

## Blockchain Exp - 3

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

### Theory:

#### 1. Blockchain Overview

Blockchain is a **distributed and decentralized ledger** that stores information in a series of linked blocks.

Each block contains:

- Transaction data
- Timestamp
- Previous block's hash
- Its own unique hash (digital fingerprint)

Once data is recorded in a blockchain, it becomes **immutable** because altering one block would require recalculating all subsequent blocks.

#### 2. Mining

Mining is the process of:

1. Collecting pending transactions into a block.
2. Performing a computational puzzle (Proof-of-Work) to find a valid hash.
3. Adding the new block to the blockchain.  
Broadcasting it to all connected peers.

Miners are rewarded with cryptocurrency for successfully mining a block.

#### 3. Multi-Node Blockchain Network

In this lab, we simulate **three independent blockchain nodes** (5001, 5002, 5003).

Each node:

- Runs on a separate port.
- Maintains its own copy of the blockchain.
- Can connect with peers to share and validate blocks.

#### 4. Consensus Mechanism

We use the **Longest Chain Rule**:

- If multiple versions of the chain exist, the **longest valid chain** is chosen.
- This ensures all nodes agree on a single transaction history.

#### 5. Transactions & Mining Reward

Each transaction has:

- Sender
- Receiver
- Amount

When mining a block:

- Pending transactions are added to the block.
- A **reward transaction** is added automatically to pay the miner.

## 6. Chain Replacement

When /replace\_chain is called:

1. Node requests chains from peers.
2. If it finds a longer and valid chain, it replaces its own.
3. This keeps the blockchain consistent across all nodes.

## Tools & Libraries Used

- **Python 3.x**
- **Flask** – Web framework for API endpoints  
pip install Flask
- **Requests** – For HTTP communication between nodes  
pip install requests==2.18.4
- **Postman** – For testing API requests
- Python Standard Libraries:
  - datetime
  - jsonify
  - hashlib
  - uuid4
  - urlparse
  - request

**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

from urllib.parse import urlparse

# Generate a unique id that is in hex

# 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 transaction 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 transaction to the list of transactions
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                                         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 function
                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
            one                                                     # 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 = 500*)
```



**Output :****1. Connecting nodes: Node 1 -> 2,3**

The screenshot shows the Postman interface for a POST request to `http://127.0.0.1:5001/connect_node`. The request body is a JSON object:

```
1 {
2   "nodes": [
3     "http://127.0.0.1:5002",
4     "http://127.0.0.1:5003"
5   ]
6 }
```

The response is a 201 CREATED status with a JSON body:

```
1 {
2   "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following
3     nodes:",
4   "total_nodes": [
5     "127.0.0.1:5003",
6     "127.0.0.1:5002"
7   ]
8 }
```

**Node 2 -> 1,3**

The screenshot shows the Postman interface for a POST request to `http://127.0.0.1:5002/connect_node`. The request body is a JSON object:

```
1 {
2   "nodes": [
3     "http://127.0.0.1:5001",
4     "http://127.0.0.1:5003"
5   ]
6 }
```

The response is a 201 CREATED status with a JSON body:

```
1 {
2   "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following
3     nodes:",
4   "total_nodes": [
5     "127.0.0.1:5003",
6     "127.0.0.1:5001"
7   ]
8 }
```

## Node 3 -> 1,2

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:5003/connect_node`
- Method:** `POST`
- Body (JSON):**

```
1 {
2   "nodes": [
3     "http://127.0.0.1:5001",
4     "http://127.0.0.1:5002"
5   ]
6 }
```
- Response:** `201 CREATED` (5 ms, 331 B)
- Response Body (JSON):**

```
1 {
2   "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following
3             nodes:",
4   "total_nodes": [
5     "127.0.0.1:5001",
6     "127.0.0.1:5002"
7   ]
8 }
```

## 2. Adding transaction from node 1

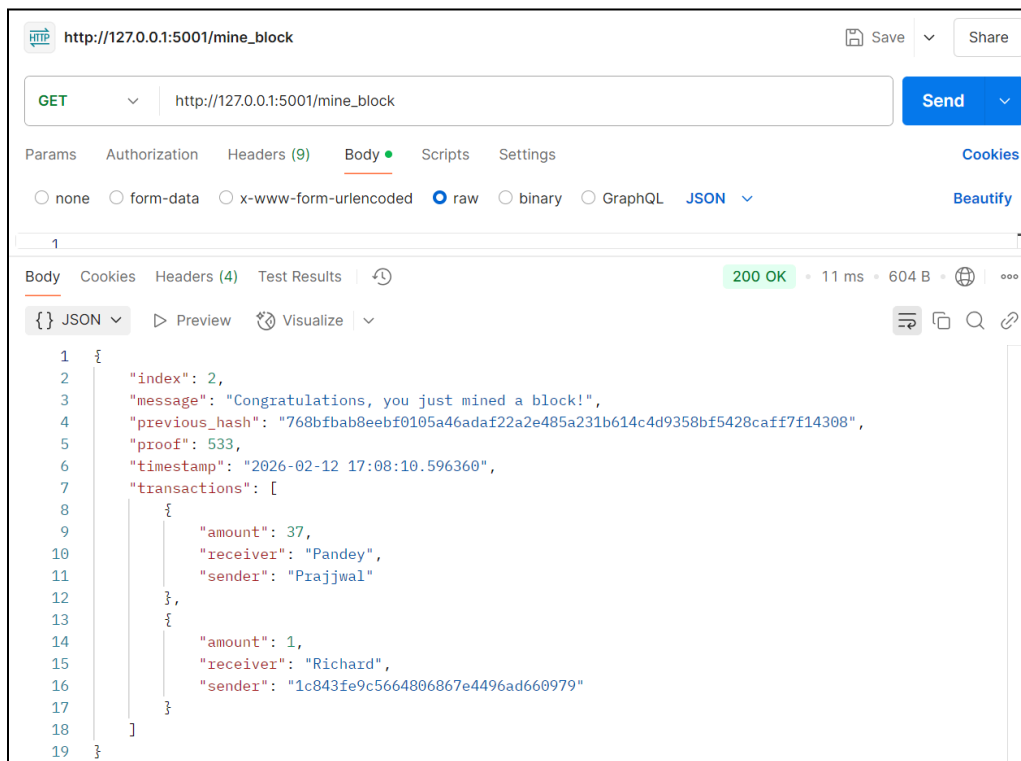
The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:5001/add_transaction`
- Method:** `POST`
- Body (JSON):**

```
1 {
2   "sender": "Prajwal",
3   "receiver": "Pandey",
4   "amount": 37
5 }
6
```
- Response:** `201 CREATED` (6 ms, 213 B)
- Response Body (JSON):**

```
1 {
2   "message": "This transaction will be added to Block 2"
3 }
```

### 3. Mining a Block (Only on 5001)



HTTP [http://127.0.0.1:5001/mine\\_block](http://127.0.0.1:5001/mine_block) Save Share

GET [http://127.0.0.1:5001/mine\\_block](http://127.0.0.1:5001/mine_block) Send

Params Authorization Headers (9) Body Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

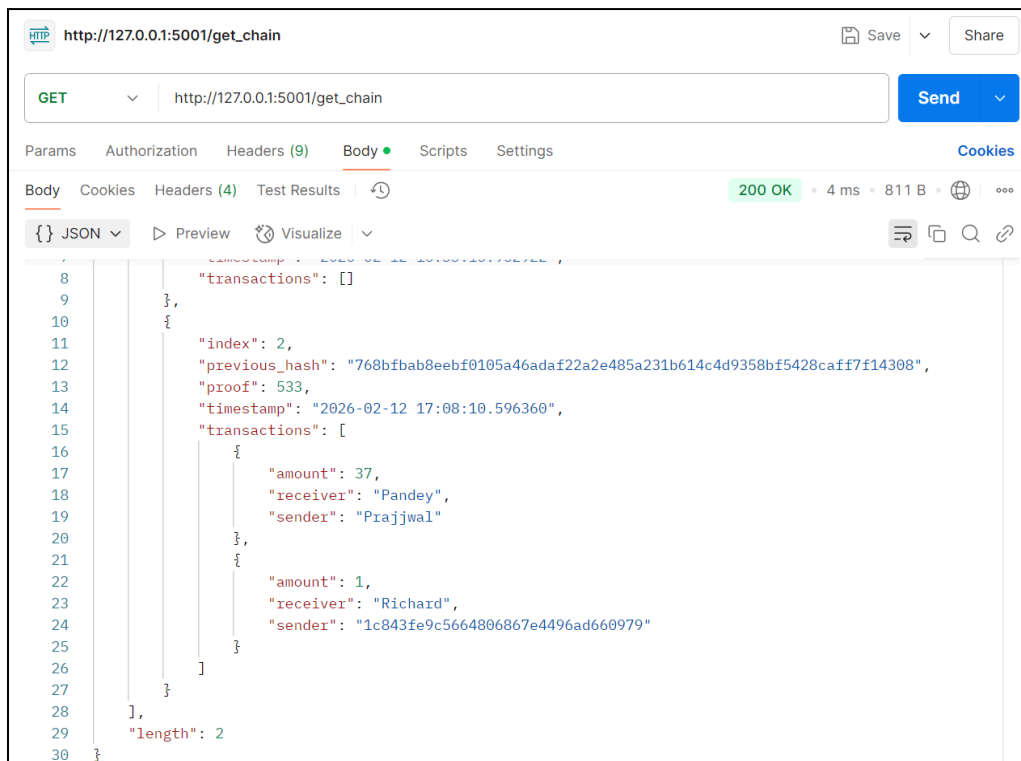
1

Body Cookies Headers (4) Test Results 200 OK • 11 ms • 604 B

{ } JSON Preview Visualize

```
1 {
2   "index": 2,
3   "message": "Congratulations, you just mined a block!",
4   "previous_hash": "768bfbab8eebf0105a46adaf22a2e485a231b614c4d9358bf5428caff7f14308",
5   "proof": 533,
6   "timestamp": "2026-02-12 17:08:10.596360",
7   "transactions": [
8     {
9       "amount": 37,
10      "receiver": "Pandey",
11      "sender": "Prajjwal"
12    },
13    {
14      "amount": 1,
15      "receiver": "Richard",
16      "sender": "1c843fe9c5664806867e4496ad660979"
17    }
18  ]
19 }
```

### 4. Checking chain length difference: Node 1



HTTP [http://127.0.0.1:5001/get\\_chain](http://127.0.0.1:5001/get_chain) Save Share

GET [http://127.0.0.1:5001/get\\_chain](http://127.0.0.1:5001/get_chain) Send

Params Authorization Headers (9) Body Scripts Settings Cookies Beautify

Body Cookies Headers (4) Test Results 200 OK • 4 ms • 811 B

{ } JSON Preview Visualize

```
8   "timestamp": "2026-02-12 17:08:10.596360",
9   "transactions": []
10  },
11  {
12    "index": 2,
13    "previous_hash": "768bfbab8eebf0105a46adaf22a2e485a231b614c4d9358bf5428caff7f14308",
14    "proof": 533,
15    "timestamp": "2026-02-12 17:08:10.596360",
16    "transactions": [
17      {
18        "amount": 37,
19        "receiver": "Pandey",
20        "sender": "Prajjwal"
21      },
22      {
23        "amount": 1,
24        "receiver": "Richard",
25        "sender": "1c843fe9c5664806867e4496ad660979"
26      }
27    ]
28  },
29  "length": 2
30 }
```

## Node 2

HTTP **http://127.0.0.1:5002/get\_chain** Save Share

GET **http://127.0.0.1:5002/get\_chain** Send

Params Authorization Headers (9) **Body** Scripts Settings Cookies

Body Cookies Headers (4) Test Results 200 OK • 4 ms • 339 B •

JSON Preview Visualize

```
1 {
2   "chain": [
3     {
4       "index": 1,
5       "previous_hash": "0",
6       "proof": 1,
7       "timestamp": "2026-02-12 16:35:01.448893",
8       "transactions": []
9     }
10  ],
11  "length": 1
12 }
```

## Node 3

HTTP **http://127.0.0.1:5003/get\_chain** Save Share

GET **http://127.0.0.1:5003/get\_chain** Send

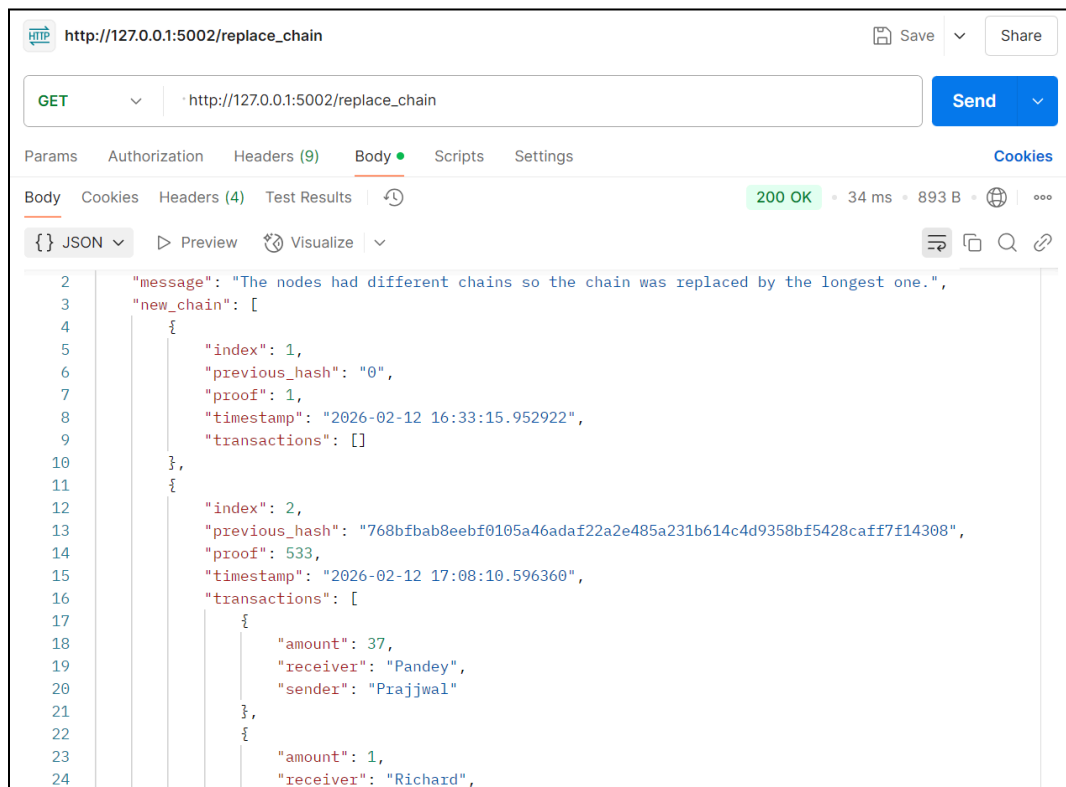
Params Authorization Headers (9) **Body** Scripts Settings Cookies

Body Cookies Headers (4) Test Results 200 OK • 5 ms • 339 B •

JSON Preview Visualize

```
1 {
2   "chain": [
3     {
4       "index": 1,
5       "previous_hash": "0",
6       "proof": 1,
7       "timestamp": "2026-02-12 16:35:28.083070",
8       "transactions": []
9     }
10  ],
11  "length": 1
12 }
```

## 5. Replace Chain (Consensus Mechanism) : Node 2



HTTP [http://127.0.0.1:5002/replace\\_chain](http://127.0.0.1:5002/replace_chain) Save Share

GET [http://127.0.0.1:5002/replace\\_chain](http://127.0.0.1:5002/replace_chain) Send

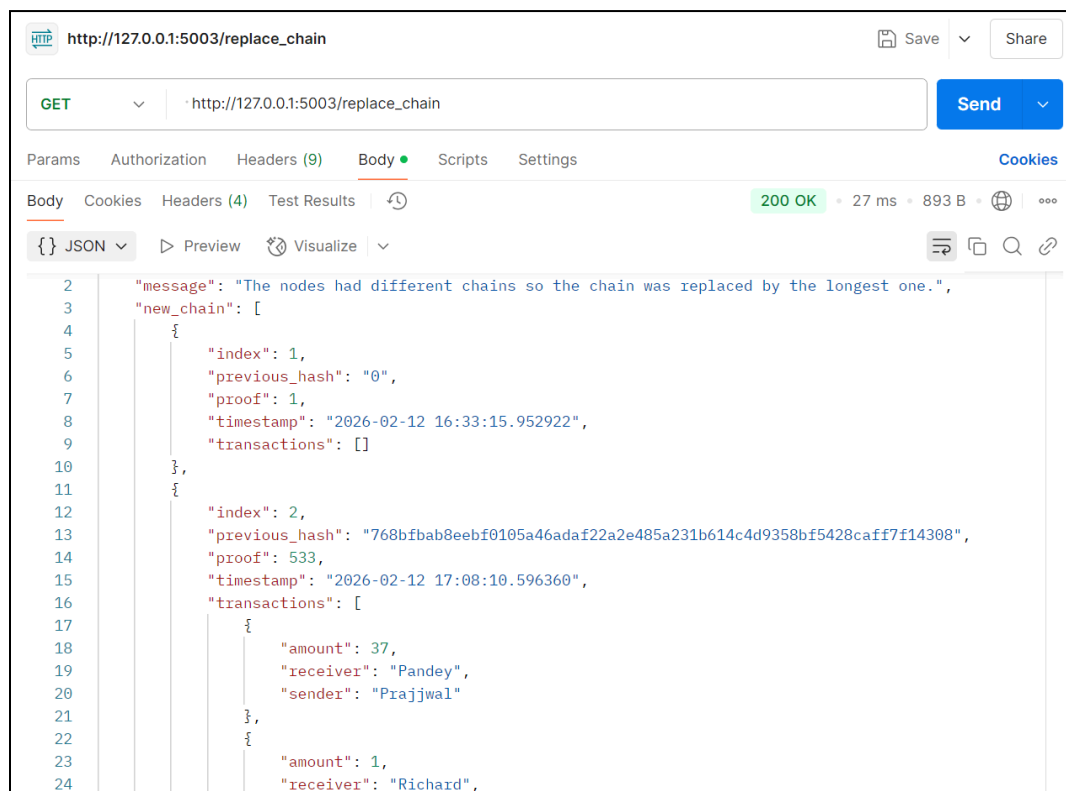
Params Authorization Headers (9) **Body** Scripts Settings Cookies

Body Cookies Headers (4) Test Results 200 OK • 34 ms • 893 B

{ } JSON Preview Visualize

```
2  "message": "The nodes had different chains so the chain was replaced by the longest one.",
3  "new_chain": [
4    {
5      "index": 1,
6      "previous_hash": "0",
7      "proof": 1,
8      "timestamp": "2026-02-12 16:33:15.952922",
9      "transactions": []
10   },
11   {
12     "index": 2,
13     "previous_hash": "768bfbab8eebf0105a46adaf22a2e485a231b614c4d9358bf5428caff7f14308",
14     "proof": 533,
15     "timestamp": "2026-02-12 17:08:10.596360",
16     "transactions": [
17       {
18         "amount": 37,
19         "receiver": "Pandey",
20         "sender": "Prajjwal"
21       },
22       {
23         "amount": 1,
24         "receiver": "Richard",
```

## Node 3



HTTP [http://127.0.0.1:5003/replace\\_chain](http://127.0.0.1:5003/replace_chain) Save Share

GET [http://127.0.0.1:5003/replace\\_chain](http://127.0.0.1:5003/replace_chain) Send

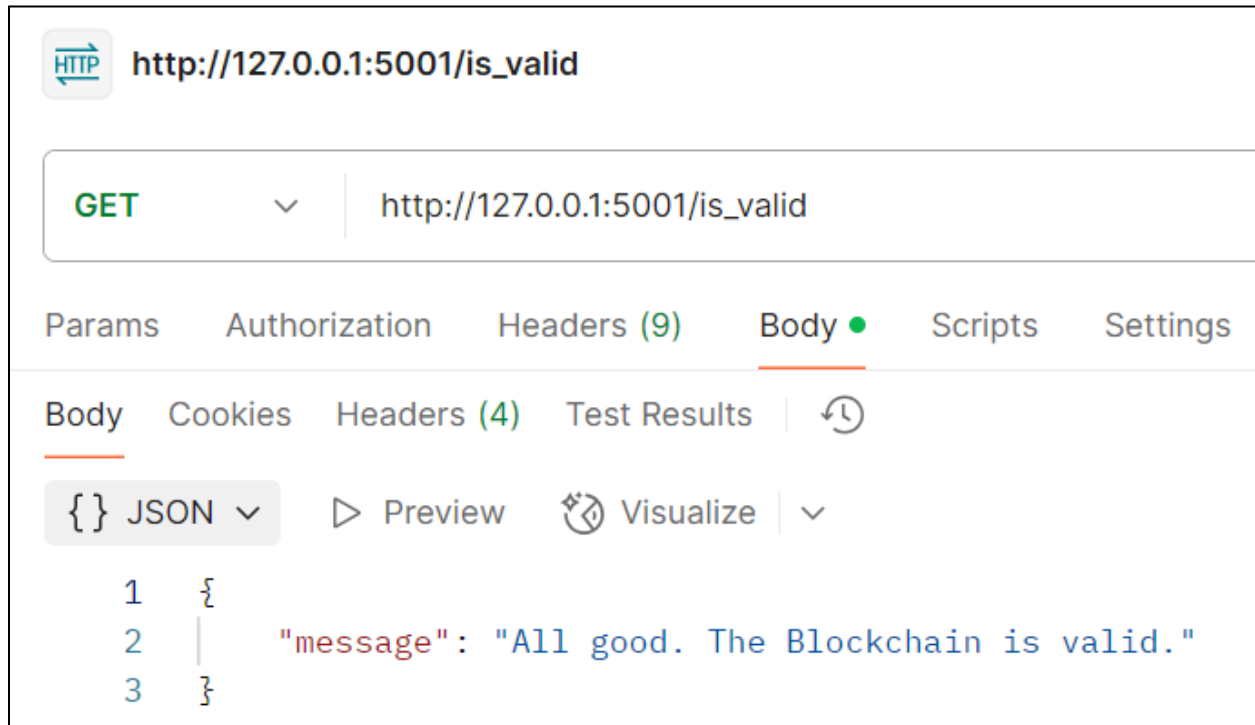
Params Authorization Headers (9) **Body** Scripts Settings Cookies

Body Cookies Headers (4) Test Results 200 OK • 27 ms • 893 B

{ } JSON Preview Visualize

```
2  "message": "The nodes had different chains so the chain was replaced by the longest one.",
3  "new_chain": [
4    {
5      "index": 1,
6      "previous_hash": "0",
7      "proof": 1,
8      "timestamp": "2026-02-12 16:33:15.952922",
9      "transactions": []
10   },
11   {
12     "index": 2,
13     "previous_hash": "768bfbab8eebf0105a46adaf22a2e485a231b614c4d9358bf5428caff7f14308",
14     "proof": 533,
15     "timestamp": "2026-02-12 17:08:10.596360",
16     "transactions": [
17       {
18         "amount": 37,
19         "receiver": "Pandey",
20         "sender": "Prajjwal"
21       },
22       {
23         "amount": 1,
24         "receiver": "Richard",
```

## 6. Final Validation



HTTP **http://127.0.0.1:5001/is\_valid**

**GET** **http://127.0.0.1:5001/is\_valid**

Params Authorization Headers (9) **Body** Scripts Settings


Body Cookies Headers (4) Test Results

{ } JSON Preview Visualize

```

1  {
2  |   "message": "All good. The Blockchain is valid."
3  }
```

## 7. Terminal Activity



```

(venv) PS C:\Users\prajj\Downloads\Lab_3_Create a Cryptocurrency-20260211T182232Z-1-001\Lab_3_Create a Cryptocurrency> python hadcoin_node_5001.py
>>
File "C:\Users\prajj\Downloads\Lab_3_Create a Cryptocurrency-20260211T182232Z-1-001\Lab_3_Create a Cryptocurrency\hadcoin_node_5001.py", line 164, in connect_node
    nodes = json.get('nodes')
AttributeError: 'NoneType' object has no attribute 'get'
# Get the nodes from json file
127.0.0.1 - - [12/Feb/2026 16:50:52] "POST /connect_node HTTP/1.1" 500 -
127.0.0.1 - - [12/Feb/2026 16:54:03] "POST /connect_node HTTP/1.1" 201 -
127.0.0.1 - - [12/Feb/2026 17:06:56] "POST /add_transaction HTTP/1.1" 201 -
127.0.0.1 - - [12/Feb/2026 17:07:55] "POST /mine_block HTTP/1.1" 405 -
127.0.0.1 - - [12/Feb/2026 17:08:10] "GET /mine_block HTTP/1.1" 200 -
127.0.0.1 - - [12/Feb/2026 17:11:22] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [12/Feb/2026 17:15:11] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [12/Feb/2026 17:16:11] "GET /get_chain HTTP/1.1" 200 -
127.0.0.1 - - [12/Feb/2026 17:17:08] "GET /is_valid HTTP/1.1" 200 -

```

**Conclusion:** In this experiment, we successfully created a basic cryptocurrency using Python and Flask and implemented mining in a multi-node blockchain network. We demonstrated how transactions are added, stored temporarily, and included in a block during the mining process using a Proof-of-Work mechanism. By running three separate nodes, we simulated a peer-to-peer network and applied the Longest Chain Rule to maintain consensus across all nodes. The experiment helped us understand blockchain structure, decentralized networking, mining rewards, and chain synchronization in a practical manner.