Name: Pranav Pol   Class : D20A   Roll no: 44   Batch: C

EXPERIMENT 3

AIM: Create a Cryptocurrency using Python and perform mining in the Blockchain created.

THEORY:

1. Blockchain Overview

A **blockchain** is a **distributed, decentralized, and cryptographically secured digital ledger** used to record transactions in a chronological and immutable manner. Unlike traditional centralized databases that rely on a trusted third party, blockchain operates on a **peer-to-peer network**, where trust is established through cryptographic verification and consensus protocols.

Each block in a blockchain contains:

- A **set of verified transactions**
- A **timestamp** indicating the time of block creation
- The **hash of the previous block**, which links blocks together
- A **nonce**, used in the mining process
- The block's **own cryptographic hash**

The linking of blocks using cryptographic hashes creates a chain structure. Any attempt to modify data in a block changes its hash, which invalidates all subsequent blocks. This feature ensures **immutability**, **data integrity**, and **tamper resistance**. Because copies of the blockchain are stored across multiple nodes, transparency and fault tolerance are also achieved.

2. Mining

Mining is the process by which new blocks are created, validated, and added to the blockchain. It plays a crucial role in maintaining the security and reliability of the blockchain network. Mining performs two major functions:

1. Verification and validation of transactions
2. Securing the blockchain against malicious attacks

Mining is based on the **Proof-of-Work (PoW)** algorithm. In PoW, miners compete to solve a computationally complex mathematical puzzle. This involves repeatedly modifying a value called a **nonce** and recalculating the block's hash until it meets the network's difficulty criteria (such as a specific number of leading zeroes).

The difficulty level dynamically adjusts to control the rate at which blocks are mined. Once a miner successfully finds a valid hash, the block is broadcast to all other nodes for verification. Due to the high computational cost involved, PoW makes attacks like double spending and data manipulation extremely difficult.

3. Multi-Node Blockchain Network

A **multi-node blockchain network** consists of several independent nodes participating in transaction validation and block propagation. In this experiment, three nodes are deployed on different ports (5001, 5002, and 5003) to simulate a real-world decentralized environment.

Each node in the network:

- Operates autonomously
- Maintains a complete copy of the blockchain
- Communicates with peer nodes using a peer-to-peer protocol

This architecture eliminates the need for a central server and ensures **decentralization**, **data redundancy**, and **high availability**. Even if one node fails or behaves maliciously, the network continues to function correctly. This setup clearly demonstrates how blockchain systems maintain reliability and consistency across distributed systems.

4. Consensus Mechanism

A **consensus mechanism** is a protocol that enables all nodes in a blockchain network to agree on a single, consistent version of the ledger. Since nodes operate independently and blocks may be mined simultaneously, temporary inconsistencies or forks can occur.

In this system, the **Longest Chain Rule** is used as the consensus mechanism. According to this rule:

- The blockchain with the maximum number of blocks is considered valid
- The longest chain represents the greatest cumulative computational effort
- Nodes discard shorter chains and adopt the longest valid chain

This rule ensures eventual consistency and prevents permanent divergence in the blockchain. Consensus mechanisms are essential for maintaining trust, preventing double spending, and ensuring the integrity of distributed ledgers.

5. Transactions and Mining Reward

A **transaction** is a digitally signed record representing the transfer of value between participants in a blockchain network. Each transaction contains:

- Sender's address
- Receiver's address
- Transaction amount

Transactions are verified using cryptographic techniques and temporarily stored in a transaction pool until they are included in a block.

To incentivize miners, the system introduces a **mining reward mechanism**. When a block is successfully mined, a special transaction called a **coinbase transaction** is created. This transaction generates new cryptocurrency and awards it to the miner. The reward mechanism encourages honest participation and provides economic security to the network.

6. Chain Replacement

Chain replacement is a synchronization mechanism that ensures all nodes in the blockchain network maintain a consistent and up-to-date ledger. Due to network latency or simultaneous mining, nodes may temporarily have different versions of the blockchain.

When the `/replace_chain` endpoint is invoked:

1. A node requests blockchain data from all peer nodes
2. The received chains are validated for correctness and integrity
3. The node replaces its own blockchain if a longer and valid chain is found

This mechanism resolves conflicts, eliminates forks, and ensures that all nodes converge to a single authoritative blockchain. Chain replacement is essential for maintaining decentralization without central coordination.

CODE

```
import datetime

import hashlib

import json

from flask import Flask,jsonify,request

import requests

from uuid import uuid4

from urllib.parse import urlparse

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_has
h,'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('-','')

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('/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)
```

OUTPUT:

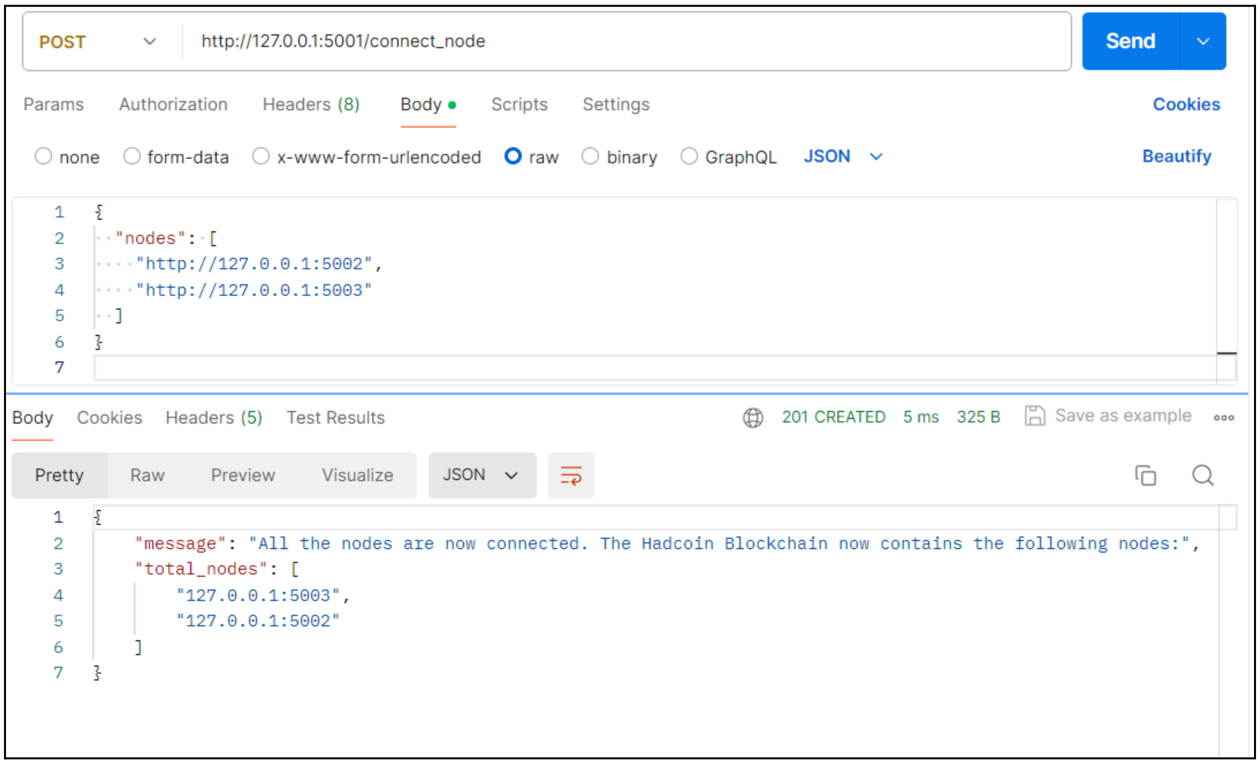
1. Connect Nodes (POST)

URL (from any node):

```
POST http://127.0.0.1:5001/connect_node
```

Body → raw → JSON

```
{
  "nodes": [
    "http://127.0.0.1:5002",
    "http://127.0.0.1:5003"
  ]
}
```



POST http://127.0.0.1:5001/connect_node

Body → raw → JSON

```
{
  "nodes": [
    "http://127.0.0.1:5002",
    "http://127.0.0.1:5003"
  ]
}
```

201 CREATED 5 ms 325 B

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

2. Add Transaction (POST)

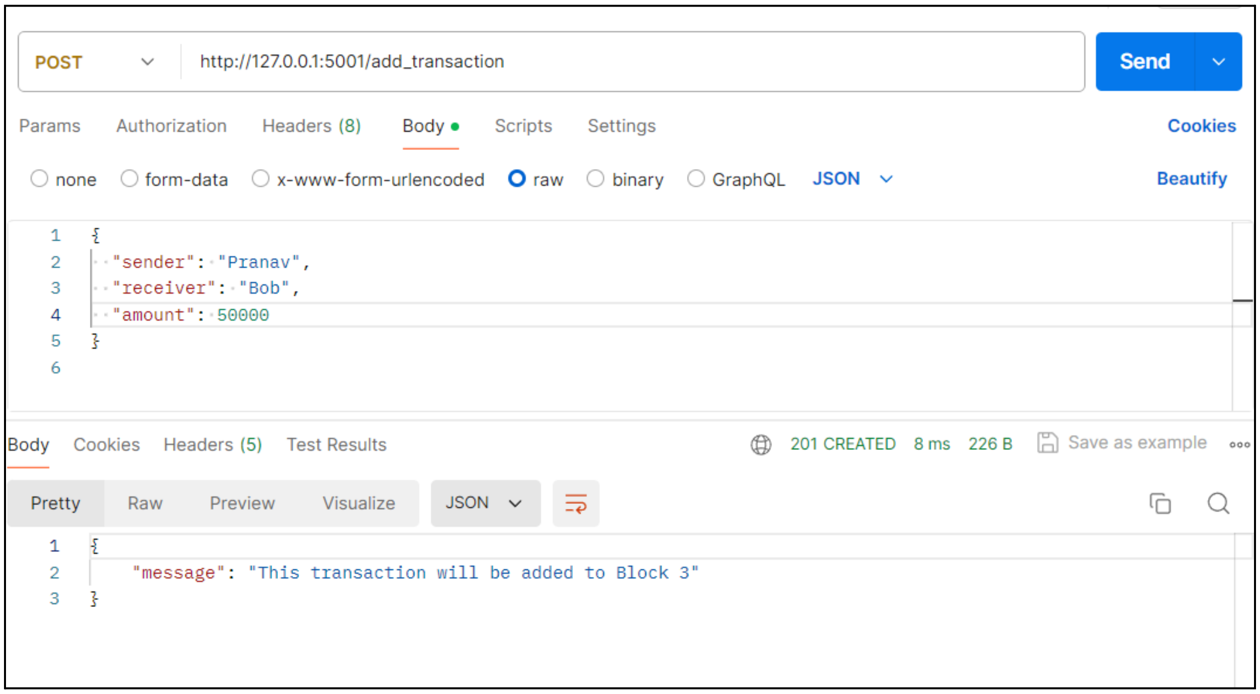
URL

```
POST http://127.0.0.1:5001/add_transaction
```

Body

```
{
  "sender": "Pranav",
  "receiver": "Bob",
  "amount": 50
}
```

Transaction goes into **mempool**, NOT block yet.



POST http://127.0.0.1:5001/add_transaction

Body: raw, JSON

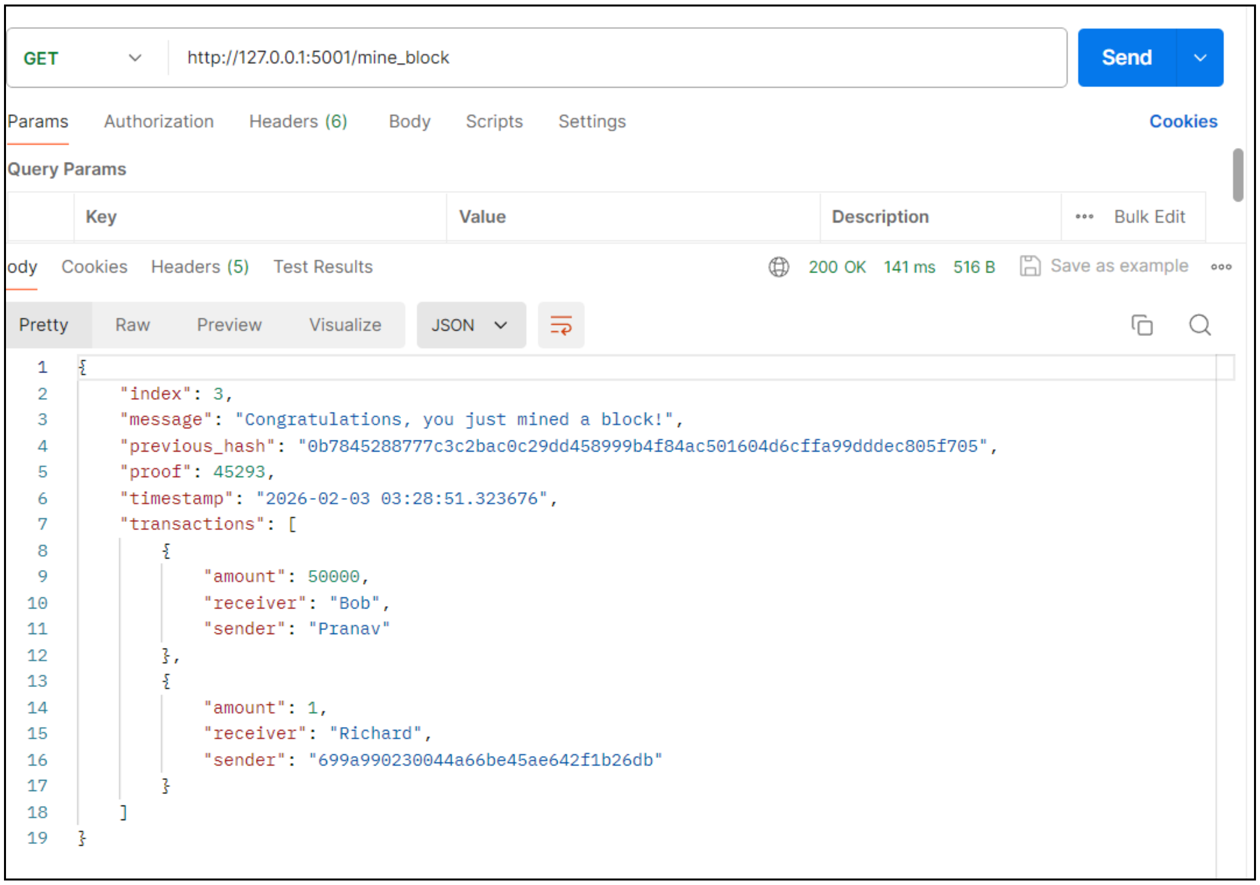
```
{
  "sender": "Pranav",
  "receiver": "Bob",
  "amount": 50000
}
```

201 CREATED 8 ms 226 B

```
{
    "message": "This transaction will be added to Block 3"
}
```

3. Mine Block (GET)

```
GET http://127.0.0.1:5001/mine_block
```



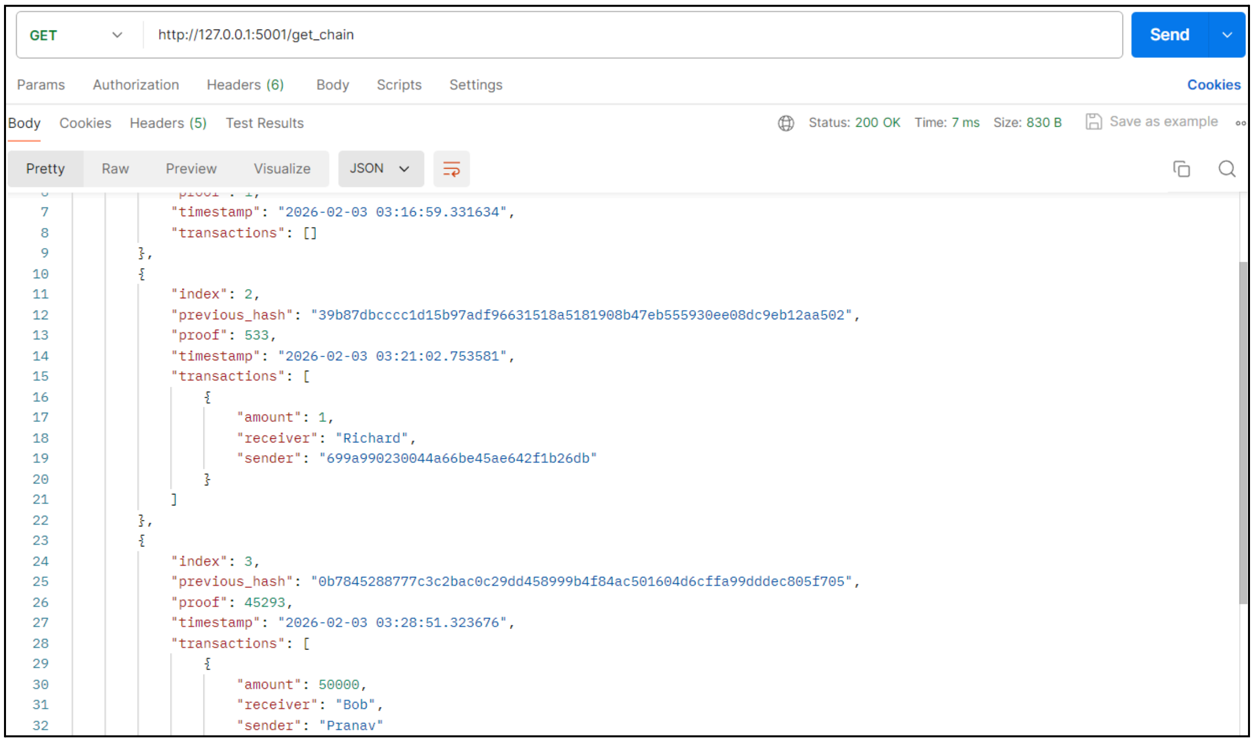
```
{
    "index": 3,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "0b7845288777c3c2bac0c29dd458999b4f84ac501604d6cffa99dddec805f705",
    "proof": 45293,
    "timestamp": "2026-02-03 03:28:51.323676",
    "transactions": [
        {
            "amount": 50000,
            "receiver": "Bob",
            "sender": "Pranav"
        },
        {
            "amount": 1,
            "receiver": "Richard",
            "sender": "699a990230044a66be45ae642f1b26db"
        }
    ]
}
```

4. Get Blockchain (GET)

`GET http://127.0.0.1:5001/get_chain`

You'll see:

- Transactions inside blocks
- `transactions: []` for new pending list


GET http://127.0.0.1:5001/get_chain

Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Body Cookies Headers (5) Test Results Status: 200 OK Time: 7 ms Size: 830 B Save as example

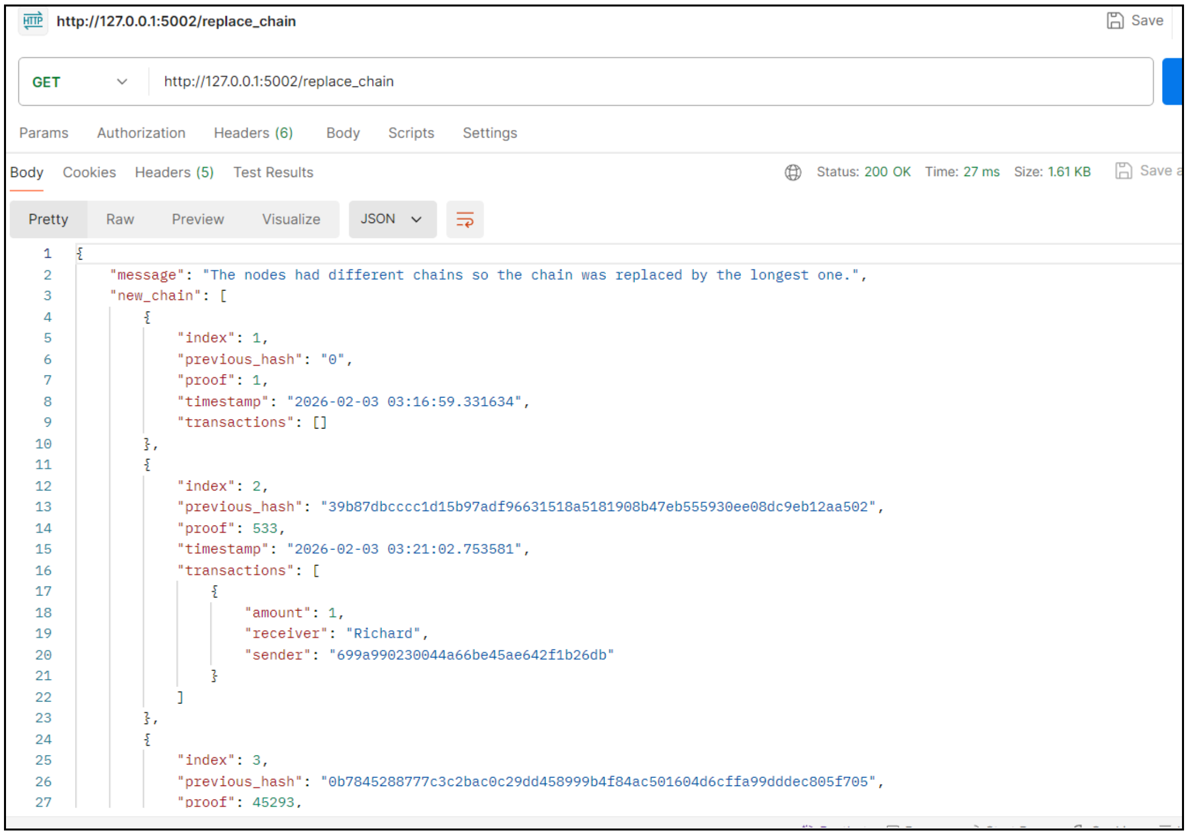
Pretty Raw Preview Visualize JSON

```
            "timestamp": "2026-02-03 03:16:59.331634",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "39b87dbcccc1d15b97adf96631518a5181908b47eb555930ee08dc9eb12aa502",
            "proof": 533,
            "timestamp": "2026-02-03 03:21:02.753581",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "699a990230044a66be45ae642f1b26db"
                }
            ]
        },
        {
            "index": 3,
            "previous_hash": "0b7845288777c3c2bac0c29dd458999b4f84ac501604d6cffa99dddec805f705",
            "proof": 45293,
            "timestamp": "2026-02-03 03:28:51.323676",
            "transactions": [
                {
                    "amount": 50000,
                    "receiver": "Bob",
                    "sender": "Pranav"
```

5. Replace Chain (Consensus)

`GET http://127.0.0.1:5002/replace_chain`

Shorter chains get replaced by the longest valid one.


http://127.0.0.1:5002/replace_chain
Save
GET
http://127.0.0.1:5002/replace_chain
Params Authorization Headers (6) Body Scripts Settings
Body Cookies Headers (5) Test Results
Status: 200 OK Time: 27 ms Size: 1.61 KB
Pretty Raw Preview Visualize JSON

```
{
    "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": "2026-02-03 03:16:59.331634",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "39b87dbcccc1d15b97adf96631518a5181908b47eb555930ee08dc9eb12aa502",
            "proof": 533,
            "timestamp": "2026-02-03 03:21:02.753581",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "699a990230044a66be45ae642f1b26db"
                }
            ]
        },
        {
            "index": 3,
            "previous_hash": "0b7845288777c3c2bac0c29dd458999b4f84ac501604d6cffa99dddec805f705",
            "proof": 45293,
```

CONCLUSION:

In this experiment, a simple cryptocurrency was successfully created using Python by implementing core blockchain concepts such as block creation, hashing, Proof-of-Work mining, transactions, and decentralization. The blockchain system was developed using Flask, allowing multiple nodes to communicate and maintain their own copies of the blockchain.

Mining was performed by solving the Proof-of-Work algorithm, through which new blocks were added to the blockchain and miners were rewarded with cryptocurrency. The experiment also demonstrated transaction handling, where transactions were verified and included in mined blocks. Overall, this experiment provided a practical understanding of how blockchain technology works, including mining, decentralization, and consensus, and highlighted how cryptocurrencies operate in real-world distributed systems.