Sonam chhabaidiya
D20A 16

EXPERIMENT NO.3

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

1. Blockchain Overview

Blockchain is a **distributed and decentralized digital ledger** that records transactions across a network of computers (nodes) in a secure, transparent, and tamper-resistant manner. It eliminates the need for a central authority by using cryptography and consensus.

Data is stored in a series of linked **blocks** forming a chronological chain. Each block typically contains:

- A list of **transaction data** (e.g., sender, receiver, amount)
- A **timestamp** indicating when the block was created
- The **hash of the previous block** (creating the link)
- Its own **unique hash** (a cryptographic digital fingerprint generated from the block's contents)

Once a block is added to the chain and confirmed by the network, its data becomes **immutable** — changing any information in a block would alter its hash, breaking the link to the next block and requiring recalculation of all subsequent blocks (which is computationally infeasible in a large network). This structure ensures security, transparency, and trust without intermediaries.

2. Mining

Mining is the process of validating transactions, creating new blocks, and securing the blockchain through computational work, primarily using the **Proof-of-Work (PoW)** consensus mechanism (as in Bitcoin).

Steps in mining:

1. Miners collect **pending transactions** from the network (mempool) and group them into a candidate block.
2. They perform a **computational puzzle** — repeatedly hashing the block header (including a variable nonce) until the resulting hash meets the network's difficulty target (e.g., starts with a required number of leading zeros).
3. The first miner to find a valid hash **adds the new block** to their copy of the blockchain and **broadcasts** it to all peers for verification.
4. If accepted, the block becomes part of the official chain.

Successful miners receive a **reward** (newly minted cryptocurrency + transaction fees) as an incentive for their work. Mining prevents double-spending, maintains decentralization, and makes tampering extremely expensive.

3. Multi-Node Blockchain Network

In a real blockchain (and in this lab simulation), the network operates as a **peer-to-peer (P2P)** system with multiple independent **nodes** (computers) instead of a central server.

In the lab setup:

- Three nodes run on separate ports (e.g., 5001, 5002, 5003).
- Each node maintains its **own full copy** of the blockchain ledger.

- Nodes communicate directly with each other (P2P) to share new blocks, transactions, and chain information.
- This decentralization ensures no single point of failure — if one node goes offline, others continue operating and can sync later.

Nodes validate incoming data and propagate valid information across the network, enabling global consensus without trusting any central entity.

4. Consensus Mechanism

Consensus is the process by which all nodes in a decentralized network agree on the valid state of the blockchain (i.e., which chain and transactions are legitimate).

In this lab (and Bitcoin), the mechanism used is the **Longest Chain Rule** (also called Nakamoto Consensus in PoW systems):

- When multiple valid chains exist (e.g., due to near-simultaneous block mining), nodes always adopt the **longest valid chain** (the one with the most accumulated proof-of-work / blocks).
- This rule resolves conflicts automatically — shorter/forked chains are eventually discarded as miners continue building on the longest one.
- It ensures eventual agreement on a single transaction history across the entire network.

This simple yet powerful rule achieves reliable consensus in a trustless environment.

5. Transactions & Mining Reward

A **transaction** in blockchain is a record of value transfer between parties, typically including:

- **Sender** address
- **Receiver** address
- **Amount** transferred
- Digital signature (to prove ownership)

Transactions are collected in the mempool, verified, and included in blocks during mining.

When a miner successfully creates a new block:

- All selected pending transactions are added to the block body.
- An automatic **reward transaction** (coinbase transaction) is included as the first transaction in the block.
- This coinbase transaction pays the miner a fixed **block reward** (newly created coins) plus all **transaction fees** paid by users.

The reward incentivizes miners to secure the network and introduces new coins into circulation in a controlled manner.

6. Chain Replacement

In a decentralized network, nodes may temporarily have different chain versions due to network delays or simultaneous mining.

The **/replace_chain** endpoint (or equivalent resolve function) implements conflict resolution:

1. The node requests the current blockchain from all connected peers.
2. It compares the lengths (and validity) of received chains against its own.

3. If a **longer and valid chain** is found (following the longest chain rule), the node replaces its local chain with the longer one.
4. The node discards its shorter/forked chain and adopts the majority-accepted version.

This process keeps the blockchain synchronized and consistent across all nodes, ensuring everyone eventually agrees on the same transaction history.

CODE:-

```
import datetime,hashlib,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,'pr
evious_hash':previous_hash,'transactions':self.transactions}
```

Sonam chhabaidiya
D20A 16

```
        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()).hexdige
st()

            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)
```

Sonam chhabaidiya
D20A 16

```
    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():
```

Sonam chhabaidiya
D20A 16

```
    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'],'previou
s_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['am
ount'])

    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)

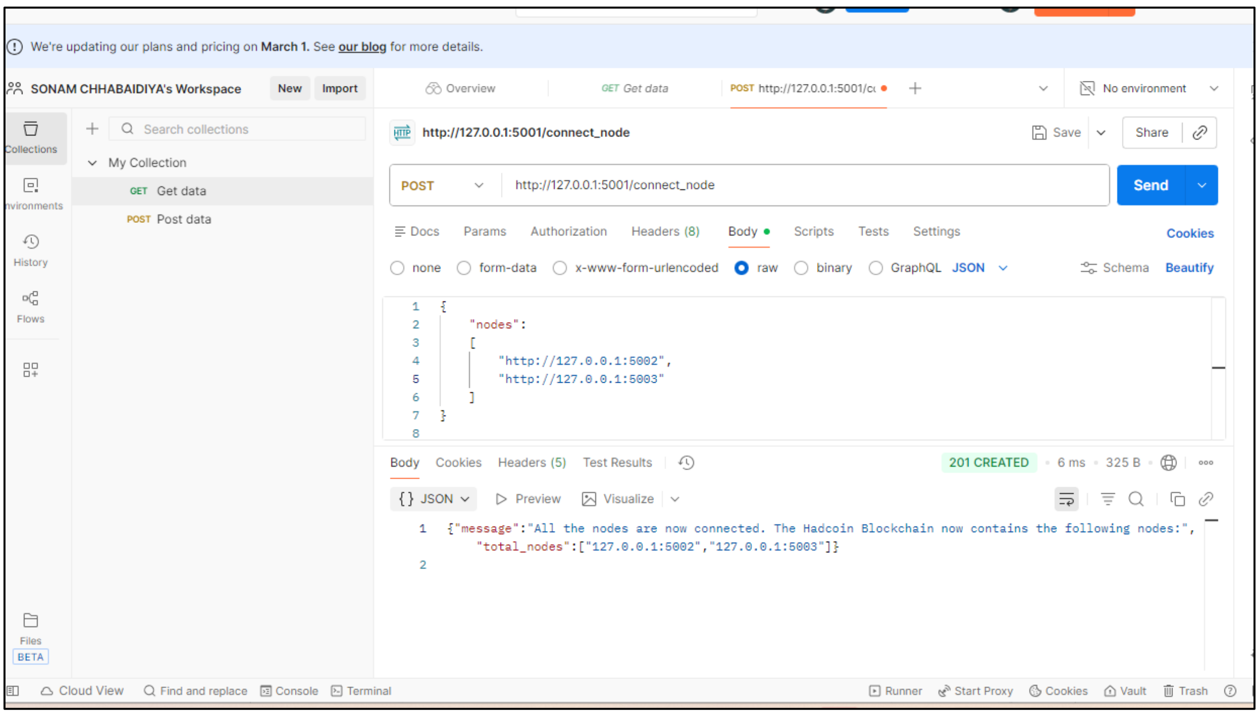
URL (from any node):

POST http://127.0.0.1:5001/connect_node

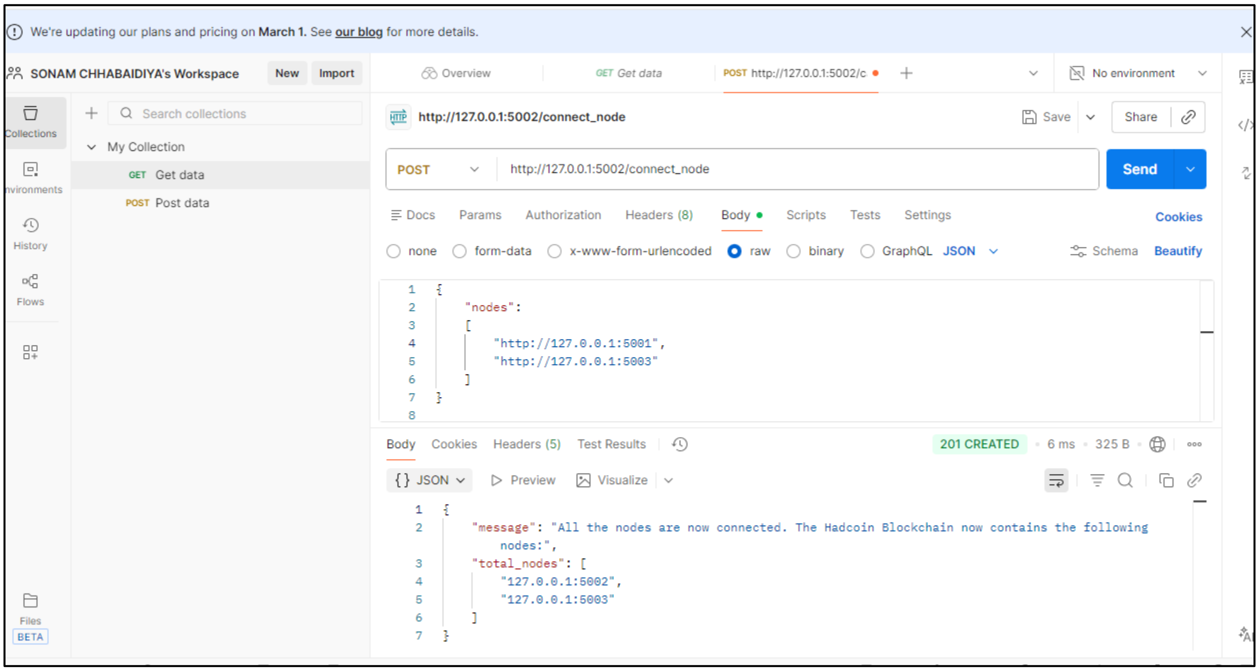
Body → raw → JSON

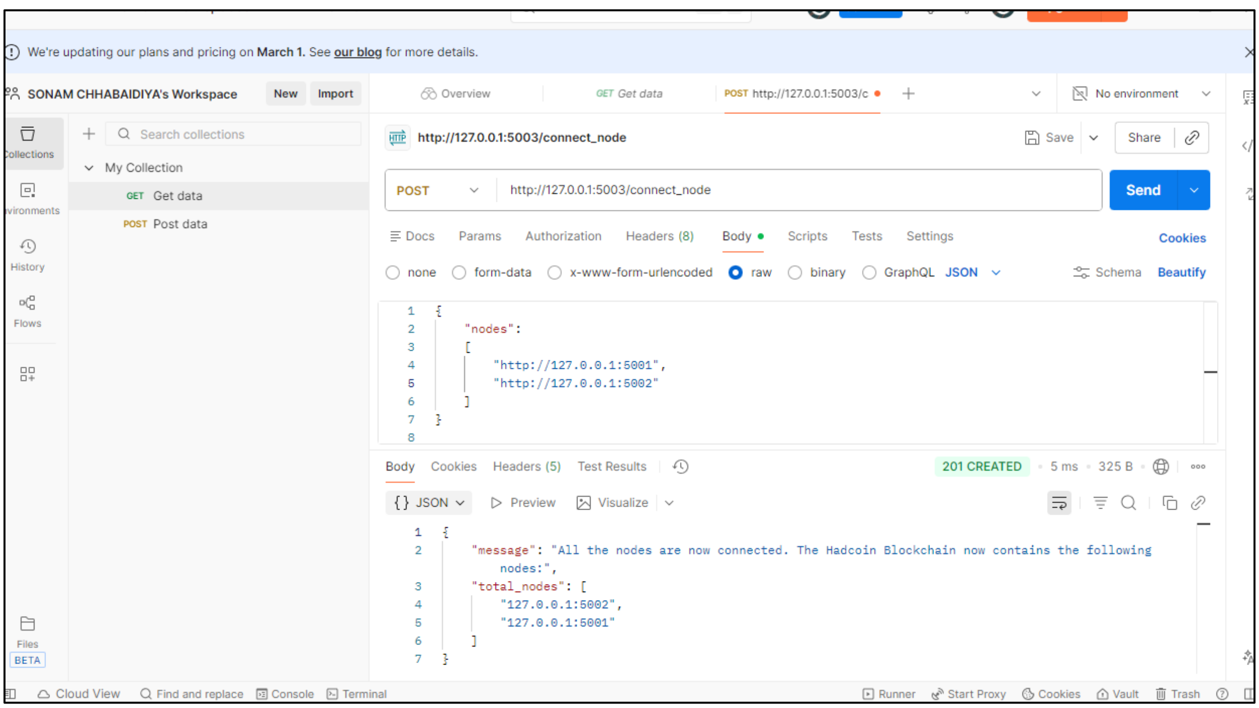
Sonam chhabaidiya
D20A 16

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



Sonam chhabaidiya
D20A 16





Sonam chhabaidiya
D20A 16

2. Add Transaction (POST)

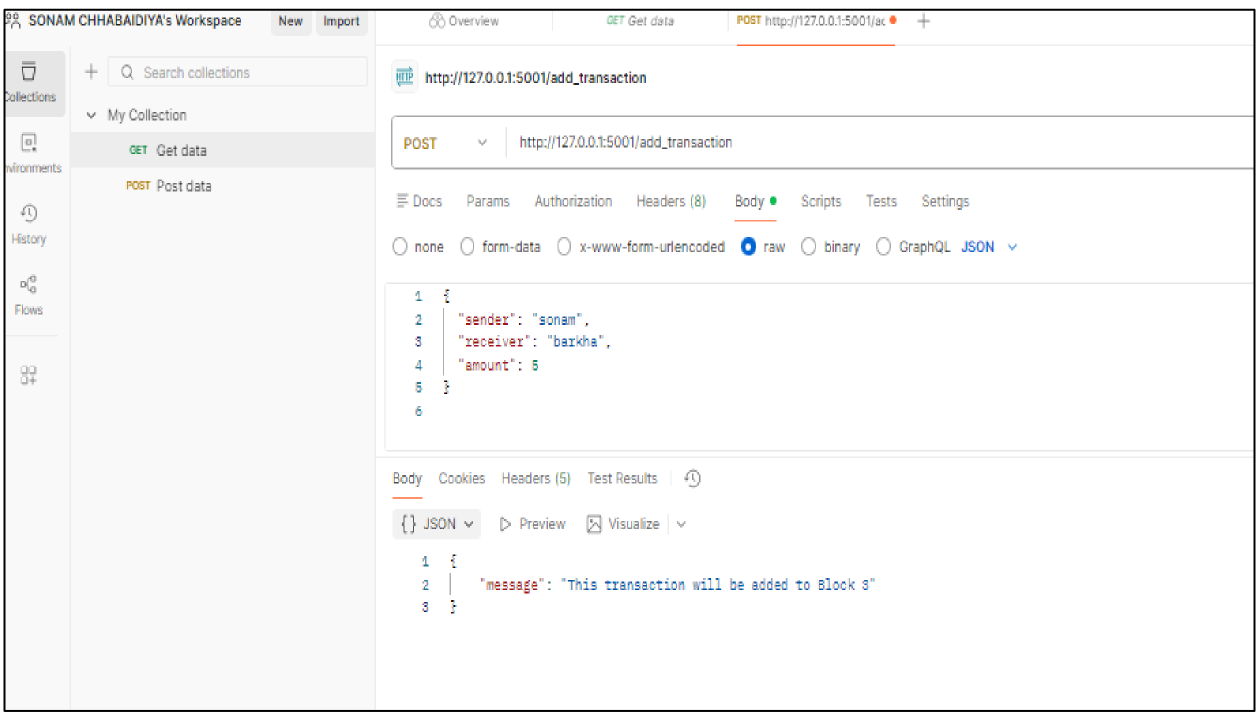
URL

POST http://127.0.0.1:5001/add_transaction

Body

```
{
 "sender": "sonam",
 "receiver": "barkha",
 "amount": 5
}
```

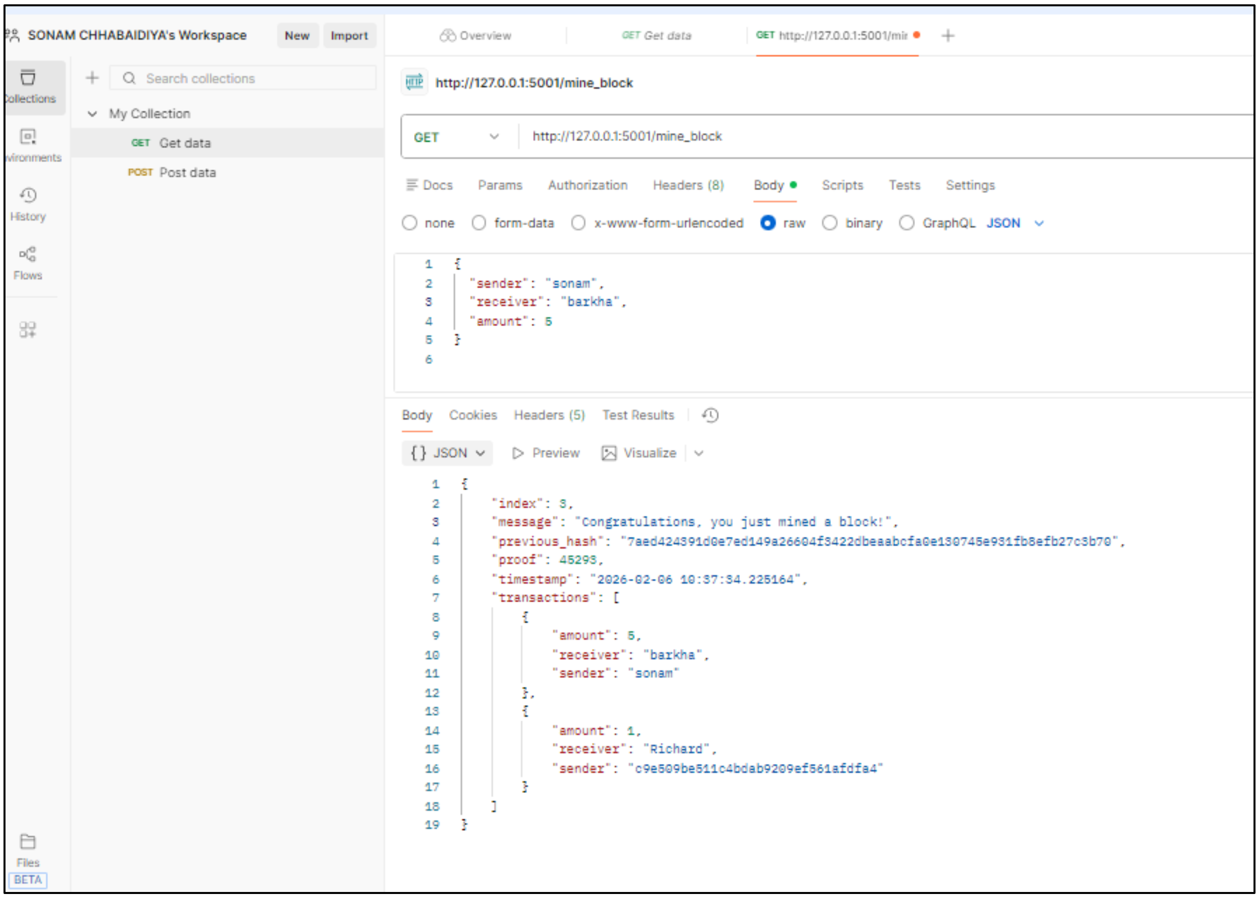
Transaction goes into **mempool**, NOT block yet



Sonam chhabaidiya
D20A 16

3. Mine Block (GET)

GET http://127.0.0.1:5001/mine_block



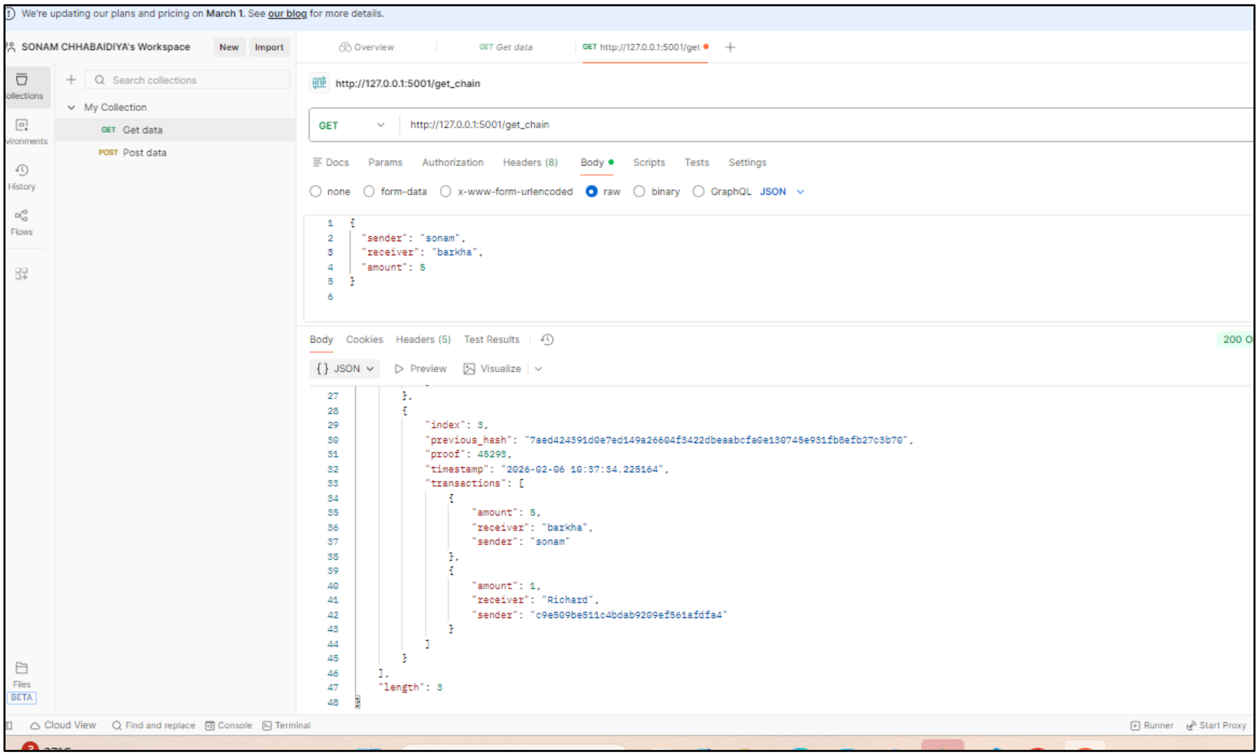
4. Get Blockchain (GET)

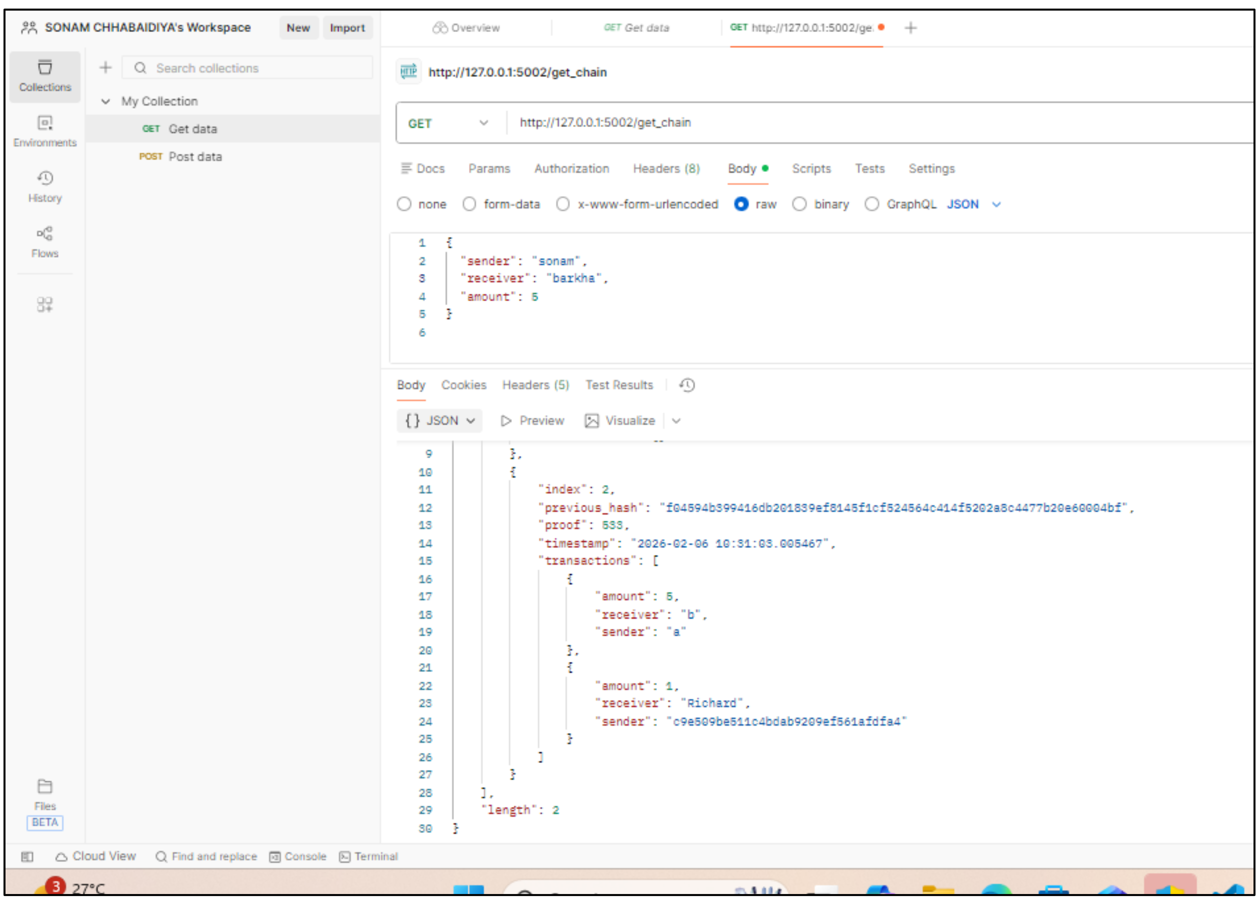
GET http://127.0.0.1:5001/get_chain

You'll see:

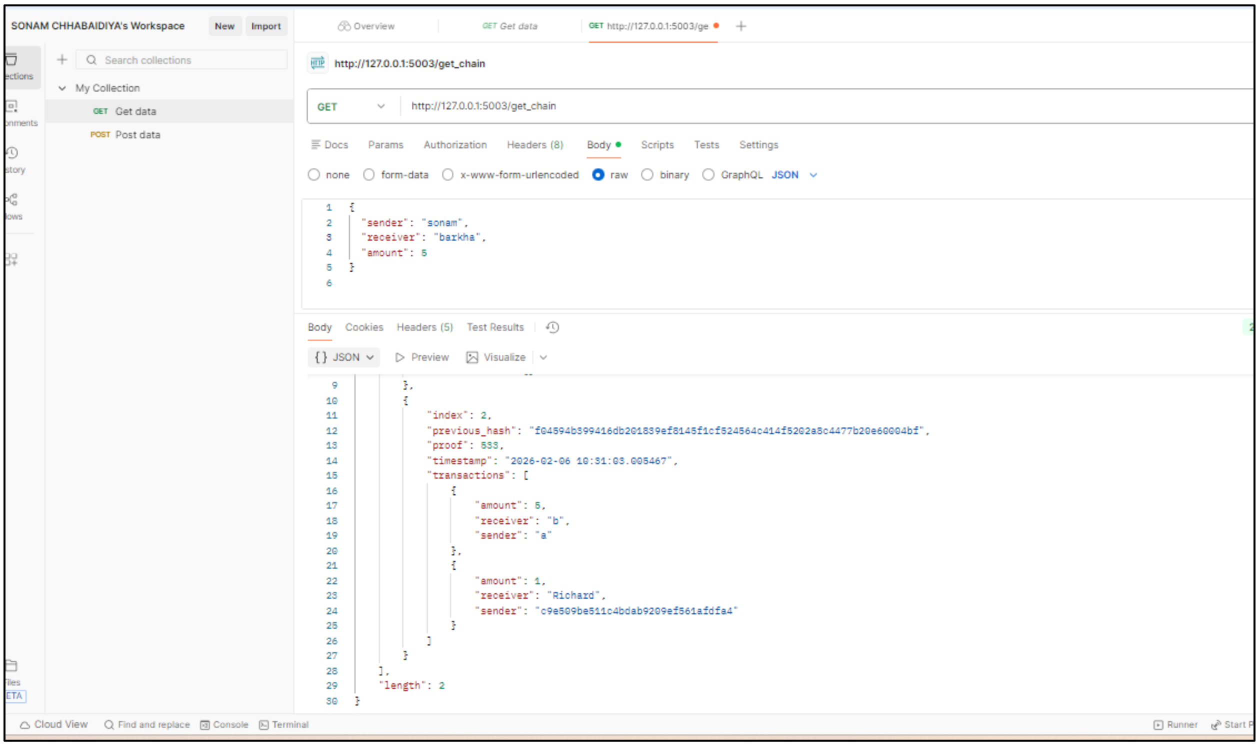
- Transactions inside blocks
- transactions: [] for new pending list

Sonam chhabaidiya
D20A 16





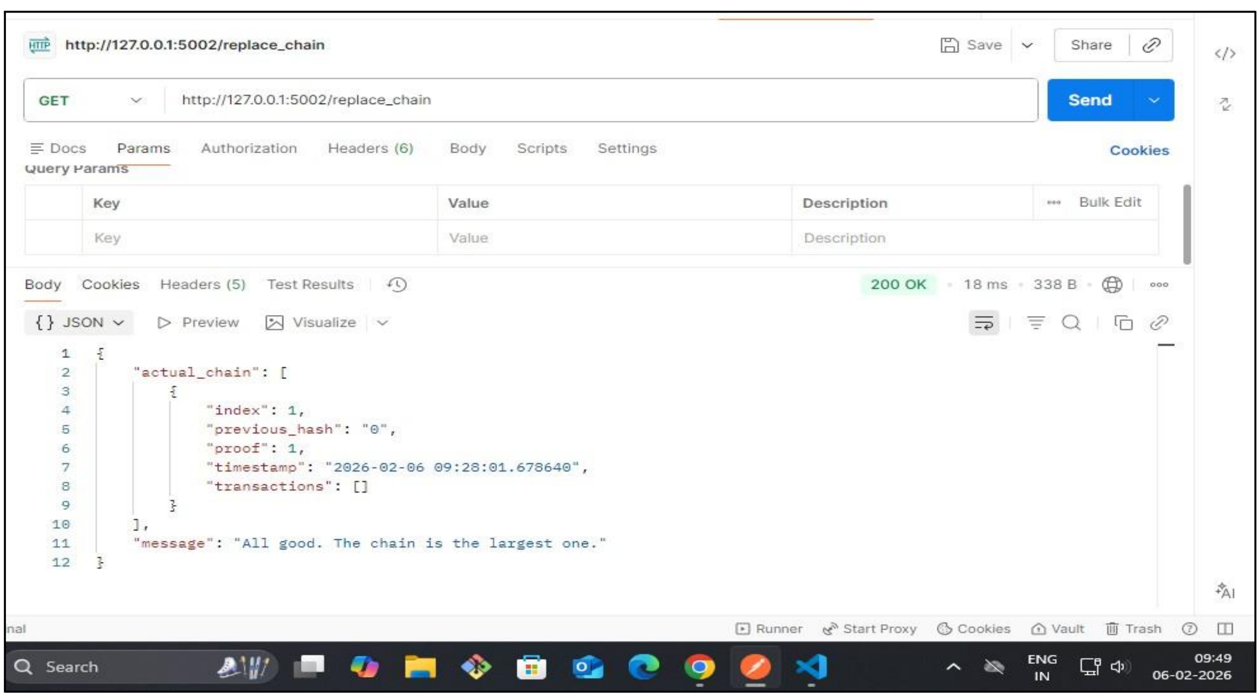
Sonam chhabaidiya
D20A 16



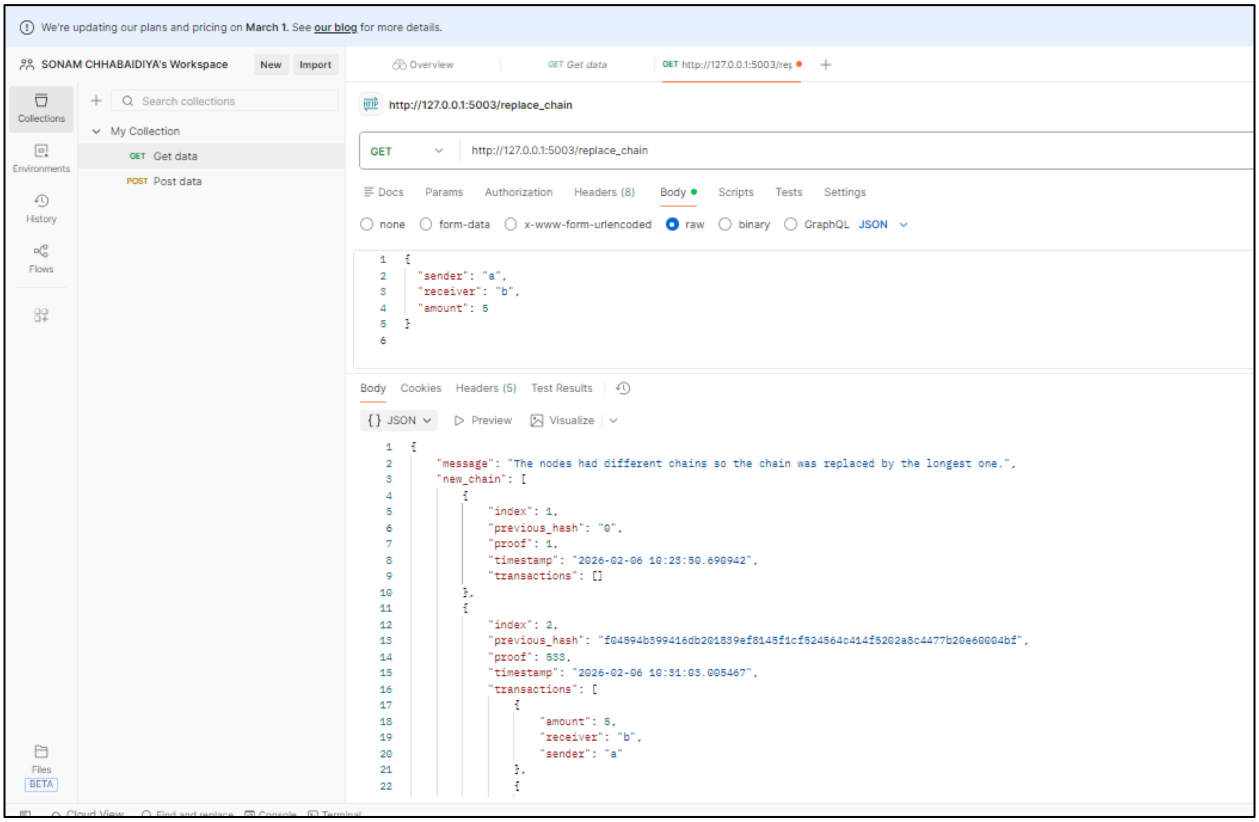
5.Replace Chain (Consensus)

GET http://127.0.0.1:5002/replace_chain

Shorter chains get replaced by the longest valid one



Sonam chhabaidiya
D20A 16



Sonam chhabaidiya
D20A 16

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.

Sonam chhabaidiya
D20A 16