Ria Chaudhari D20A14

Blockchain Lab Experiment 3

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

THEORY:

Q1.Challenges in P2P Networks

- No central authority to control the network.
- All nodes must agree on one blockchain (**consensus problem**).
- Some nodes may send fake or malicious data.
- Network delays can cause different chain versions.
- Difficult to maintain synchronization as network grows.
- Nodes may go offline → system must still work.

Q2.How Transactions are Performed on the Network

- The user sends a transaction request.
- Transaction is broadcast to all nodes.
- It is stored temporarily in the **mempool**.
- Miner selects transactions from mempool.
- Miner solves the Proof of Work.
- New block is created.
- Block is added to the blockchain.
- Updated chain is shared with other nodes

Q3.Role of Mempool

A mempool (memory pool) is a temporary storage area for unconfirmed transactions.
Functions of Mempools:

● Stores pending transactions before they are added to a block.

● Helps miners select transactions for block creation.

● Prevents network congestion by organizing transaction queues.

Q4.Libraries & Tools Used

Tools

- **Postman** → send GET & POST requests.
- **VS Code / Terminal** → run servers.

Python Libraries

- **Flask** → create APIs/endpoints.
- **datetime** → timestamp for each block.
- **hashlib** → SHA-256 hashing.
- **jsonify** → send response in JSON format.
- **request** → get data from POST request.
- **uuid4** → generate unique miner ID.
- **urlparse** → extract node address.
- **requests** → communicate with other nodes.

CODE:

```
# Module 2 - Create a Cryptocurrency

# To be installed:

# Flask==0.12.2: pip install Flask==0.12.2

# Postman HTTP Client: https://www.getpostman.com/

# requests==2.18.4: pip install requests==2.18.4

# Importing the libraries

import datetime

import hashlib

import json
```

```
from flask import Flask, jsonify, request

import requests

from uuid import uuid4 # Generate a unique id that is in hex

from urllib.parse import urlparse # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:

def __init__(self):

self.chain = []

self.transactions = [] # Adding transactions before they are added to a block

self.create_block(proof = 1, previous_hash = '0')

self.nodes = set() # Set is used as there is no order to be maintained as the nodes can be from all around the globe

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} # Adding transactions to make the blockchain a cryptocurrency

self.transactions = [] # The list of transacction should become empty after they are added to a block

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 check_proof is False:

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

# This method will add the trqnsaction to the list of trnsactions
```

```
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 # It will return the block index to which the transaction should be added

# This function will add the node containing an address to the set of nodes created in init function

def add_node(self, address):

parsed_url = urlparse(address) # urlparse will parse the url from the address

self.nodes.add(parsed_url.netloc) # Add is used and not append as it's a set. Netloc will only return '127.0.0.1:5000'

# Consensus Protocol. This function will replace all the shorter chain with the longer chain in all the nodes on the network

def replace_chain(self):

network = self.nodes # network variable is the set of nodes all around the globe

longest_chain = None # It will hold the longest chain when we scan the network

max_length = len(self.chain) # This will hold the length of the chain held by the node that runs this function

for node in network:

response = requests.get(f'http://{node}/get_chain') # Use get chain method already created to get the length of the chain

if response.status_code == 200:

length = response.json()['length'] # Extract the length of the chain from get_chain fiunction

chain = response.json()['chain']

if length > max_length and self.is_chain_valid(chain): # We check if the length is bigger and if the chain is valid then

max_length = length # We update the max length
```

```
longest_chain = chain # We update the longest chain

if longest_chain: # If longest_chain is not none that means it was replaced

self.chain = longest_chain # Replace the chain of the current node with the longest chain

return True

return False # Return false if current chain is the longest one

# Part 2 - Mining our Blockchain

# Creating a Web App

app = Flask(__name__)

# Creating an address for the node on Port 5000. We will create some other nodes as well on different ports

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

# Creating a Blockchain

blockchain = Blockchain()

# Mining a new block

@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) # Hadcoins to mine the block (A Reward). So the node gives 1 hadcoin to Abcde for mining the block

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

# Getting the full Blockchain

@app.route('/get_chain', methods = ['GET'])

def get_chain():

response = {'chain': blockchain.chain,

'length': len(blockchain.chain)}

return jsonify(response), 200

# Checking if the Blockchain is valid

@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': 'Houston, we have a problem. The Blockchain is not valid.'}

return jsonify(response), 200

# Adding a new transaction to the Blockchain

@app.route('/add_transaction', methods = ['POST']) # Post method as we have to pass
something to get something in return

def add_transaction():

json = request.get_json() # This will get the json file from postman. In Postman we will create
a json file in which we will pass the values for the keys in the json file

transaction_keys = ['sender', 'receiver', 'amount']

if not all(key in json for key in transaction_keys): # Checking if all keys are available in json

return 'Some elements of the transaction are missing', 400
```

```
index = blockchain.add_transaction(json['sender'], json['receiver'], json['amount'])

response = {'message': f'This transaction will be added to Block {index}'}

return jsonify(response), 201 # Code 201 for creation

# Part 3 - Decentralizing our Blockchain

# Connecting new nodes

@app.route('/connect_node', methods = ['POST']) # POST request to register the new nodes from the json file

def connect_node():

json = request.get_json()

nodes = json.get('nodes') # Get the nodes from json file

if nodes is None:

return "No node", 400

for node in nodes:

blockchain.add_node(node)

response = {'message': 'All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:',

'total_nodes': list(blockchain.nodes)}

return jsonify(response), 201

# Replacing the chain by the longest chain if needed

@app.route('/replace_chain', methods = ['GET'])

def replace_chain():

is_chain_replaced = blockchain.replace_chain()

if is_chain_replaced:

response = {'message': 'The nodes had different chains so the chain was replaced by the longest one.',

'new_chain': blockchain.chain}

else:

response = {'message': 'All good. The chain is the largest one.',
```

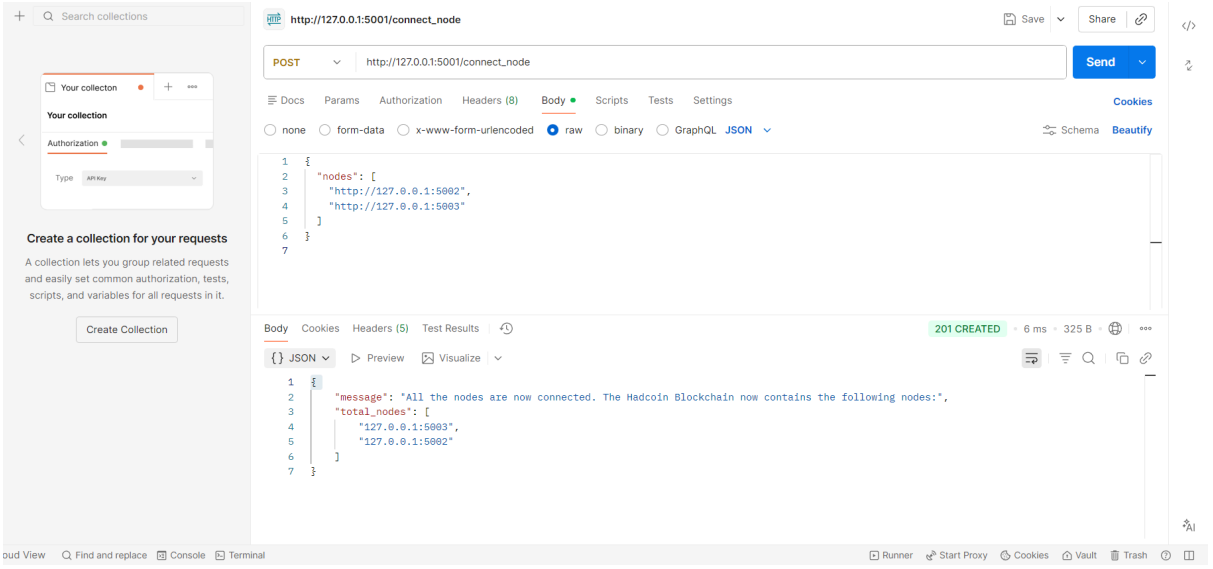
'actual_chain': blockchain.chain}

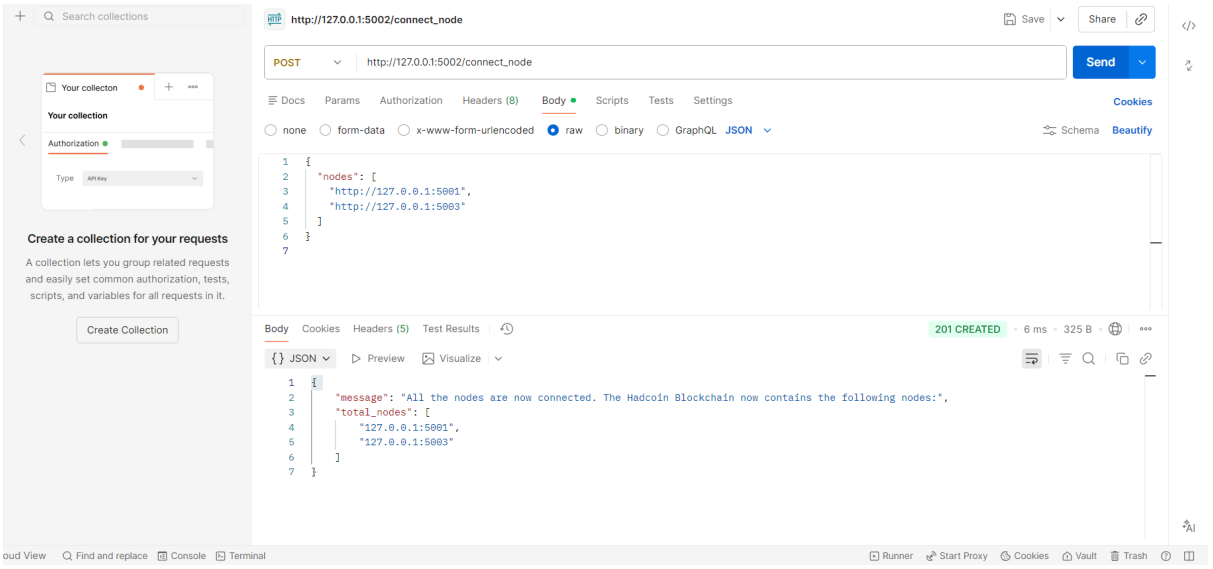
return jsonify(response), 200

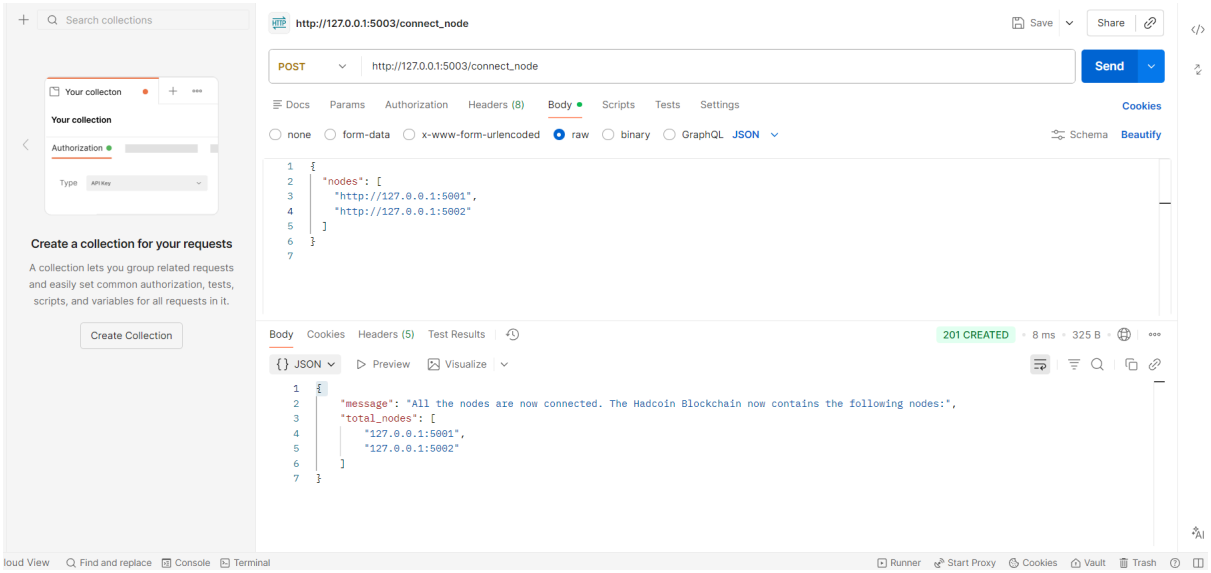
Running the app

app.run(host = '0.0.0.0', port = 5001)

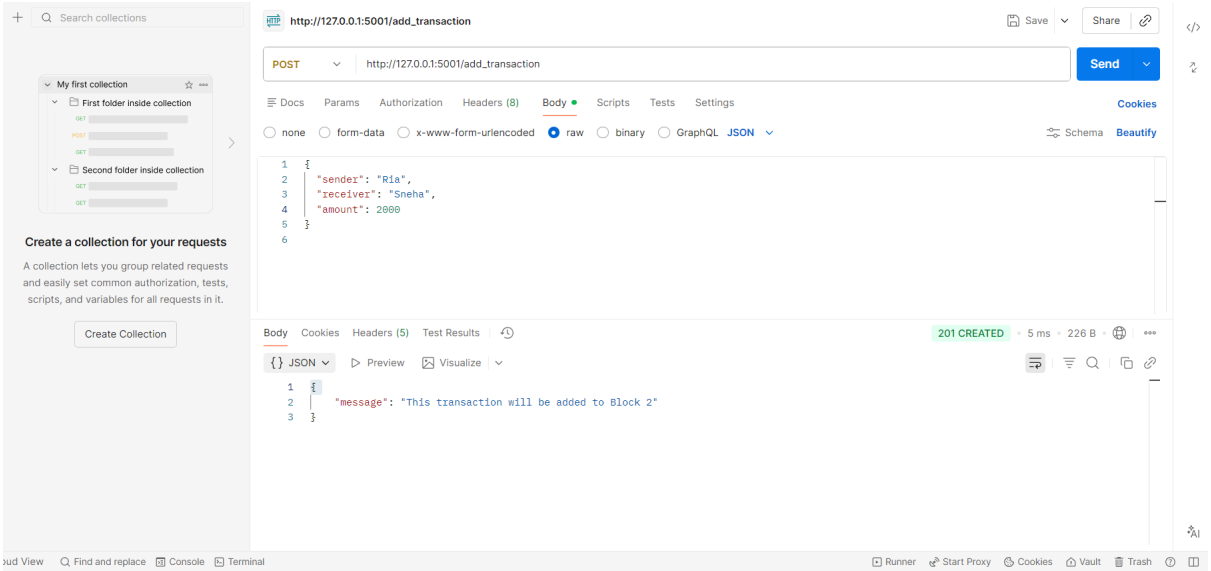
Step 1: Connect nodes

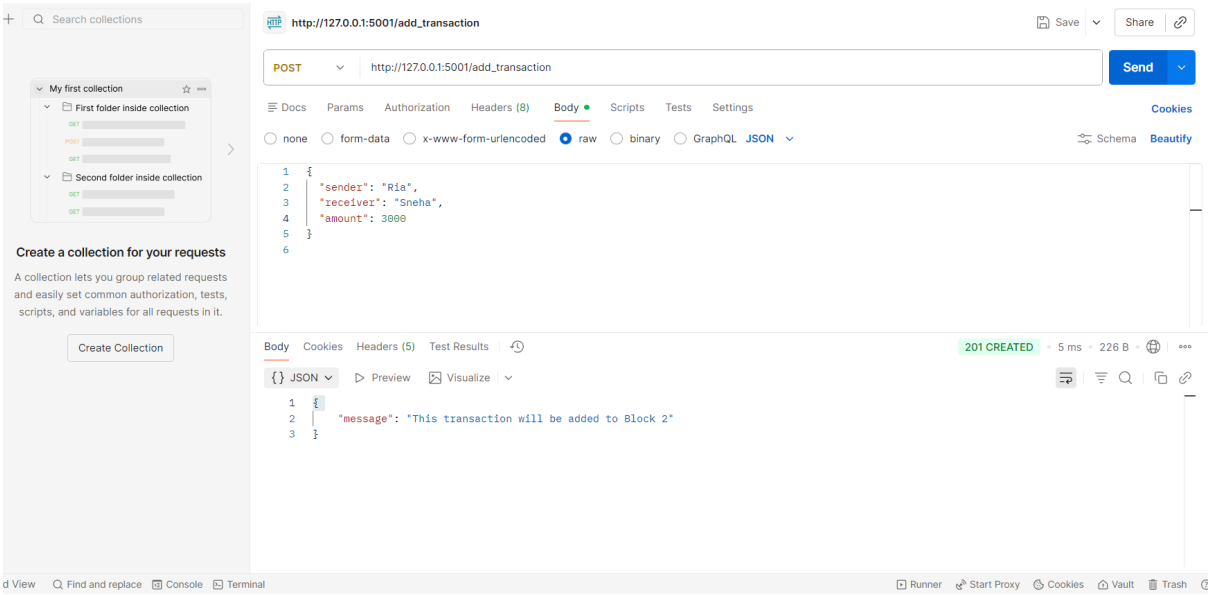




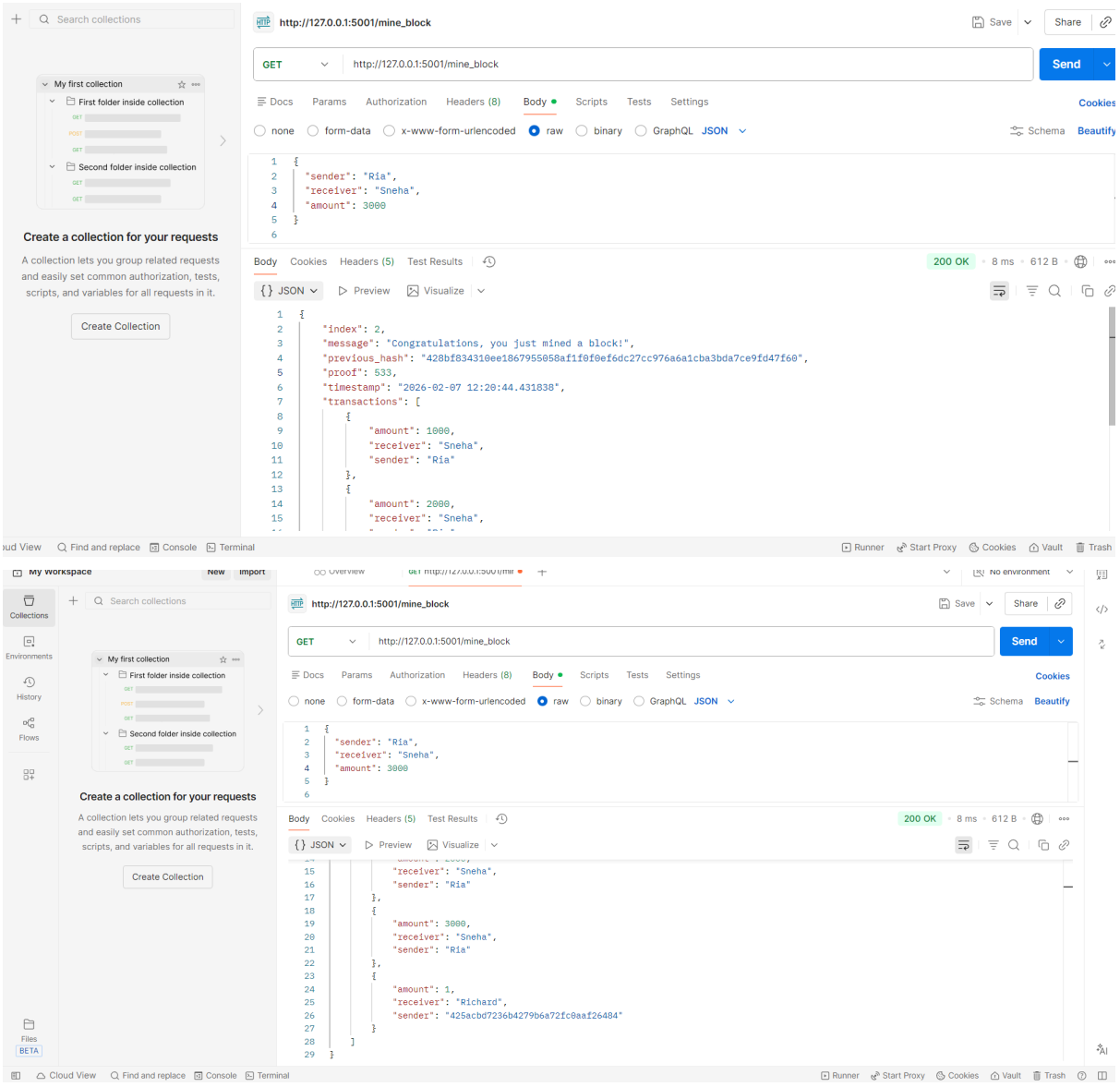


Step 2: Add transaction

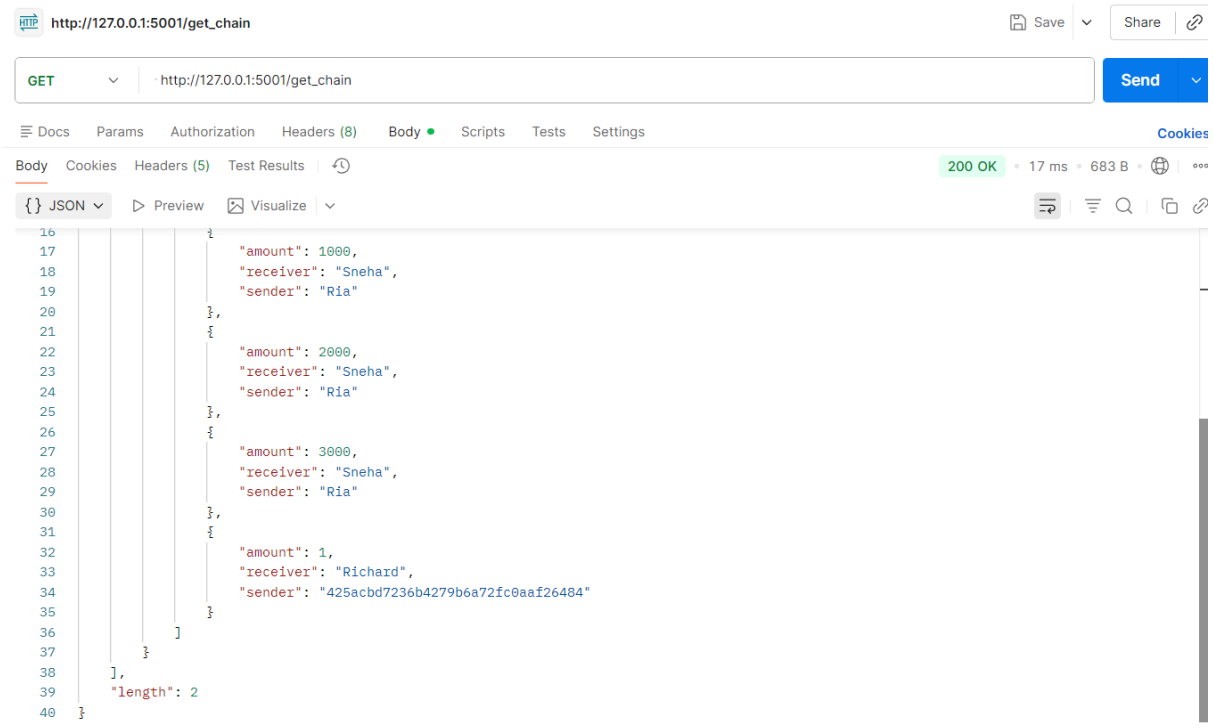


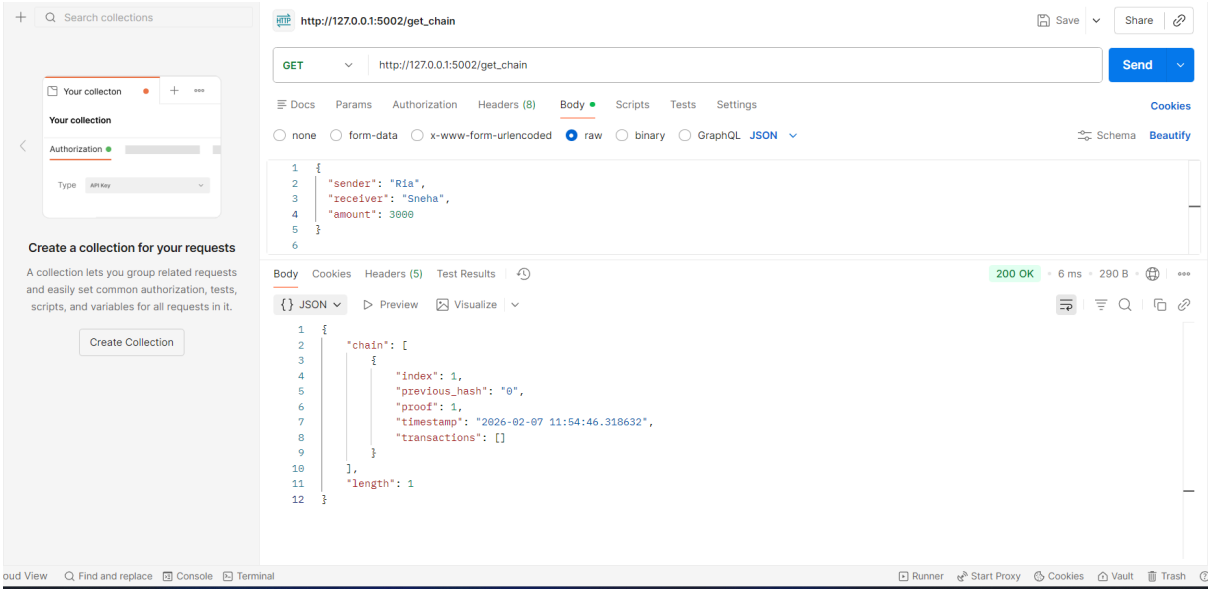


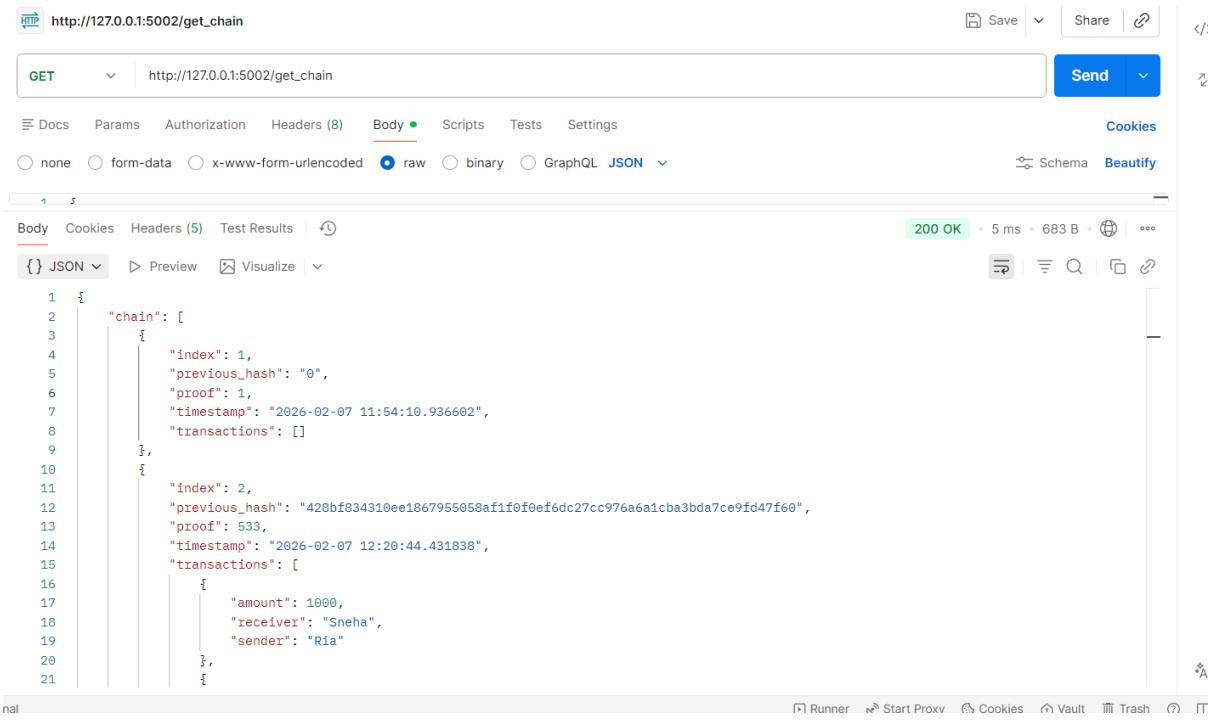
Step 3: Mine blocks



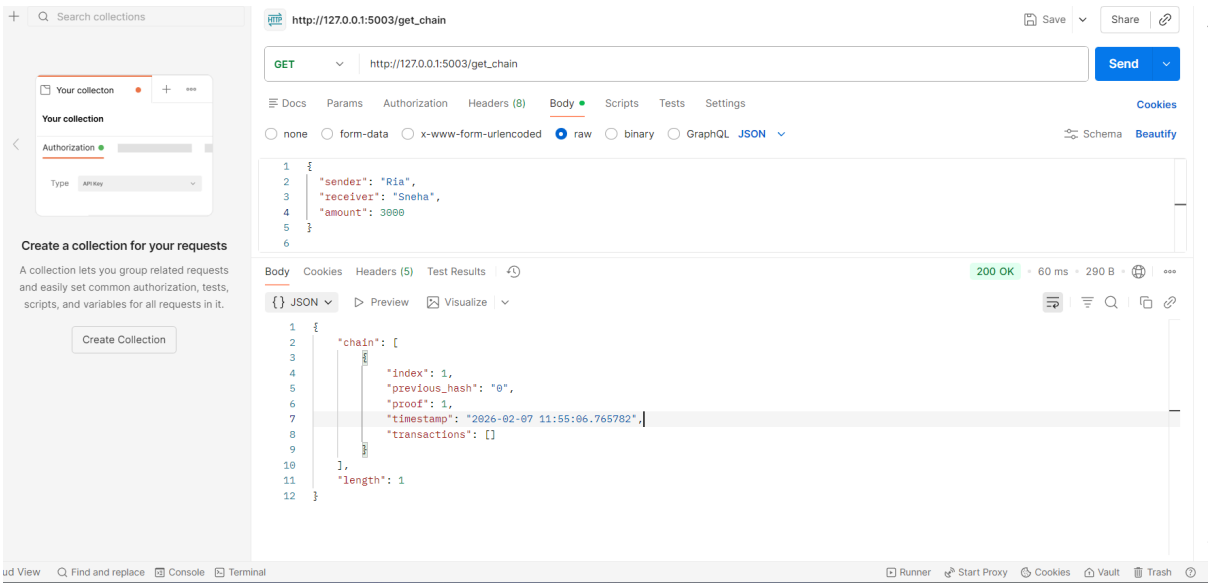
Step 4: Check chain before consensus

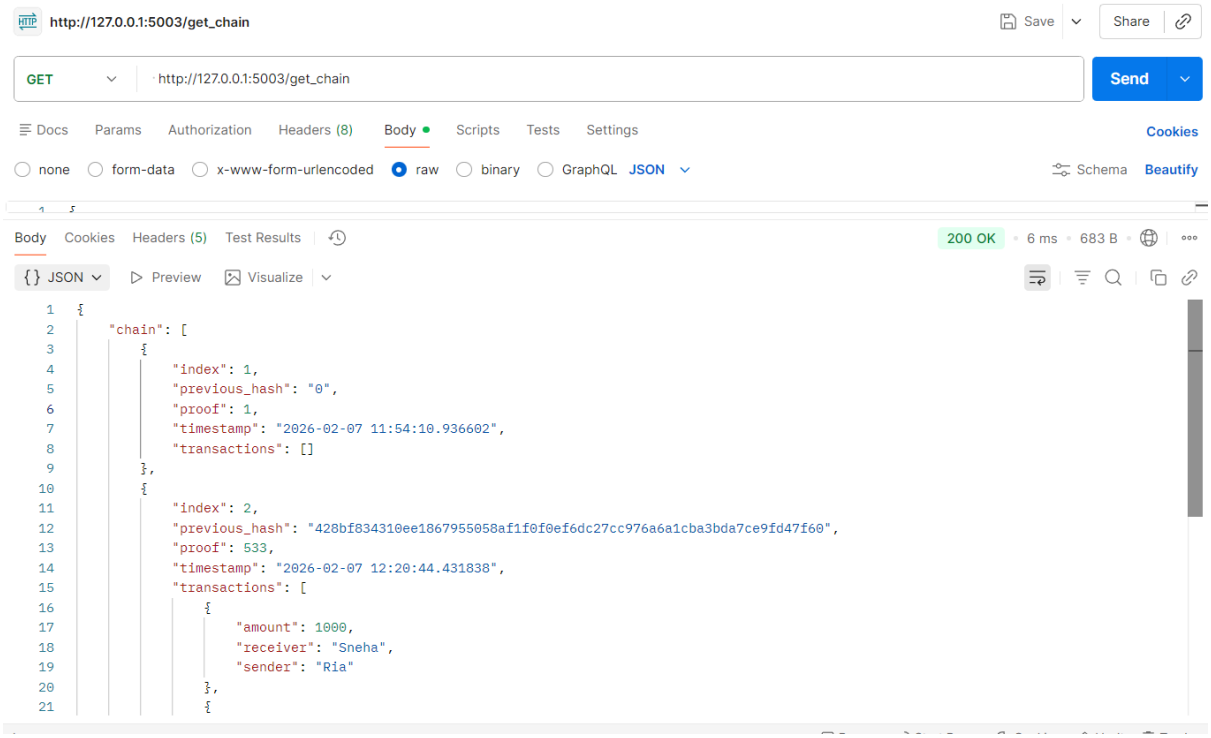






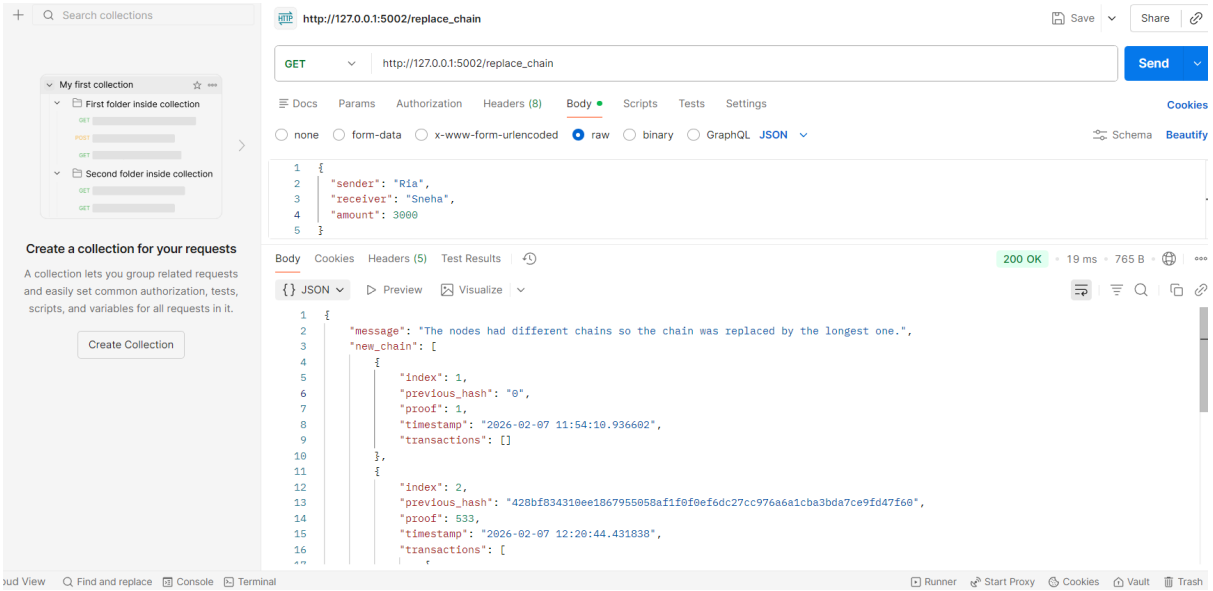

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-07 11:54:10.936602",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "428bf834310ee1867955058af1f0f0ef6dc27cc976a6a1cba3bda7ce9fd47f60",
            "proof": 533,
            "timestamp": "2026-02-07 12:20:44.431838",
            "transactions": [
                {
                    "amount": 1000,
                    "receiver": "Sneha",
                    "sender": "Ria"
                },
                {
                    "amount": 2000,
                    "receiver": "Sneha",
                    "sender": "Ria"
                },
                {
                    "amount": 3000,
                    "receiver": "Sneha",
                    "sender": "Ria"
                },
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "425acbd7236b4279b6a72fc0aaf26484"
                }
            ]
        }
    ],
    "length": 2
}
```


Search collections
Your collecton
Your collection
Authorization
Type
API Key
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
http://127.0.0.1:5003/get_chain
Save
Share
GET
http://127.0.0.1:5003/get_chain
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"sender": "Ria",
"receiver": "Sneha",
"amount": 3000
}
Body
Cookies
Headers (5)
Test Results
200 OK
60 ms
290 B
{ } JSON
Preview
Visualize
{
"chain": [
{
"index": 1,
"previous_hash": "0",
"proof": 1,
"timestamp": "2026-02-07 11:55:06.765782",
"transactions": []
}
],
"length": 1
}
ud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash

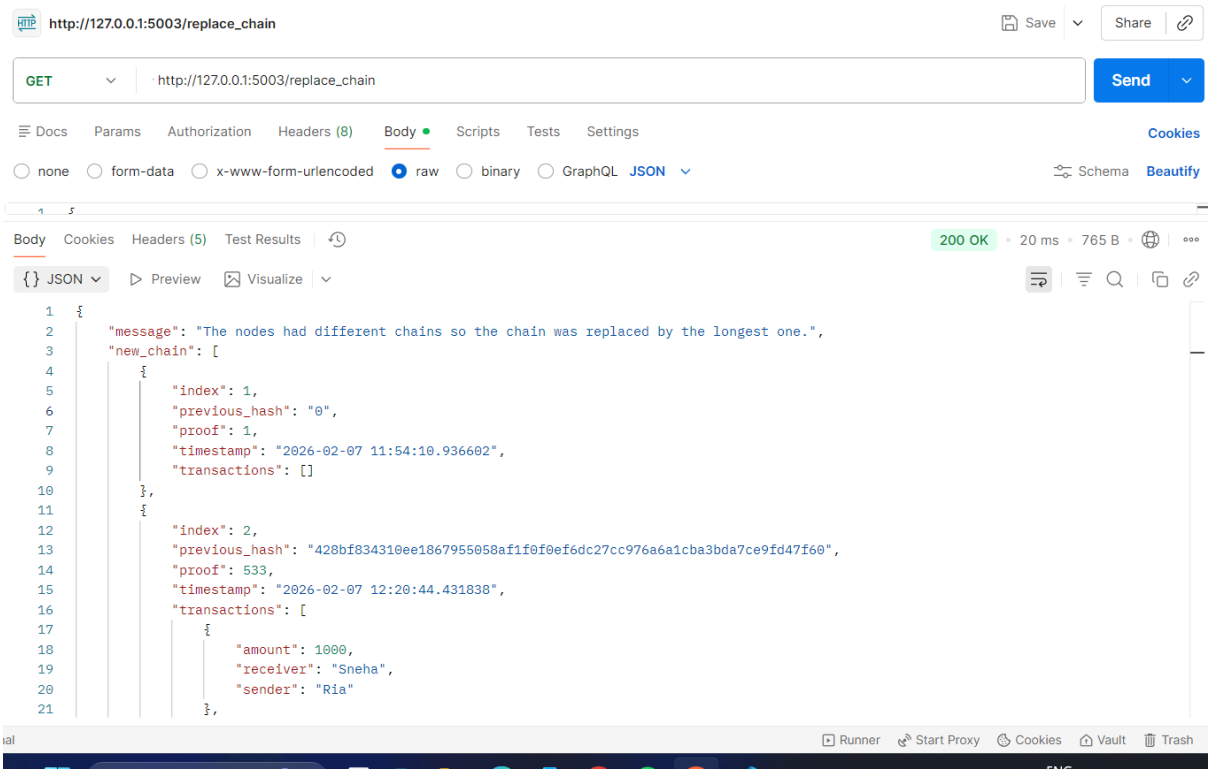

http://127.0.0.1:5003/get_chain
Save
Share
GET
http://127.0.0.1:5003/get_chain
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
Body
Cookies
Headers (5)
Test Results
200 OK
6 ms
683 B
{ } JSON
Preview
Visualize
{
"chain": [
{
"index": 1,
"previous_hash": "0",
"proof": 1,
"timestamp": "2026-02-07 11:54:10.936602",
"transactions": []
},
{
"index": 2,
"previous_hash": "428bf834310ee1867955058af1f0f0ef6dc27cc976a6a1cba3bda7ce9fd47f60",
"proof": 533,
"timestamp": "2026-02-07 12:20:44.431838",
"transactions": [
{
"amount": 1000,
"receiver": "Sneha",
"sender": "Ria"
},
{

```
            {
                "amount": 2000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 3000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 1,
                "receiver": "Richard",
                "sender": "425acbd7236b4279b6a72fc0aaf26484"
            }
        ]
    }
],
"length": 2
}
```


Step 5:Apply consensus



```
        "transactions": [
            {
                "amount": 1000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 2000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 3000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 1,
                "receiver": "Richard",
                "sender": "425acbd7236b4279b6a72fc0aaf26484"
            }
        ]
    }
]
}
```


```
{
    "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": "2026-02-07 11:54:10.936602",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "428bf834310ee1867955058af1f0f0ef6dc27cc976a6a1cba3bda7ce9fd47f60",
            "proof": 533,
            "timestamp": "2026-02-07 12:20:44.431838",
            "transactions": [
                {
                    "amount": 1000,
                    "receiver": "Sneha",
                    "sender": "Ria"
                },
```

```
            {
                "amount": 2000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 3000,
                "receiver": "Sneha",
                "sender": "Ria"
            },
            {
                "amount": 1,
                "receiver": "Richard",
                "sender": "425acbd7236b4279b6a72fc0aaf26484"
            }
        ]
    }
]
}
```


CONCLUSION:

In this experiment, a cryptocurrency blockchain was successfully created using Python and Flask, where multiple nodes performed transactions and mining independently. Different blockchain lengths were generated across nodes and the consensus mechanism based on the longest chain rule was applied using the replace_chain function. After consensus, all nodes synchronized to the longest valid chain, demonstrating how decentralized networks maintain consistency, security, and agreement without a central authority