AIM:

Create a Blockchain using Python

Task to be performed:

1. Building a Blockchain
2. Create POW/POS/POB/POET etc.
3. Mining the Blockchain

Tools & Libraries used:

- Flask : pip install Flask
- Download Postman from https://www.postman.com/
- Python Libraries : datetime, jsonify, hashlib

CODE:

```
from hashlib import sha256
from time import time
from flask import Flask, jsonify, request

app = Flask(__name__)

class Block:
        def __init__(self, index, proof_no, prev_hash, data,
timestamp=None):
        self.index = index
        self.proof_no = proof_no
        self.prev_hash = prev_hash
        self.data = data
        self.timestamp = timestamp or time()

    @property
    def calculate_hash(self):
        block_of_string = "{}{}{}{}{}{}".format(
            self.index,
            self.proof_no,
            self.prev_hash,
            self.data,
            self.timestamp
        )
```

```
        return sha256(block_of_string.encode()).hexdigest()

class Blockchain:
    def __init__(self):
        self.chain = []
        self.current_data = []
        self.nodes = set()
        self.construct_genesis()

    def construct_genesis(self):
        self.construct_block(proof_no=0, prev_hash=0)

    def construct_block(self, proof_no, prev_hash):
        block = Block(
            index=len(self.chain),
            proof_no=proof_no,
            prev_hash=prev_hash,
            data=self.current_data
        )
        self.current_data = []
        self.chain.append(block)
        return block

    @staticmethod
    def proof_of_work(last_proof):
        proof_no = 0
          while Blockchain.verifying_proof(proof_no, last_proof)
is False:
            proof_no += 1
        return proof_no

    @staticmethod
    def verifying_proof(last_proof, proof):
        guess = f'{last_proof}{proof}'.encode()
        guess_hash = sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"

    @property
    def latest_block(self):
        return self.chain[-1]
```

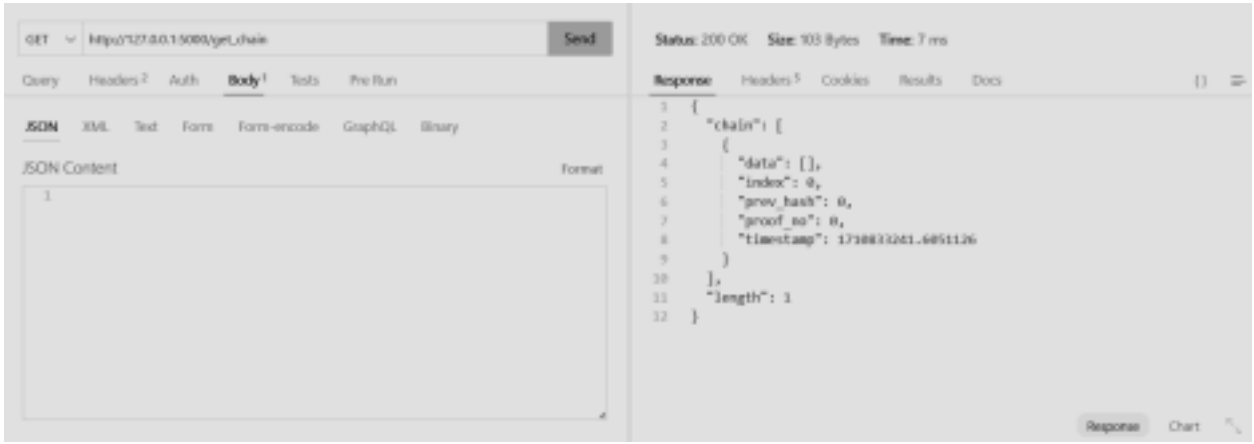
```
    def create_node(self, address):
        self.nodes.add(address)
        return True

    def new_transaction(self, sender, recipient,
        quantity): self.current_data.append({
            'sender': sender,
            'recipient': recipient,
            'quantity': quantity
        })
        return True

    def mine_block(self, miner_address):
        last_block = self.latest_block
        last_proof_no = last_block.proof_no
        proof_no = self.proof_of_work(last_proof_no)

        self.new_transaction(
            sender="0",
            recipient=miner_address,
            quantity=1
        )

        last_hash = last_block.calculate_hash
        block = self.construct_block(proof_no,
        last_hash) return vars(block)

    def verify_chain_integrity(self):
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i - 1]

               # Check if the previous hash in the current block
matches the hash of the previous block
                                    if current_block.prev_hash !=
previous_block.calculate_hash:
                return False

                      # Check if the hash of the current block is
correctly calculated
```

```
                                if current_block.calculate_hash !=
current_block.calculate_hash:
                return False

        return True

    def tamper_block_data(self, block_index, new_data): if
        block_index < 0 or block_index >= len(self.chain):
        return False

        block_to_tamper = self.chain[block_index]
        block_to_tamper.data = new_data
        return True

blockchain = Blockchain()

@app.route('/mine_block', methods=['GET'])
def mine_block():
    miner_address =
    request.args.get('miner_address') if not
    miner_address:
              return jsonify({'message': 'Missing miner_address
parameter'}), 400

    block = blockchain.mine_block(miner_address)
    response = {
        'message': 'New Block Mined!',
        'block': block
    }
    return jsonify(response), 200

@app.route('/add_node', methods=['POST'])
def add_node():
    node_address = request.json.get('node_address')
    if not node_address:
               return jsonify({'message': 'Missing node_address
parameter'}), 400

    blockchain.create_node(node_address)
    response = {
        'message': 'New node added',
        'total_nodes': list(blockchain.nodes)
```

```
    }
    return jsonify(response), 201

@app.route('/new_transaction', methods=['POST'])
def new_transaction():
    values = request.get_json()
    required_fields = ['sender', 'recipient', 'quantity'] if not
all(field in values for field in required_fields): return
jsonify({'message': 'Missing required fields'}), 400

                    blockchain.new_transaction(values['sender'],
values['recipient'], values['quantity'])
    response = {'message': 'Transaction added to
    block'} return jsonify(response), 201

@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {
        'chain': [vars(block) for block in
        blockchain.chain], 'length': len(blockchain.chain)
    }
    return jsonify(response), 200

@app.route('/verify_chain', methods=['GET'])
def verify_chain():
    is_valid = blockchain.verify_chain_integrity()
    if is_valid:
         return jsonify({'message': 'Chain integrity verified. No
tampering detected.'}), 200
    else:
         return jsonify({'message': 'Chain integrity verification
failed. Tampering detected!'}), 400

@app.route('/tamper_block_data',
methods=['POST']) def tamper_block_data():
    values = request.get_json()
    required_fields = ['block_index', 'new_data']
    if not all(field in values for field in required_fields):
return jsonify({'message': 'Missing required fields'}), 400
```

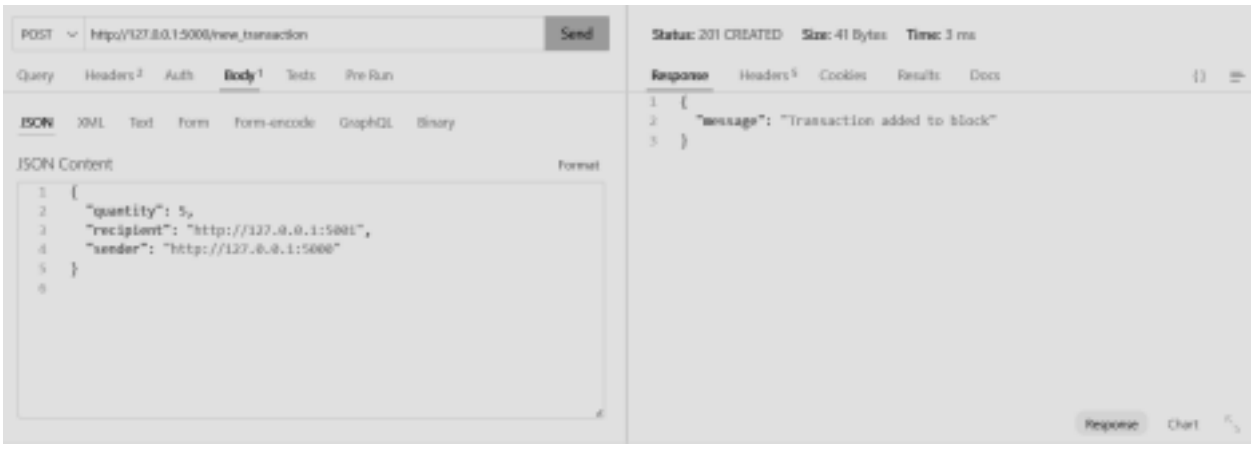
```
    block_index = values['block_index']
    new_data = values['new_data']

          tampered = blockchain.tamper_block_data(block_index,
new_data)
    if tampered:
         return jsonify({'message': f'Data of block {block_index}
tampered successfully.'}), 200
    else:
          return jsonify({'message': f'Failed to tamper data of
block {block_index}. Block index out of range.'}), 400

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

View the chain: **`http://127.0.0.1:5000/get_chain`**

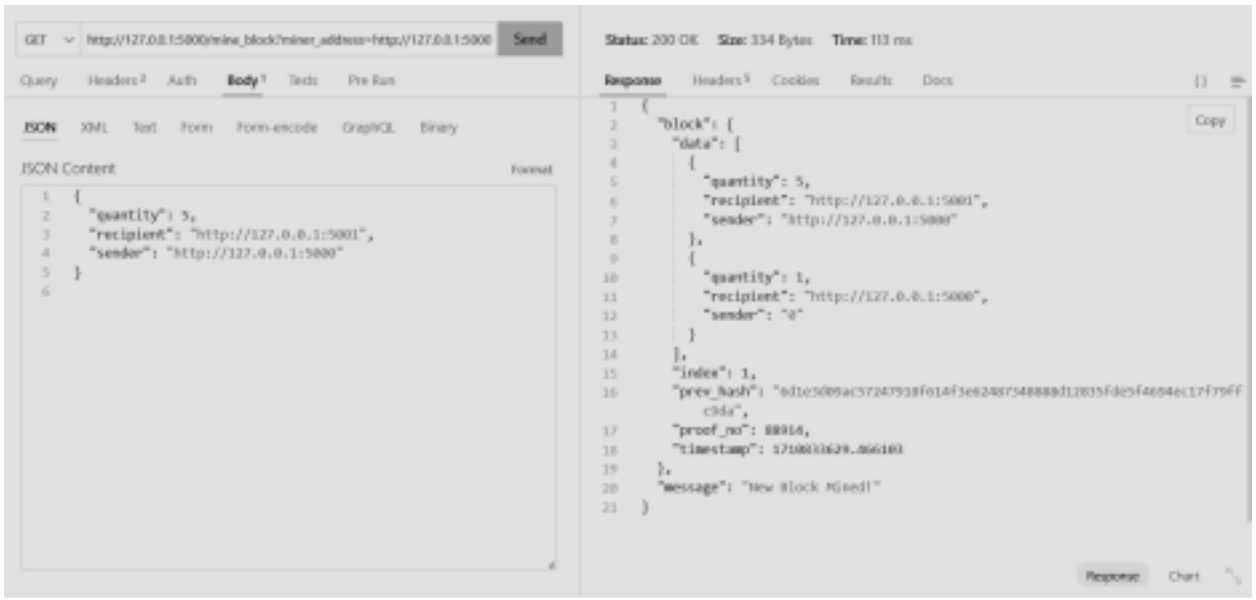


Add a New Node: **`http://127.0.0.1:5000/add_node`**

Adding a new transaction: **`http://127.0.0.1:5000/new_transaction`**

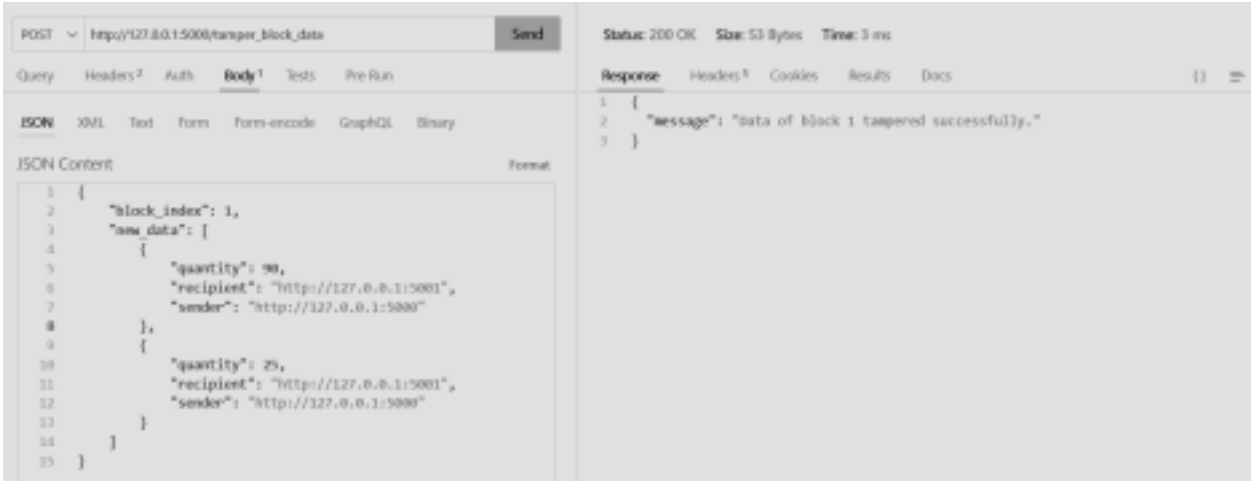


Mining the block:
`http://127.0.0.1:5000/mine_block?miner_address=http://127.0.0.1:5000`



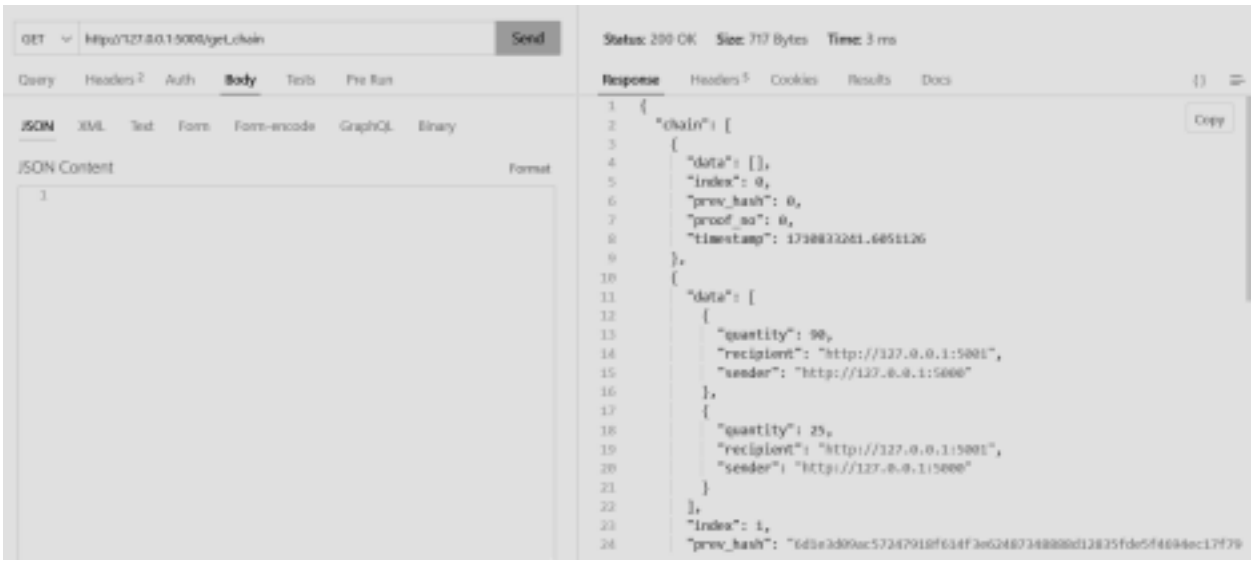
Verifying the chain: **`http://127.0.0.1:5000/verify_chain`**

Tampering the chain: **`http://127.0.0.1:5000/tamper_block_data`**



Verifying the chain after tampering: **`http://127.0.0.1:5000/verify_chain`**

Viewing the chain again: **`http://127.0.0.1:5000/get_chain`**



CONCLUSION:

We have successfully created a blockchain using python and also created functions to get the chain, add a transaction, mine a block and also to check the immutability.