EXPERIMENT 3

Name-Bhagyesh Patil
Class-D20A
Roll no-03

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

Theory:

1. Blockchain Overview

A blockchain is a distributed, decentralized digital ledger used to record transactions securely across multiple computers (nodes). Instead of relying on a central authority, every node in the network maintains its own copy of the blockchain.
Each block in the blockchain contains:

- A list of transactions
- A timestamp
- The hash of the previous block
- Its own cryptographic hash

The blocks are linked together using hashes, forming a chain. Any attempt to modify data in a block changes its hash, which breaks the chain and makes tampering easily detectable. This property ensures data integrity, transparency, and immutability.

2. Mining

Mining is the process by which new blocks are added to the blockchain. It involves:

- Collecting pending transactions
- Creating a block header
- Solving a computationally difficult problem known as Proof-of-Work (PoW)

In Proof-of-Work, miners repeatedly change a nonce value to generate a hash that satisfies the network's difficulty requirement (such as leading zeroes). Once a valid hash is found, the block is added to the blockchain and shared with other nodes. As an incentive, miners receive a cryptocurrency reward for their computational effort.

3. Multi-Node Blockchain Network

In this experiment, a multi-node blockchain network is simulated using three nodes running on different ports (5001, 5002, and 5003).

Each node:

- Operates independently
- Maintains its own copy of the blockchain
- Communicates with peer nodes to share newly mined blocks

This setup demonstrates how blockchain achieves decentralization and synchronization without a central server.

4. Consensus Mechanism

To ensure consistency across all nodes, the Longest Chain Rule is used as the consensus mechanism.

When different versions of the blockchain exist, the node accepts the longest valid chain as the correct one. This rule ensures that all nodes eventually agree on a single transaction history, even if temporary differences arise.

5. Transactions and Mining Reward

Transactions in the cryptocurrency system include:

- Sender address
- Receiver address
- Transaction amount

When a block is mined, all pending transactions are added to the block. Additionally, a special transaction is created to reward the miner with newly generated cryptocurrency. This reward mechanism motivates miners to maintain and secure the network.

6. Chain Replacement

The chain replacement process ensures network consistency. When the /replace_chain endpoint is triggered:

- A node requests blockchain data from its peers
- Validates the received chains
- Replaces its own chain if a longer and valid chain is found.

This mechanism helps resolve conflicts and keeps all nodes synchronized.

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 = 'Abcde', 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()
```

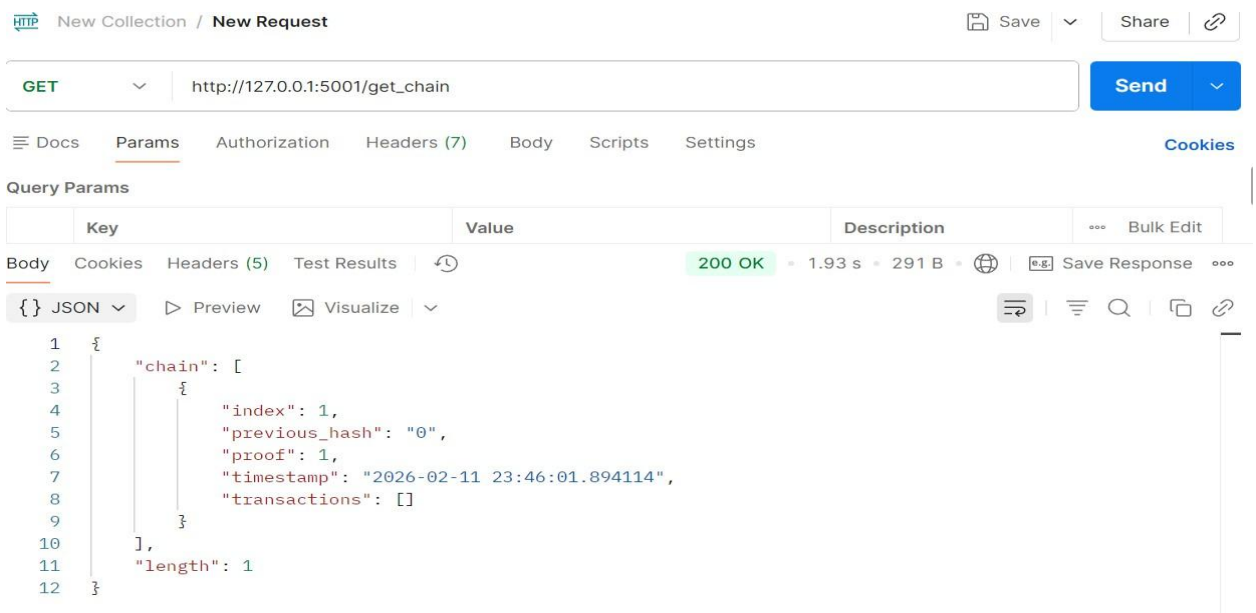
```
    if is_chain_replaced:
        response = {'message': 'The nodes had different chains so the chain was replaced by the
longest one.',
                    'new_chain': blockchain.chain}
    else:
        response = {'message': 'All good. The chain is the largest one.',
                    'actual_chain': blockchain.chain}
    return jsonify(response), 200

# Running the app
app.run(host = '0.0.0.0', port = 5003)
```

Output:

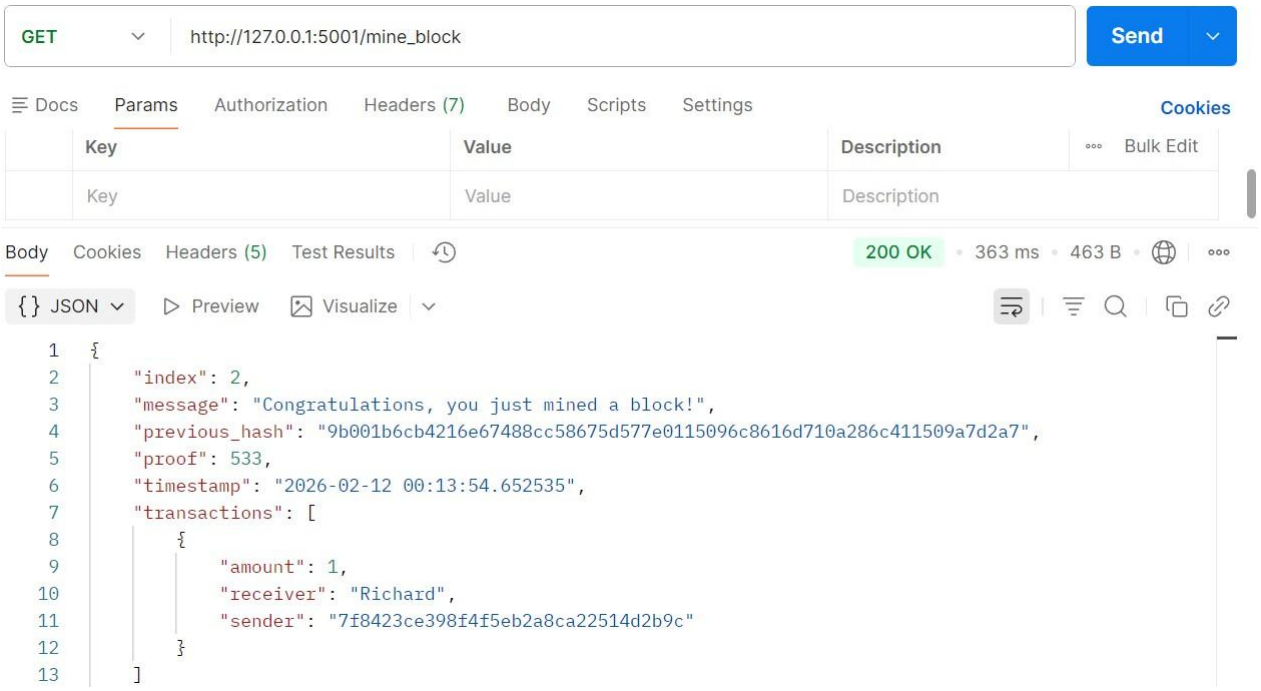
1)/get_chain

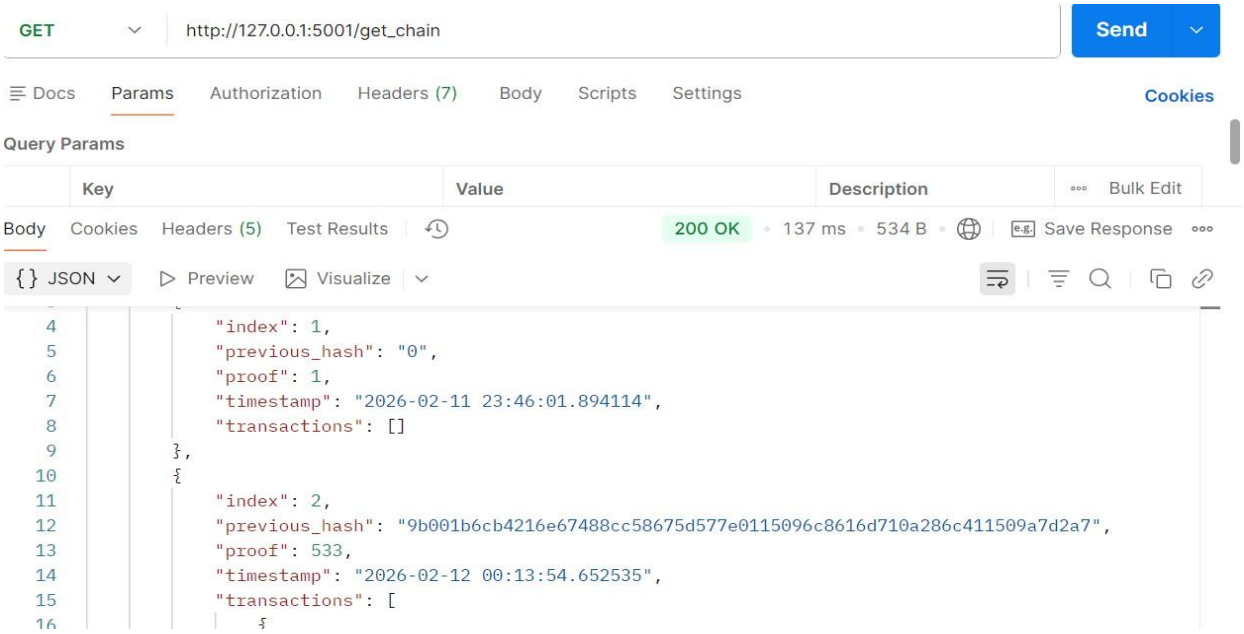
This GET request retrieves the complete blockchain from the node, showing all mined blocks and stored transactions.



2)/mine_block

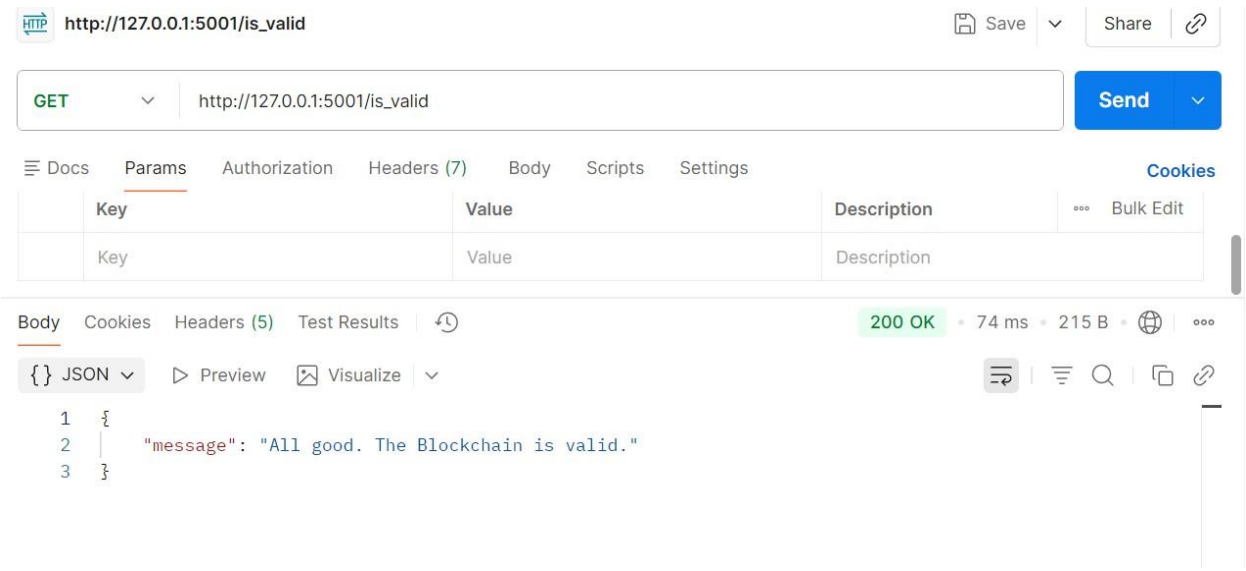
This GET request performs the mining process using Proof-of-Work and adds a new block to the blockchain.





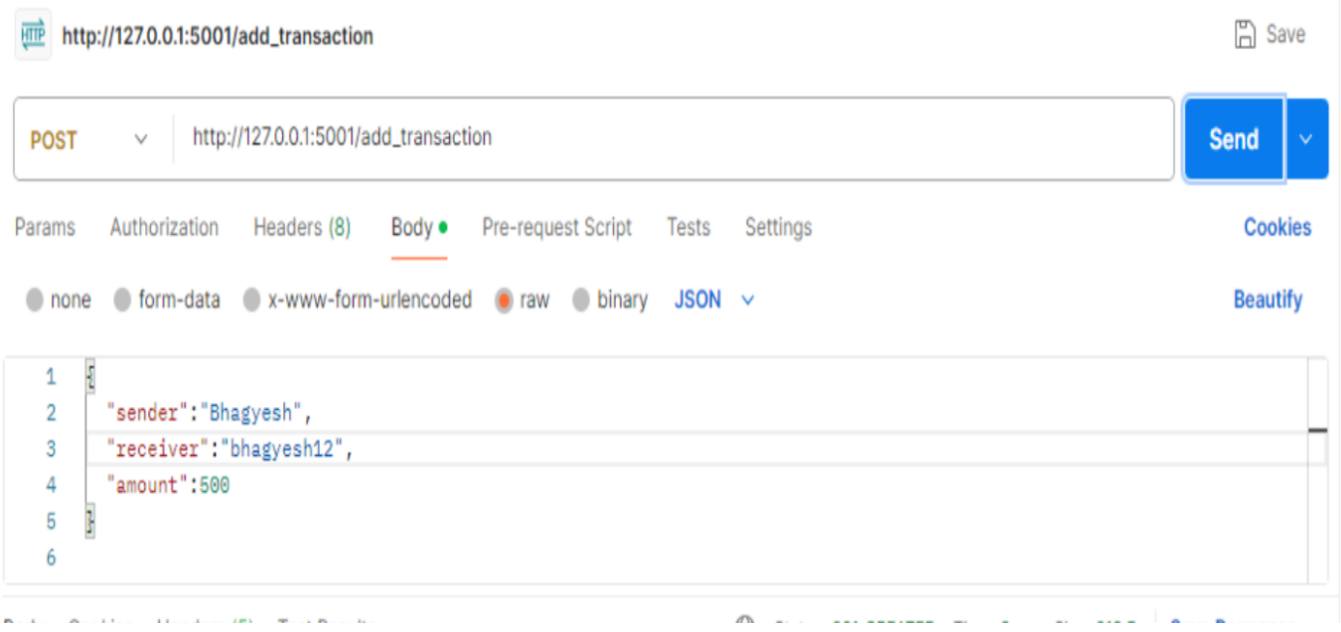
3)/is_valid

This GET request checks whether the current blockchain is valid and untampered.



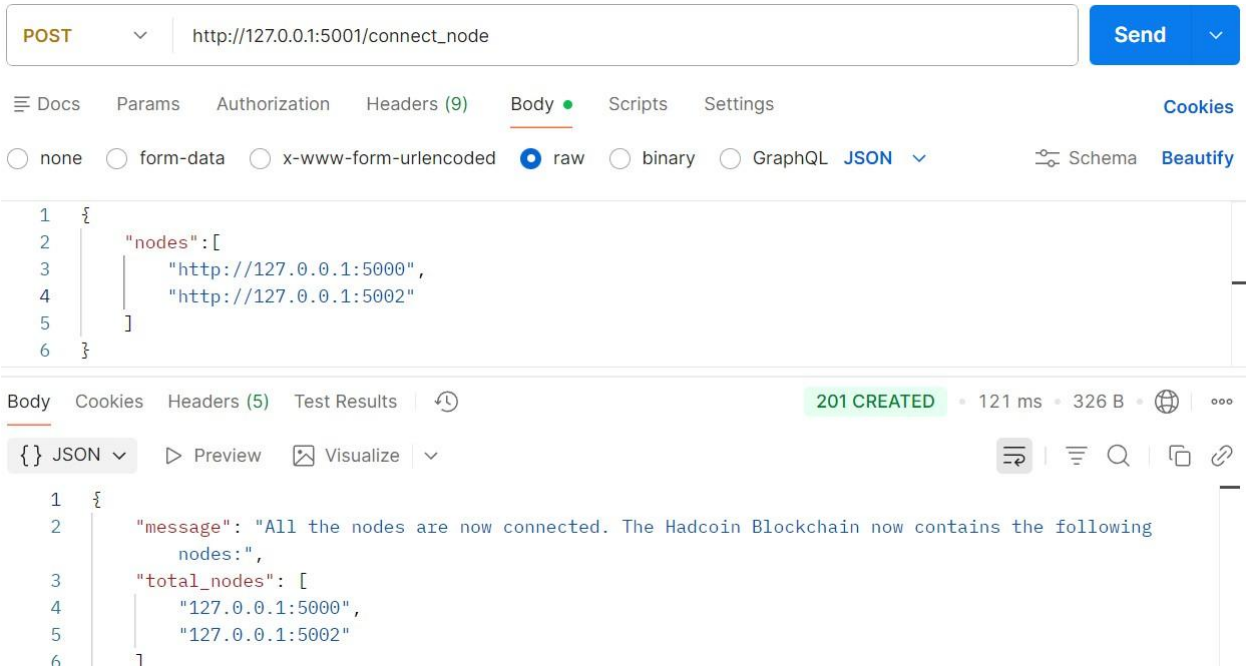
4)/add_transaction

This POST request submits a new transaction (sender, receiver, amount) to be added to the next mined block.



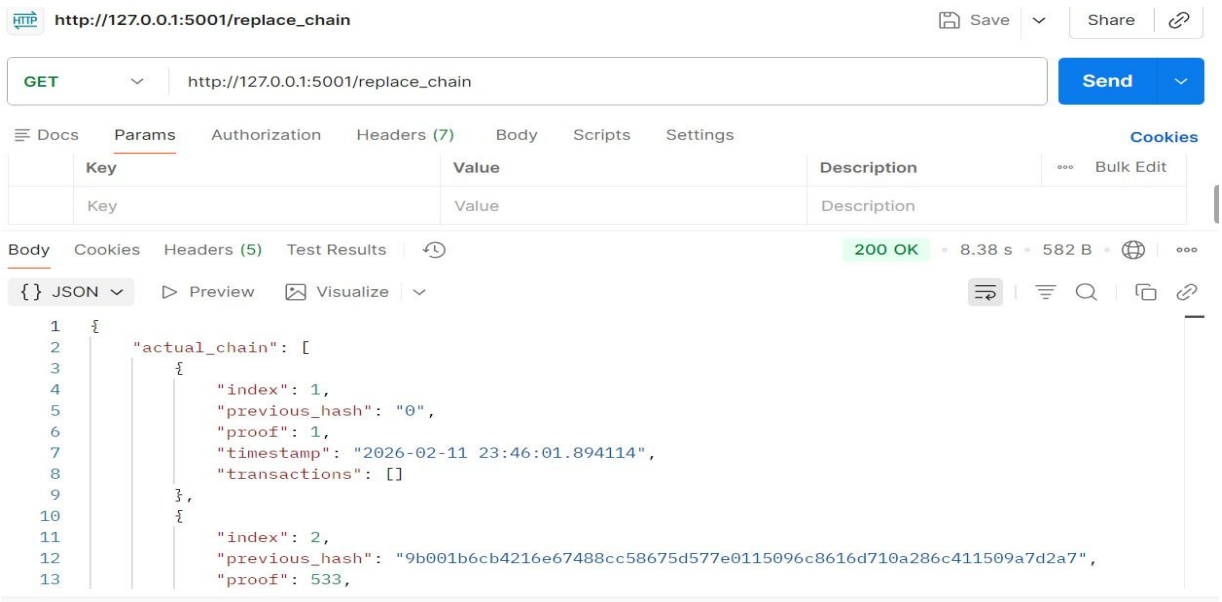
5)/connect_node

This POST request connects the current node with other peer nodes in the blockchain network.



6)/replace_chain

This GET request checks all connected peer nodes and replaces the current blockchain with the longest valid chain, ensuring network-wide consistency.



```
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "7f8423ce398f4f5eb2a8ca22514d2b9c"
                }
            ]
        }
    ],
    "message": "All good. The chain is the largest one."
}
```


Conclusion:

In this experiment, a basic cryptocurrency system was designed and implemented using Python by applying fundamental blockchain principles such as block generation, cryptographic hashing, Proof-of-Work mining, transaction processing, and decentralized networking. A Flask-based framework was used to enable communication between multiple nodes, with each node maintaining its own copy of the blockchain ledger.

The mining process required solving the Proof-of-Work challenge, after which new blocks were added to the chain and miners received rewards. The system also supported transaction validation and inclusion of verified transactions into newly mined blocks. Through this implementation, the experiment offered hands-on insight into the functioning of blockchain technology, demonstrating how consensus, decentralization, and mining together enable the secure operation of cryptocurrencies in distributed environments.