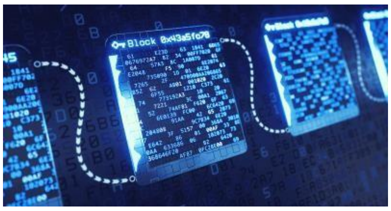
Palestine Technical University (Kadoorie)

Faculty of Engineering and Technology

Department of Computer Systems Engineering

BULIDING BLOCKCHAIN USING PYTHON



Special Topics in Computer Systems Engineering

By:

Mays Khalil Najjar 201811598

Supervised:

Dr. Mahmoud Swalha.

Tulkarem, Palestine

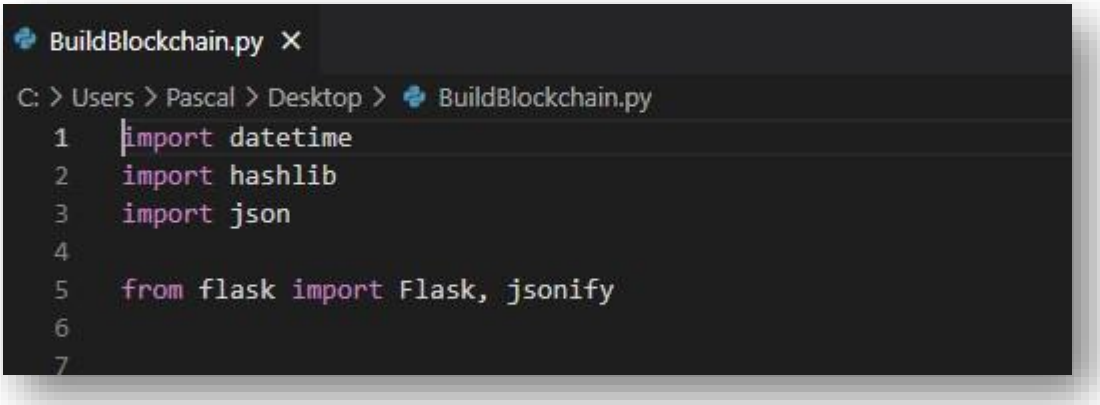
Semester: 1st Academic Year: 2022-2023

Objective

I want to build an application that generates and simulates the operation of blockchain.

In this project I use python language to build this application.

The libraries that we need are:



```
import datetime
import hashlib
import json

from flask import Flask, jsonify
```

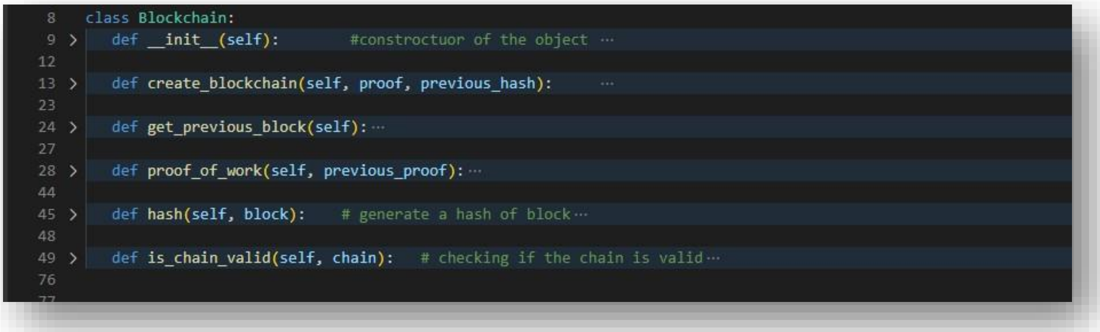
Datetime --> to get the time stamp.

Hashlib --> to calculate the hash function.

Json --> saving different type of data each other.

Flask --> to make this app run by using API of flask.

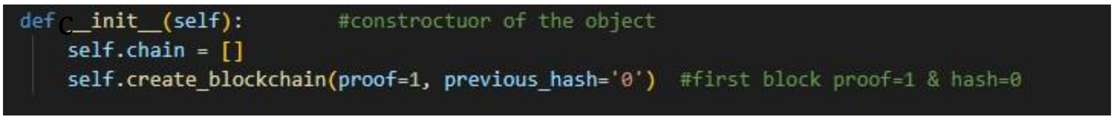
Blockchain class:-



```
class Blockchain:
    def __init__(self):        #constructuor of the object ...

    def create_blockchain(self, proof, previous_hash):     ...

    def get_previous_block(self):...

    def proof_of_work(self, previous_proof):...

    def hash(self, block):    # generate a hash of block...

    def is_chain_valid(self, chain):   # checking if the chain is valid...
```

The functions we have:-

1. __init__(self): create list of chain object.

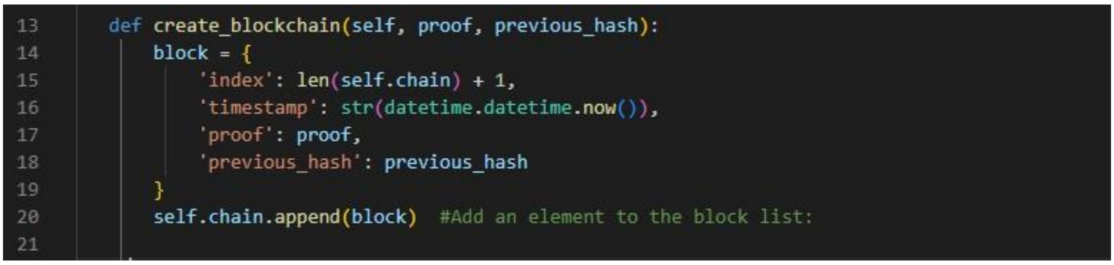


```
def __init__(self):        #constructuor of the object
    self.chain = []
    self.create_blockchain(proof=1, previous_hash='0')  #first block proof=1 & hash=0
```

2. create_blockchain(self, proof, previous_hash):

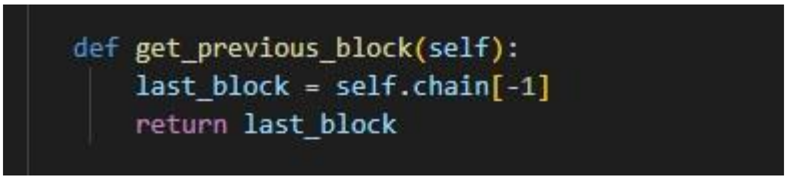
In this function we will create the blockchain,

First, we build a block with index, timestamp, proof, previous_hash. Then we add this block to the chain list.



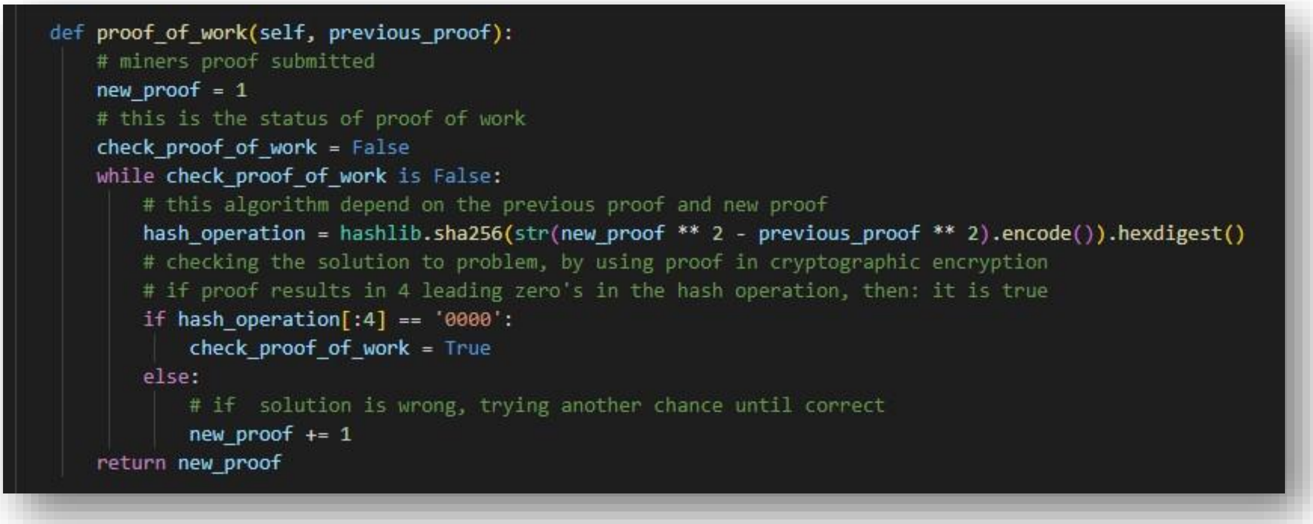
```
    def create_blockchain(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash
        }
        self.chain.append(block)  #Add an element to the block list:
```

3. get_previous_block(self):



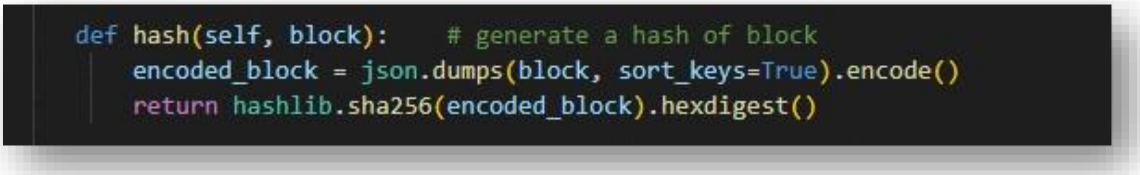
```
def get_previous_block(self):
    last_block = self.chain[-1]
    return last_block
```

4. proof_of_work(self, previous_proof):



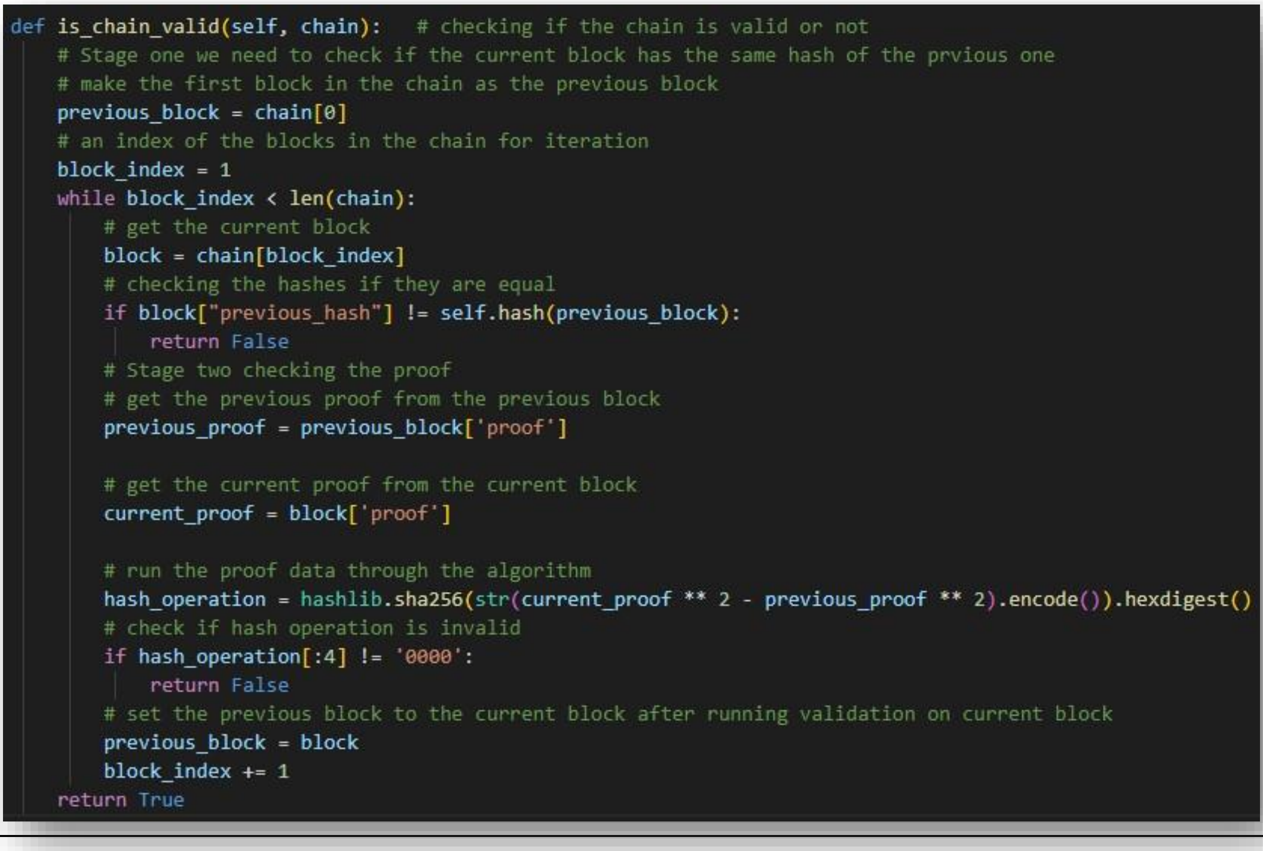
```python
def proof_of_work(self, previous_proof):
    # miners proof submitted
    new_proof = 1
    # this is the status of proof of work
    check_proof_of_work = False
    while check_proof_of_work is False:
        # this algorithm depend on the previous proof and new proof
        hash_operation = hashlib.sha256(str(new_proof ** 2 - previous_proof ** 2).encode()).hexdigest()
        # checking the solution to problem, by using proof in cryptographic encryption
        # if proof results in 4 leading zero's in the hash operation, then: it is true
        if hash_operation[:4] == '0000':
            check_proof_of_work = True
        else:
            # if  solution is wrong, trying another chance until correct
            new_proof += 1
    return new_proof
```

5. hash(self, block): we can get sha256 by using the library hashlib.



```python
def hash(self, block):    # generate a hash of block
    encoded_block = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(encoded_block).hexdigest()
```

6. is_chain_valid(self, chain): Checking in two stages:

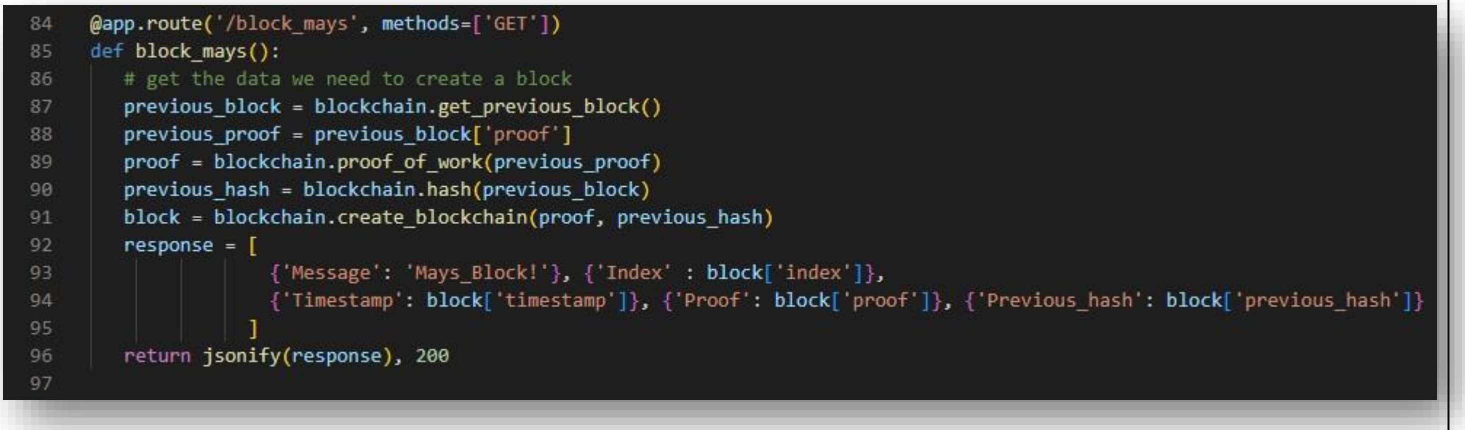


```python
def is_chain_valid(self, chain):   # checking if the chain is valid or not
    # Stage one we need to check if the current block has the same hash of the prvious one
    # make the first block in the chain as the previous block
    previous_block = chain[0]
    # an index of the blocks in the chain for iteration
    block_index = 1
    while block_index < len(chain):
        # get the current block
        block = chain[block_index]
        # checking the hashes if they are equal
        if block["previous_hash"] != self.hash(previous_block):
            return False
        # Stage two checking the proof
        # get the previous proof from the previous block
        previous_proof = previous_block['proof']

        # get the current proof from the current block
        current_proof = block['proof']

        # run the proof data through the algorithm
        hash_operation = hashlib.sha256(str(current_proof ** 2 - previous_proof ** 2).encode()).hexdigest()
        # check if hash operation is invalid
        if hash_operation[:4] != '0000':
            return False
        # set the previous block to the current block after running validation on current block
        previous_block = block
        block_index += 1
    return True
```

Flask API

Now after we build a blockchain class with the its function, we want to run it as an app on flask API:-

- First install the flask library from cmd [pip install flask].
- Then we want to build this app like this

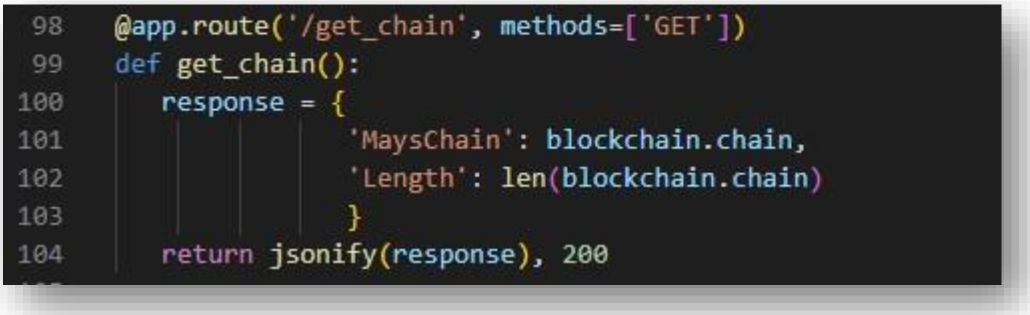


```python
@app.route('/block_mays', methods=['GET'])
def block_mays():
    # get the data we need to create a block
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
    block = blockchain.create_blockchain(proof, previous_hash)
    response = [
                {'Message': 'Mays_Block!'}, {'Index' : block['index']},
                {'Timestamp': block['timestamp']}, {'Proof': block['proof']}, {'Previous_hash': block['previous_hash']}
               ]
    return jsonify(response), 200
```

If we want to create a new block we just call method block_mays.

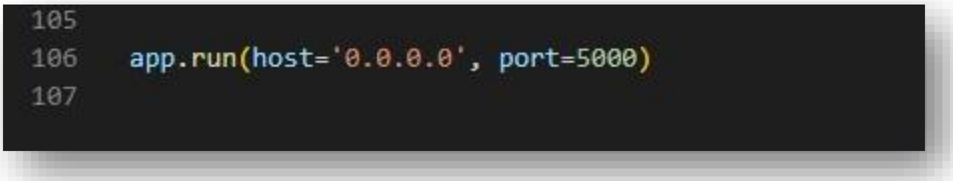
Then the output will be a josnify file to show the content of block.

- To show the chain that we build, call method get_chain, and this the implement of it



```python
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
                'MaysChain': blockchain.chain,
                'Length': len(blockchain.chain)
                }
    return jsonify(response), 200
```

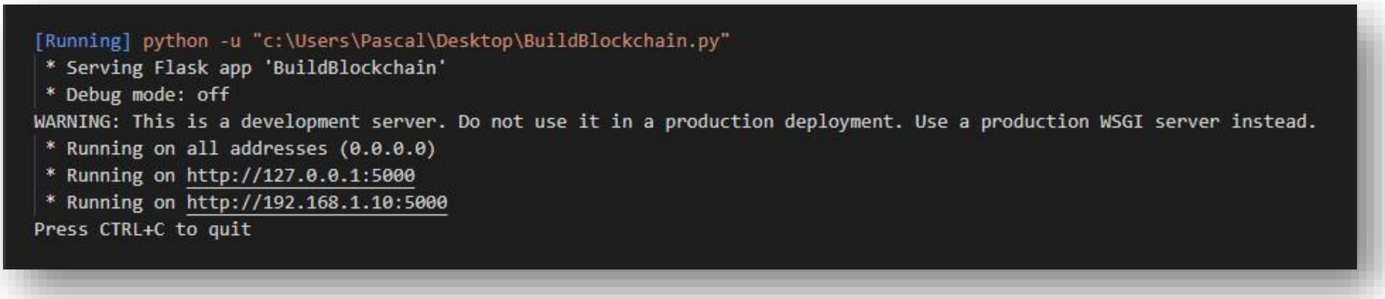
- To run the app on free port.



```python

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

```

Run The App

After click the run button this is the output:-

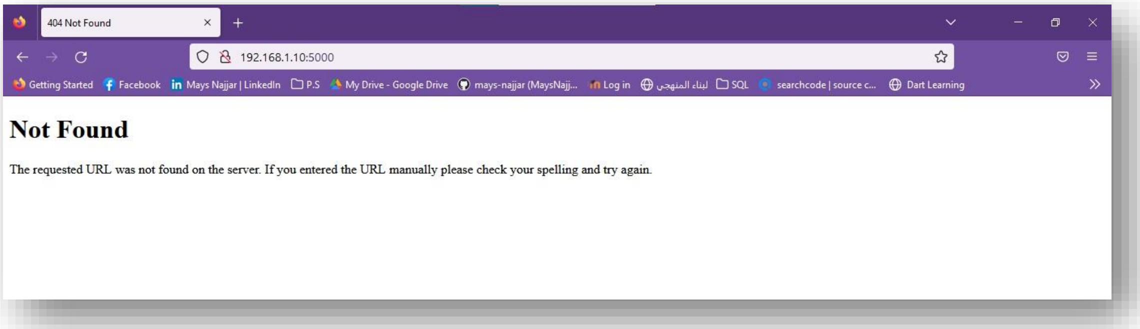


```
[Running] python -u "c:\Users\Pascal\Desktop\BuildBlockchain.py"
 * Serving Flask app 'BuildBlockchain'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.10:5000
Press CTRL+C to quit
```

So we want to go in http://192.168.1.10:5000

Or http://192.168.1.10:5000 to run our app

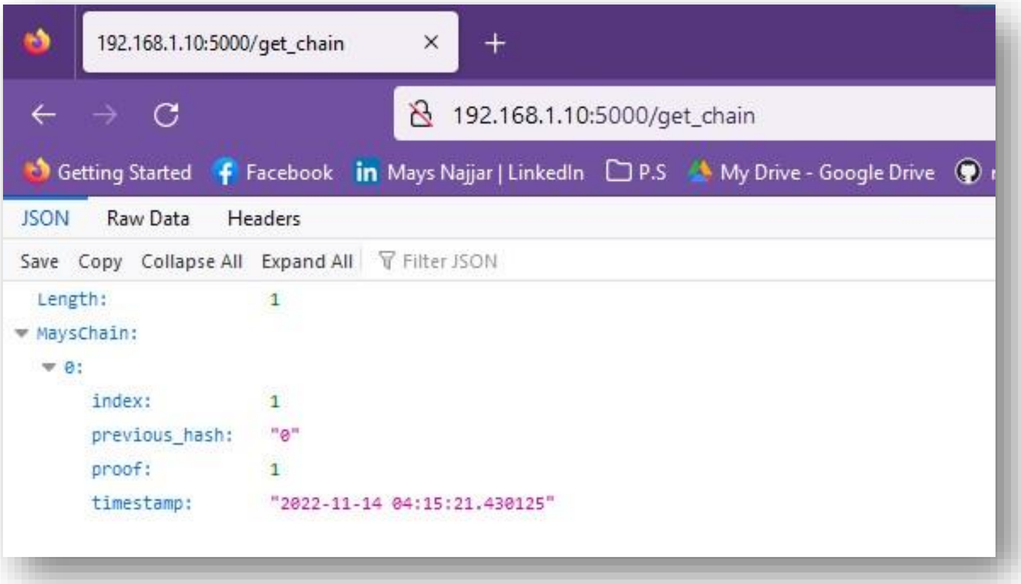
[hint to get better format of josnify use firefox browser



Not Found

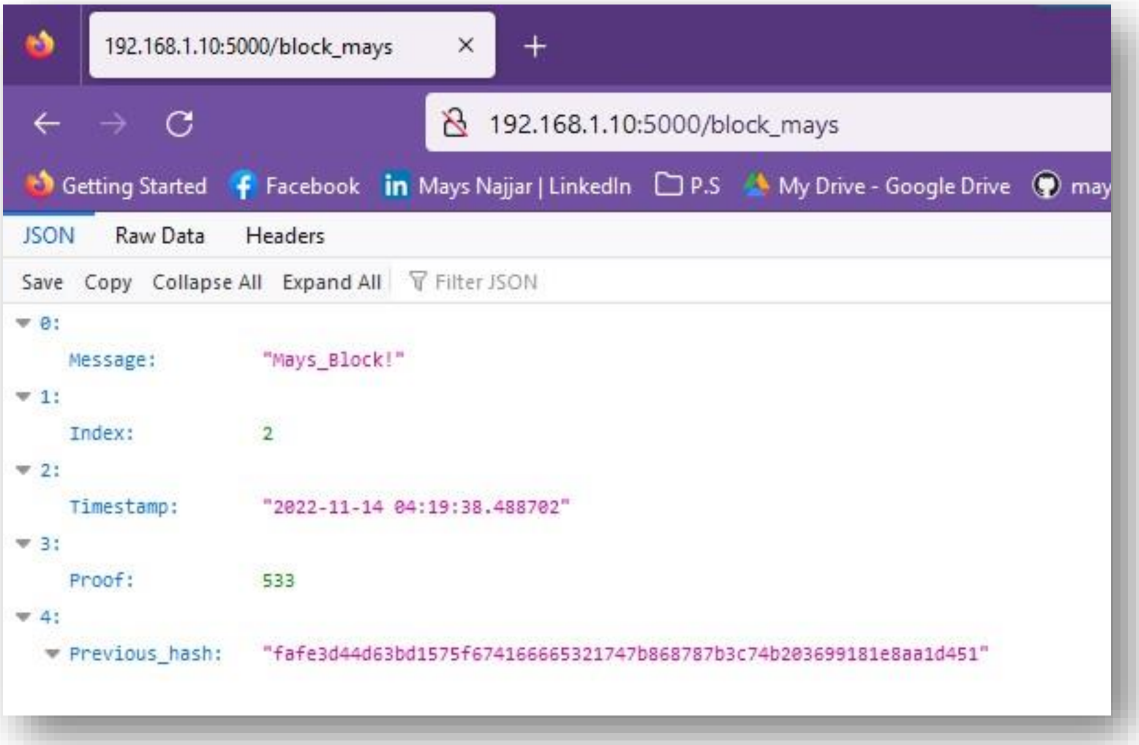
The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

This is what we get, so we want to call method get_chain to show our blockchain



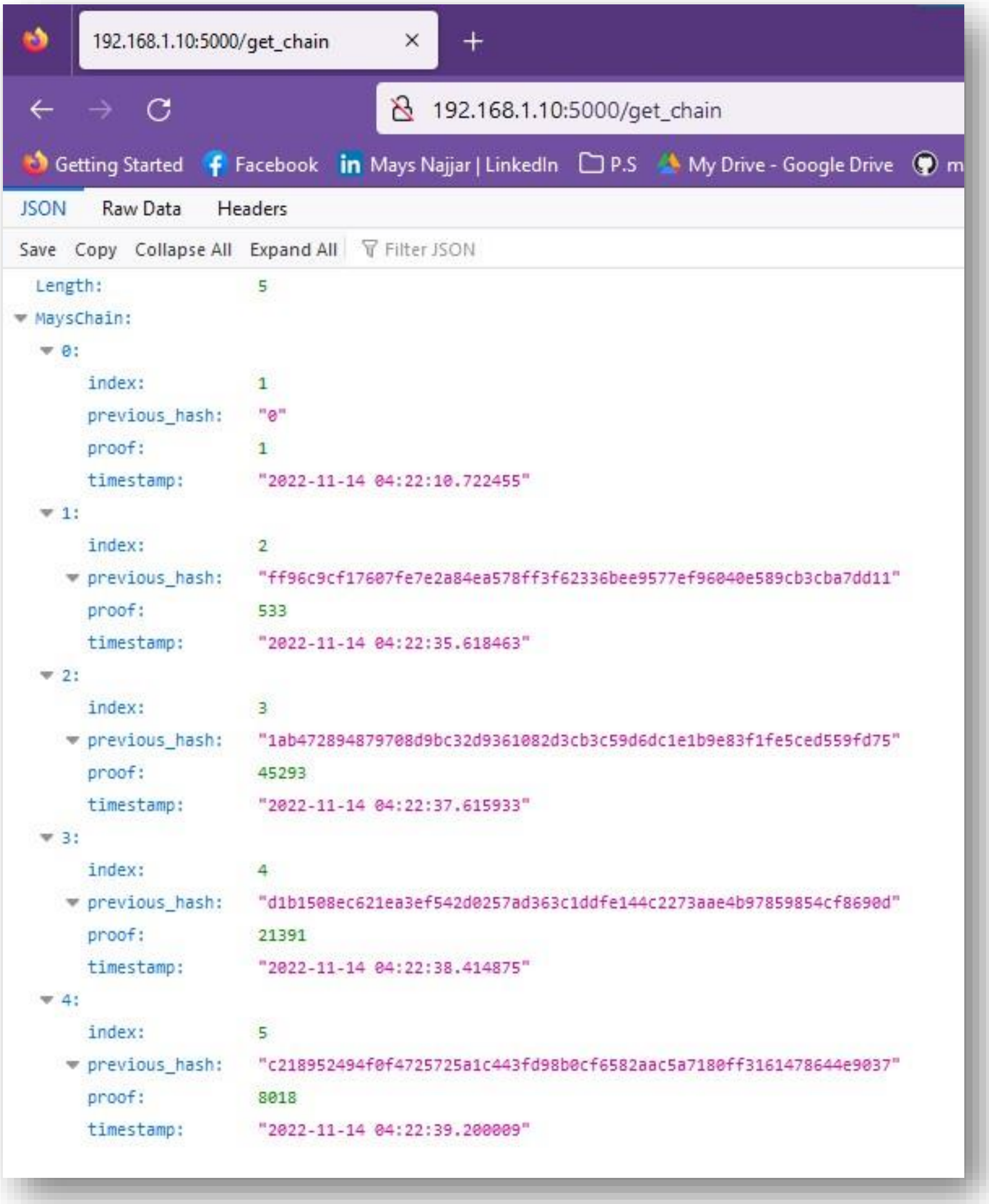
```
  Length:               1
▾ MaysChain:
  ▾ 0:
      index:            1
      previous_hash:    "0"
      proof:            1
      timestamp:        "2022-11-14 04:15:21.430125"
```

Now add a new block by calling method block_mays.



```
▼ 0:
    Message:          "Mays_Block!"
▼ 1:
    Index:            2
▼ 2:
    Timestamp:        "2022-11-14 04:19:38.488702"
▼ 3:
    Proof:            533
▼ 4:
  ▼ Previous_hash:    "fafe3d44d63bd1575f674166665321747b868787b3c74b203699181e8aa1d451"
```

Add another blocks to get 5 blocks, then call method get_chain



```
Length:              5
▼ MaysChain:
  ▼ 0:
      index:           1
      previous_hash:   "0"
      proof:           1
      timestamp:       "2022-11-14 04:22:10.722455"
  ▼ 1:
      index:           2
    ▼ previous_hash:   "ff96c9cf17607fe7e2a84ea578ff3f62336bee9577ef96040e589cb3cba7dd11"
      proof:           533
      timestamp:       "2022-11-14 04:22:35.618463"
  ▼ 2:
      index:           3
    ▼ previous_hash:   "1ab472894879708d9bc32d9361082d3cb3c59d6dc1e1b9e83f1fe5ced559fd75"
      proof:           45293
      timestamp:       "2022-11-14 04:22:37.615933"
  ▼ 3:
      index:           4
    ▼ previous_hash:   "d1b1508ec621ea3ef542d0257ad363c1ddfe144c2273aae4b97859854cf8690d"
      proof:           21391
      timestamp:       "2022-11-14 04:22:38.414875"
  ▼ 4:
      index:           5
    ▼ previous_hash:   "c218952494f0f4725725a1c443fd98b0cf6582aac5a7180ff3161478644e9037"
      proof:           8018
      timestamp:       "2022-11-14 04:22:39.200009"
```

The source code:

https://drive.google.com/file/d/1l5HTdE_8NrVFKzz41VSarFOIfgRNNNH5/view?usp=sharing