Create Your Own Private Blockchain

Meets Specifications

Congratulations 🎊 You have passed all the Project specifications 👏 and completed your project

You have done an excellent job in making this project. I would like to specially call out a few of the things that I noticed in the project -

- The `validate()` function in Block.js class is perfectly written because usually, students forget to assign the hash value back after calculating the hash value again but your code is perfectly written.
- You've nicely created an endpoint to invoke `validateChain()` function.
- The code looks very clean. It does not contain unnecessary code. It is well formatted.

Also, you can use a new VS Code extension that is an alternative to Postman, you can checkout this link.

In the next project, you will be learning about solidity language to write smart contracts. All the best for future content and project 😄

If you find anything I have missed to point out or anything bad in my review, do give me feedback.

Happy Learning 👌 and Stay Udacious 👍

Best,
Anku

Rate this review

START

Complete unfinished block.js implementation

✓ -Return a new promise to allow the method be called asynchronous.
-Create an auxiliary variable and store the current hash of the block in it (this represent the block object)
-Recalculate the hash of the entire block (Use SHA256 from crypto-js library)
-Compare if the auxiliary hash value is different from the calculated one.
-Resolve true or false depending if it is valid or not.

✅ The `validate()` function is working as expected

- The logic written to validate the block is perfect. It is returning `false` output if any of the block's property is modified.

✓ -Use hex2ascii module to decode the data
-Because data is a javascript object use JSON.parse(string) to get the Javascript Object

- Resolve with the data and make sure that you don't need to return the data for the genesis block OR reject with an error.

Complete unfinished blockchain.js implementation

✓

- Must return a Promise that will resolve with the block added OR reject if an error happen during the execution.
- height must be checked to assign the previousBlockHash
  -Assign the timestamp & the correct height
  -Create the block hash and push the block into the chain array.
  Don't for get to update the `this.height`

✓

- must return a Promise that will resolve with the message to be signed

✓

- must resolve with the Block added or
  reject with an error.
- time elapsed between when the message was sent and the current time must be less that 5 minutes
- must verify the message with wallet address and signature: bitcoinMessage.verify(message, address, signature)
- must create the block and add it to the chain if verification is valid

✓

- must return a Promise that will resolve with the Block

✓

- must return a Promise that will resolve with an array of the owner address' Stars from the chain

✓

- must return a Promise that will resolve with the list of errors when validating the chain
- must validate each block using validateBlock()
- Each Block should check with the previousBlockHash
- execute the `validateChain()` function every time a block is added
- create an endpoint that will trigger the execution of `validateChain()`

✅ The `validateChain()` function is working as expected

- It is calling the validate() function to validate the block.
- It is comparing the hash value with the previousBlockHash of the blocks.
- An endpoint is created to invoke `validateChain()` function.
- Every time a new block is added this function is getting invoked.

Test your App functionality

✓

- must use a GET call to request the Genesis block
- must use a POST call to requestValidation
- must sign message with your wallet
- must submit your Star
- must use GET call to retrieve starts owned by a particular address

DOWNLOAD PROJECT

RETURN TO PATH