**Source Code Instruction Manual**

**Software Development Team**

***Methods for Securing Blockchain-Based Transactive Energy Systems***

**Team Members**

Ismail Asorona

Gbolahan Afloabi

Mecahel Demissie

Glory Ezeogu

Boluwatife Shekoni

Emmanuel Yormesor

**Source Code**

import hashlib

import json

import time

class Block:

def \_\_init\_\_(self, block\_number, data, location, device\_name, energy, nonce=0, prev\_hash=" ",

timestamp=None):

self.block\_number = block\_number

self.data = data

self.timestamp = timestamp or time.time()

self.nonce = nonce

self.prev\_hash = prev\_hash

self.device\_name = device\_name

self.energy = energy

self.location = location

self.hash = self.calculate\_hash()

# create a didgital fingerprint for the block(hash) using sha256

def calculate\_hash(self):

block\_str = json.dumps(self.\_\_dict\_\_, sort\_keys=True)

return hashlib.sha256(block\_str.encode()).hexdigest()

def print\_hashes(self):

print("Previous hash: " + self.prev\_hash)

print("Hash: " + self.hash)

class Blockchain:

chain = []

def \_\_int\_\_(self):

self.chain = []

self.prev\_hash = self.get\_last\_block()

def genesis\_block(self):

genesis\_block = Block(0, ["First Block"], time.time(), "0", "", "", "")

genesis\_block = genesis\_block.calculate\_hash()

self.chain.append(genesis\_block)

return genesis\_block

def get\_last\_block(self):

return self.chain[-1]

def add(self, block):

self.prev\_hash = self.get\_last\_block()

self.block = block.calculate\_hash()

if self.verify(block) == False:

return "Invalid Block... sorry we cannot verify this block"

elif self.verify(block) == True:

return self.mine(block)

def mine(self, block):

difficulty = 3

block.nonce = 0

self.hash = block.calculate\_hash()

while not self.hash.startswith(

'0' \* difficulty):

block.nonce += 1

# print(self.hash)

self.hash = block.calculate\_hash()

self.chain.append(self.hash)

return "---------------------------------------------" + \

"\nMined Blocked :" + self.hash + \

"\nNonce: " + str(block.nonce) + \

"\nBlock Number: " + str(block.block\_number) + \

"\nData: " + str(block.data) + \

"\nLocation: " + str(block.location) + \

"\nEnergy Source: " + str(block.device\_name) + \

"\nEnergy Input: " + str(block.energy) + \

"\nTime: " + str(time.ctime(block.timestamp)) + \

"\nPrevious Hash: " + str(self.prev\_hash) + \

"\n "

def length\_chain(self):

return ("The blockchain length is: " + str(len(self.chain)))

def index\_chain(self):

l = 0

while len(self.chain) != l:

for i in self.chain:

print("\n Block " + str(l) + ":" + i)

l+= 1

return ""

def verify(self, block):

flag = True

return flag

blocks = Blockchain()

blocks.genesis\_block()

state = input("Enter the state abbreviation in which you live in: ")

if state == 'GA' or state == 'MD' or state == 'VA' or state == 'DC':

blockenergy = 6000.00

choice = 'a'

num = int(1)

while choice != 'q' and choice != 'Q':

print("Team Block has", blockenergy, "kilowatts stored in the bank.")

print("Enter 1 to Buy")

print("Enter 2 to Sell")

print("Enter Q to Quit")

print("\n")

choice = str(input("Enter choice: "))

print("\n")

if choice == 'q' or choice == 'Q':

print("Thank you for choosing Team Block!")

print("\n")

print(blocks.length\_chain())

print(blocks.index\_chain())

elif choice == '1':

print("You are now buying energy from Team Block.")

print("\n")

data = input("Enter name: ")

location = input("Enter your city: ")

device\_name = input("What is the energy source you will be using?: ")

energy = float(input("Enter the amount of energy to be bought (kW): "))

if energy > blockenergy:

print("Invalid. Energy preffered is greater than Team Block's energy bank.")

else:

blockenergy = blockenergy - energy

num += 1

print(blocks.add(Block(num, [data], location, device\_name, energy, "")))

num += 1

print("\n")

elif choice == '2':

print("You are now selling energy to Team Block.")

print("\n")

data = input("Enter name: ")

location = input("Enter your city: ")

device\_name = input("What is the energy source you will be using?: ")

energy = float(input("Enter the amount of energy to be sold (kW): "))

blockenergy = blockenergy + energy

print(blocks.add(Block(num, [data], location, device\_name, energy, "")))

num += 1

print("\n")

else:

print("You cannot access Team Block's energy bank.")

**Program Results**

Graphical user interface, application

Description automatically generated

The program starts with asking the user which state they live in. This helps secure access as the only valid states are Georgia, Maryland, Virginia, and the city Washington D.C.  Any other input would deny access to the blockchain system. The reason for that is Team Block has buildings that are located in the valid entries.

Graphical user interface, application

Description automatically generated

After entering the state, the user is displayed the total amount of kilowatts Team Block has stored in the back. The user is given three options where they can either buy, sell, or quit.

Text

Description automatically generated

If the user enters “1”, they choose the option to buy. This prompts them to enter their credentials such as their name, city they live in, the energy source used to transfer energy, and the amount of energy they want to buy.

Text

Description automatically generated

After entering the information, the record is recorded and stored into the blockchain system. Time, hash, and nonce has also been recorded at the time. The previous hash is also displayed.

Text

Description automatically generated

The energy bank is then updated showing the current kilowatts and the user is prompted to enter a choice again.

Text

Description automatically generated

When the user chooses to sell, the same credentials are prompted but now asks how much the user wants to sell to Team Block’s energy bank. This adds energy back into the bank and successfully records the transaction in the system.

Text

Description automatically generated

When the user finally wants to exit the program, the system stops and displays the current blockchain.