Name: Diya Theryani **Div:** D20B **Roll No.:** 66

Experiment 3

AIM: Implement peer-to-peer network and implement Mining using block chain using Python

Task to be performed:

1. Update the Blockchain created in the previous experiment
2. Create a decentralized network
3. Perform transactions among the peers.
4. Mining the Blockchain

Tools & Libraries used:

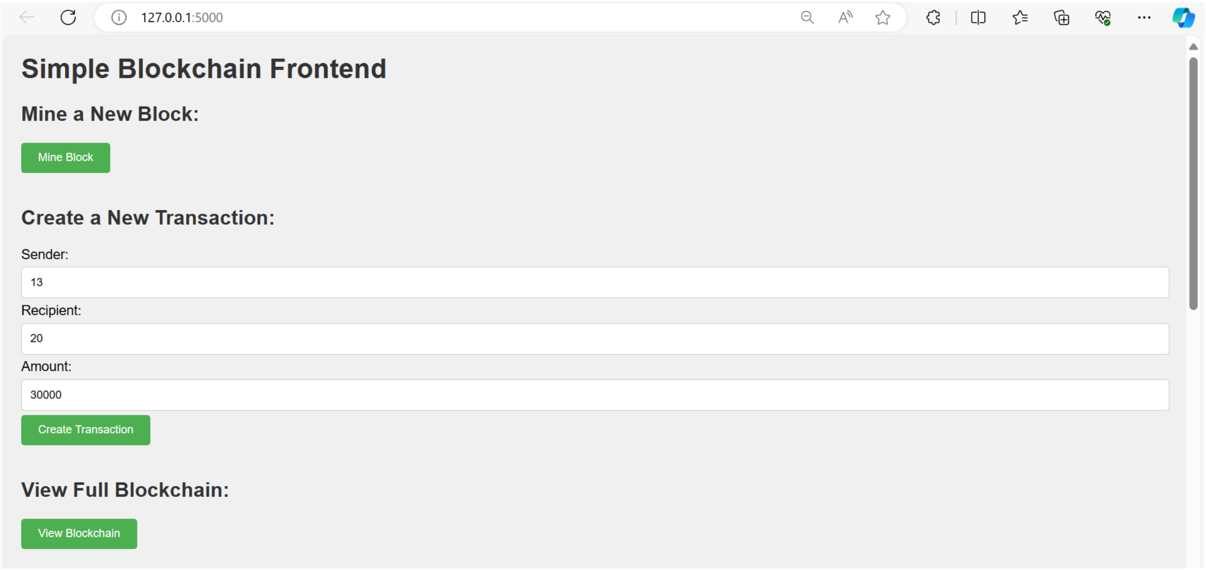
Install Flask : pip install Flask
Download Postman from https://www.postman.com/
Python Libraries : datetime, jsonify, hashlib, uuid4, urlparse, request
Install requests : pip install requests==2.18.4

CODE:

```
from flask import Flask, jsonify, request, render_template
import hashlib
import json
import random
from time import time


app = Flask(__name__)


class Blockchain:
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.nodes = set()  # Set to store node addresses
        self.new_block(previous_hash='1', proof=100)


    def register_node(self, address):
        # Add a new node to the set of nodes
        self.nodes.add(address)


    def new_block(self, proof, previous_hash=None):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or
self.hash(self.chain[-1]),
            'miner_id': random.randint(1, 1000)  # Random
miner ID
        }


        # Reset the current list of transactions
        self.current_transactions = []


        self.chain.append(block)
        return block


    def new_transaction(self, sender, recipient, amount,
node):
        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
            'node': node  # Include the node information in
the transaction
        })
        return self.last_block['index'] + 1


    @staticmethod
    def hash(block):
        block_string = json.dumps(block,
sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()


    @property
    def last_block(self):
        return self.chain[-1]
    def proof_of_work(self, last_proof):
        proof = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1
        return proof


    @staticmethod
    def valid_proof(last_proof, proof):
        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"

blockchain = Blockchain()

# Dummy nodes
dummy_nodes = ["node 1", "node 2", "node 3", "node 4", "node
5"]
for node in dummy_nodes:
    blockchain.register_node(node)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/mine', methods=['GET'])
def mine():
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)


    # Get a random node that will add the transaction
    node = random.choice(list(blockchain.nodes))


    sender = "node " + str(random.randint(1, 5))
    recipient = "node " + str(random.randint(1, 5))
    amount = random.randint(1, 10)

        blockchain.new_transaction(sender, recipient, amount,
node)
```

```
    previous_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, previous_hash)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
         'miner_id': block['miner_id'],  # Include miner's ID
in the response
         'added_by_node': node  # Include the node that added
the transaction
    }
    return jsonify(response), 200

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # Get a random node that will add the transaction
    node = random.choice(list(blockchain.nodes))

    sender = "node " + str(random.randint(1, 5))
    recipient = "node " + str(random.randint(1, 5))
    amount = values.get('amount')


        blockchain.new_transaction(sender, recipient, amount,
node)


    response = {
          'message': f'Transaction will be added to the next
block',
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
         'added_by_node': node  # Include the node that added
the transaction
    }
    return jsonify(response), 201
```

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


if __name__ == '__main__':
    app.run(debug=True)
```



Simple Blockchain Frontend

Mine a New Block:

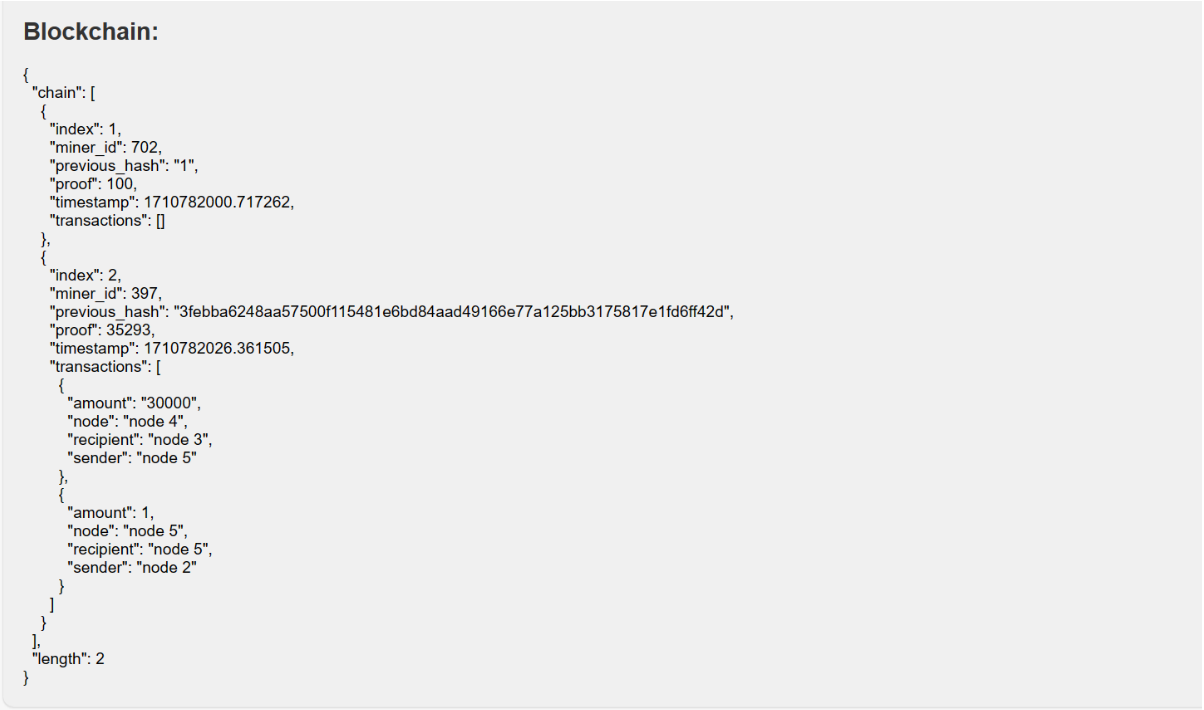
Create a New Transaction:

Sender:
13

Recipient:
20

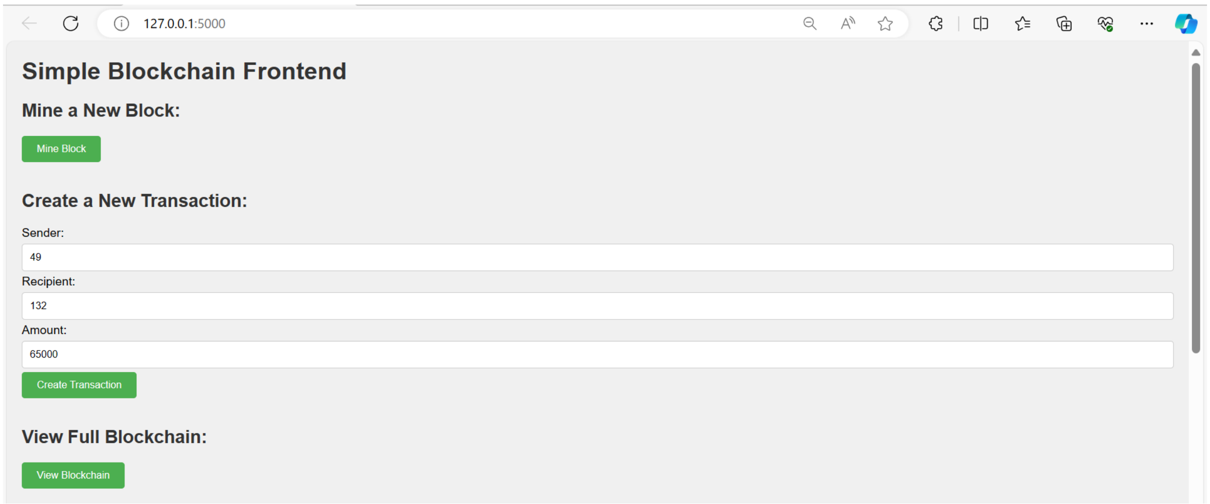
Amount:
30000

View Full Blockchain:



Blockchain:

```
{
  "chain": [
    {
      "index": 1,
      "miner_id": 702,
      "previous_hash": "1",
      "proof": 100,
      "timestamp": 1710782000.717262,
      "transactions": []
    },
    {
      "index": 2,
      "miner_id": 397,
      "previous_hash": "3febba6248aa57500f115481e6bd84aad49166e77a125bb3175817e1fd6ff42d",
      "proof": 35293,
      "timestamp": 1710782026.361505,
      "transactions": [
        {
          "amount": "30000",
          "node": "node 4",
          "recipient": "node 3",
          "sender": "node 5"
        },
        {
          "amount": 1,
          "node": "node 5",
          "recipient": "node 5",
          "sender": "node 2"
        }
      ]
    }
  ],
  "length": 2
}
```



Simple Blockchain Frontend

Mine a New Block:

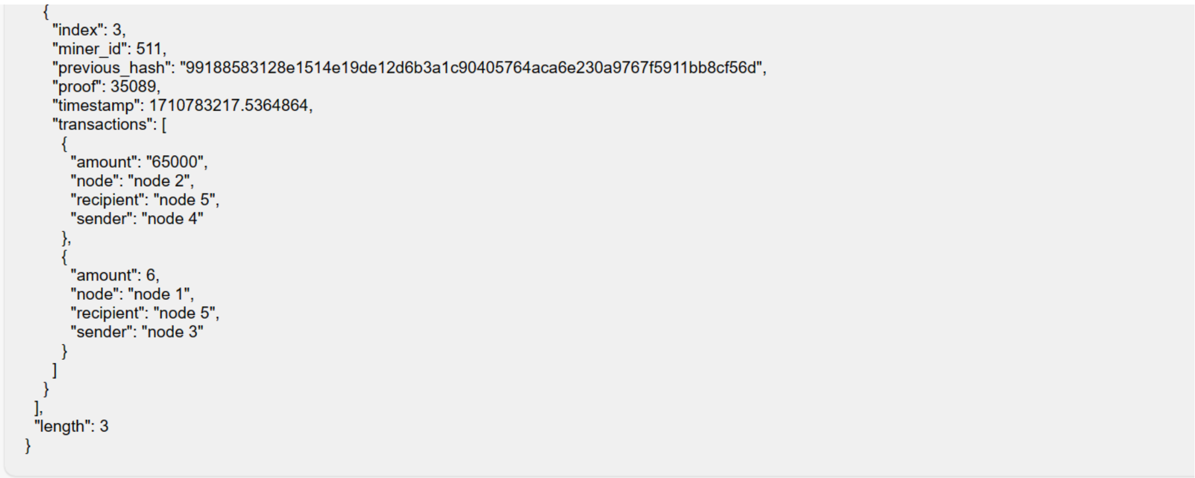
Create a New Transaction:

Sender: 49

Recipient: 132

Amount: 65000

View Full Blockchain:



```
    {
      "index": 3,
      "miner_id": 511,
      "previous_hash": "99188583128e1514e19de12d6b3a1c90405764aca6e230a9767f5911bb8cf56d",
      "proof": 35089,
      "timestamp": 1710783217.5364864,
      "transactions": [
        {
          "amount": "65000",
          "node": "node 2",
          "recipient": "node 5",
          "sender": "node 4"
        },
        {
          "amount": 6,
          "node": "node 1",
          "recipient": "node 5",
          "sender": "node 3"
        }
      ]
    }
  ],
  "length": 3
}
```

CONCLUSION:

In this experiment, we have successfully set up a peer-to-peer network and we have created dummy nodes which are responsible for mining the block when transactions are added.