Project Report

Blockchain

Submitted to:
Sir Raja Muzammal

Submitted By:

Rohab Shabir	2021-CE-19
Javeria Zaheer	2021-CE-03
Yarushah E Sardar	2021-CE-11
Laiba Saad	2021-CE-08

Department of Computer Engineering

Table of Contents

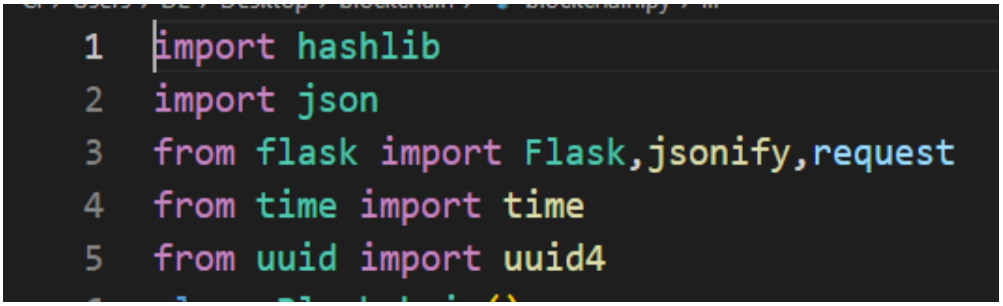
Abstract

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An *asset* can be tangible (a house, car, cash, and land) or intangible (intellectual property, patents, copyrights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved.

Components of Our Project

- ## Importing Modules

  In this part we import different python modules to include various useful functions in our code using the **import** keyword. We import **hashlib**, **json**, **flask**, **time** and **uuid**. From flask we import Flask class, jsonify method and request object.



```
1  import hashlib
2  import json
3  from flask import Flask,jsonify,request
4  from time import time
5  from uuid import uuid4
```

- ## Creating a Blockchain class :

  - Make a class named Blockchain
  - Setting parameters for chain and pending transactions
  - Creating a method to form blocks and adding it to the chain
  - Creating another method to access the last block of the chain

In this part we create a blockchain class. In the constructor we create attributes of chain and pending transactions with list data type.
Then we create a **new block** method to add new blocks to our blockchain. Our new block will be a dictionary which contains the index, timestamp, transactions, proof and previous hash. Next we create a **last block** method which returns the last block of chain. We use property decorator to use this method as a property.
Our next method is a **new transaction** method. In this method we make a dictionary which has the values of sender, recipient and amount. Then we append this dictionary to our list of pending transactions.

```
class Blockchain():
    def __init__(self):
        self.chain = []
        self.pending_transactions = []
        self.new_block(proof=1,previous_hash='GENESISBLOCK')
    def new_block(self, proof, previous_hash):
            block = {
                'index': len(self.chain) + 1,
                'timestamp': time(),
                'transactions': self.pending_transactions,
                'proof': proof,
                'previous_hash': previous_hash or self.hash(self.chain[-1])
            }
            self.pending_transactions = []
            self.chain.append(block)
            return block
```

```
def new_transaction(self, sender, recipient, amount):
        transaction = {
            'sender': sender,
            'recipient': recipient,
            'amount': amount
        }
        self.pending_transactions.append(transaction)
        return self.last_block['index'] + 1
```

- **Calculating previous block hash:**

    - Creating a Hash method by using block as a parameter
    - Using json.dumps() method to convert a block dictionary with values and keys to a string object
    - Using hashing algorithm(SHA256) to generate hash(a sequence of fixed length strings)

```
def hash(self, block):
        string_object = json.dumps(block, sort_keys=True)
        block_string = string_object.encode()
        raw_hash = hashlib.sha256(block_string)
        hex_hash = raw_hash.hexdigest()
        return hex_hash
```

Here our block dictionary is converted into a json string by using dumps method. Then we encode this string into bytes and pass it to our sha256 algorithm to generate a raw hash in the form of bits. This raw hash is converted to a hex hash by using hexdigest method and the hexadecimal hash which is a string is returned by this hash method.

- **Calculating proof of work:**

  - Using proof of work(which is a number) to validate the chain
  - Applying a condition to run the program until a specific hash is generated
  - Incrementing the new_proof until a hash starting with '00000' is generated
  - Return the proof number

```
def proof(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[:5] == '00000':
            check_proof = True
        else:
            new_proof += 1
    return new_proof
```

The while loop runs until a hash starting with five zeros is generated. In that case check proof is set to true and our loop breaks, giving us the number of times it took our code to generate a hash with our specified condition. This number is our proof and it validates our blockchain because sha256 algorithm is a very secure algorithm and only one hash can be generated for one input.

- **Setting up a Flask app:**
  - Setup Flask application in the visual studio
  - Flask helps to build a web application using a single python file

```
nodes=set()
app = Flask(__name__)
node_identifier = str(uuid4()).replace('-', '')
blockchain = Blockchain()
```

Flask is a python web framework used to create web apps with python. We create a flask object "**app**" and give it a variable __name__ which has the special value of __main__. We will run this app on our local server to interact with our blockchain.
We create a set object "**nodes**" because we will be using it later in our code while registering new nodes. **Node identifier** is a unique random id generated using uuid4 function. It represents the address of miner and we will see later why it is created. Finally, we initialize our blockchain by creating a blockchain object.

- **Creating new transactions:**

Now that we have created an app, we will add different endpoints for interacting with our blockchain. The new transactions end point has the rule URL of transactions/new and view function of new transactions. We register pou function with the url by using route decorator. This endpoint has a post method which means that client will enter data on server in json format which will be fetched by get_json method of request object. Then we will check if all our required values are entered by the user or not. If yes, we will give a response of json application/mimetype that transactions will be added to next block. Otherwise, we will give a bad request error to user.

```
@app.route('/transactions/new',methods=['POST'])
def new_transactions():
    values = request.get_json()
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message':'Transaction will be added to next Block'}
    return jsonify(response), 200
```

- **Mining a new block:**

After creating new transactions we will mine a block by calculating previous block hash and proof of work. This endpoint has the rule url **/mine_block** and view function mine_block. We will call our new block method and return our block dictionary as a json response.

```
@app.route('/mine_block', methods=['GET'])
def mine_block():
    previous_block = blockchain.last_block
    previous_proof = previous_block['proof']
    proof = blockchain.proof(previous_proof)
    previous_hash = blockchain.hash(previous_block)
    blockchain.new_transaction("sender",node_identifier,"10BTC")
    block = blockchain.new_block(proof, previous_hash)
    response = {'message': 'A block is MINED',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
                'transactions':block['transactions']}
    return jsonify(response), 200
```

- **Registering new nodes:**

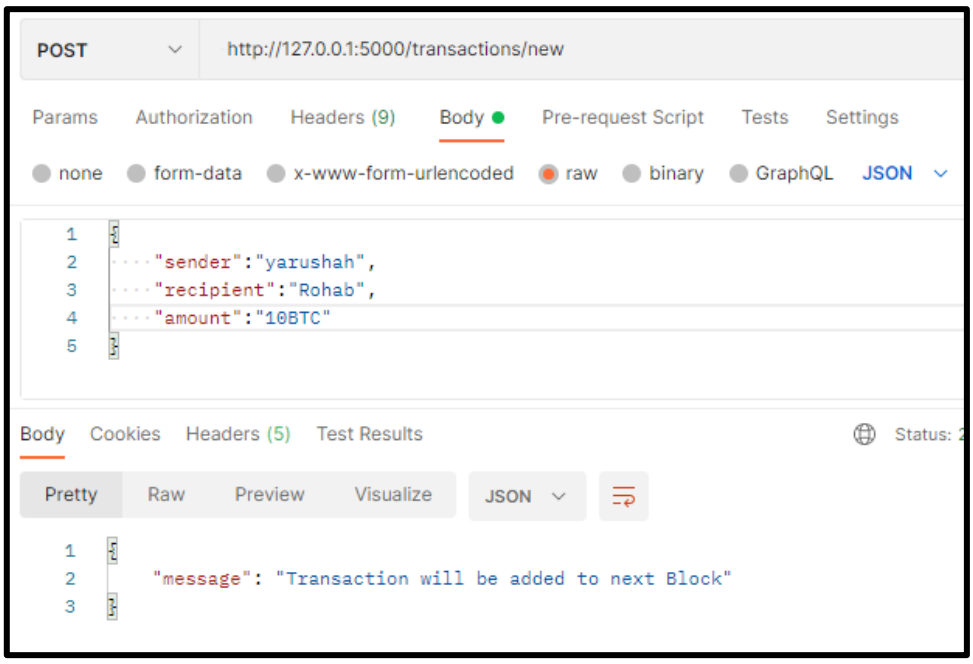
Our next endpoint is to register new nodes using post method. The user will enter a list of nodes, get_json method will get the list from server, we will loop through the list and add the nodes to our set object **nodes.** this set object will make sure that same nodes are considered as one. Then we will show our chain in response to sync the node with our current node.Now that we are done with creating endpoints we will call the run function on our flask app to run our app on our specified host and port number. Our app will run on a local development server.

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    new_nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in new_nodes:
        nodes.add(node)
    response ={'message':'new nodes have been added',
               'chain': blockchain.chain,
               'pending_transactions':blockchain.pending_transactions,
                'length': len(blockchain.chain)}
    return jsonify(response),200
app.run(host='127.0.0.1', port=5000)
```

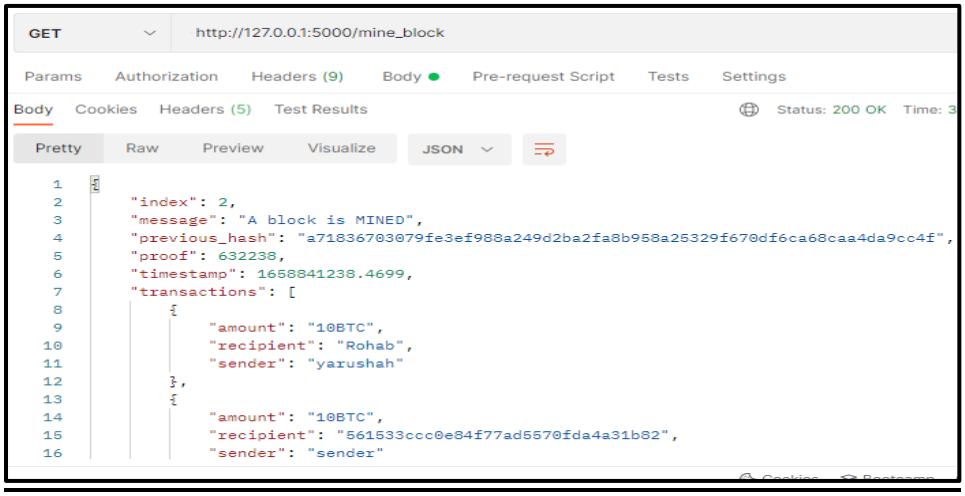
- **Interacting with blockchain postman:**

  Postman is an API platform for handling http requests. We will use get and post methods to access our functions and view the response.

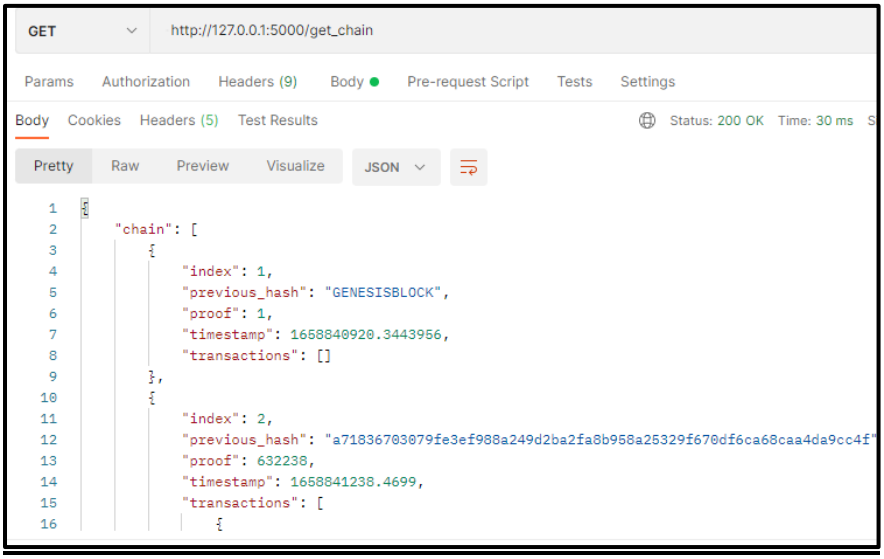
  New transactions response:



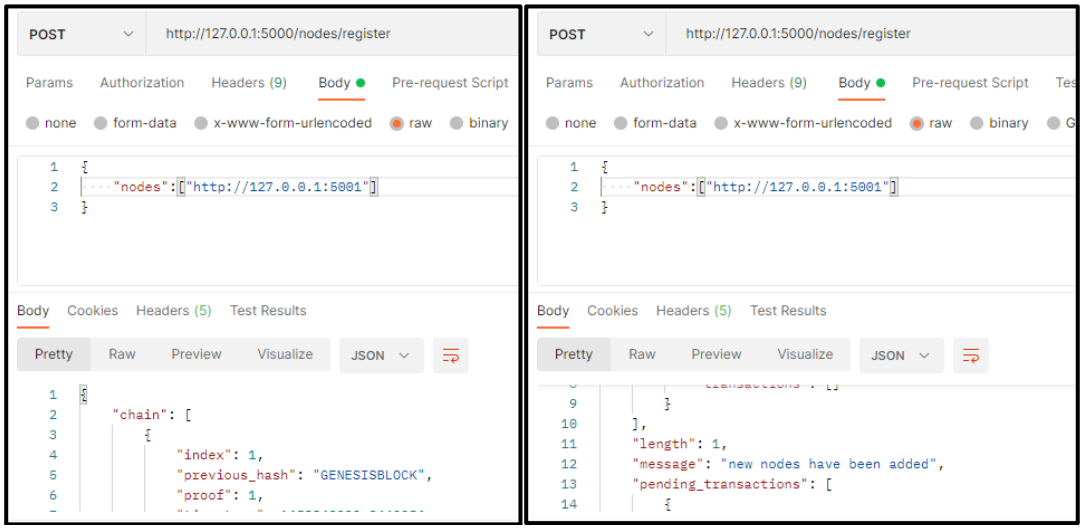
- **Mine block response:**



- **Get chain response:**



- **Registering new nodes response:**

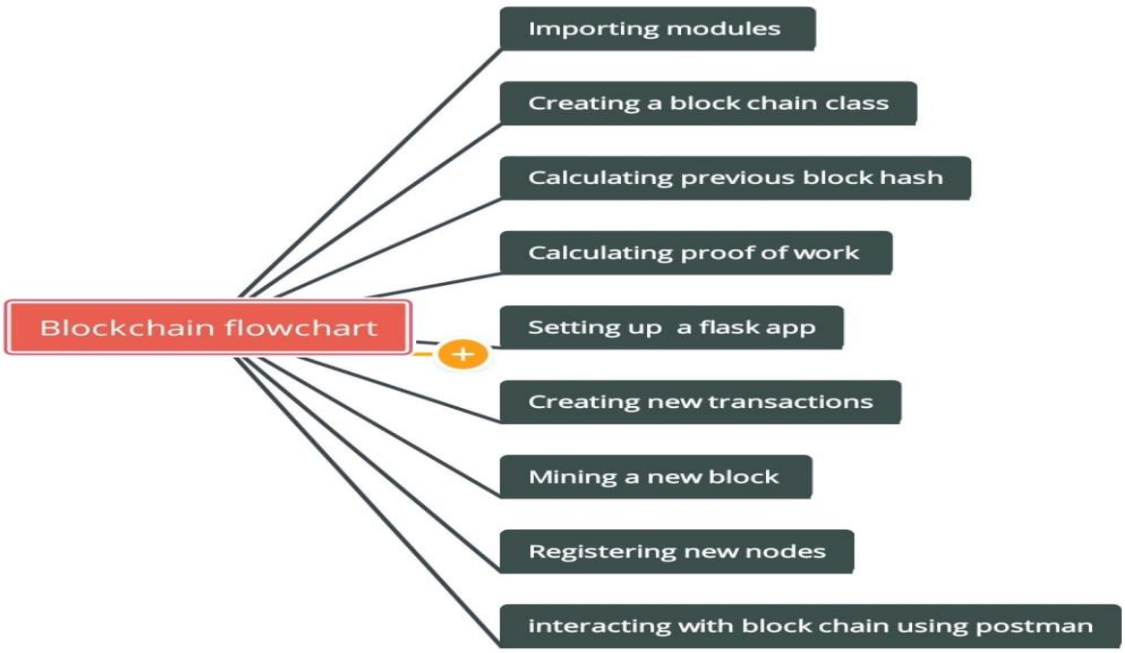


3. Socio-economic benefits:

If a hacker hacks a block in the chain, the hashes of all the later blocks will become incorrect. This gives blockchains immutability. This provides a platform to make transactions without the help of any bank or mega corporation.

Without blockchain, each organization has to keep a separate database. Because blockchain uses a distributed ledger, transactions and data are recorded identically in multiple locations. All transactions are immutability recorded, and are time- and date-stamped. This enables members to view the entire history of a transaction and virtually eliminates any opportunity for fraud.

4.Flowchart:



TOOL/SOFTWARE

- Visual Studio Code

6.REFERENCES

- https://hackernoon.com/learn-blockchains-by-building-one-117428612f46

- https://medium.com/coinmonks/python-tutorial-build-a-blockchain-713c706f6531