**Lab 3: Creating a Blockchain**

**We created a blockchain using Spyder (Python 3.8) from the Anaconda Data Science Toolkit.**

**As part of the first lab instructions, we build the architecture of the blockchain.**

**We then create some functions:**

1. **Retrieve the state of the blockchain to display in the Postman interface**
2. **Mine a new block**

A screen shot of a computer

Description automatically generated

**Import libraries**

*(****Datetime****)* allocate date and timestamps for block mined

***(Hashlib****)* interface for hashing messages to hash blocks

*(****Json****)* simple command line interface to validate and pretty print JSON objects

**Jsonify** function to return messages in Postman (**json** format returns index of the new block, proof of new block, and previous hash)

**Creating a class named Blockchain**

**Dumps function** from library to encode blocks before they are **hashed. Import** flask class from flask.

**Create block**

**Get previous block**

**Proof of work**

**Blockchain:** Genesis block, Initialisation, Create Block Function and add new blocks, Get previous block, Proof of work and validate block chain.

Starting the class by defining the **Init** method (same argument **self**) **self** variables to the objects (as additional methods are created)

**Genesis block**

variable ***self*.chain = []** contains list of blocks (squares brackets initialize list)

define **create\_block** method contains 3 arguments (self, proof, previous\_hash)

block has 4 keys (*index* (new block number*), timestamp* (when block mined), *proof* (of work), *previous\_hash* (most recent block in chain) self.chain is a list, so append function to append new block created.

define **get\_previous\_block** method retrieves the last block of current chain

define **proof\_of\_work** method contains 2 arguments (self, previous\_proof) previous proof is the element miners will need to control to find new proof.

**new\_proof** is initialised to 1 because the variable is incremented by 1 with each iteration of the while loop until the correct proof is found.

**check\_proof** variable to check if new proof is correct proof, if False then while loop iterates through proofs until **check\_proof** is true.

While loop (2 lines) defines what the problem miners have to solve (combination of SHA256 cryptographic hash function and hex digest function) to return string of 64 characters. Output string needs to start with 4 leading zeros for the proof to be valid and new block mined.

Two checks need to be made. One is each block in chain has valid proof of work (cryptographic has with 4 leading zeros) and second if previous hash of each block is equal to the hash of the previous block.

define **hash** function (method of the class) containing 2 arguments (self, block) taking any block in the block chain as an input and returning a SHA256 hash of the block. Each block contains a dictionary of four keys (index, timestamp proof and previous hash). **Json.dumps** function will convert the dictionary to a string and in the format to be accepted by SHA256 function.

A screen shot of a computer program

Description automatically generated

**Validate blockchain**

define **is\_chain\_valid** function (method of the class) containing 2 arguments (self, chain).

Fundamentals are in place.

**Mining the Blockchain**

Create a web app (**Flask**) and use Postman to create the blockchain by getting the blockchain to show the genesis block. Mine 20-30 blocks periodically requesting details of full blockchain.

**Create the blockchain** (blockchain = Blockchain())

Mine a new block through HTTP GET Request from Postman

define **mine\_block()** function

Request the full Blockchain

define **get\_chain()** function

Check if Blockchain is valid

define **is\_valid()** function

**Running the app**

Run the 223120-blockchain-lab2.py in Spyder

A computer screen shot of a computer code

Description automatically generated

**Open Postman and initiate a HTTP GET Request to the Flask web app (get\_chain)**

**Postman is an API platform (Application Programming Interface testing our blockchain)**

A screenshot of a computer

Description automatically generated

Initiate a HTTP GET Request to Flask web app http://127.0.0.1:5000/get\_chain

The timestamp shows us the genesis block of the blockchain

**Postman and initiate a HTTP GET Request to the Flask web app (mine\_block)**

A screenshot of a computer

Description automatically generated

To mine a new block we send a HTTP GET Request to <http://127.0.0.1:5000/mine_block>

**A screenshot of a computer

Description automatically generatedFew more blocks mined**

**Now sending a HTTP GET request to check whether or not the Block chain is valid**

A screenshot of a computer

Description automatically generated

**Summary of the request**

**A screenshot of a computer

Description automatically generated**

**Lab 3: Create a blockchain continued**

**Tutorial to extend the functionality of the blockchain developed in python**

[**Develop a blockchain application from scratch in Python | RUOCHI.AI (zhangruochi.com)**](https://zhangruochi.com/Develop-a-blockchain-application-from-scratch-in-Python/2020/04/21/)

**Blockchain is a way of storing digital data. (Bitcoin is logs of transfers from one account to the other). Data stored in the form of blocks linked (chained) together using cryptographic hashes.**

**The magic is in the way data is stored and added to the blockchain. Blockchain is essentially linked list that contains ordered data with few constraints**

* Blocks can’t be modified once added (append only)
* Specific rules for appending data to it
* Architecture distributed

Benefits of enforcing constraints: Immutability and durability of data, no single point of control or failure, verifiable audit trail of the order in which data was added.

**Store transactions into blocks (storing data) using JSON (post stored in blockchain)**

**A screenshot of a computer screen

Description automatically generated**

**Generic term “data” is replaced by the term “transactions”**

Transactions are packed into blocks and each block can contain one or many transactions. These are generated frequently and added to the blockchain. Multiple blocks will all have unique ID.

**A screen shot of a computer code

Description automatically generated**

**Add digital fingerprints to the blocks**

Cryptographic hash functions to prevent any tampering of data stored in the block. Hash function takes data of any size and produces a data of fixed size (Deterministic – same data result same hash, random and easy to compute).

**Chain the blocks**

Making a way that any change in previous blocks invalidates the entire chain. Bitcoin way is to create dependency among consecutive blocks by chaining them with the hash of the block immediately previous to them.

**Implement a proof of work algorithm**

Creating a hash difficult and random (adding a constraint by leading zeros – a new nonce).

**Add blocks to the chain**

Verify data has not been tampered with (proof of work is correct) and order of transactions is preserved (previous hash) add to the hash of latest block in chain

**Mining**

Process of putting unconfirmed transactions in a block and computing proof of work is known as the mining of blocks.

**Create Interfaces**

Blockchain node to interact with an application called Flask. A rest api that interacts with and invokes various operations in blockchain node.

**Establish consensus and decentralization**

Relinking the block with hashes and applying the PoW constraint. Data needs to be distributed and multiple nodes to maintain the blockchain. Create a mechanism to let a new node aware of peers in the network (single node to a peer-to-peer network). Achieving a simple consensus algorithm to agree upon the longest valid chain when chains of different participating nodes in the network appear to diverge.

**Build the application**

Application needs to connect to a node in the blockchain network to fetch the data and submit new data. Application has a HTML form to take user input and makes a post request to a connected node to add the transaction into unconfirmed transactions pools.

A screenshot of a computer

Description automatically generated

**I’ve tried numerous ways to extend the blockchain and test this in postman (we can see mine\_block, get\_chain and is\_valid functions work**

A screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**I would try extending by adding a transaction from the node\_server.py however had difficulties trying to get this to work**

**A screen shot of a computer code

Description automatically generated**

**Here are two in comparison and needing to tweak this to make it work**

A screen shot of a computer code

Description automatically generated

**This was my common response, however the extension of the blockchain has been added (using similar codes from the given sources) I am unsure if this is a process that can be checked via Postman. Otherwise the code has been extended with one extra functionality (new\_transaction).**

A screenshot of a computer error

Description automatically generated

# -\*- coding: utf-8 -\*-

"""

Created on Tue Jul 30 09:17:32 2024

@author: Andrew.Graff

"""

# libraries to import

#FIRST PART Building the architecture of the blockchain

import datetime

import hashlib

import json

from flask import Flask, jsonify

class Blockchain:

def \_\_init\_\_(self):

self.chain = []

self.create\_block(proof = 1, previous\_hash = '0')

def create\_block(self, proof, previous\_hash):

block = {'index': len(self.chain) + 1,

'timestamp': str(datetime.datetime.now()),

'proof': proof,

'previous\_hash': previous\_hash}

self.chain.append(block)

return block

def get\_previous\_block(self):

return self.chain[-1]

def proof\_of\_work(self, previous\_proof):

new\_proof = 1

check\_proof = False

while check\_proof is False:

hash\_operation = hashlib.sha256(str(new\_proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:4] == '0000':

check\_proof = True

else:

new\_proof += 1

return new\_proof

def hash(self, block):

encoded\_block = json.dumps(block, sort\_keys = True).encode()

return hashlib.sha256(encoded\_block).hexdigest()

def is\_chain\_valid(self, chain):

previous\_block = chain[0]

block\_index = 1

while block\_index < len(chain):

block = chain[block\_index]

if block['previous\_hash'] != self.hash(previous\_block):

return False

previous\_proof = previous\_block['proof']

proof = block['proof']

hash\_operation = hashlib.sha256(str(proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:4] != '0000':

return False

previous\_block = block

block\_index += 1

return True

#Mining the Blockchain

#Create the Web App (http://flask.palletsprojects.com/en/rtd/quickstart/)

app = Flask(\_\_name\_\_)

app.config['JSONIFY\_PRETTYPRINT\_REGULAR'] = False

#Create the Blockchain

blockchain = Blockchain()

#Mine a new block through a HTTP GET Request from Postman

@app.route('/mine\_block', methods = ['GET'])

def mine\_block():

previous\_block = blockchain.get\_previous\_block()

previous\_proof = previous\_block['proof']

proof = blockchain.proof\_of\_work(previous\_proof)

previous\_hash = blockchain.hash(previous\_block)

block = blockchain.create\_block(proof, previous\_hash)

response = {'message': 'Whakamihi. You successfully mined a block!',

'index': block['index'],

'timestamp': block['timestamp'],

'proof': block['proof'],

'previous\_hash': block['previous\_hash']}

return jsonify(response), 200

#Request the full Blockchain

@app.route('/get\_chain', methods = ['GET'])

def get\_chain():

response = {'chain': blockchain.chain,

'length': len(blockchain.chain)}

return jsonify(response), 200

#Check if the Blockchain is valid

@app.route('/is\_valid', methods = ['GET'])

def is\_valid():

is\_valid = blockchain.is\_chain\_valid(blockchain.chain)

if is\_valid:

response = {'mesage': 'The Blockchain is in a valid state.'}

else:

response = {'message': 'The Blockchain is no longer valid.'}

return jsonify(response), 200

# Running the app

app.run(host = '0.0.0.0', port = 5000)