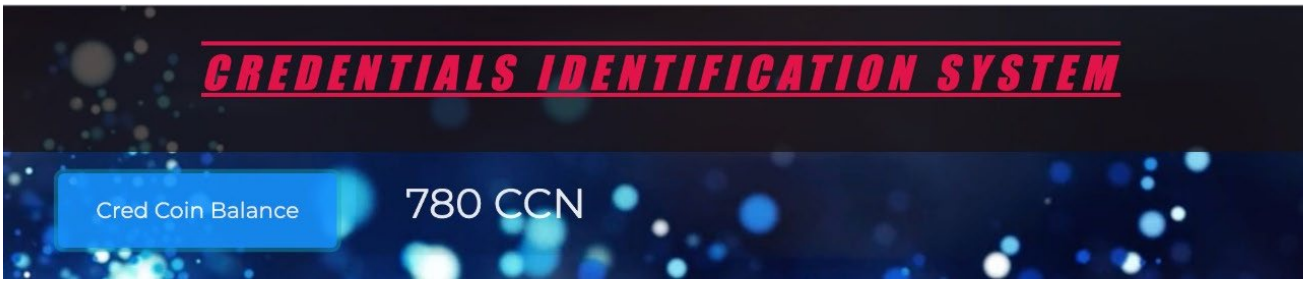


Fig 5: Cred Coin Balance

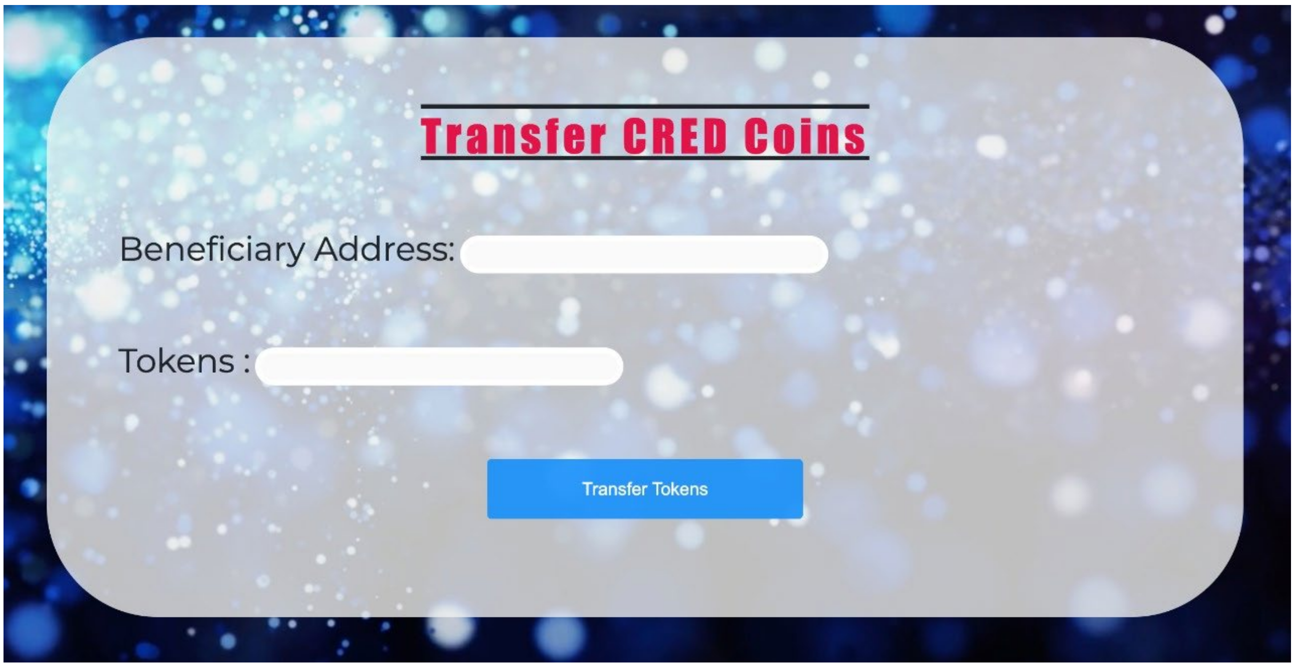


Fig 6: Transfer Cred Coins

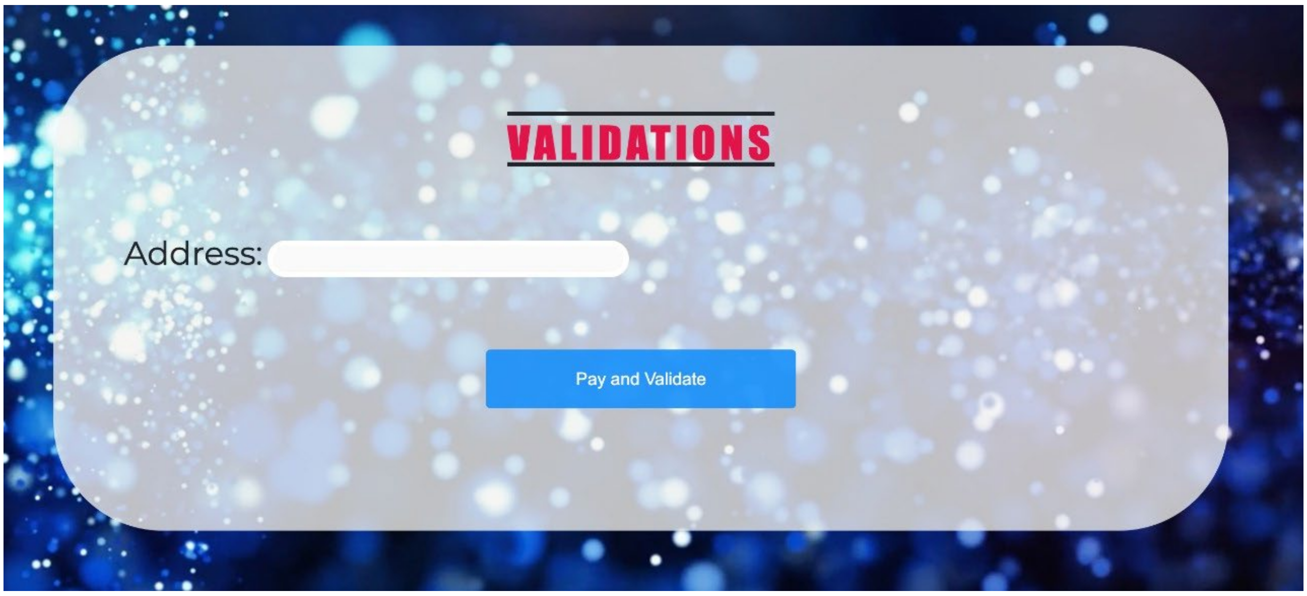


Fig 7: Pay and Validate Credence

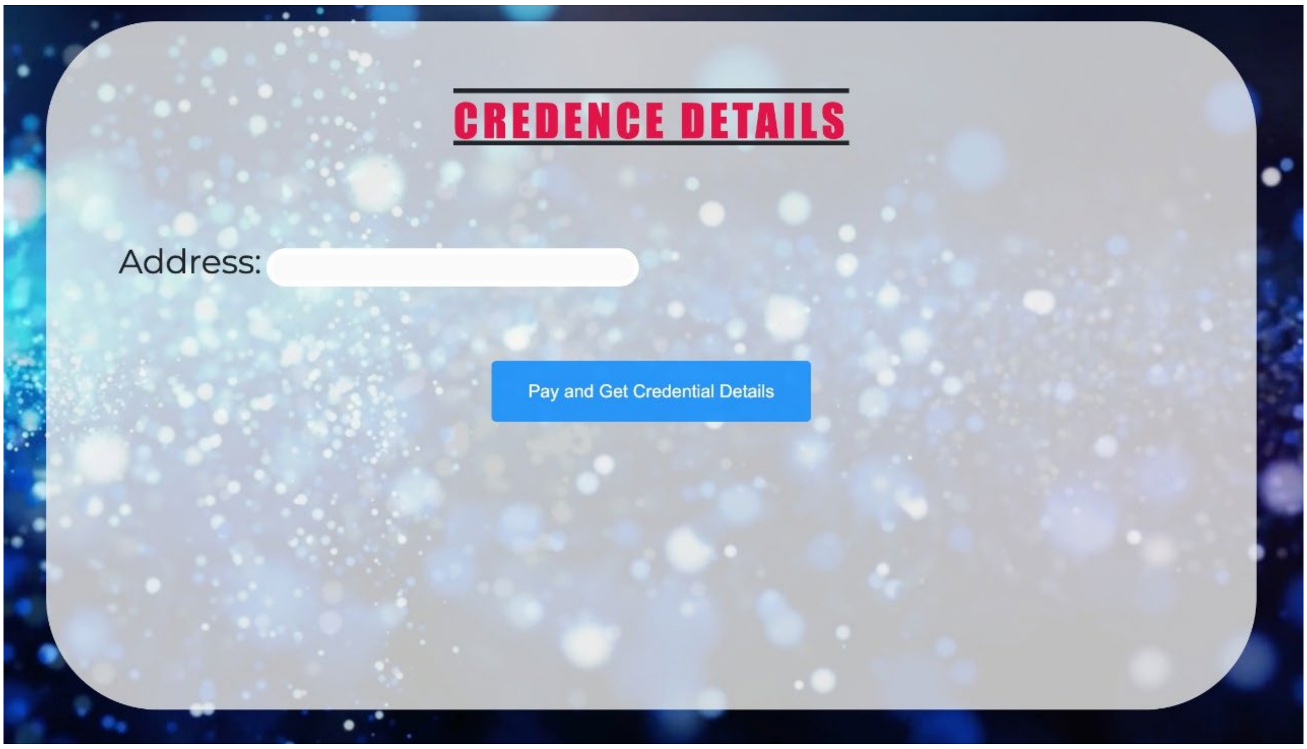


Fig 8: Pay and Get Credential Details

<u>**References**</u> :

(1) Textbook: Blockchain in Action – *Prof. Bina Ramamurthy*
(2) https://www.coursera.org/learn/blockchain-basics/
(3) Blockchain Think lab - Power2peer

[1] = We have Demonstrated Video and divided it into chapters. We have shown deployment and also demonstrated with chairperson, vendor and user point of view.
[2] = Removed from Heroku as it was getting charged on weekly basis. Suggested by Mr.Sean Sanders on Piazza.