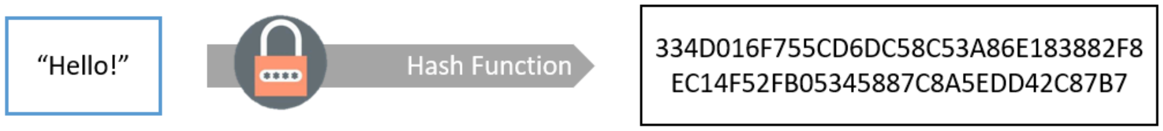
Practical 2

Aim: To create a blockchain using Python.

Theory

Cryptographic Hash Function

Hashing is converting an original piece of data into a digest or hash. The process uses cryptographic hash functions for the irreversible conversion of the message.


"Hello!"
Hash Function
334D016F755CD6DC58C53A86E183882F8
EC14F52FB05345887C8A5EDD42C87B7

Cryptographic hash functions are irreversible. That means it's a 1-way function, and one can't generate the message back using the digest. There are a bunch of cryptographic hash functions. For example, SHA-224, SHA-256, SHA-512, KECCAK-256, Whirlpool, etc.

Merkle Tree

Merkle trees, also known as Binary hash trees, are a prevalent sort of data structure in computer science. In bitcoin and other cryptocurrencies, they're used to encrypt blockchain data more efficiently and securely. It's a mathematical data structure made up of hashes of various data blocks that summarize all the transactions in a block. It also enables quick and secure content verification across big datasets and verifies the consistency and content of the data.

Benefits of Merkle Tree

- Validate the data's integrity: It can be used to validate the data's integrity effectively.
- Takes little disk space: Compared to other data structures, the Merkle tree takes up very little disk space.
- Tiny information across networks: Merkle trees can be broken down into small pieces of data for verification.
- Efficient Verification: The data format is efficient, and verifying the data's integrity takes only a few moments.

Cryptographic Puzzle

A cryptographic puzzle in proof-of-work blockchains is a mathematical challenge miners must solve to add a new block to the chain. To solve the puzzle, miners must find a block with a hash value that starts with a specific number of zeros, known as mining difficulty. They do this by trying different numbers, called nonces, until they get a hash that meets the difficulty requirement. The first miner to solve the puzzle gets to add the block to the blockchain and earn a reward. This process helps secure the blockchain and control the rate at which new blocks are added.

Working of Merkle Trees

A Merkle tree totals all transactions in a block and generates a digital fingerprint of the entire set of operations, allowing the user to verify whether it includes a transaction in the block.

- Merkle trees are made by hashing pairs of nodes repeatedly until only one hash remains; this hash is known as the Merkle Root or the Root Hash.
- They're built from the bottom, using Transaction IDs, which are hashes of individual transactions.
- Each non-leaf node is a hash of its previous hash, and every leaf node is a hash of transactional data.

Mining in blockchain

Mining is the process that Bitcoin and several other cryptocurrencies use to generate new coins and verify new transactions.

Mining in blockchain involves using computers to solve complex mathematical problems that validate and record transactions on the blockchain. When a miner successfully solves these problems, they add a new block of transactions to the blockchain and are rewarded with cryptocurrency. This process ensures the integrity and security of the blockchain by preventing fraud and double-spending.

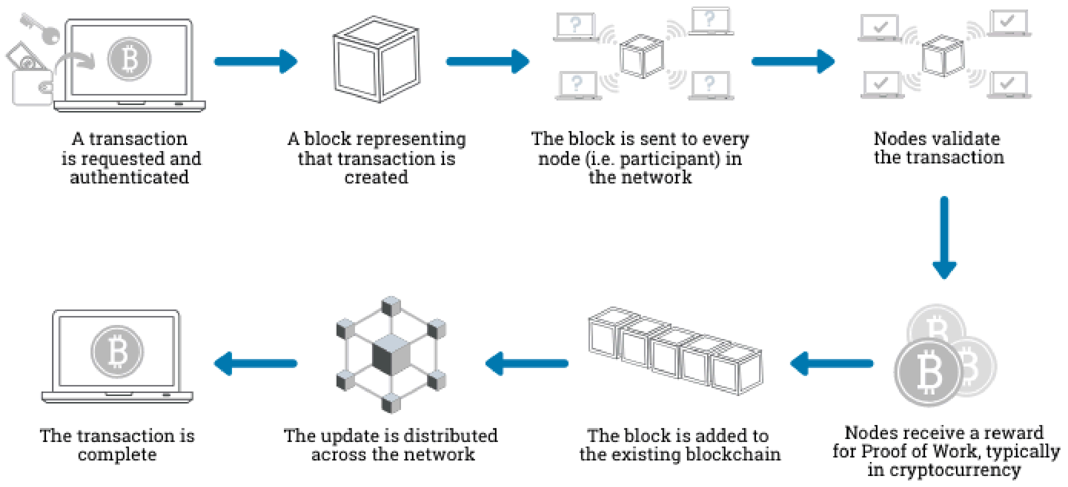
Checking validity of the blockchain

To check the validity of blocks, nodes verify that each block's hash matches the expected value, confirm that all transactions within the block are legitimate and follow the blockchain's rules, and ensure the block's previous hash correctly references the previous block. This ensures the integrity and continuity of the blockchain.

Challenges in P2P network

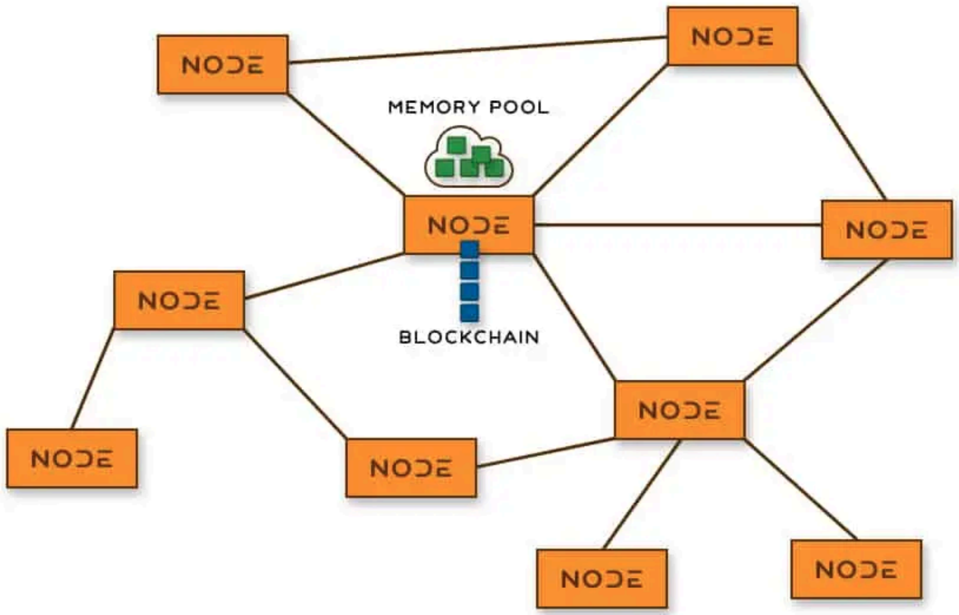
Peer-to-peer (P2P) networks face challenges such as managing data consistency and synchronization, dealing with network security and privacy issues, handling varying node reliability and performance, and addressing scalability as the number of nodes increases.

How are transactions performed on a network?


A transaction is requested and authenticated
A block representing that transaction is created
The block is sent to every node (i.e. participant) in the network
Nodes validate the transaction
Nodes receive a reward for Proof of Work, typically in cryptocurrency
The block is added to the existing blockchain
The update is distributed across the network
The transaction is complete

What is mempool?

A mempool, or memory pool, is where unconfirmed transactions are stored before being included in a block. It acts as a waiting area where transactions are kept until they are validated by miners and added to the blockchain.



Libraries and tools used during implementation

datetime: Essential for timestamping transactions and blocks to ensure accurate recording of when events occur on the blockchain. For instance, blocks might include timestamps to record when they were mined or transactions were initiated.

jsonify: Useful for creating responses in web APIs that interact with blockchain nodes. For example, it can be used to format blockchain data, like transaction details or block information, into JSON to be sent over HTTP.

hashlib: Critical for creating cryptographic hashes of blocks and transactions. Hashing ensures data integrity by generating unique, fixed-size representations of variable-size input data. In blockchain, hashes are used to link blocks and secure transaction data.

uuid4: Useful for creating unique IDs for transactions or blocks. This helps avoid duplication and ensures that each transaction or block has a distinct identifier, which is important for tracking and referencing in the blockchain.

urlparse: Helps in handling and processing URLs in blockchain applications, such as when interacting with APIs or network nodes. It allows for easier extraction and manipulation of URL components needed for network communication.

request: Facilitates communication between different blockchain nodes or between a client and server. It is used to send and receive data over the internet, such as querying block data from a node or sending transaction information.

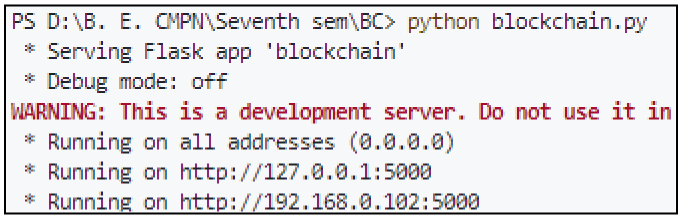
Code

```python
import datetime
import hashlib
import json
from flask import Flask, jsonify, request

# Part 1 - Building a Blockchain
class Blockchain:
    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(self.proof_of_work(self.get_temp_block('0')))

    def create_block(self, block):
        self.chain.append(block)
        self.transactions = self.transactions[5:]
        return block

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

    def hash(self, block):
        encoded_block = json.dumps(block, sort_keys = True).encode()
        return hash(encoded_block)

    def proof_of_work(self, temp_block):
        new_proof = 1
        check_proof = False
        while check_proof is False:
            temp_block['nonce'] = new_proof
            hash_operation = self.hash(temp_block)
            if hash_operation.startswith('000'):
                check_proof = True
            else:
                new_proof += 1
        return temp_block

    def create_transaction(self, sender, receiver, amount):
        self.transactions.append({'sender': sender, 'receiver': receiver,
                                  'amount': amount})
        previous_block = self.get_previous_block()
        return "Your transaction has been added to pool."

    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
            if not self.hash(block).startswith('000'):
                return False
            previous_block = block
            block_index += 1
        return True
    def get_temp_block(self, previous_hash):
        block = {'index': len(self.chain) + 1,
                 'nonce': 1,
                 'timestamp': str(datetime.datetime.now()),
                 'previous_hash': previous_hash,
                 'transactions': self.transactions[:5],
                 'merkle_root': get_merkle_root(self.transactions[:5])}
        return block

# Create merkle tree
def get_merkle_root(transactions):
    if len(transactions) == 0:
        return None
    if len(transactions) == 1:
        return hash(transactions[0])
    # hash all transactions before building merkle tree
    hashed_transactions = []
    for i in range(len(transactions)):
        hashed_transactions.append(hash(transactions[i]))
    # build merkle tree
    level = 0
    while len(hashed_transactions) > 1:
        if len(hashed_transactions) % 2 != 0:
            hashed_transactions.append(hashed_transactions[-1])

        new_transactions = []
        for i in range(0, len(hashed_transactions), 2):
            combined = hashed_transactions[i] + hashed_transactions[i+1]
            hash_combined = hash(combined)
            new_transactions.append(hash_combined)
        hashed_transactions = new_transactions
        level += 1
    return hashed_transactions[0]

# Helper function to hash
def hash(value):
    return hashlib.sha256(str(value).encode('utf-8')).hexdigest()

# Part 2 - Mining our Blockchain
app = Flask(__name__)
b = Blockchain()

# Mining a new block
@app.route('/mine_block', methods = ['GET'])
def mine_block():
    if len(b.transactions) > 0:
        previous_block = b.get_previous_block()
        previous_hash = b.hash(previous_block)
        temp_block = b.proof_of_work(b.get_temp_block(previous_hash))
        block = b.create_block(temp_block)
        response = {'message': 'Congratulations, you just mined a block!',
                    'index': block['index'], 'timestamp': block['timestamp'],
                    'nonce': block['nonce'],
                    'previous_hash': block['previous_hash']}
    else:
        response = {'message': 'No transactions to mine'}
    return jsonify(response), 200
```

```
# Getting the full Blockchain
@app.route('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': b.chain, 'length': len(b.chain)}
    return jsonify(response), 200

# Checking if the Blockchain is valid
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = b.is_chain_valid(b.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Houston, we have a problem.
                                 The Blockchain is not valid.'}
    return jsonify(response), 200

@app.route('/add_transaction', methods = ['POST'])
def add_transaction():
    # Method to create transaction in the b
    json = request.get_json()
    sender = json['sender']
    receiver = json['receiver']
    amount = json['amount']
    response = {'message': b.create_transaction(sender, receiver, amount)}
    return jsonify(response), 201

@app.route('/get_transactions', methods = ['GET'])
def get_transactions():
    response = {'transactions': b.transactions}
    return jsonify(response), 200
# Running the app
app.run(host = '0.0.0.0', port = 5000)
```

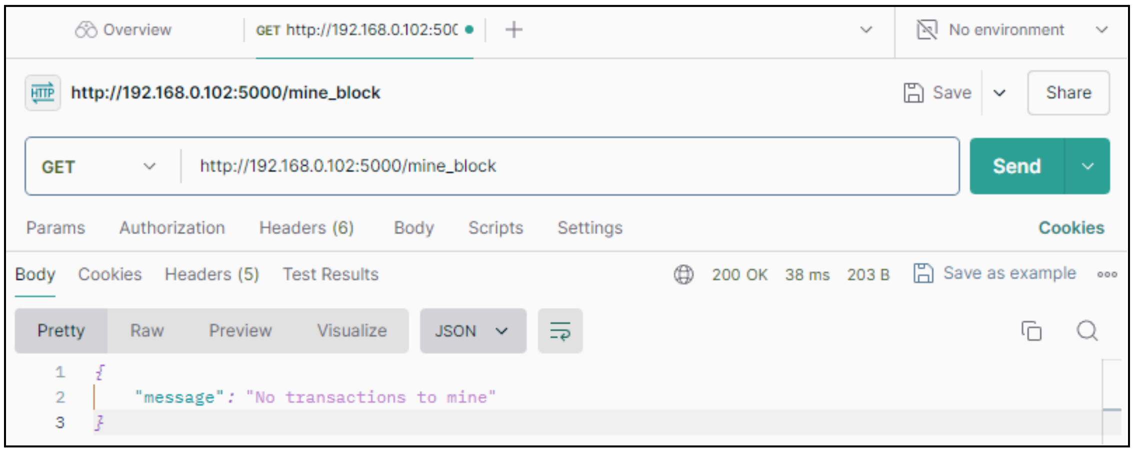
Outputs

Execute flask application



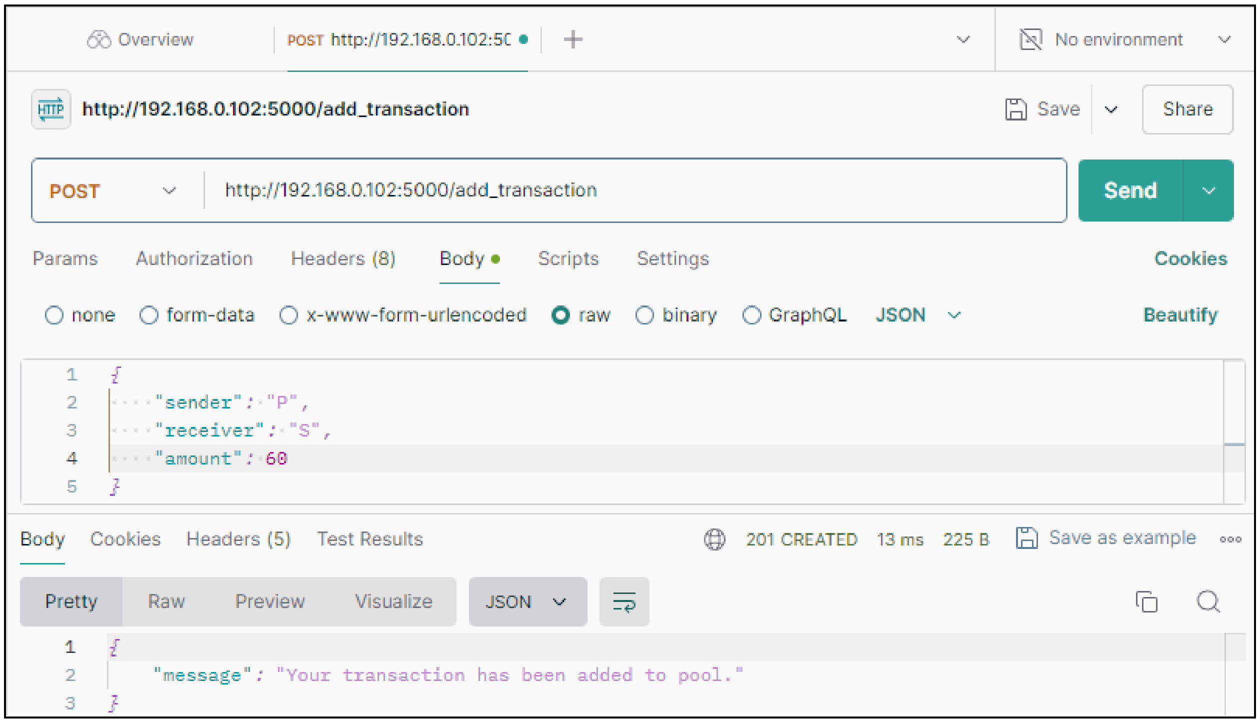
```
PS D:\B. E. CMPN\Seventh sem\BC> python blockchain.py
 * Serving Flask app 'blockchain'
 * Debug mode: off
WARNING: This is a development server. Do not use it in
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.102:5000
```

Make request to the API



```
{
    "message": "No transactions to mine"
}
```

Add few transactions to blockchain



http://192.168.0.102:5000/add_transaction

POST http://192.168.0.102:5000/add_transaction

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

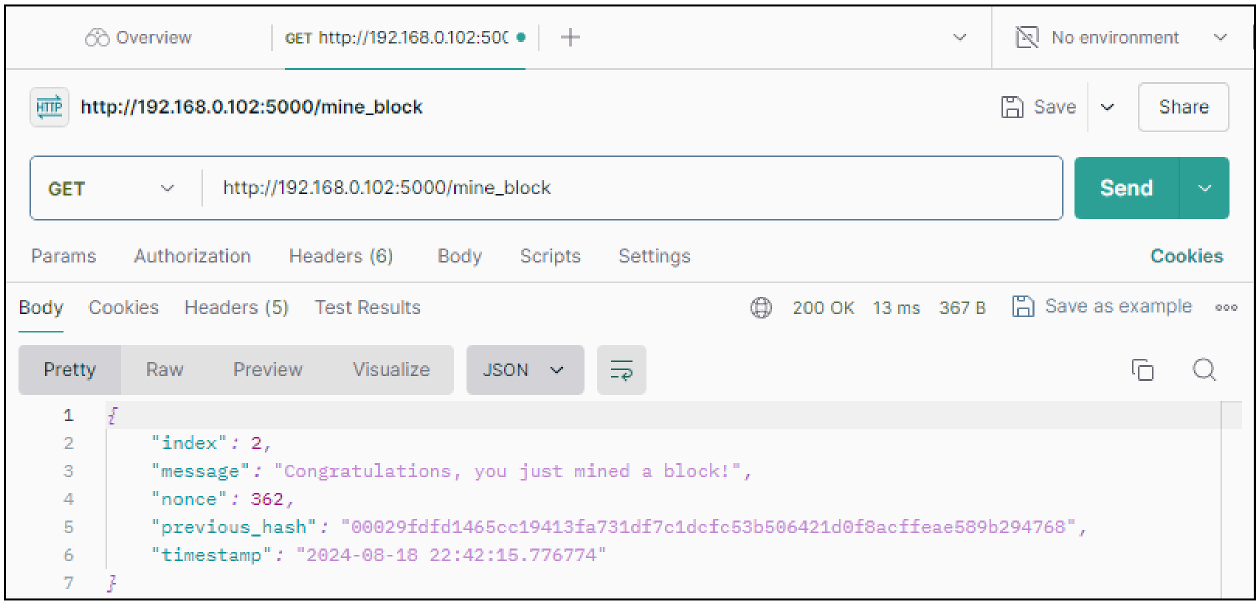
```
{
    "sender": "P",
    "receiver": "S",
    "amount": 60
}
```

201 CREATED 13 ms 225 B

JSON

```
{
    "message": "Your transaction has been added to pool."
}
```

Mine block



http://192.168.0.102:5000/mine_block

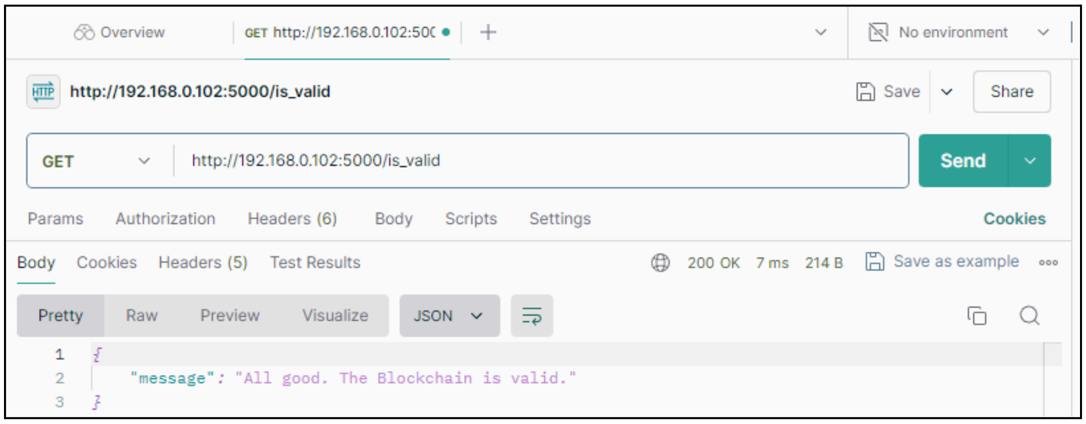
GET http://192.168.0.102:5000/mine_block

200 OK 13 ms 367 B

JSON

```
{
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "nonce": 362,
    "previous_hash": "00029fdfd1465cc19413fa731df7c1dcfc53b506421d0f8acffeae589b294768",
    "timestamp": "2024-08-18 22:42:15.776774"
}
```

Check if blockchain is valid



http://192.168.0.102:5000/is_valid

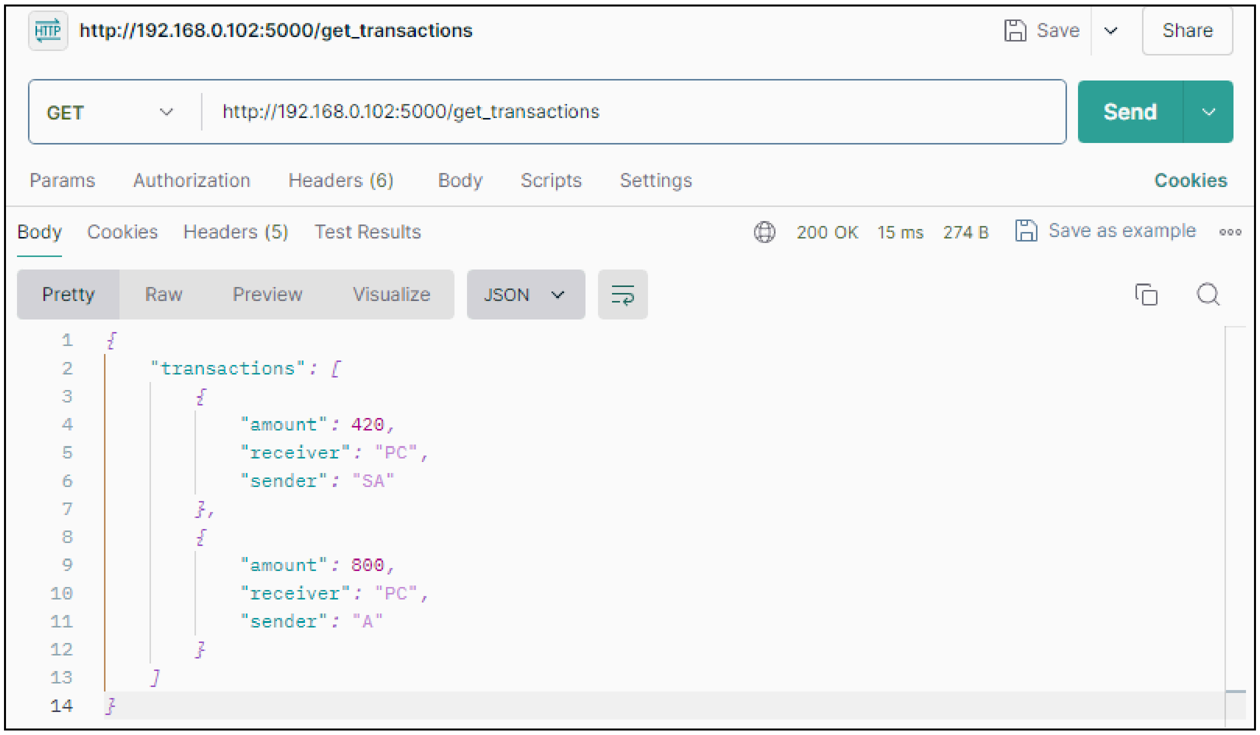
GET http://192.168.0.102:5000/is_valid

200 OK 7 ms 214 B

JSON

```
{
    "message": "All good. The Blockchain is valid."
}
```

Viewing transactions in mempool



http://192.168.0.102:5000/get_transactions

GET http://192.168.0.102:5000/get_transactions

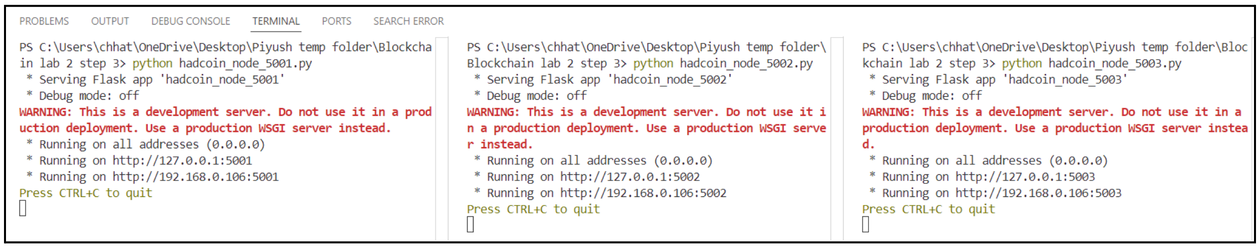
200 OK 15 ms 274 B

```
{
    "transactions": [
        {
            "amount": 420,
            "receiver": "PC",
            "sender": "SA"
        },
        {
            "amount": 800,
            "receiver": "PC",
            "sender": "A"
        }
    ]
}
```

Viewing the chain at http://192.168.0.102:5000/get_chain

```
{
    "chain": [
        {
            "index": 1, "merkle_root": null, "nonce": 1995,
            "previous_hash": "0", "timestamp": "2024-08-18 22:36:01.669454",
            "transactions": []
        },
        {
            "index": 2, "nonce": 362,
            "merkle_root":
            b335f26f17941cc0372d5795f3de3a8300083c98931ec2d0a10c9d4b294659b3,
            "previous_hash":
            00029fdfd1465cc19413fa731df7c1dcfc53b506421d0f8acffeae589b294768,
            "timestamp": "2024-08-18 22:42:15.776774",
            "transactions": [
                {
                    "amount": 60, "receiver": "S", "sender": "P"
                },
                {
                    "amount": 120, "receiver": "Shop", "sender": "S"
                },
                {
                    "amount": 100, "receiver": "A", "sender": "P"
                }
            ]
        },
        {
            "index": 3, "nonce": 3117,
            "merkle_root":
            bf2128b4769f631f5ec97c156b6a635c87f4e53f554a6490249bd3f4121be8ad,
            "previous_hash":
            000a1ea954e2997ddc9dc128b84dd2543f92e41abdf8dac865c17b327f26e516,
            "timestamp": "2024-08-18 22:51:15.778886",
            "transactions": [
                {
                    "amount": 420,
                    "receiver": "PC",
                    "sender": "SA"
                },
                {
                    "amount": 800,
                    "receiver": "PC",
                    "sender": "A"
                }
            ]
        }
    ],
    "length": 3
}
```

Running a blockchain with 3 peer nodes

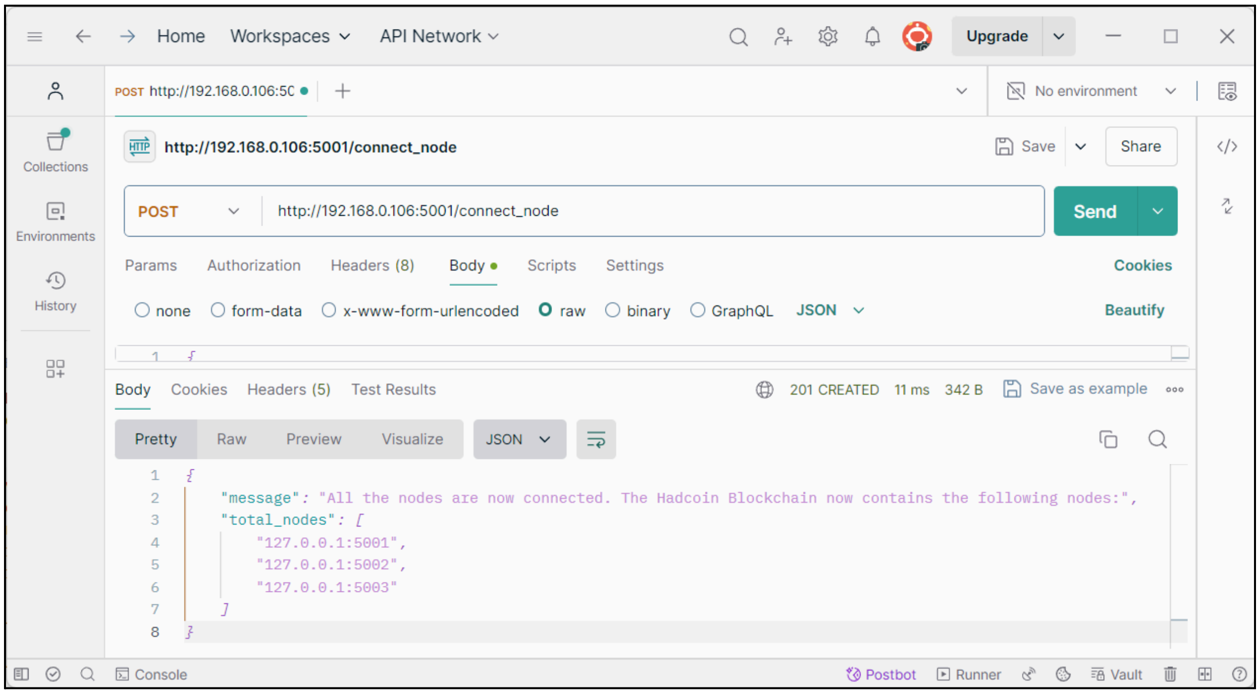


```
PS C:\Users\chhat\OneDrive\Desktop\Piyush temp folder\Blockcha
in lab 2 step 3> python hadcoin_node_5001.py
 * Serving Flask app 'hadcoin_node_5001'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a prod
uction deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.0.106:5001
Press CTRL+C to quit
```

```
PS C:\Users\chhat\OneDrive\Desktop\Piyush temp folder\
Blockchain lab 2 step 3> python hadcoin_node_5002.py
 * Serving Flask app 'hadcoin_node_5002'
 * Debug mode: off
WARNING: This is a development server. Do not use it i
n a production deployment. Use a production WSGI serve
r instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://192.168.0.106:5002
Press CTRL+C to quit
```

```
PS C:\Users\chhat\OneDrive\Desktop\Piyush temp folder\Bloc
kchain lab 2 step 3> python hadcoin_node_5003.py
 * Serving Flask app 'hadcoin_node_5003'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instea
d.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5003
 * Running on http://192.168.0.106:5003
Press CTRL+C to quit
```

Connect node 1 with peer nodes



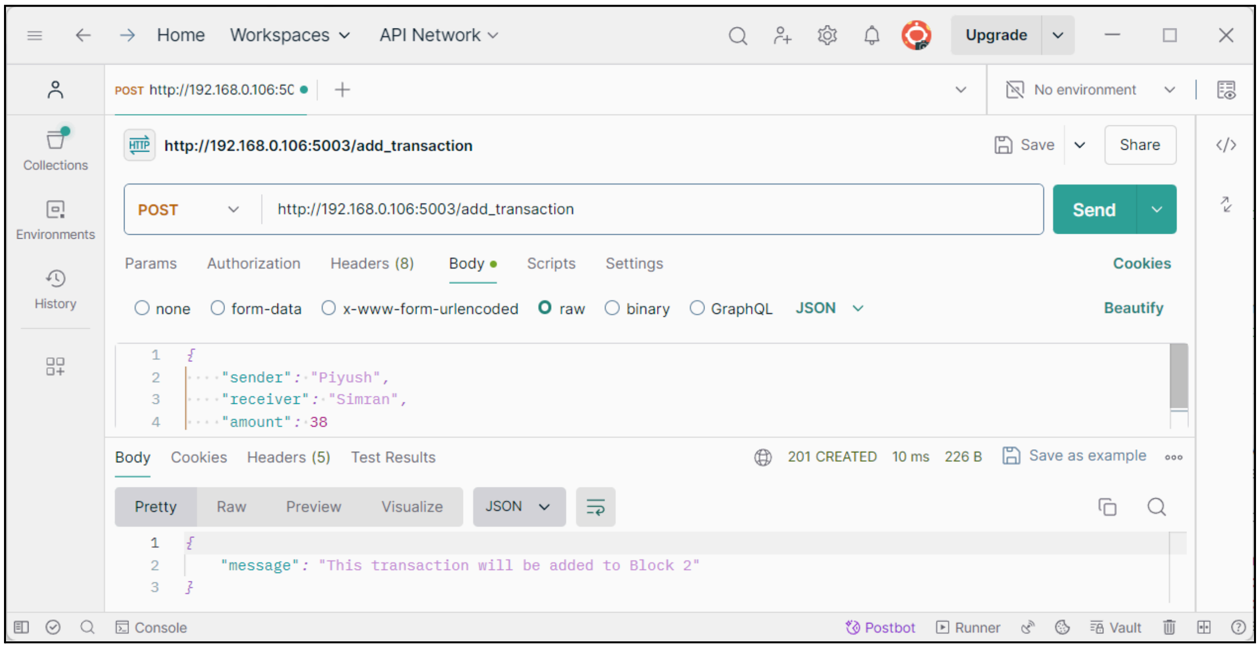
POST http://192.168.0.106:5001/connect_node

Body: [✓ raw] JSON

201 CREATED 11 ms 342 B

```
{
    "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5001",
        "127.0.0.1:5002",
        "127.0.0.1:5003"
    ]
}
```

Add different transactions to all blockchain nodes



http://192.168.0.106:5003/add_transaction

POST http://192.168.0.106:5003/add_transaction

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

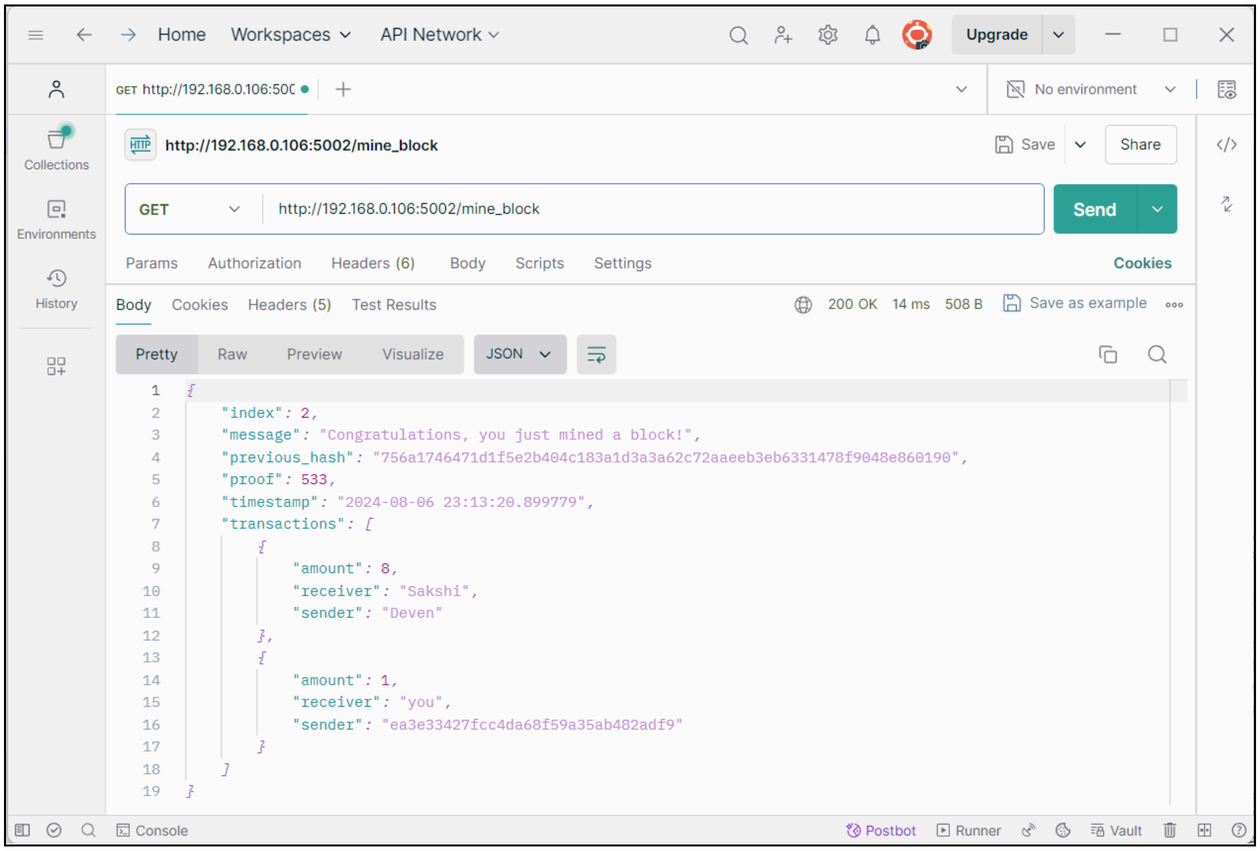
```
{
    "sender": "Piyush",
    "receiver": "Simran",
    "amount": 38
```

Body | Cookies | Headers (5) | Test Results — 201 CREATED 10 ms 226 B — Save as example

Pretty | Raw | Preview | Visualize | JSON

```
{
    "message": "This transaction will be added to Block 2"
}
```

Mine blocks on different nodes



http://192.168.0.106:5002/mine_block

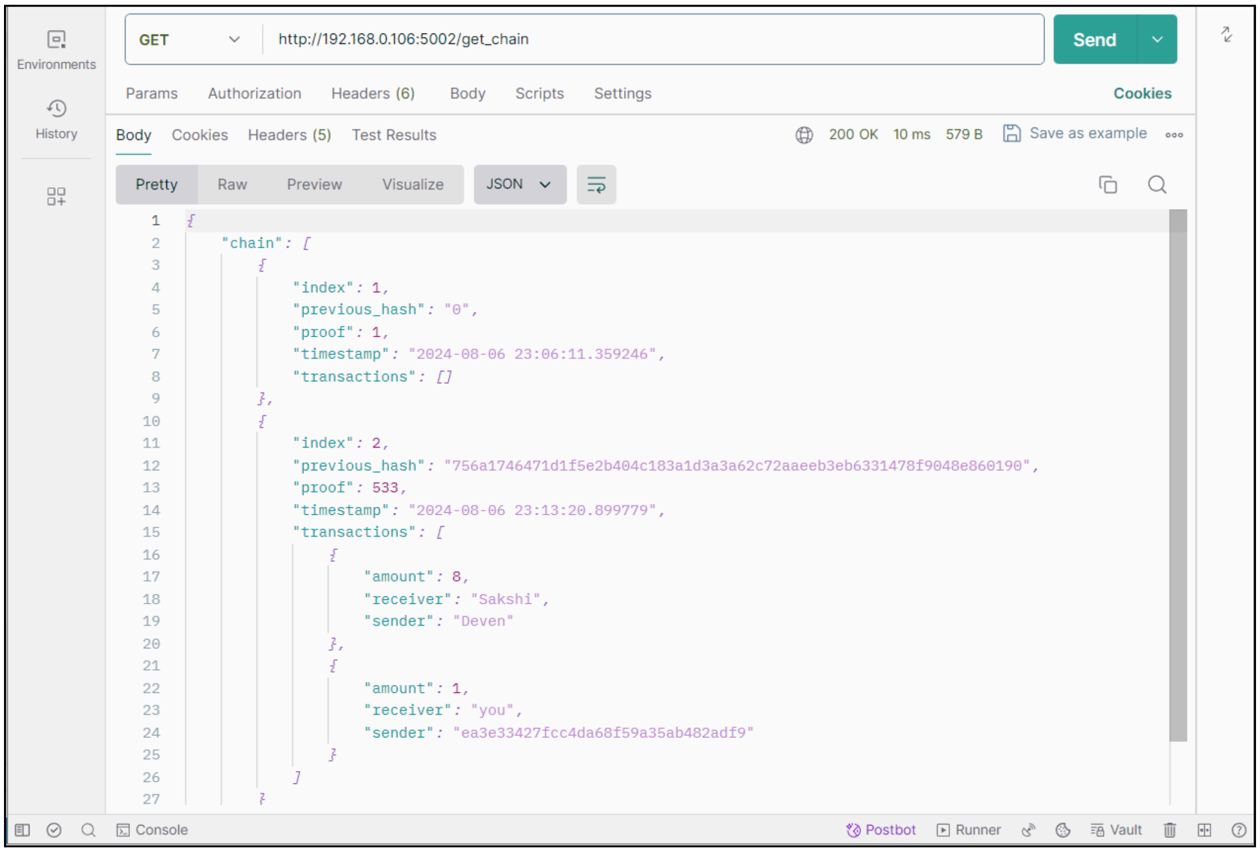
GET http://192.168.0.106:5002/mine_block

Body | Cookies | Headers (5) | Test Results — 200 OK 14 ms 508 B — Save as example

Pretty | Raw | Preview | Visualize | JSON

```
{
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "756a1746471d1f5e2b404c183a1d3a3a62c72aaeeb3eb6331478f9048e860190",
    "proof": 533,
    "timestamp": "2024-08-06 23:13:20.899779",
    "transactions": [
        {
            "amount": 8,
            "receiver": "Sakshi",
            "sender": "Deven"
        },
        {
            "amount": 1,
            "receiver": "you",
            "sender": "ea3e33427fcc4da68f59a35ab482adf9"
        }
    ]
}
```

Viewing the blockchain from node 2

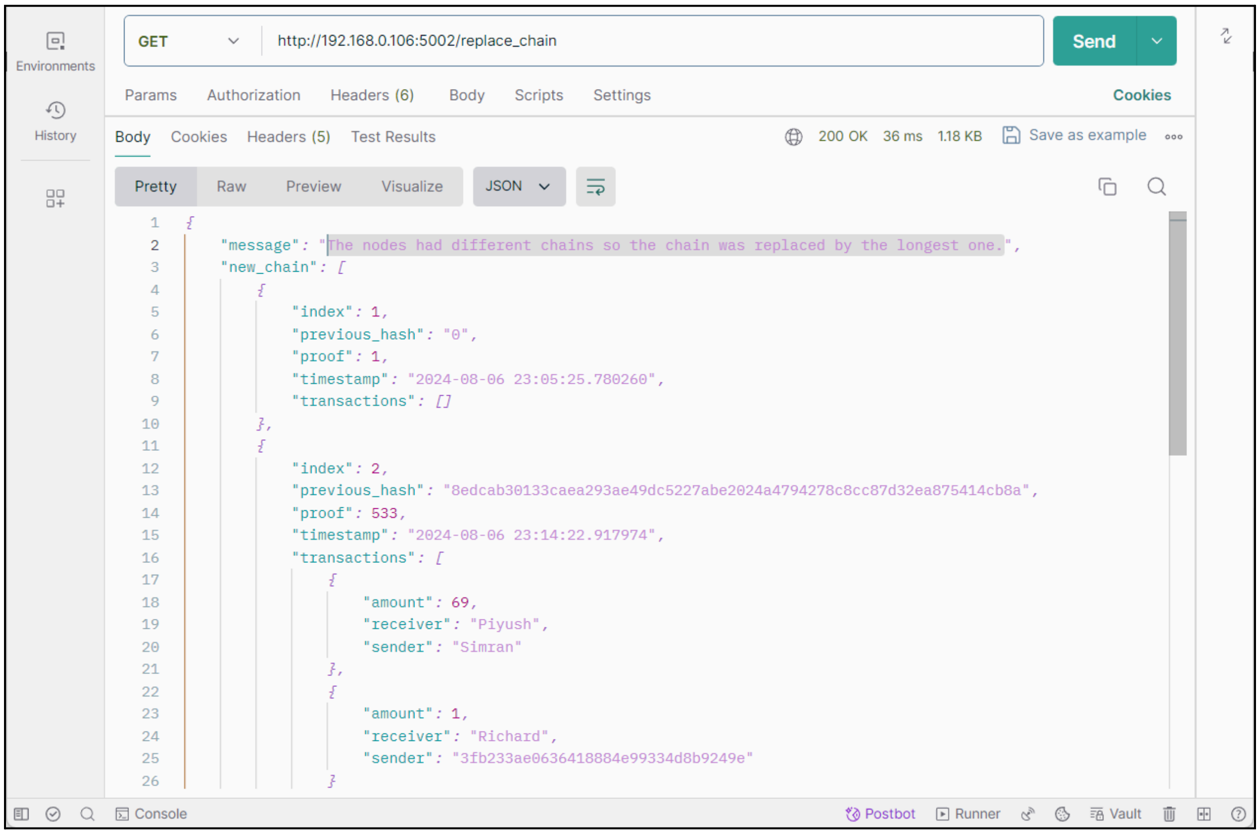


GET http://192.168.0.106:5002/get_chain

200 OK 10 ms 579 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2024-08-06 23:06:11.359246",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "756a1746471d1f5e2b404c183a1d3a3a62c72aaeeb3eb6331478f9048e860190",
            "proof": 533,
            "timestamp": "2024-08-06 23:13:20.899779",
            "transactions": [
                {
                    "amount": 8,
                    "receiver": "Sakshi",
                    "sender": "Deven"
                },
                {
                    "amount": 1,
                    "receiver": "you",
                    "sender": "ea3e33427fcc4da68f59a35ab482adf9"
                }
            ]
        }
```

Blocks had different chains, replace chain in block 2 with longest one



GET http://192.168.0.106:5002/replace_chain

200 OK 36 ms 1.18 KB

```
{
    "message": "The nodes had different chains so the chain was replaced by the longest one.",
    "new_chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2024-08-06 23:05:25.780260",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "8edcab30133caea293ae49dc5227abe2024a4794278c8cc87d32ea875414cb8a",
            "proof": 533,
            "timestamp": "2024-08-06 23:14:22.917974",
            "transactions": [
                {
                    "amount": 69,
                    "receiver": "Piyush",
                    "sender": "Simran"
                },
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "3fb233ae0636418884e99334d8b9249e"
                }
```

Conclusion

Understood the challenges in P2P networks, how transactions are performed and how a miner mines a block to be added in a blockchain. Implemented a Cryptocurrency in Python using Flask, Postman and Python libraries such as datetime, jsonify, hashlib, uuid4, urlparse, request. Successfully mined the blocks among a P2P network with 3 nodes. Performed transactions via the network. Successfully updated the block across the network.