1. Construire structura blockchain

De stiut:

Un portofel(wallet) dintr-un blockchain este reprezentat de o pereche cheie privata si cheie publica.  
Cheia privata este doar a ta si cu ajutorul ei dovedesti ca tu esti proprietarul portofelului. De asemenea, tranzactiile sunt realizate cu cheia privata.

Cheia publica este adresa ta publica din blockchain. Blockchain-ul fiind o structura de date decentralizata si transparenta iti ofera oportunitatea de a vede toate tranzactiile realizate de un portofel pana in momentul de fata ( in schimb, pentru a il corela cu o persoana ar trebui sa ii stii cheia publica).

Pentru a realiza un transfer de la o persoana la alta se face o scriere de tranzactie in „blockchain” intre cele 2 adrese publice ale celor 2 persoane. Tranzactia este semnata de emitent prin cheia privata.

Un block contine in general mai multe tranzactii. O lista simplu inlantuita de blockuri reprezinta un blockchain. Blockurile sunt legate intre ele prin hash.

Hash-ul este o forma de encryptare a datelor ireversibila. Avand aceleasi date de intrare vei avea intotdeauna acelasi output. In schimb daca variaza o litera hash-ul va fi altul. De asemenea nu poti ajunge de la hash la date.

Fiecare block are o referinta la block-ul anterior. Pe pozitia 0 se creaza un block numit genesisBlock. Acesta face referinta catre null. Orice alt block ce il urmeaza are ca proprietate un „previousHash” cu valoarea hash-ului blockului anterior.

Dintr-o cheie privata se poate afla o cheie publica, invers nu.

In caz ca vreti sa vedeti o structura, doar ca e facuta in js: https://github.com/gandolh/discord-bot/tree/master/node/blockchain

Construire clasa „Transaction”. In aceasta clasa:

* Se va crea un Constructor. Acesta primeste urmatorii parametrii:  
  self(referinta la obiectul curent, asa functioneaza python), fromAdress, toAdress, amount (default 1)
* In constructor i se vor atribui obiectului „self” parametrii mentionati mai sus, dar si un timestamp ( Data si ora in care se realizeaza atribuirea)
* Se va crea o metoda calculateHash. Va forma un string cu self.fromAdress, self.toAdress, self.amount si self.timestamp. String-ul format va fi prelucrat cu un algoritm de hashing (SHA256 de preferat) si va fi salvat intr-o variabila locala „hash”. Aceasta va trebui sa fie returnata de functie. (recomand formatul „hex”, nu „bin” daca aveti alternativa asta)
* Se va crea o metoda signTransaction ce va primi ca parametru un string (signingKey) (acesta reprezinta cheia privata cu care va fi semnata tranzactia).   
  Se va face o verificare, daca cheia publica aferenta cheii private „signingKey” este diferita de self.fromAdress atunci se va arunca un mesaj de eroare.  
  Daca nu s-a aruncat mesajul de eroare atunci:  
  -Se calculeaza hash-ul cu metoda descrisa mai sus.  
  -se semneaza hash-ul cu cheia privata si valoarea lui se salveaza intr-o variabila (sig?). Recomand salvarea hash-ului semnat in formatul base64. (ar trebui sa existe o functie pentru asta).  
  Exemplu de apel: sig= signingKey.sign(hashTx,’base64’)  
  -self.signature ia valoarea hash-ului semnat convertit la DER.( ar trebui sa existe o functie si pentru asta. Exemplu de apel: self.signature.toDER()
* Se va crea o metoda isValid. Nu va primii parametrii. Aceasta va verifica daca Tranzactia curenta este valida. Se aplica urmatoarele reguli:  
  -daca self.fromAdress este null atunci se returneaza true (in cazul acesta este genesis block)

-Daca self.signature este nedefinita sau are lungime 0 se va arunca o eroare in care se spune ca nu exista o semnatura.  
- Se va lua cheia publica in format hex din self.fromAdress si se va genera wallet-ul cu ea. (libraria de encriptie ar trebui sa aiba un utilitar pentru asta). Wallet-ul ar trebui sa aiba o functie de verify prin care se verifice daca self.signature este cea a keyPublice curente.

Construire clasa „Block”. In aceasta clasa:

* Se va crea un Constructor. Acesta primeste urmatorii parametrii:  
  self (referinta la obiectul curent), timestamp, transactions, previousHash
* In constructor i se vor atribui obiectului „self” parametrii mentionati mai sus, un catchphrase (primit de la user) si un „hash”. Hash-ul trebuie initializat cu rezultatul metodei de calculare a hash-ului din aceasta clasa ( self.calculateHash())
* Se va crea o metoda „calculateHash”, fara parametrii. Aceasta realizeaza un string format din concatenarea datelor obiectului self si il encripteaza prin sha256. Rezultatul este returnat de catre functie. (recomand formatul „hex”, nu „bin” daca aveti alternativa asta)
* O metoda „mineBlock” fara parametrii. Se va adauga un sleep 2 minute in aceasta functie si opreste temporal functionalitatea programului
* O metoda „hasValidTransactions() fara parametrii. Aceasta itereaza prin lista self.transactions, iar daca o tranzactie nu este valida atunci se returneaza False. Altfel se returneaza True.  
  Fiecare element din self.transactions este un obiect ce are o metode denumita isValid(). Pentru a verifica daca o tranzactie este valida apelati: self.transactions[i].isValid()

Construire clasa „Blockchain”. In aceasta clasa:

* Constructorul nu primeste nici-un parametru. In acesta se initializeaza:  
  - self.chain = [ self.createGenesisBlock() ] – metoda createGenesisBlock va fi creata ulterior.   
  Este obligatoriu un genesis block pentru integritatea blockchainului

- self.pendingTransactions – o lista cu tranzactiile ce n-au fost adaugate momentan intr-un block. (goala initial)

* Se va crea o metoda „createGenesisBlock”, fara parametrii, ce creaza primul block din Blockchain.  
  Aceasta metoda trebuie sa returneze un nou Block cu timestamp= data de 26/07/2002,transactions = [] si previousHash=0.   
  De ex: return new Block(new Date(),[],0)
* Se va crea o metoda getLatestBlock() , fara parametrii. Aceasta returneaza ultimul element din self.chain
* Se va crea o metoda minePendingTransactions() , fara parametrii.   
  -Aici se va crea un nou block cu urmatoarele date: data curenta, self.pendingTransactions, hash-ul ultimului block creat in chain (se va folosi metoda getLatestBlock() si proprietate „hash” a instantei)  
  -Se apeleaza metoda mineBlock() a blocului creat.  
  Block-ul creat va fi adaugat la final in lista self.chain  
  lista self.pendingTransaction se va goli. O atribuire cu lista goala ar trebui sa rezolve totul.
* Se va crea o metoda getBallanceFromAdress. Primeste ca parametru o adresa ce reprezinta cheia publica a unui portofel. Pentru fiecare tranzactie a blockurilor din self.chain se vor face urmatoarele operatii:  
  -Daca fromAdress-ul tranzactiei curente este egal cu adresa primita ca parametru atunci sold-ul curent scade cu valoarea tranzactiei (trans.amount)  
  -DacaToAdress-ul tranzactiei curente este egal cu adresa primita ca parametru atunci sold-ul curent creste cu valoarea tranzactiei.  
  Se va returna valoarea sold-ului pentru adresa respectiva.
* Se va crea o metoda addTransaction. Aceasta primeste ca parametru o tranzactie.  
  Sunt necesare urmatoarele verificari:  
  -daca campul fromAdress sau toAdress este gol atunci se va arunca un mesaj de eroare.  
  -daca tranzactia nu este valida se va arunca un mesaj de eroare. Verificarea unei tranzactii daca este valida se va face prin invocarea metodei isValid() a tranzactiei  
  -Daca amount-ul tranzactiei este negativ se va arunca un mesaj de eroare.  
  -Daca sold-ul pentru fromAdress este mai mic decat valoarea tranzactiei (transaction.ammout) se va arunca un mesaj de eroare.   
  (sold-ul se va verifica cu metoda getBallance a acestei clase)  
  -in caz contrar, daca s-a ajuns aici fara un mesaj de eroare, se va adauga la final tranzactia in self.pendingTransactions.
* Se va crea o metoda getAllTransactionsForWallet. Aceasta primeste un parametru „adress”. Pentru fiecare tranzactie din fiecare block se verifica daca campul toAdress sau campul fromAdress este egal cu parametrul primit „adress”. Daca da, aceasta tranzactie se va adauga la finalul unei liste ( de ex txs). Se va returna lista construita.
* Se va crea o metoda isChainValid(). Aceasta nu va primi nici-un parametru.  
  -Se va crea un genesis block,se va salva intr-o variabila si se va compara cu cu self.chain[0] (blockul 0 din blockchain). Daca sunt diferite se returneaza false;
* Se va itera prin fiecare block al chain-ului,incepand de la al-2lea block, si pentru fiecare se vor verifica:  
  -Daca proprietate „hash” a block-ului anterior este diferita de proprietate „previousHash” a block-ului curent se returneaza false  
  - Daca block-ul curent nu are tranzactii valide se va returna false.  
  (se va folosi metoda hasValidTransactions() a clasei Block. Exemplu de apel: curentBlock.hasValidTransactions() )  
  -Daca hash-ul blockului curent este diferit de curentBlock.calculateHash() atunci se va returna false. Acest lucru ne ajuta sa vedem daca tranzactia a fost modificata intre timp.  
  -In cazul in care toate verificariile au fost trecute pentru fiecare block, atunci se marcheaza chain-ul ca valid.

1. **Generare de chei**

Pentru inceput trebuie sa existe o functie ce va genera un wallet random. Functia va crea un wallet in secp256k1 cu elliptic si va returna o cheie publica si una privata

1. **Salvare state**

De fiecare data cand programul se va inchide se va scrie intr-un fisier obiectul din python sub format json.  
La pornirea blockchainului,json-ul va fi importat, transformat intr-un obiect de python si va suprascrie datele din blockchain.