Name : Devansh Wadhwani **Class: D20A** **Roll No: 62**

BLOCKCHAIN LAB

EXPERIMENT NO - 3

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

THEORY :

1. Blockchain Overview

Blockchain is a decentralized and distributed digital ledger technology used to securely store transaction records across multiple systems known as nodes. Unlike traditional systems, blockchain does not depend on a central authority, as each node in the network maintains an identical copy of the ledger.

Each block in the blockchain contains:
- A list of transactions
- A timestamp
- The hash of the previous block
- Its own cryptographic hash

Blocks in a blockchain are connected through cryptographic hash values, creating a continuous and secure chain structure. Each block depends on the hash of the previous one, so even a small change in stored data alters the hash and disrupts the chain. This makes any unauthorized modification immediately visible to the network. As a result, blockchain maintains **data integrity,** allows transparent verification of records, and ensures **immutability** by preventing changes once data is recorded.

A collection of verified transactions that represent the exchange of data or digital assets between participants in the blockchain network.

The exact date and time of block creation, which helps maintain the proper sequence and chronological order of all blocks in the blockchain.

A reference hash of the preceding block, which securely links the current block to the previous one and maintains continuity in the chain.

A unique cryptographic hash generated for the current block, used to identify the block and ensure its data has not been altered.

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)

Collecting pending transactions:
Before a new block is created, all unconfirmed transactions waiting in the network are gathered. These transactions are verified to ensure they are valid and then prepared to be included in the next block.

Creating a block header:
After transactions are selected, a block header is formed. It contains important metadata such as the previous block's hash, timestamp, nonce, and Merkle root, which together define the identity and structure of the block.

Solving Proof-of-Work (PoW):
In this step, miners solve a complex mathematical puzzle by repeatedly changing a nonce value to generate a hash that satisfies the required difficulty level. Successfully solving this **Proof-of-Work** confirms the block and allows it to be added to the blockchain.

Mining is the process of adding new blocks to the blockchain by collecting pending transactions and solving a complex **Proof-of-Work problem**. Miners compete to find a **valid hash**, and the first one to succeed adds the block to the chain and receives a reward.

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

Operates independently:
Each node functions on its own, validates data independently, and does not rely on a central authority for operation.

Maintains its own copy of the blockchain:
Every node stores and updates a complete copy of the blockchain, ensuring data security and fault tolerance.

Communicates with peer nodes:
Nodes exchange newly mined blocks and updates with other nodes to maintain consistency across the network.

Each node in a blockchain network works independently without relying on a central authority. It stores and maintains its own complete copy of the blockchain to ensure **reliability and fault tolerance**. Nodes continuously communicate with other nodes to exchange newly mined blocks, which helps keep the entire network **synchronized and consistent.**

4. Consensus Mechanism

Key Points of Consensus

- Consensus is a mechanism used to achieve agreement among all nodes in a blockchain network.
- It ensures that every node maintains the same and correct version of the blockchain.
- Consensus eliminates the need for a central authority to validate transactions.
- It helps prevent double spending and fraudulent transactions.
- Different blockchain systems use different consensus algorithms such as Proof-of-Work and Longest Chain Rule.
- Consensus maintains trust, security, and synchronization in a decentralized environment.

A consensus mechanism in blockchain is used to ensure that all nodes in the network agree on the same valid version of the blockchain. Since there is no central authority, consensus allows nodes to independently **verify transactions** and blocks before accepting them. It helps prevent fraudulent activities such as double spending and ensures that only valid data is added to the chain. By resolving conflicts between different versions of the blockchain, consensus keeps all nodes **synchronized and maintains trust, security, and consistency** across the decentralized network.

5. Transactions and Mining Reward

Transactions in the cryptocurrency system include:

● Sender address

● Receiver address

● Transaction amount

Sender address: Identifies the account from which the cryptocurrency is sent.

Receiver address: Identifies the account that receives the cryptocurrency.

Transaction amount: Specifies the quantity of cryptocurrency being transferred.

6. Chain Replacement

The chain replacement process ensures network consistency.

When the /replace_chain endpoint is triggered:

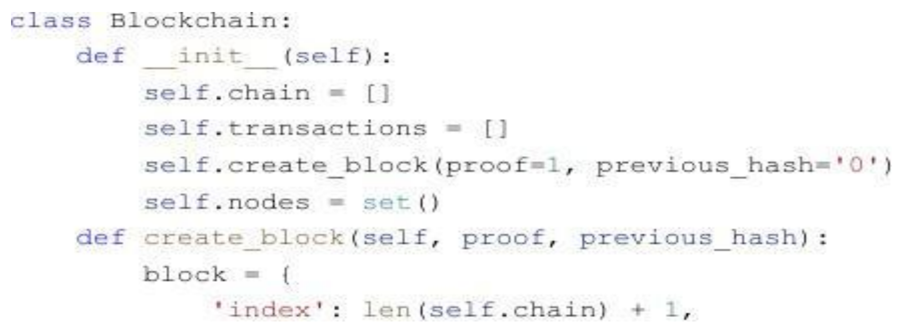
● A node requests blockchain data from its peers

● Validates the received chains

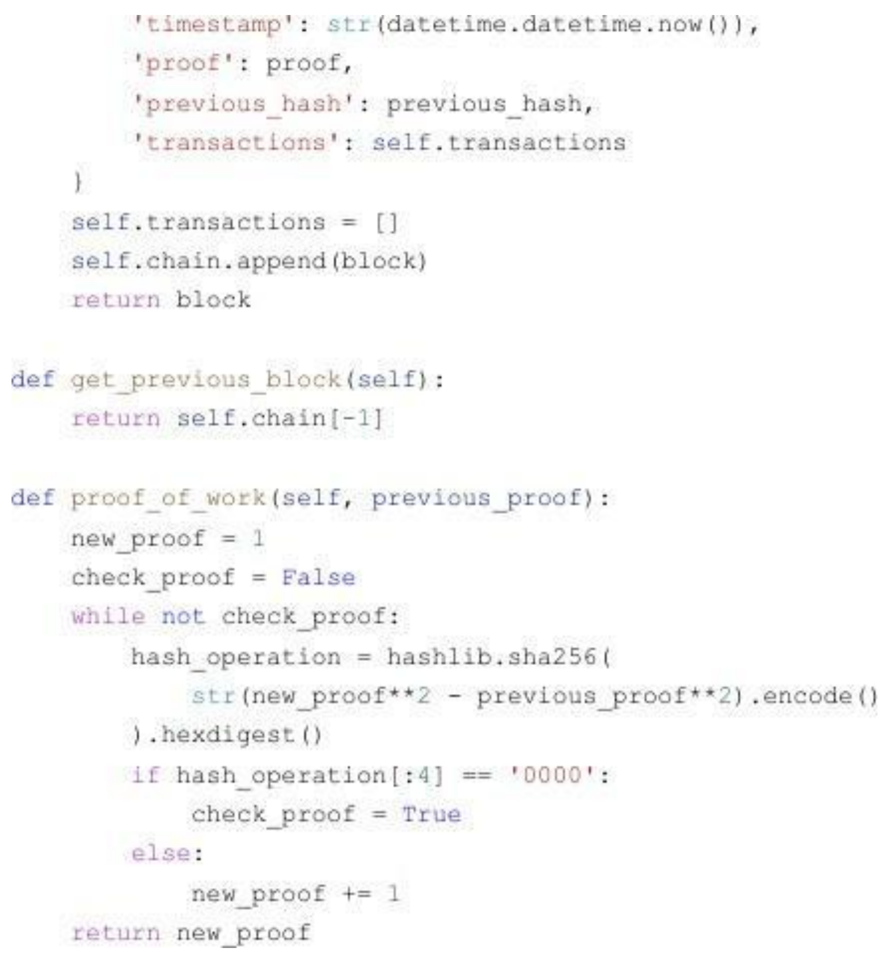
● Replaces its own chain if a longer and valid chain is found

This mechanism helps resolve conflicts and keeps all nodes synchronized.

CODE :



```python
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,
```

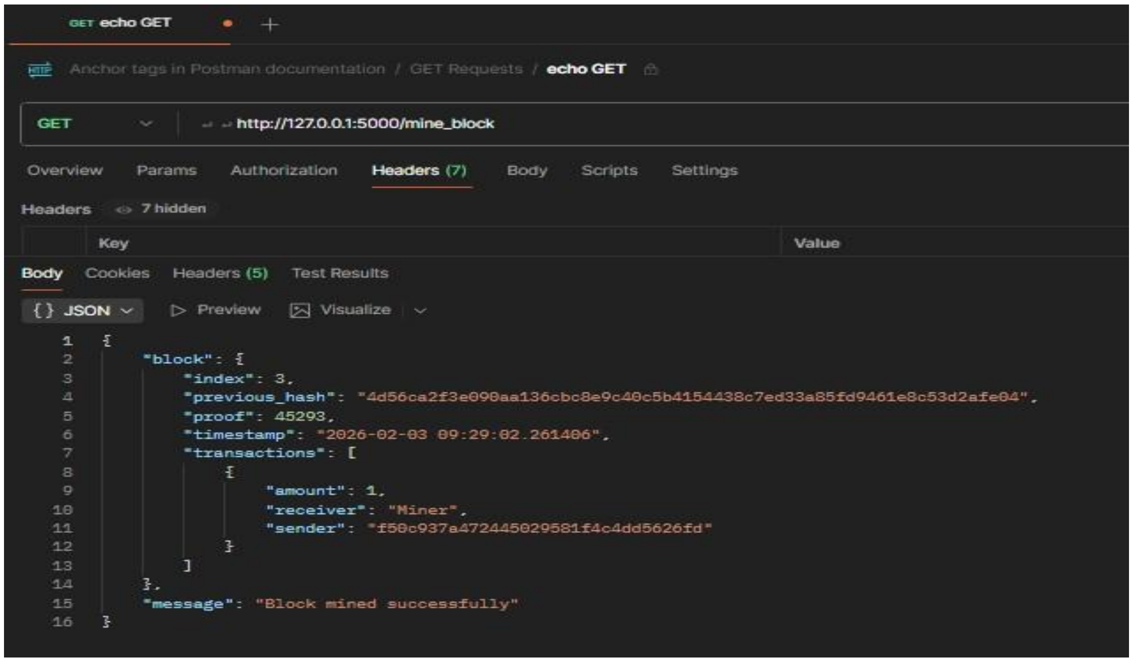


```python
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash,
            'transactions': self.transactions
        }
        self.transactions = []
        self.chain.append(block)
        return block

    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()
            ).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
        if longest_chain:
            self.chain = longest_chain
            return True
        return False

app = Flask(__name__)

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

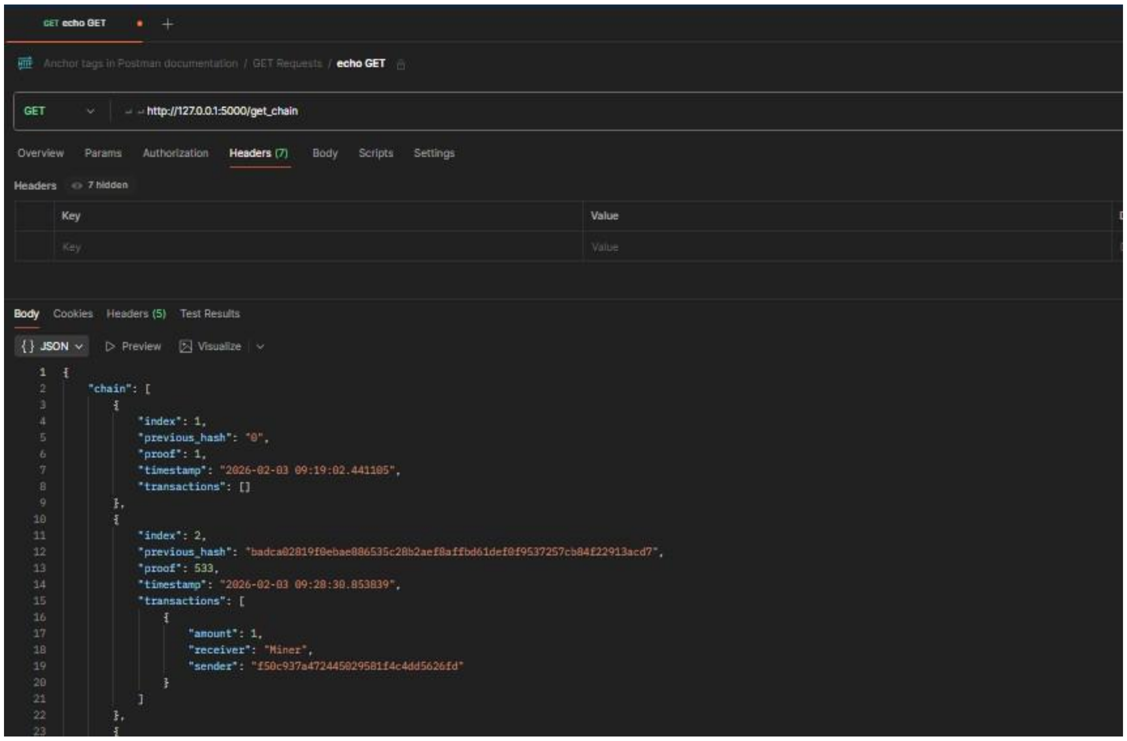
```python
blockchain = Blockchain()

@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
```

```python
@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': 'The Blockchain is not valid.'}
    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['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_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)
```

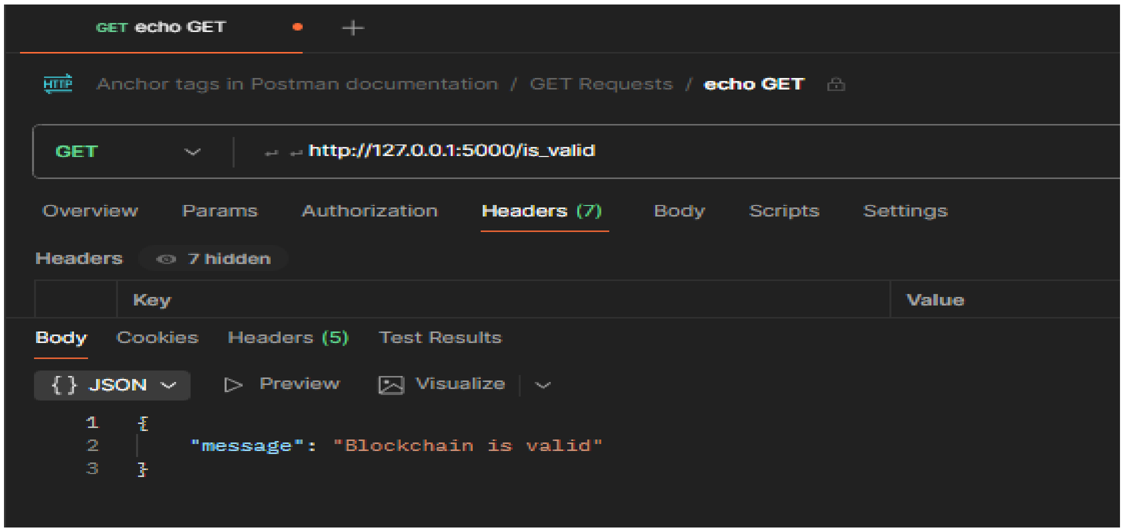
OUTPUT :


GET echo GET
Anchor tags in Postman documentation / GET Requests / echo GET
GET
http://127.0.0.1:5000/mine_block
Overview Params Authorization Headers (7) Body Scripts Settings
Headers 7 hidden
Key Value
Body Cookies Headers (5) Test Results
{ } JSON Preview Visualize

```
{
    "block": {
        "index": 3,
        "previous_hash": "4d56ca2f3e090aa136cbc8e9c40c5b4154438c7ed33a85fd9461e8c53d2afe04",
        "proof": 45293,
        "timestamp": "2026-02-03 09:29:02.261406",
        "transactions": [
            {
                "amount": 1,
                "receiver": "Miner",
                "sender": "f50c937a472445029581f4c4dd5626fd"
            }
        ]
    },
    "message": "Block mined successfully"
}
```


GET echo GET
Anchor tags in Postman documentation / GET Requests / echo GET
GET
http://127.0.0.1:5000/get_chain
Overview Params Authorization Headers (7) Body Scripts Settings
Headers 7 hidden
Key Value
Key Value
Body Cookies Headers (5) Test Results
{ } JSON Preview Visualize

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-03 09:19:02.441105",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "badca02819f0ebae886535c28b2aef8affbd61def0f9537257cb84f22913acd7",
            "proof": 533,
            "timestamp": "2026-02-03 09:28:30.853839",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Miner",
                    "sender": "f50c937a472445029581f4c4dd5626fd"
                }
            ]
        },
        {
```



http://127.0.0.1:5000/is_valid

```
{
    "message": "Blockchain is valid"
}
```

Conclusion:

This implementation demonstrates how blockchain technology works by creating a decentralized ledger using Python. The code shows how blocks are securely linked using cryptographic hashes, how transactions are recorded, and how **Proof-of-Work** is used to mine new blocks. The use of consensus through chain validation and replacement ensures data integrity and synchronization across nodes. Overall, the system highlights the core principles of blockchain such as **decentralization, transparency, security, and immutability.**