EXPERIMENT NO 03

Name: Tejas Gunjal **Class:** D20A

Roll: No: 24 **Batch:** B

Aim: Create a cryptocurrency using python and perform mining in the Blockchain created.

Theory:

1. Blockchain Introduction

Blockchain is a distributed digital ledger used to record transactions in a secure and organized manner. Instead of storing data in a single central database, the information is shared across multiple systems (called nodes) in a network. This makes the system decentralized and more reliable.
The data in a blockchain is stored in units called blocks. These blocks are arranged sequentially, and each block is linked to the previous one using a cryptographic hash value. This connection forms a continuous chain of blocks, which is why it is called a blockchain.

Each block contains:

- Transaction details
- Timestamp
- Hash of the previous block
- Its own hash value

The hash is generated using a cryptographic algorithm (such as SHA-256). Even a small change in the block data produces a completely different hash value. Since every block stores the hash of the previous block, modifying one block would change its hash and break the entire chain. Therefore, any tampering can be easily detected.

2. Mining Process

Mining is the process of verifying transactions and adding them to the blockchain by creating a new block. It is an important part of any cryptocurrency system because it ensures security and prevents fraudulent activities such as double spending.

In our blockchain implementation, mining is done using a method called Proof of Work (PoW). This method requires the miner to solve a computational puzzle before a block can be added to the chain.

The mining process works as follows:

- New transactions are collected and stored temporarily.
- The miner creates a new block containing these transactions.
- A special value called a nonce is used.

- The system repeatedly changes the nonce and generates a hash for the block.
- The goal is to generate a hash that satisfies a specific condition, such as starting with a certain number of leading zeros.

This process requires multiple attempts and computational effort. Once the correct hash is found, the block is successfully mined and added to the blockchain.

3. Multi – Node Blockchain Setup

A multi-node blockchain setup means that the blockchain network consists of multiple independent systems, called nodes, which are connected to each other. Each node maintains its own copy of the blockchain and participates in validating and updating the ledger. Unlike a single-node system, where everything runs on one server, a multi-node setup ensures decentralization. This means there is no central authority controlling the network. All nodes work together to maintain the integrity of the blockchain.

In these practical, multiple nodes are created by running the blockchain application on different ports. Each node:

- Stores its own copy of the blockchain
- Maintains a list of transactions
- Communicates with other nodes in the network.

When a new block is mined on one node, other nodes can verify and update their chains accordingly. If any node has a longer valid chain, other nodes can replace their existing chain using the consensus mechanism.

The advantages of a multi-node setup include:

- Increased reliability and fault tolerance
- Improved security
- True decentralization
- No single point of failure

4. Consensus Mechanism

In a decentralized blockchain network, there is no central authority to decide which transactions are valid. Therefore, all the nodes in the network must agree on a single version of the blockchain. This agreement process is known as the consensus mechanism.

A consensus mechanism ensures that:

- All nodes maintain the same valid blockchain
- Invalid or fraudulent blocks are rejected
- The network remains synchronized

In this practical, the consensus mechanism used is based on the Longest Chain Rule, which works along with Proof of Work.

According to this rule:

- Each node checks the blockchain of other connected nodes.
- The chain with the greater length (more mined blocks) is considered valid.
- If a node finds a longer valid chain, it replaces its current chain with the longer one.

This process is also called chain synchronization.

The reason the longest chain is considered valid is that it represents the maximum computational work done by the network. This makes it difficult for malicious users to create fake chains. The consensus mechanism plays a very important role in maintaining trust, consistency, and security in the blockchain network. Without consensus, different nodes might have different versions of the blockchain, which would create conflicts and reduce reliability.

5. Transaction and Mining

Transactions

A transaction represents the transfer of cryptocurrency from one user to another. It is the basic unit of activity in a blockchain network.

Each transaction typically contains:

- Sender's address
- Receiver's address
- Amount to be transferred

When a transaction is created, it is not immediately added to the blockchain. Instead, it is stored temporarily in a pool of pending transactions. These transactions are later collected and added to a block during the mining process.

Before being added to the blockchain, transactions are validated to ensure that the sender has sufficient balance and that the transaction format is correct. Once the block containing the transactions is mined, the transactions become permanent and cannot be altered.

Mining Incentives

Mining requires computational effort and time. To encourage participants to contribute their computing power to maintain the network, a reward system is introduced. This is known as the mining incentive.

Whenever a miner successfully mines a block:

- A reward transaction is automatically generated.
- The miner receives a fixed amount of cryptocurrency as a reward.

This reward serves two main purposes:

- It motivates users to participate in the mining process.
- It helps in the circulation of new cryptocurrency into the system.

Code:

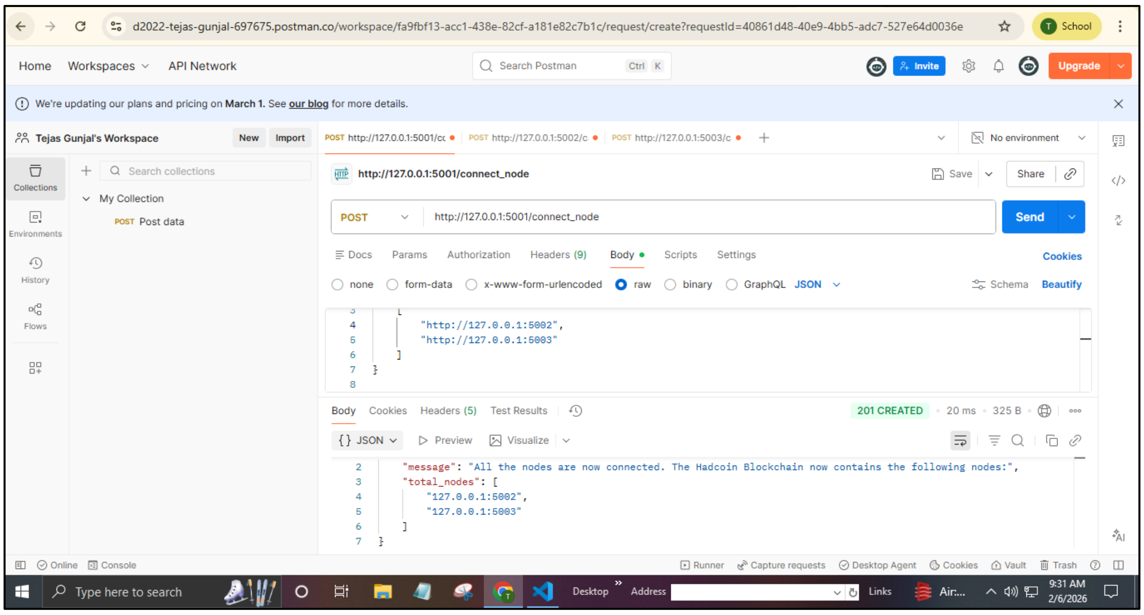
hadcoin_node_5001.py , hadcoin_node_5002.py , hadcoin_node_5003.py

```
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_hash,
                 'transactions':
self.transactions}
        self.transactions =
[]
        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

    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
            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('/get_chain', methods = ['GET'])
def get_chain():
    response = {'chain': blockchain.chain,
                'length': len(blockchain.chain)}
    return jsonify(response), 200

@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

@app.route('/add_transaction', methods =
['POST'])
def add_transaction():
    json =
request.get_json()

    transaction_keys = ['sender', 'receiver',
'amount']
    if not all(key in json for key in
transaction_keys):
        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

@app.route('/connect_node', methods =
['POST'])
def connect_node():
    json =
request.get_json()

    nodes =
json.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. The Hadcoin Blockchain now
contains the following nodes:',
                '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 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

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

Output :

➢ A new node is registered and the updated network node list is displayed.



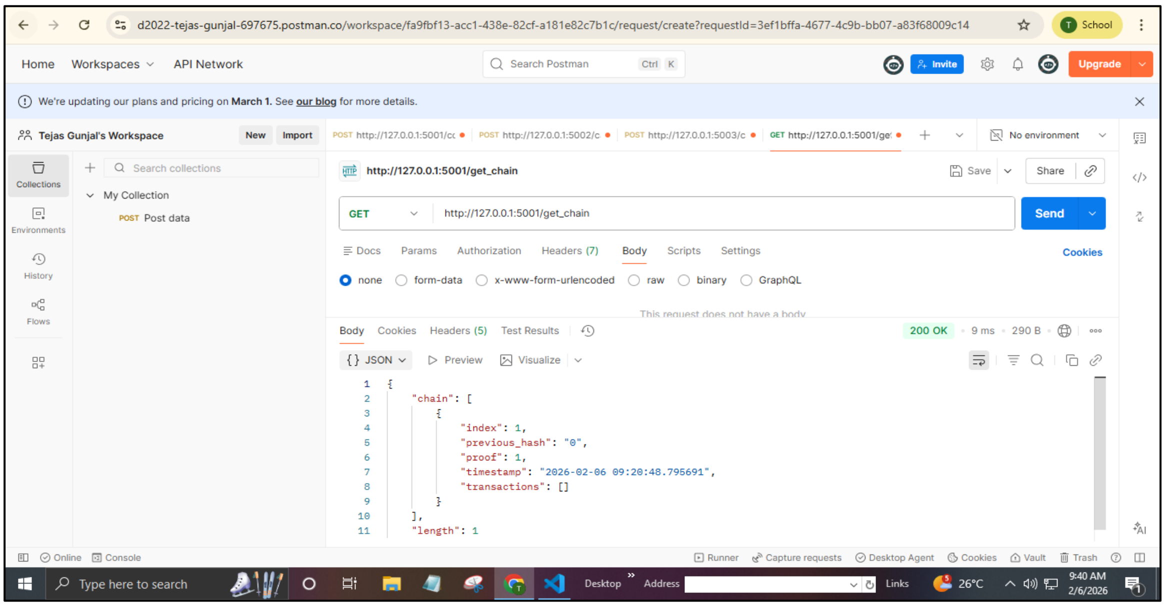
POST http://127.0.0.1:5001/connect_node

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

201 CREATED • 20 ms • 325 B

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

➢ The blockchain is retrieved from Node 5001, 5002, 5003 showing the genesis block with no transactions.



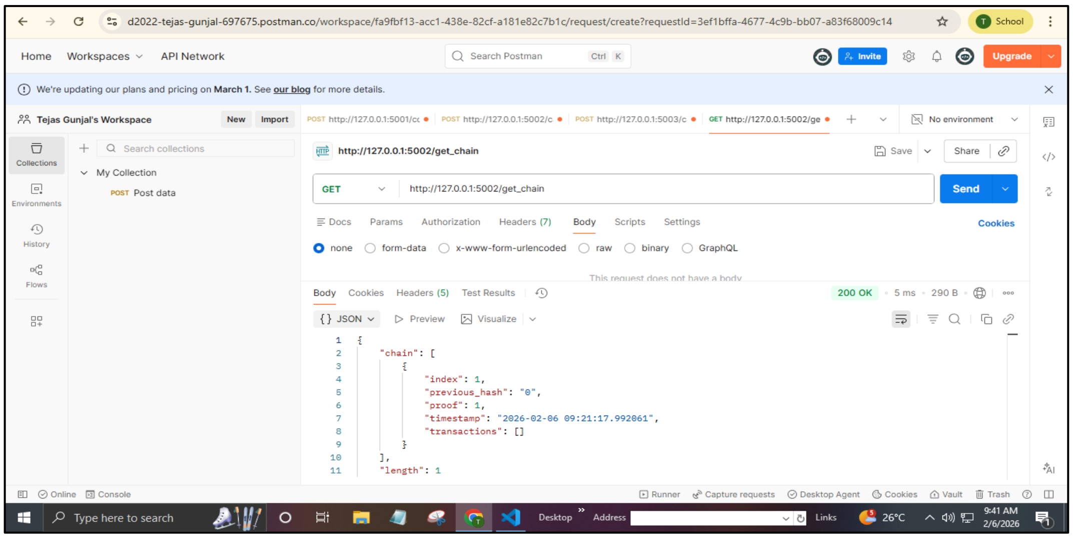
GET http://127.0.0.1:5001/get_chain

Body: none

This request does not have a body

200 OK • 9 ms • 290 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:20:48.795691",
            "transactions": []
        }
    ],
    "length": 1
```

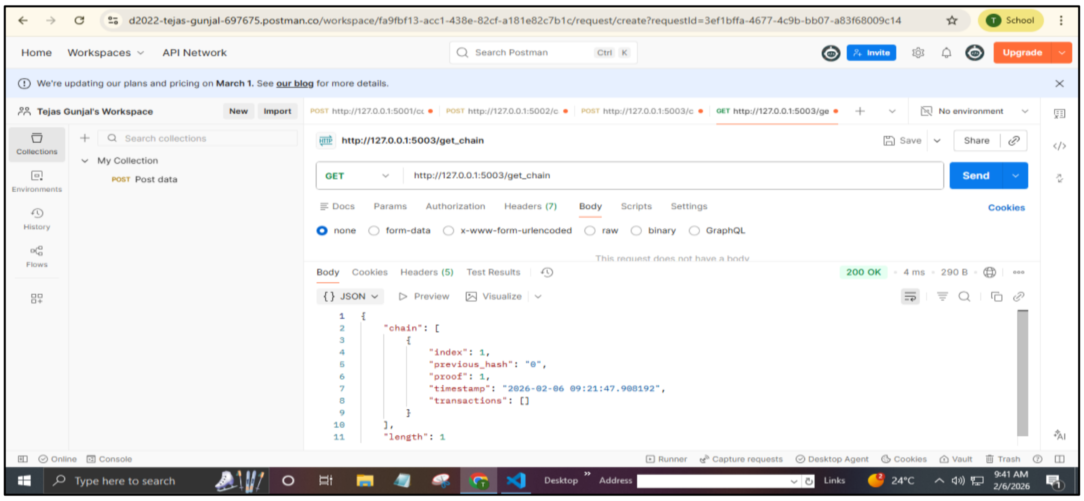


d2022-tejas-gunjal-697675.postman.co/workspace/fa9fbf13-acc1-438e-82cf-a181e82c7b1c/request/create?requestId=3ef1bffa-4677-4c9b-bb07-a83f68009c14

GET http://127.0.0.1:5002/get_chain

200 OK 5 ms 290 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:21:17.992061",
            "transactions": []
        }
    ],
    "length": 1
```



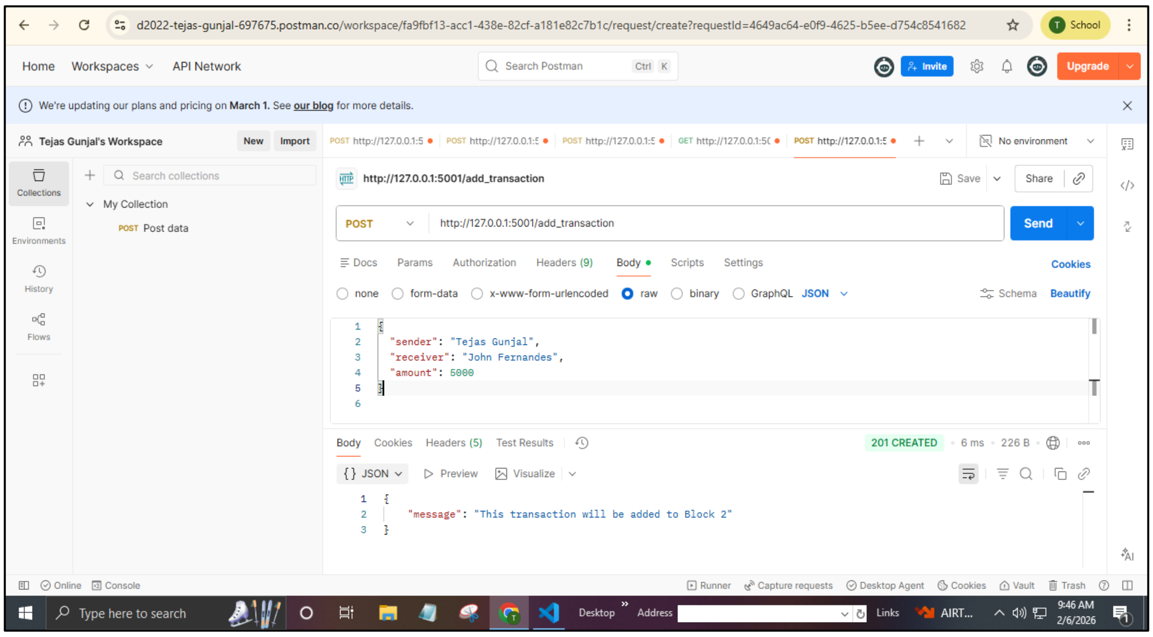
d2022-tejas-gunjal-697675.postman.co/workspace/fa9fbf13-acc1-438e-82cf-a181e82c7b1c/request/create?requestId=3ef1bffa-4677-4c9b-bb07-a83f68009c14

GET http://127.0.0.1:5003/get_chain

200 OK 4 ms 290 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:21:47.908192",
            "transactions": []
        }
    ],
    "length": 1
```

➢ A new transaction is created and added to the pending transaction pool.



d2022-tejas-gunjal-697675.postman.co/workspace/fa9fbf13-acc1-438e-82cf-a181e82c7b1c/request/create?requestId=4649ac64-e0f9-4625-b5ee-d754c8541682

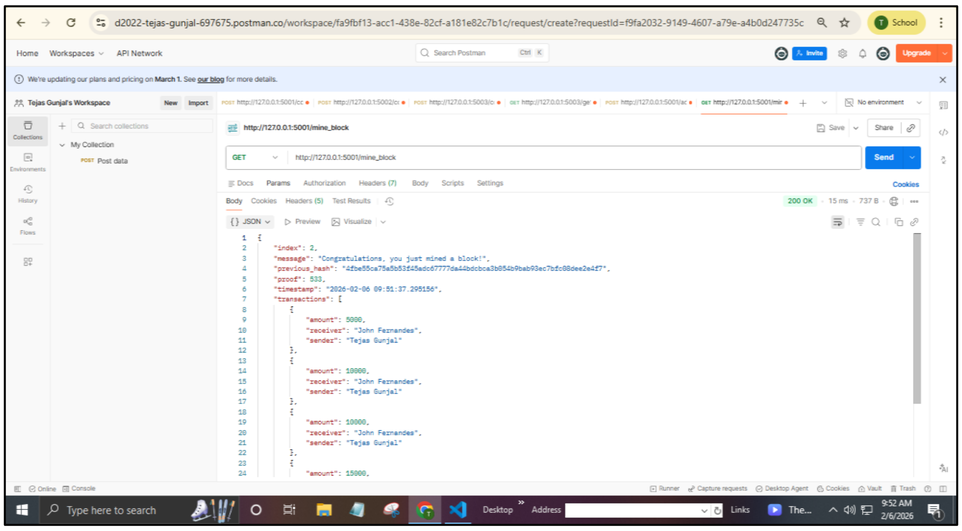
POST http://127.0.0.1:5001/add_transaction

```
{
"sender": "Tejas Gunjal",
"receiver": "John Fernandes",
"amount": 5000
}
```

201 CREATED 6 ms 226 B

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

- A new block is successfully mined on Node 5001, with proof and the previous hash displayed.

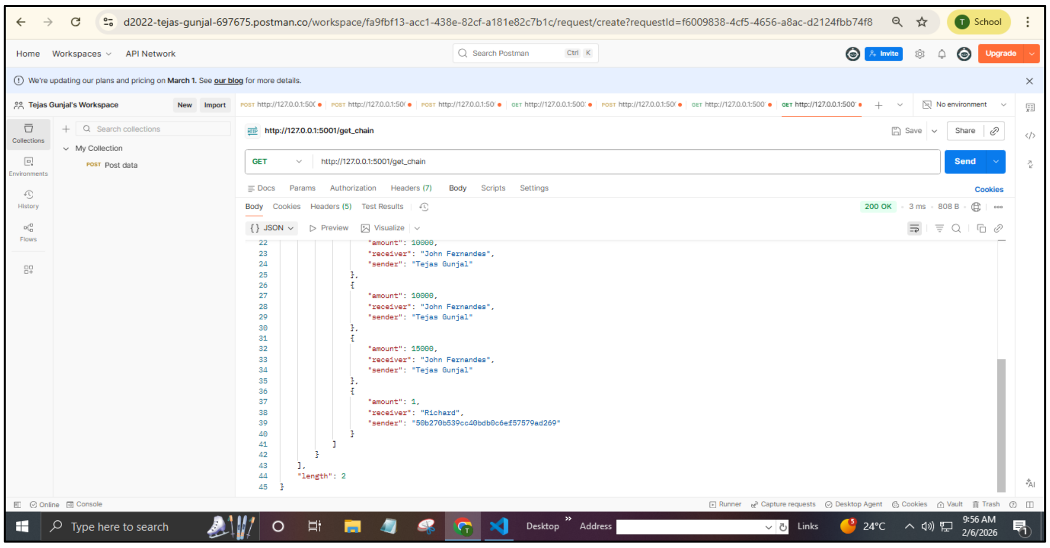


GET http://127.0.0.1:5001/mine_block

200 OK · 15 ms · 737 B

```
{
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "4fbe55ca75a5b53f45adc67777da44bdcbca3b054b9bab93ec7bfc08dee2e4f7",
    "proof": 533,
    "timestamp": "2026-02-06 09:51:37.295156",
    "transactions": [
        {
            "amount": 5000,
            "receiver": "John Fernandes",
            "sender": "Tejas Gunjal"
        },
        {
            "amount": 10000,
            "receiver": "John Fernandes",
            "sender": "Tejas Gunjal"
        },
        {
            "amount": 10000,
            "receiver": "John Fernandes",
            "sender": "Tejas Gunjal"
        },
        {
            "amount": 15000,
```

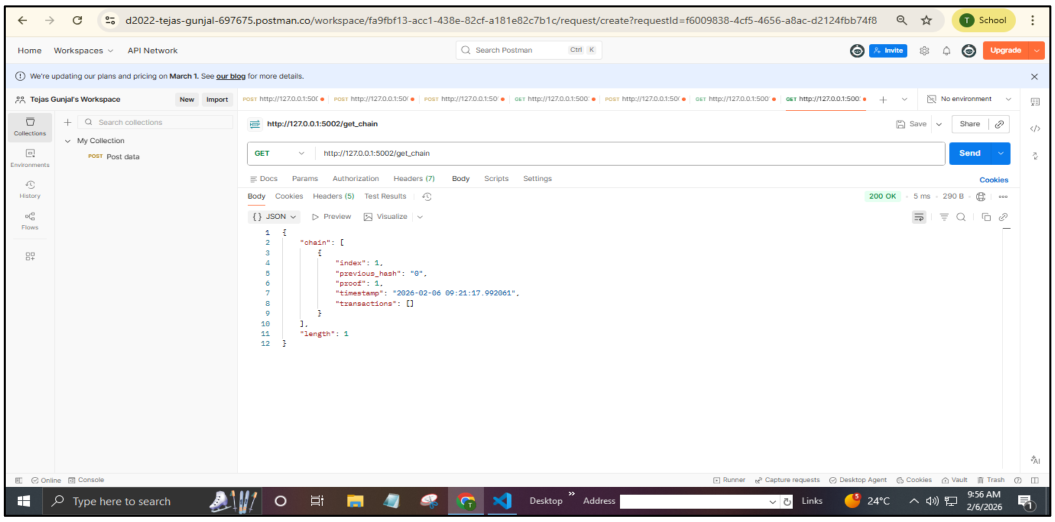
- Node 5001 successfully mines Block 2 while Node 5002 and Node 5003 remain on the genesis block, showing inconsistent chain lengths across the network.



GET http://127.0.0.1:5001/get_chain

200 OK · 3 ms · 808 B

```
                    "amount": 10000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 10000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 15000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "50b270b539cc40bdb0c6ef57579ad269"
                }
            ]
        }
    ],
    "length": 2
}
```

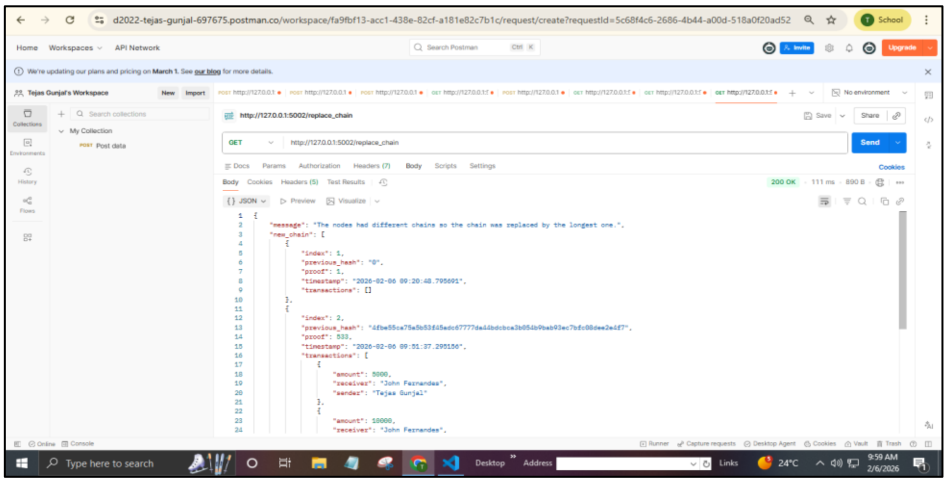


GET http://127.0.0.1:5002/get_chain

200 OK · 5 ms · 290 B

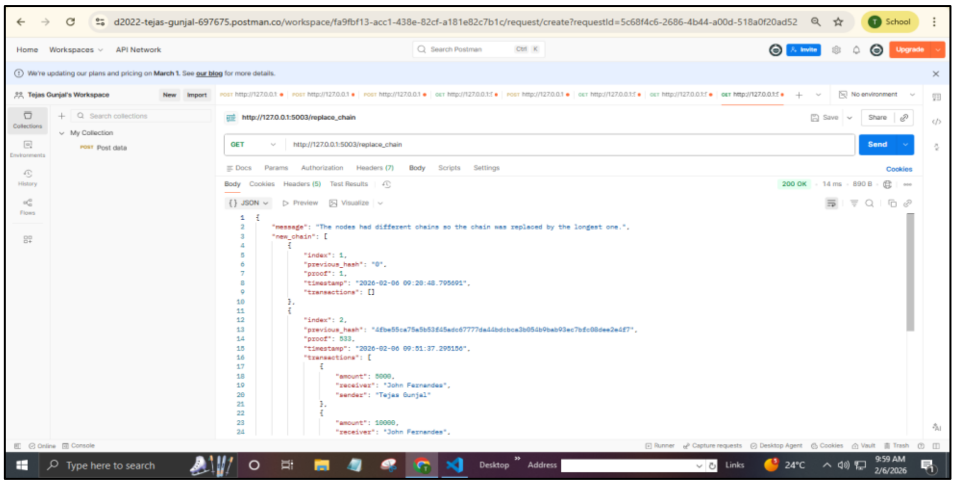
```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:21:17.992061",
            "transactions": []
        }
    ],
    "length": 1
}
```

➢ Consensus mechanism is executed on Node 5002 and Node 5003, successfully replacing their shorter chains with the longer valid chain from Node 5001.


GET http://127.0.0.1:5002/replace_chain

200 OK · 111 ms · 890 B

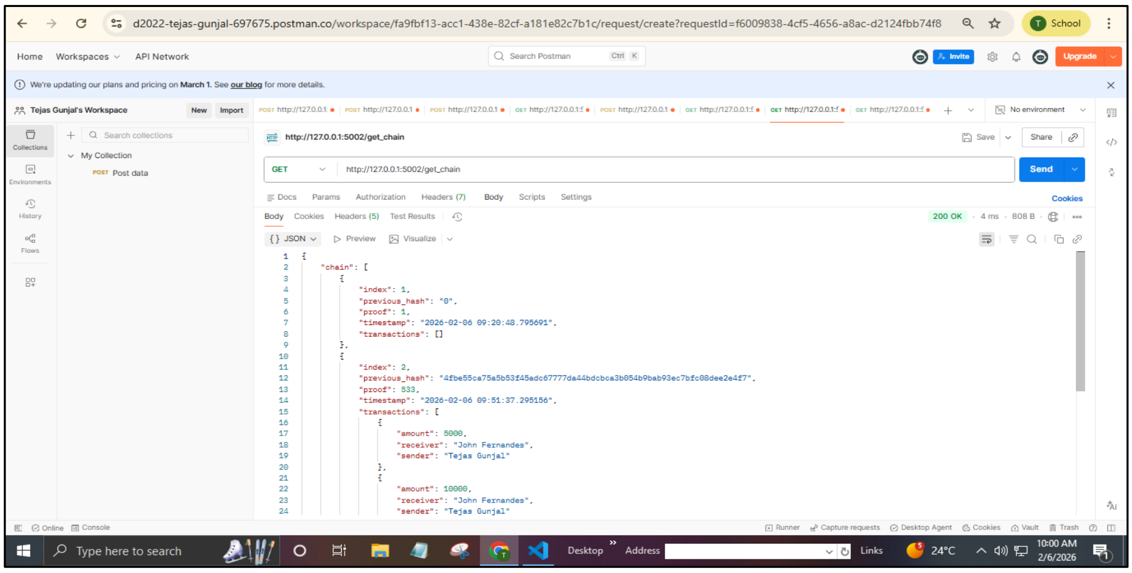
```
{
    "message": "The nodes had different chains so the chain was replaced by the longest one.",
    "new_chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:20:48.795691",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "4fbe55ca75a5b53f45adc67777da44bdcbca3b054b9bab93ec7bfc08dee2e4f7",
            "proof": 533,
            "timestamp": "2026-02-06 09:51:37.295156",
            "transactions": [
                {
                    "amount": 5000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 10000,
                    "receiver": "John Fernandes",
```


GET http://127.0.0.1:5003/replace_chain

200 OK · 14 ms · 890 B

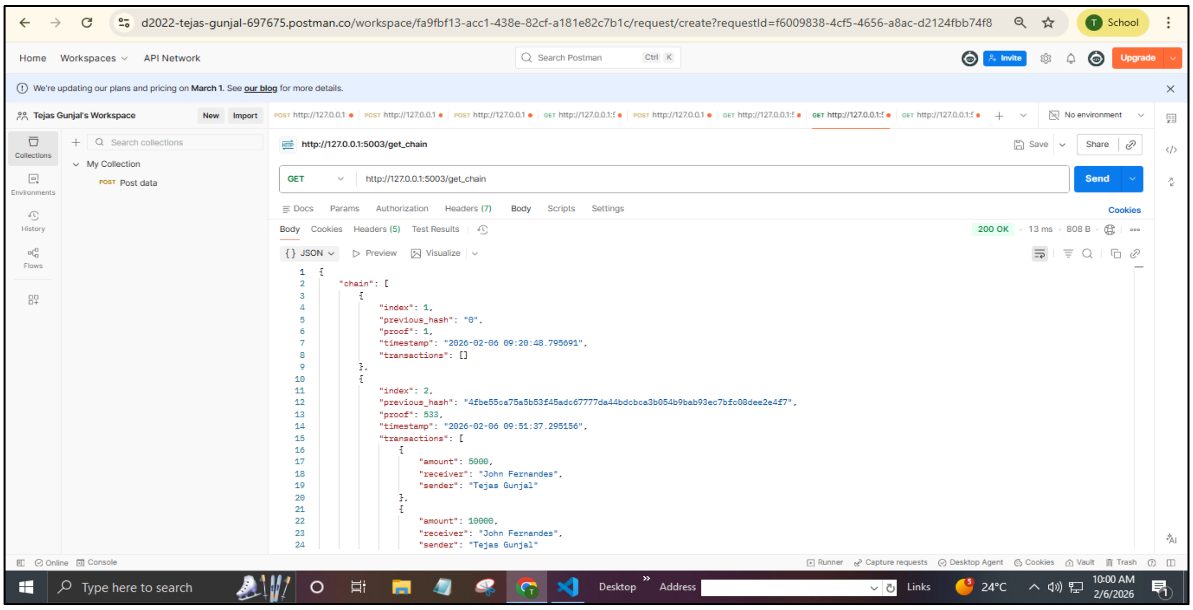
```
{
    "message": "The nodes had different chains so the chain was replaced by the longest one.",
    "new_chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:20:48.795691",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "4fbe55ca75a5b53f45adc67777da44bdcbca3b054b9bab93ec7bfc08dee2e4f7",
            "proof": 533,
            "timestamp": "2026-02-06 09:51:37.295156",
            "transactions": [
                {
                    "amount": 5000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 10000,
                    "receiver": "John Fernandes",
```

➢ Node 5002 and Node 5003 are successfully synchronized, both now displaying the updated blockchain with Block 2 containing two transactions and consistent chain lengths across the network.


GET http://127.0.0.1:5002/get_chain

200 OK · 4 ms · 808 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:20:48.795691",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "4fbe55ca75a5b53f45adc67777da44bdcbca3b054b9bab93ec7bfc08dee2e4f7",
            "proof": 533,
            "timestamp": "2026-02-06 09:51:37.295156",
            "transactions": [
                {
                    "amount": 5000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 10000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
```

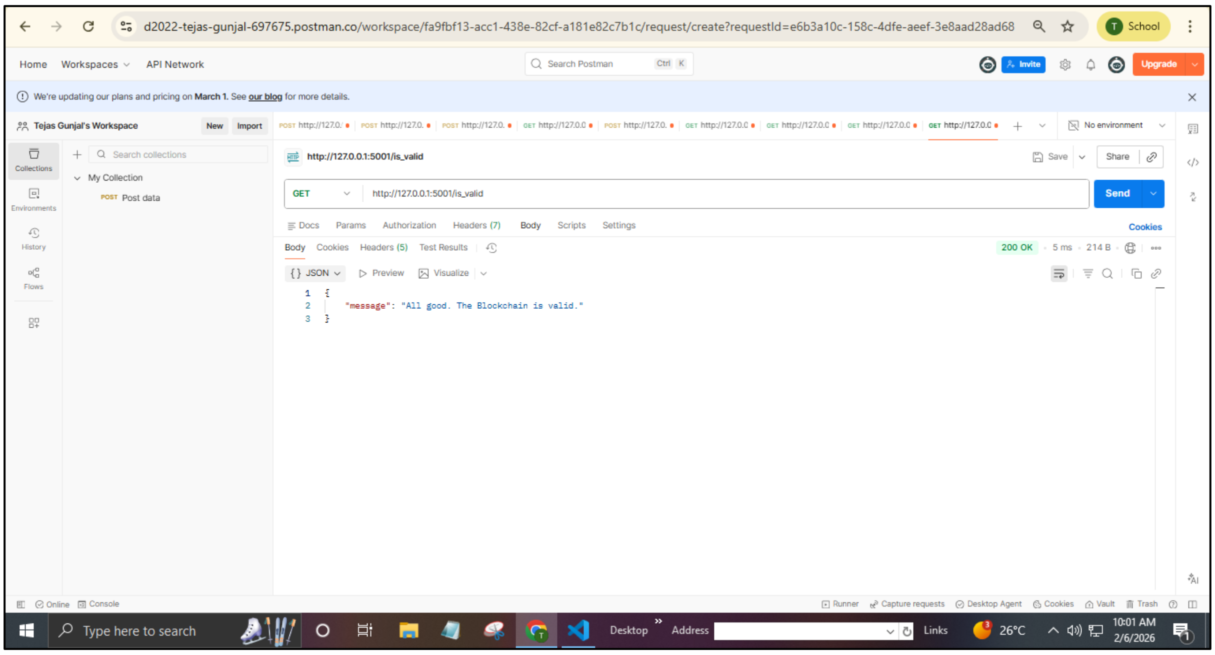


GET http://127.0.0.1:5003/get_chain

200 OK · 13 ms · 808 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 09:20:48.795691",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "4fbe55ca75a5b53f45adc67777da44bdcbca3b054b9bab93ec7bfc08dee2e4f7",
            "proof": 533,
            "timestamp": "2026-02-06 09:51:37.295156",
            "transactions": [
                {
                    "amount": 5000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
                },
                {
                    "amount": 10000,
                    "receiver": "John Fernandes",
                    "sender": "Tejas Gunjal"
```

- ➢ Blockchain validity is checked on Node 5001



GET http://127.0.0.1:5001/is_valid

200 OK · 5 ms · 214 B

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

Conclusion :

In this practical, we successfully created a basic cryptocurrency system using Python and implemented a working blockchain. We learned how blocks are linked using cryptographic hashes and how Proof of Work is used to mine new blocks securely. The experiment demonstrated how transactions are added, verified, and permanently recorded in the blockchain. By running multiple nodes, we understood the concept of decentralization and how different nodes maintain their own copies of the ledger. The consensus mechanism using the longest chain rule helped synchronize all nodes and ensured consistency across the network. Overall, this practical provided a clear understanding of blockchain structure, mining, transactions, and network consensus.