Pranav Titambe, D20A, 60

Blockchain Lab
EXP 3

Aim - Create a Cryptocurrency using Python and perform mining in the Blockchain created.

Theory -

1. Blockchain Overview

A blockchain is a **distributed, decentralized digital ledger** used to record transactions securely across multiple computers (nodes). Instead of relying on a central authority, every node in the network maintains its own copy of the blockchain.
Each block in the blockchain contains:

- A list of transactions
- A timestamp
- The hash of the previous block
- Its own cryptographic hash

The blocks are linked together using hashes, forming a chain. Any attempt to modify data in a block changes its hash, which breaks the chain and makes tampering easily detectable. This property ensures **data integrity, transparency, and immutability**.

2. Mining

Mining is the process by which new blocks are added to the blockchain. It involves:

- Collecting pending transactions
- Creating a block header
- Solving a computationally difficult problem known as **Proof-of-Work (PoW)**

In Proof-of-Work, miners repeatedly change a nonce value to generate a hash that satisfies the network's difficulty requirement (such as leading zeroes). Once a valid hash is found, the block is added to the blockchain and shared with other nodes. As an incentive, miners receive a **cryptocurrency reward** for their computational effort.

3. Multi-Node Blockchain Network

In this experiment, a **multi-node blockchain network** is simulated using three nodes running on different ports (5001, 5002, and 5003).
Each node:

- Operates independently
- Maintains its own copy of the blockchain
- Communicates with peer nodes to share newly mined blocks

This setup demonstrates how blockchain achieves decentralization and synchronization without a central server.

4. Consensus Mechanism
To ensure consistency across all nodes, the **Longest Chain Rule** is used as the consensus mechanism.
When different versions of the blockchain exist, the node accepts the **longest valid chain** as the correct one. This rule ensures that all nodes eventually agree on a single transaction history, even if temporary differences arise.

5. Transactions and Mining Reward
Transactions in the cryptocurrency system include:

- Sender address
- Receiver address
- Transaction amount

When a block is mined, all pending transactions are added to the block. Additionally, a special transaction is created to reward the miner with newly generated cryptocurrency. This reward mechanism motivates miners to maintain and secure the network.

6. Chain Replacement
The chain replacement process ensures network consistency. When the /replace_chain endpoint is triggered:

- A node requests blockchain data from its peers
- Validates the received chains
- Replaces its own chain if a longer and valid chain is found

This mechanism helps resolve conflicts and keeps all nodes synchronized.

Code -

```
class Blockchain:
    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(proof=1, previous_hash='0')
        self.nodes = set()
    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
```

```
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash,
            'transactions': self.transactions
        }
        self.transactions = []
        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 not check_proof:
            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()
```

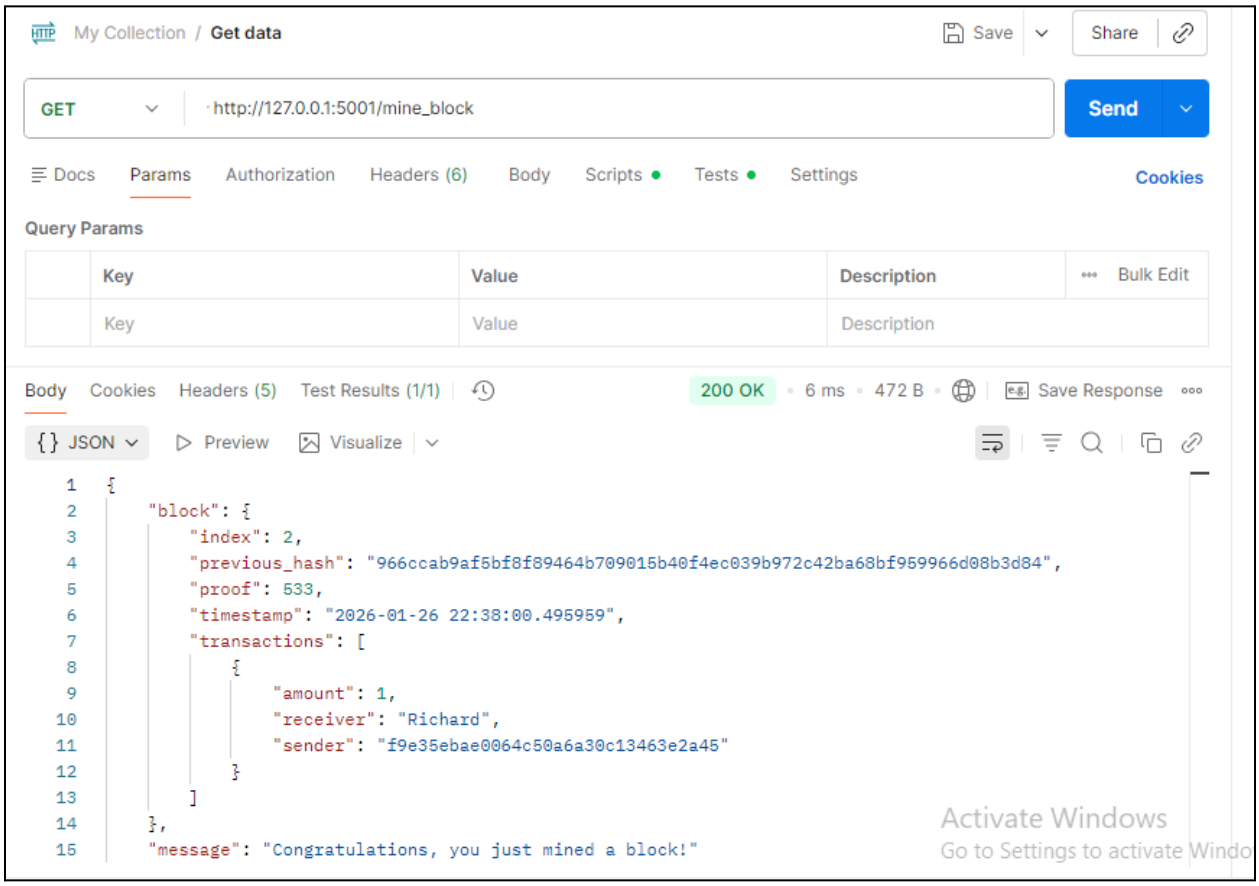
```
            ).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
            block_index += 1
        return True

    def add_transaction(self, sender, receiver, amount):
        self.transactions.append({
            'sender': sender,
            'receiver': receiver,
            'amount': amount
        })
        previous_block = self.get_previous_block()
        return previous_block['index'] + 1

    def add_node(self, address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)

    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            response = requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):
                    max_length = length
                    longest_chain = chain
        if longest_chain:
            self.chain = longest_chain
            return True
        return False

app = Flask(__name__)

node_address = str(uuid4()).replace('-', '')
```

Pranav Titambe, D20A, 60

```
blockchain = Blockchain()

@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)
    blockchain.add_transaction(
        sender=node_address,
        receiver='Richard',
        amount=1
    )
    block = blockchain.create_block(proof, previous_hash)
    response = {
        'message': 'Congratulations, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
        'transactions': block['transactions']
    }
    return jsonify(response), 200

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

@app.route('/is_valid', methods=['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'The Blockchain is not valid.'}
```

```
    return jsonify(response), 200

@app.route('/add_transaction', methods=['POST'])
def add_transaction():
    json_data = request.get_json()
    transaction_keys = ['sender', 'receiver', 'amount']
    if not all(key in json_data for key in transaction_keys):
        return 'Some elements of the transaction are missing', 400
    index = blockchain.add_transaction(
        json_data['sender'],
        json_data['receiver'],
        json_data['amount']
    )
    response = {
        'message': f'This transaction will be added to Block {index}'
    }
    return jsonify(response), 201

@app.route('/connect_node', methods=['POST'])
def connect_node():
    json_data = request.get_json()
    nodes = json_data.get('nodes')
    if nodes is None:
        return "No node", 400
    for node in nodes:
        blockchain.add_node(node)
    response = {
        'message': 'All the nodes are now connected.',
        'total_nodes': list(blockchain.nodes)
    }
    return jsonify(response), 201

@app.route('/replace_chain', methods=['GET'])
def replace_chain():
    is_chain_replaced = blockchain.replace_chain()
    if is_chain_replaced:
        response = {
            'message': 'The chain was replaced by the longest one.',
            'new_chain': blockchain.chain
        }
```

```
    else:
        response = {
            'message': 'The chain is already the largest one.',
            'actual_chain': blockchain.chain
        }
    return jsonify(response), 200

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



Pranav Titambe, D20A, 60