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

Guidelines

Lab Objectives: To implement public and private Blockchain.

Lab Outcomes (LO): Demonstrate the concept of Blockchain in real-world Applications (LO4)

Task to be performed :

1. Download the code from folder, Lab_3
2. Install requests in the virtual environment created in the Lab 2. (Follow the instructions)
3. Run the files - hadcoin_node_5001.py, hadcoin_node_5002.py, hadcoin_node_5003.py in 3

different terminals.

4. Open Postman, from each node - invoke connect_node() and pass the peers as POST requests.
5. Perform the following functions
   - ■ Add Transactions - invoke add_transactions() as a POST request.
   - ■ mining - mine_block(),
   - ■ fetch the chain - get_chain(),
   - ■ replace the longest chain - replace_chain()
6. Modify the code such that transactions are removed after they are added to the block.

Tools & Libraries used :

- ● Install Flask : pip install Flask
- ● Download Postman from https://www.postman.com/
- ● Python Libraries : datetime, jsonify, hashlib, uuid4, urlparse, request
- ● Install requests : pip install requests==2.18.4

Instructions : (Prepare for viva for the following topics)

1. Challenges in P2P networks
2. How transactions are performed on the network?
3. Explain the role of mempools
4. Write briefly about the libraries and the tools used during implementation.

Outcome :

1. Understood the challenges in P2P networks, how transactions are performed and how a miner mines a block to be added in a blockchain.
2. Implemented a Cryptocurrency in Python using Flask, Postman and Python libraries such as datetime, jsonify, hashlib, uuid4, urlparse, request.
3. Successfully mined the blocks among a P2P network with 3 nodes.
4. Performed transactions via the network.
5. Successfully updated the block across the network
6. Prepare a document with Aim, Tasks performed, Program, Output and Conclusion.
7. Submit the hardcopy by the 2nd week of August 2023
   (As per the instructions, submit a hard copy of the
   same).

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

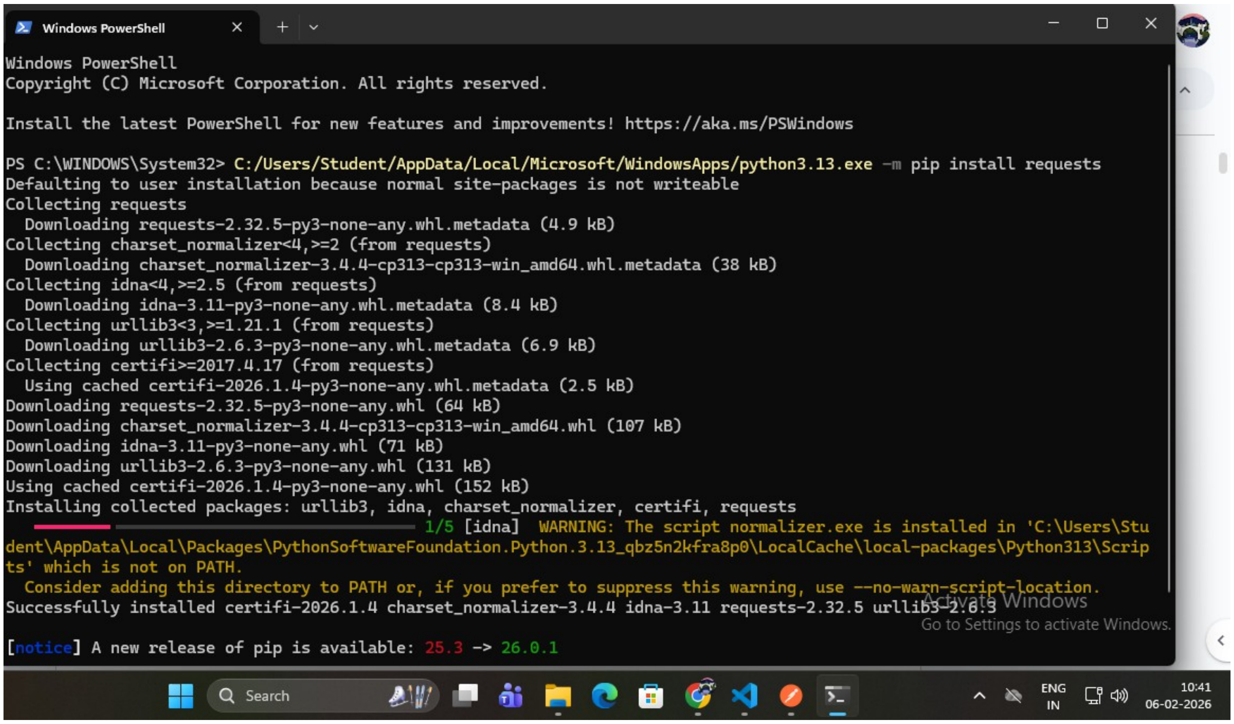
  ```
  pip install Flask
  ```
- **Requests** – For HTTP communication between nodes

  ```
  pip install requests==2.18.4
  ```
- **Postman** – For testing API requests
- Python Standard Libraries:
  - `datetime`
  - `jsonify`

- `hashlib`
- `uuid4`
- `urlparse`
- `request`

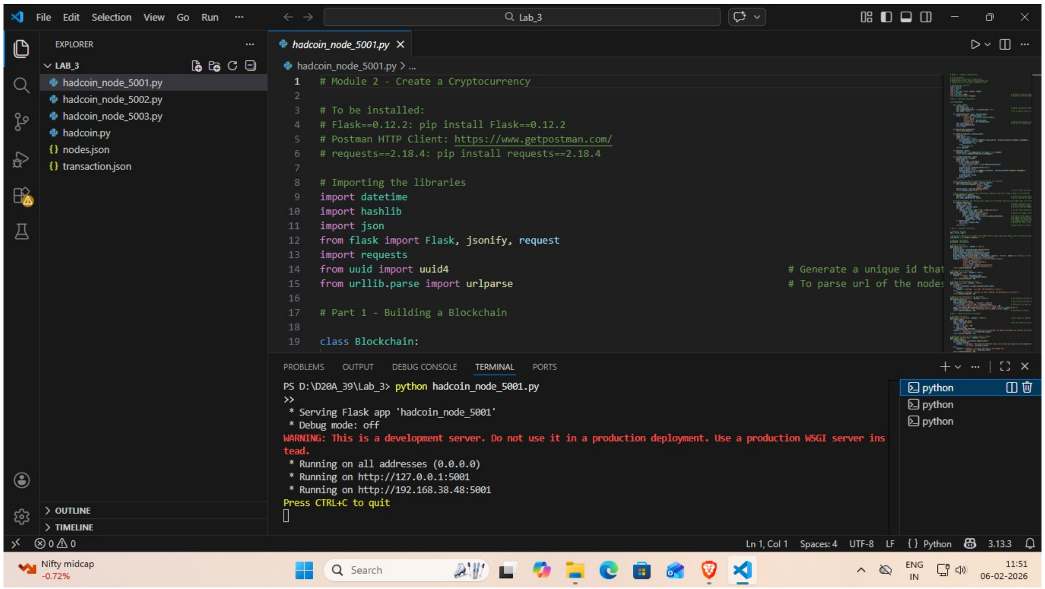
Procedure and Output:

1. Download `hadcoin_node_5001.py`, `hadcoin_node_5002.py`, `hadcoin_node_5003.py`.
2. Install required packages.



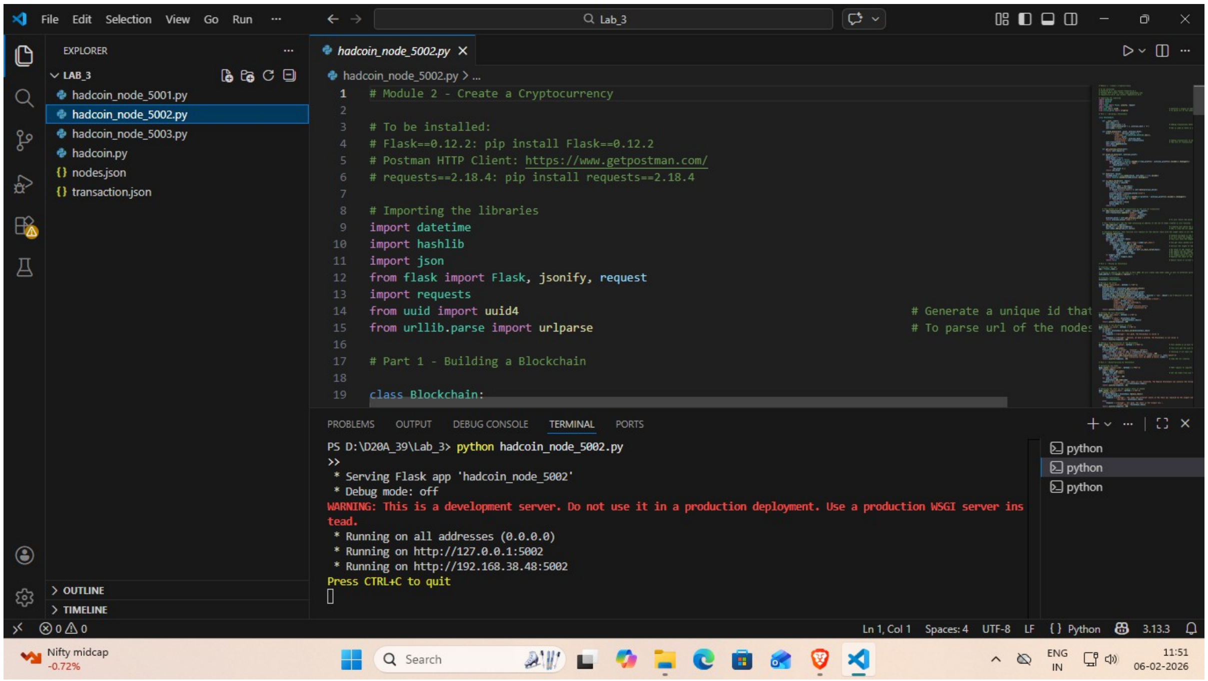
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\System32> C:/Users/Student/AppData/Local/Microsoft/WindowsApps/python3.13.exe -m pip install requests
Defaulting to user installation because normal site-packages is not writeable
Collecting requests
  Downloading requests-2.32.5-py3-none-any.whl.metadata (4.9 kB)
Collecting charset_normalizer<4,>=2 (from requests)
  Downloading charset_normalizer-3.4.4-cp313-cp313-win_amd64.whl.metadata (38 kB)
Collecting idna<4,>=2.5 (from requests)
  Downloading idna-3.11-py3-none-any.whl.metadata (8.4 kB)
Collecting urllib3<3,>=1.21.1 (from requests)
  Downloading urllib3-2.6.3-py3-none-any.whl.metadata (6.9 kB)
Collecting certifi>=2017.4.17 (from requests)
  Using cached certifi-2026.1.4-py3-none-any.whl.metadata (2.5 kB)
Downloading requests-2.32.5-py3-none-any.whl (64 kB)
Downloading charset_normalizer-3.4.4-cp313-cp313-win_amd64.whl (107 kB)
Downloading idna-3.11-py3-none-any.whl (71 kB)
Downloading urllib3-2.6.3-py3-none-any.whl (131 kB)
Using cached certifi-2026.1.4-py3-none-any.whl (152 kB)
Installing collected packages: urllib3, idna, charset_normalizer, certifi, requests
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1/5 [idna]  WARNING: The script normalizer.exe is installed in 'C:\Users\Stu
dent\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-packages\Python313\Scrip
ts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed certifi-2026.1.4 charset_normalizer-3.4.4 idna-3.11 requests-2.32.5 urllib3-2.6.3

[notice] A new release of pip is available: 25.3 -> 26.0.1
```

3. Run each node in separate terminals.



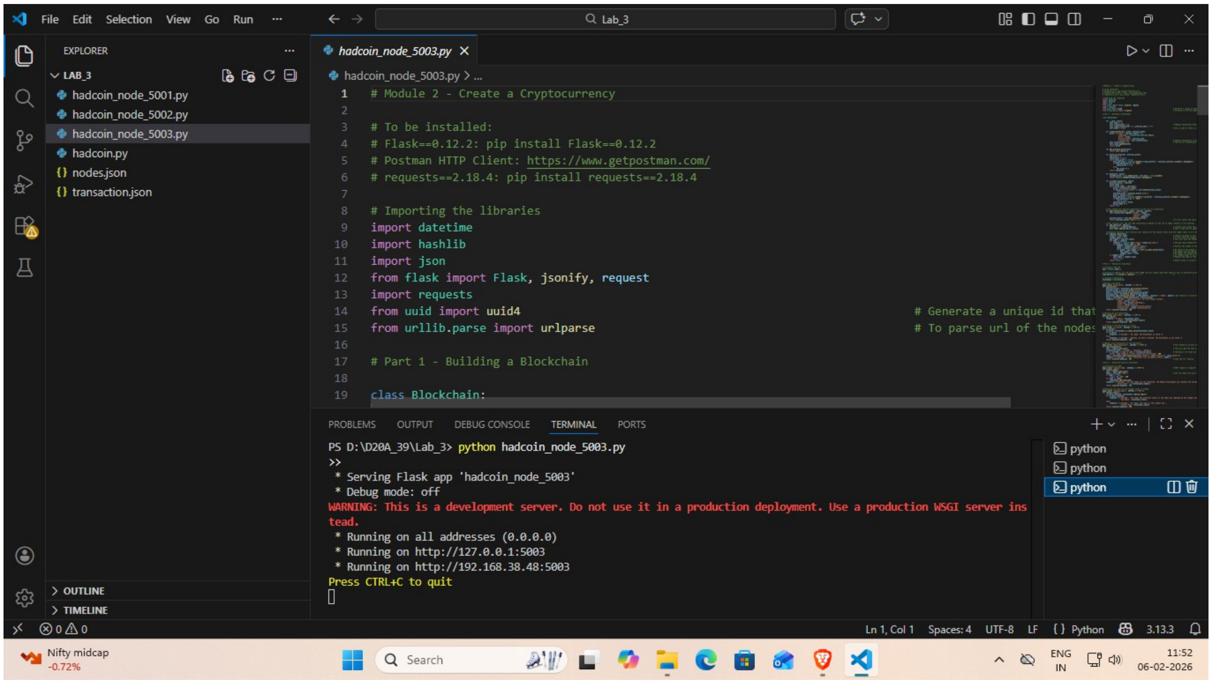
```python
# Module 2 - Create a Cryptocurrency

# 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
from urllib.parse import urlparse                                               # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:
```

```
PS D:\D20A_39\Lab_3> python hadcoin_node_5001.py
>>
 * Serving Flask app 'hadcoin_node_5001'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server ins
tead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.38.48:5001
Press CTRL+C to quit
```



```python
# 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
from urllib.parse import urlparse                                               # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:
```

```
PS D:\D20A_39\Lab_3> python hadcoin_node_5002.py
>>
 * Serving Flask app 'hadcoin_node_5002'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server ins
tead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://192.168.38.48:5002
Press CTRL+C to quit
```



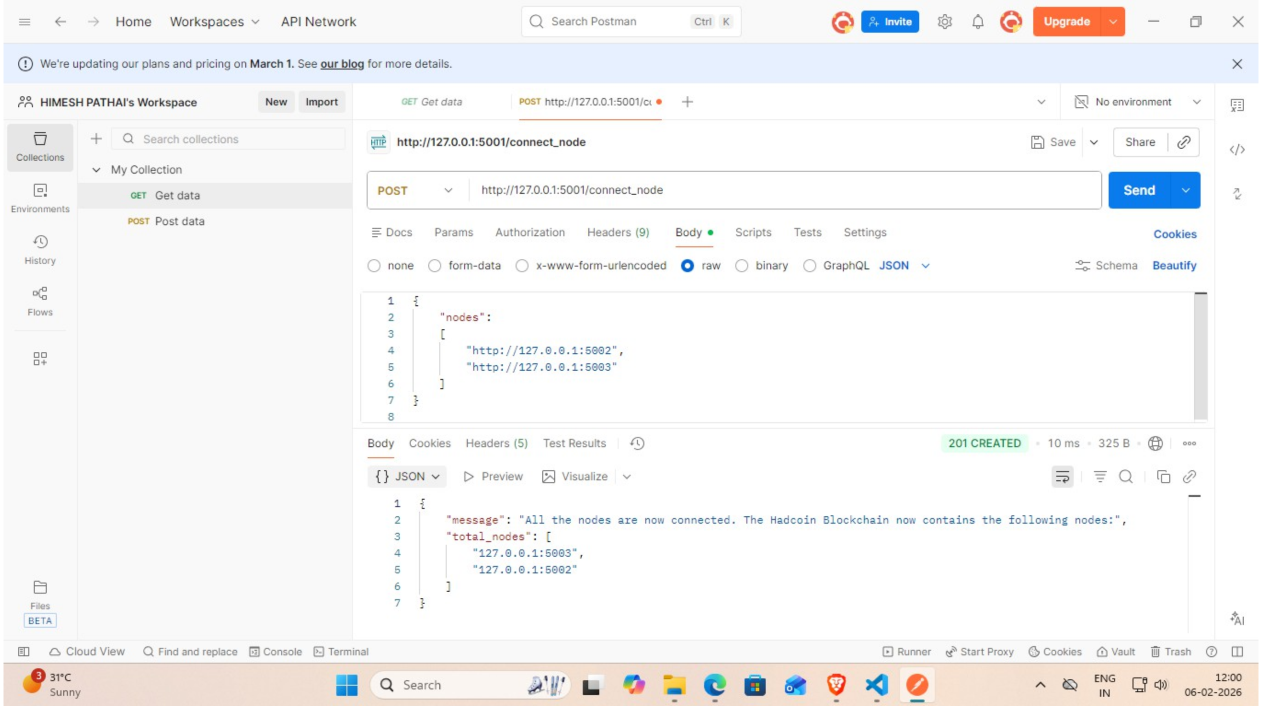
```python
# Module 2 - Create a Cryptocurrency

# 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
from urllib.parse import urlparse                                               # To parse url of the nodes

# Part 1 - Building a Blockchain

class Blockchain:
```

```
PS D:\D20A_39\Lab_3> python hadcoin_node_5003.py
>>
 * Serving Flask app 'hadcoin_node_5003'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server ins
tead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5003
 * Running on http://192.168.38.48:5003
Press CTRL+C to quit
```

4. Connect nodes using Postman – POST `/connect_node`.

```
{
  "nodes":
  [
    "http://127.0.0.1:5002",
    "http://127.0.0.1:5002"
  ]
}
```

5.



http://127.0.0.1:5001/connect_node

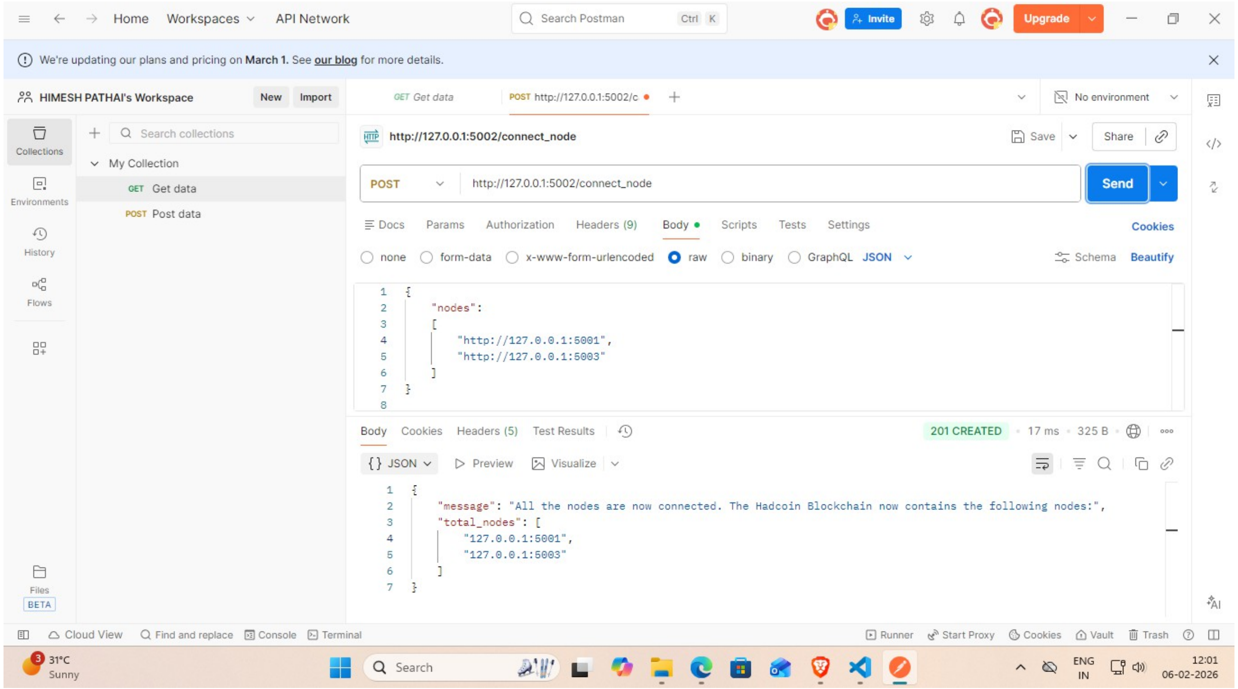
POST http://127.0.0.1:5001/connect_node

Body: ● raw JSON

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

Body — 201 CREATED · 10 ms · 325 B — JSON

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



http://127.0.0.1:5002/connect_node

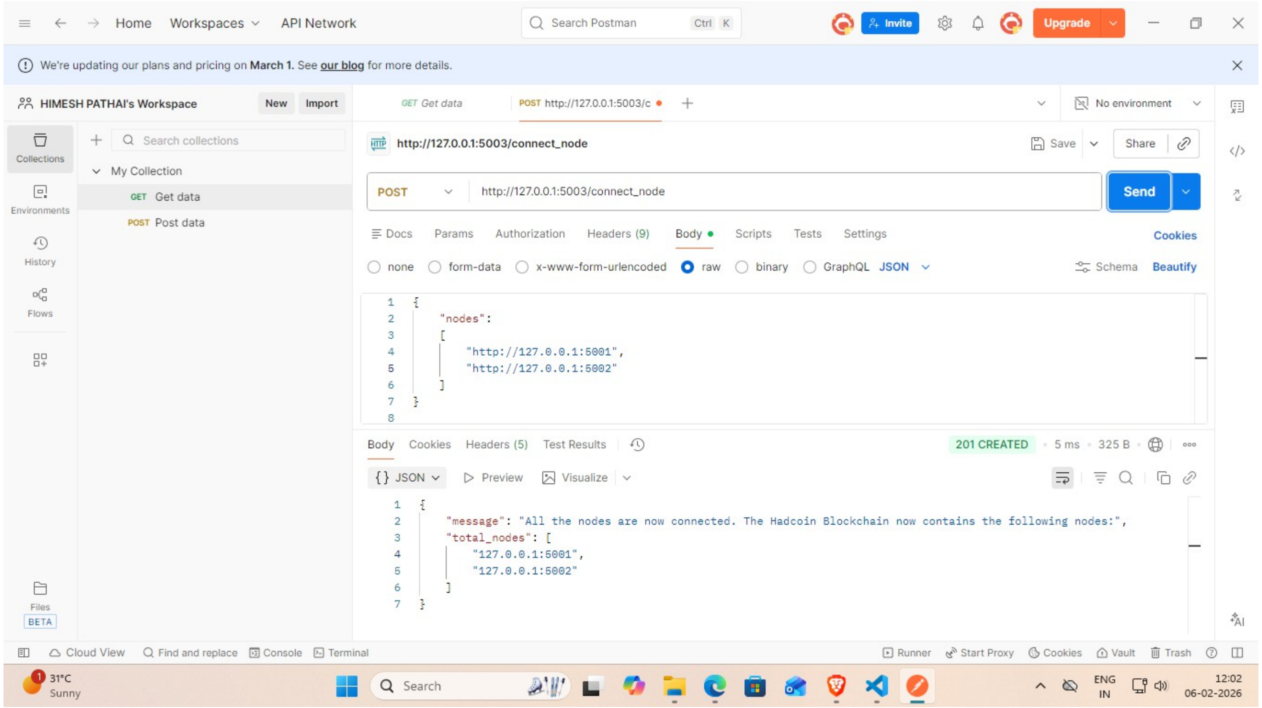
POST http://127.0.0.1:5002/connect_node

Body: ● raw JSON

```
{
    "nodes":
    [
        "http://127.0.0.1:5001",
        "http://127.0.0.1:5003"
    ]
}
```

Body — 201 CREATED · 17 ms · 325 B — JSON

```
{
    "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5001",
        "127.0.0.1:5003"
    ]
}
```



http://127.0.0.1:5003/connect_node

POST http://127.0.0.1:5003/connect_node

```
{
    "nodes":
    [
        "http://127.0.0.1:5001",
        "http://127.0.0.1:5002"
    ]
}

```

201 CREATED • 5 ms • 325 B

```
{
    "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5001",
        "127.0.0.1:5002"
    ]
}
```

6. Add transactions – POST
7. Add 3 transactions in 5001
8. Condition 2 — Each node must have **different chain lengths**

Example:

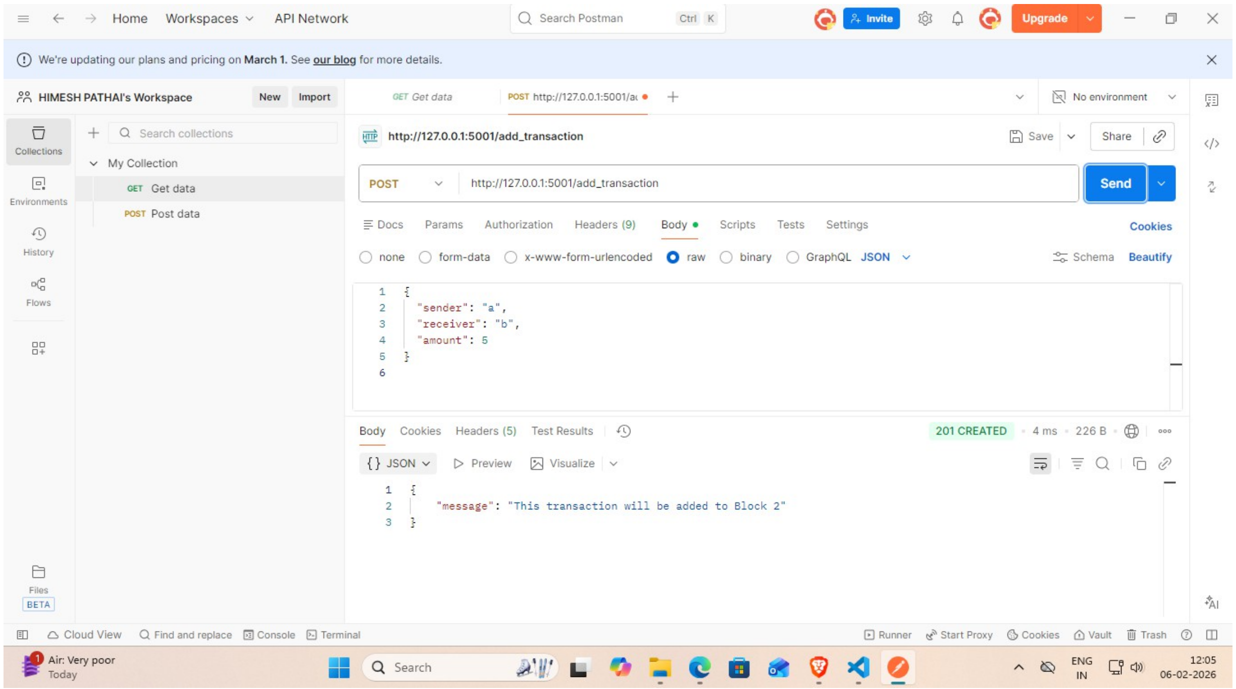
If you mined 1 block on 5001, but did not mine on 5002 or 5003 →

- **5001 chain length = 2**
- **5002 chain length = 1**
- **5003 chain length = 1**

STEP 2 — Perform all transactions ONLY from one node (e.g., 5001)

POST → `http://127.0.0.1:5001/add_transaction`

```
{

  "sender": "a",

  "receiver": "b",

  "amount": 5

}
```



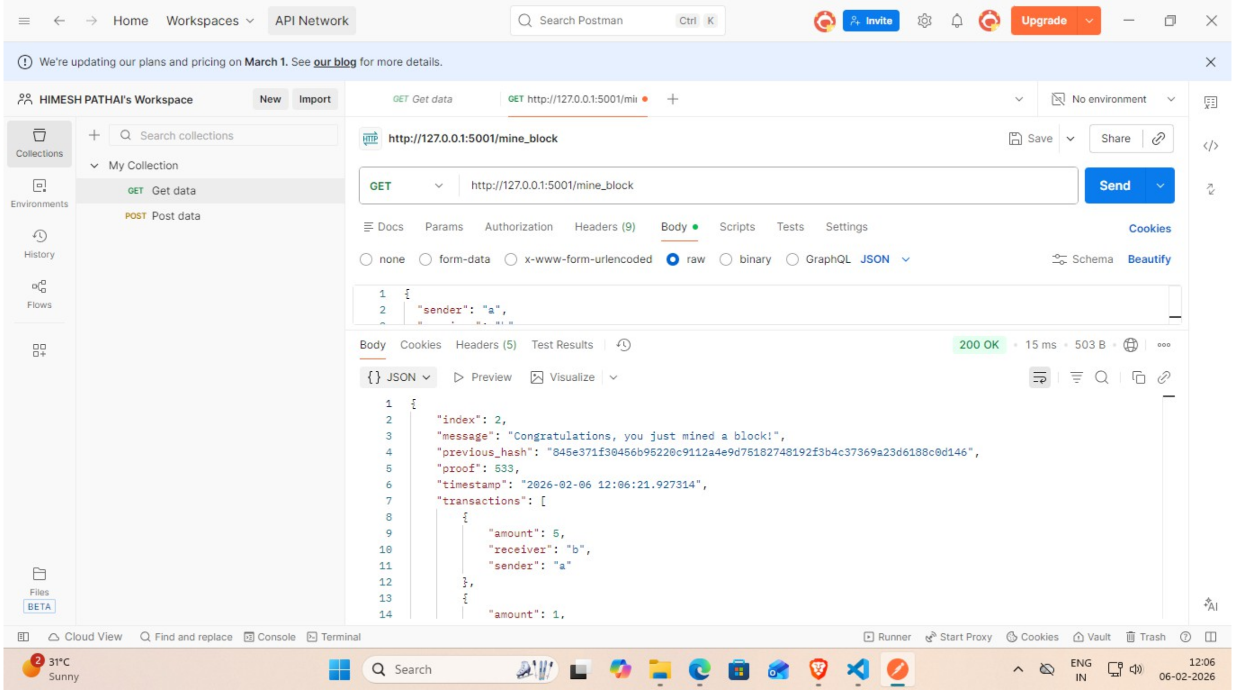
POST http://127.0.0.1:5001/add_transaction

```
{
  "sender": "a",
  "receiver": "b",
  "amount": 5
}
```

201 CREATED · 4 ms · 226 B

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

Then 5001 → GET `/mine_block`



GET http://127.0.0.1:5001/mine_block

Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
  "sender": "a",
```

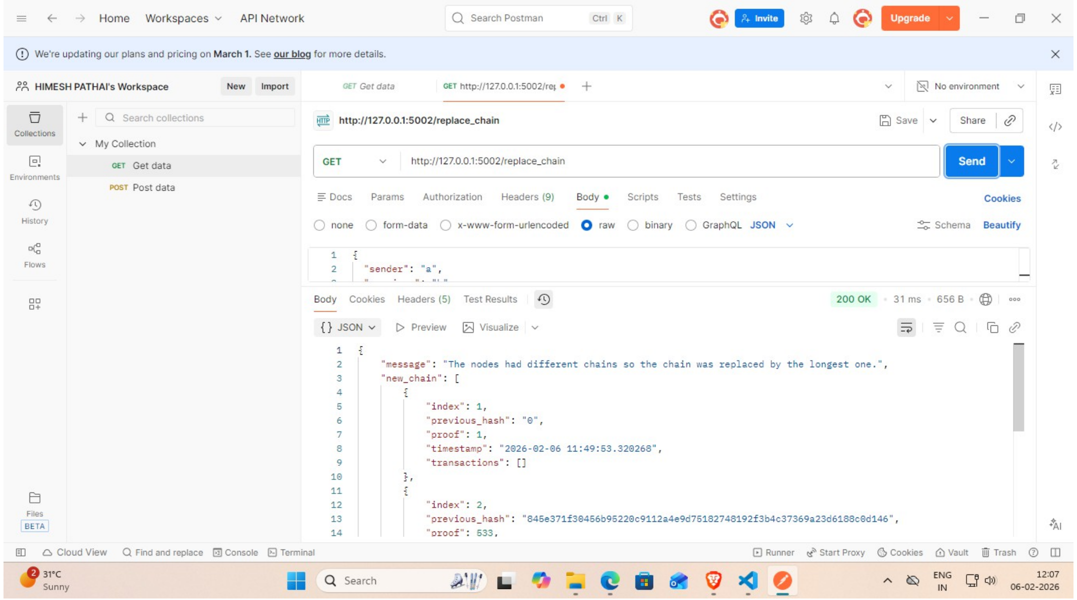
200 OK · 15 ms · 503 B

{} JSON

```
{
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "845e371f30456b95220c9112a4e9d75182748192f3b4c37369a23d6188c0d146",
    "proof": 533,
    "timestamp": "2026-02-06 12:06:21.927314",
    "transactions": [
        {
            "amount": 5,
            "receiver": "b",
            "sender": "a"
        },
        {
            "amount": 1,
```

STEP 3 — Now go to 5002 and run:

GET → `/replace_chain`



http://127.0.0.1:5002/replace_chain

GET http://127.0.0.1:5002/replace_chain

Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
  "sender": "a",
```

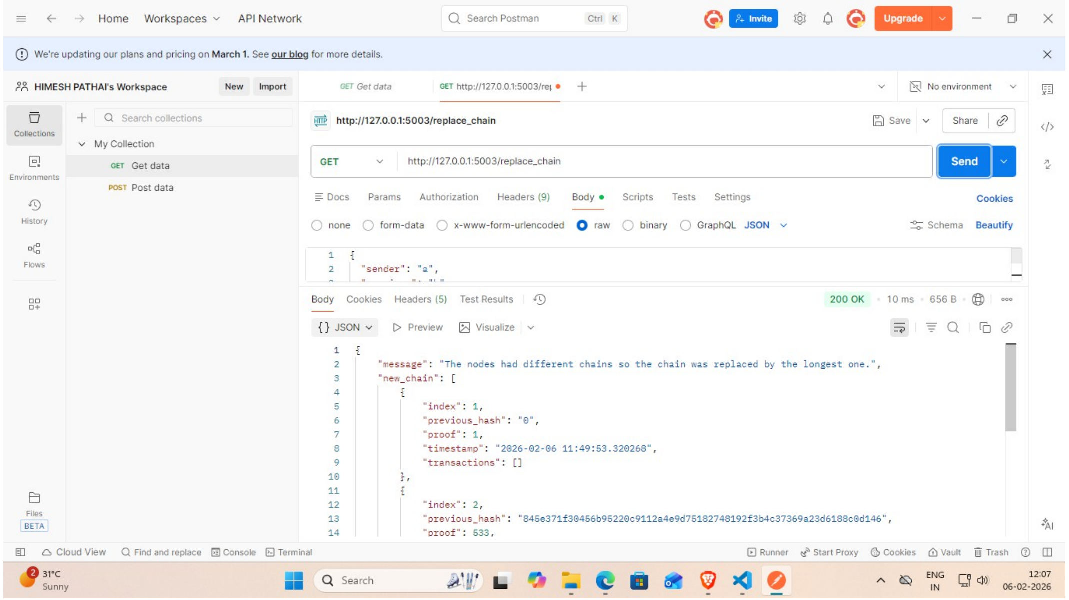
Body — 200 OK · 31 ms · 656 B — {} JSON

```
{
    "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 11:49:53.320268",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "845e371f30456b95220c9112a4e9d75182748192f3b4c37369a23d6188c0d146",
            "proof": 533,
```

STEP 4 — Repeat on 5003

GET → `/replace_chain`

Same result.



http://127.0.0.1:5003/replace_chain

GET http://127.0.0.1:5003/replace_chain

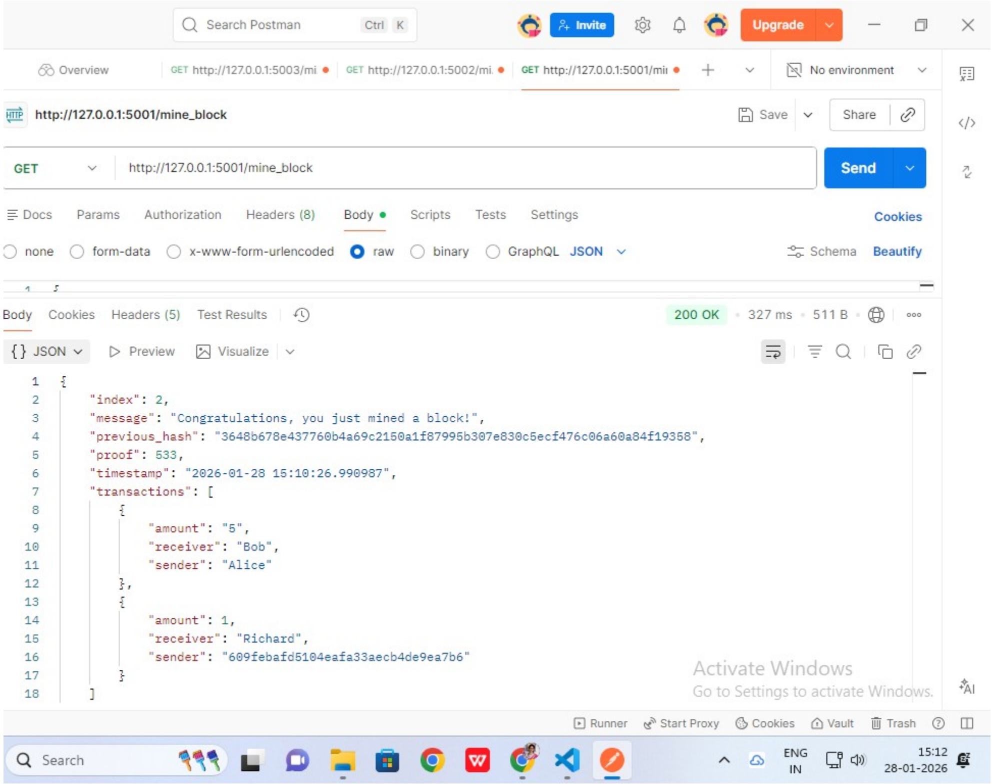
Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
  "sender": "a",
```

Body — 200 OK · 10 ms · 656 B — {} JSON

```
{
    "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 11:49:53.320268",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "845e371f30456b95220c9112a4e9d75182748192f3b4c37369a23d6188c0d146",
            "proof": 533,
```

9. Mine blocks – GET `/mine_block`.



http://127.0.0.1:5001/mine_block

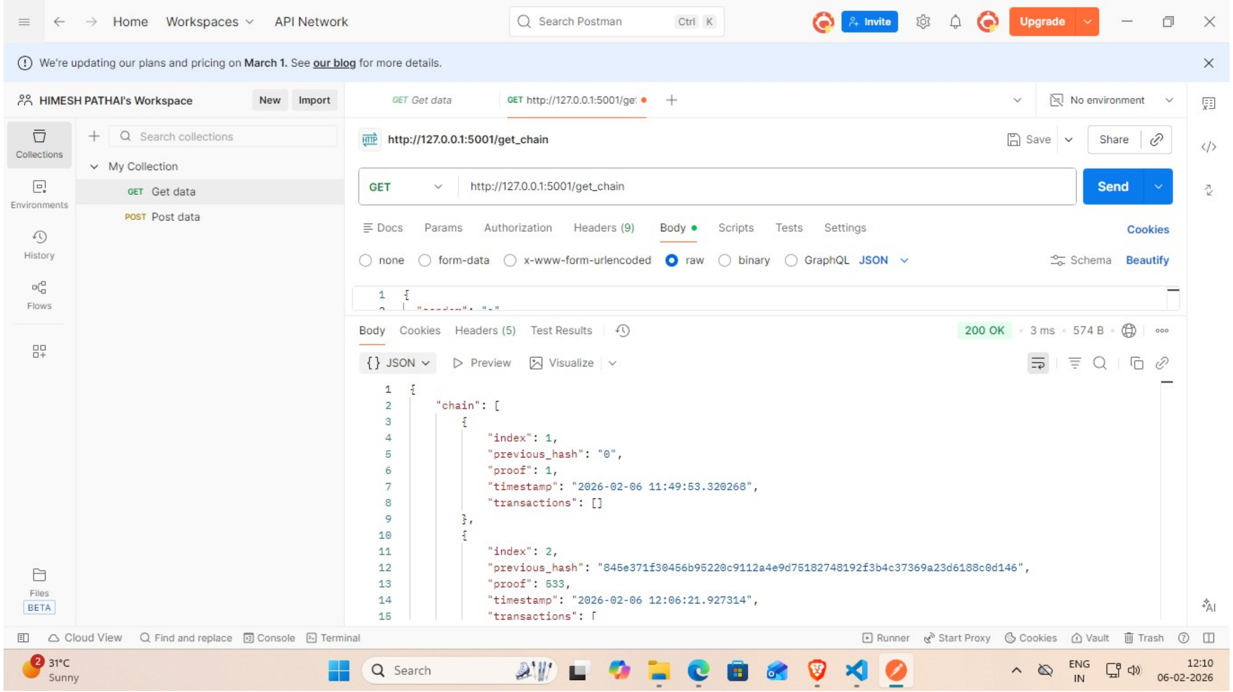
GET http://127.0.0.1:5001/mine_block

Body ● raw JSON

200 OK · 327 ms · 511 B

```
{
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "3648b678e437760b4a69c2150a1f87995b307e830c5ecf476c06a60a84f19358",
    "proof": 533,
    "timestamp": "2026-01-28 15:10:26.990987",
    "transactions": [
        {
            "amount": "5",
            "receiver": "Bob",
            "sender": "Alice"
        },
        {
            "amount": 1,
            "receiver": "Richard",
            "sender": "609febafd5104eafa33aecb4de9ea7b6"
        }
    ]
```

10. Fetch blockchain – GET `/get_chain`.



HIMESH PATHAI's Workspace

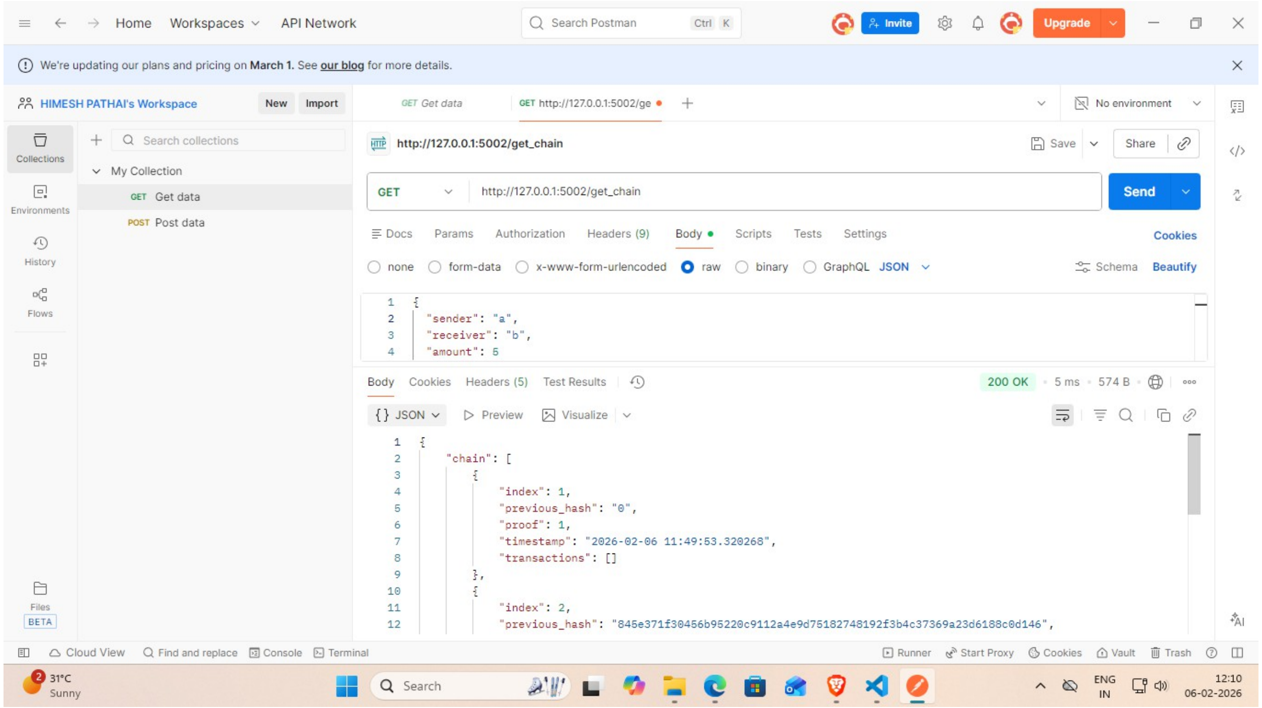
http://127.0.0.1:5001/get_chain

GET http://127.0.0.1:5001/get_chain

Body ● raw JSON

200 OK · 3 ms · 574 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 11:49:53.320268",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "845e371f30456b95220c9112a4e9d75182748192f3b4c37369a23d6188c0d146",
            "proof": 533,
            "timestamp": "2026-02-06 12:06:21.927314",
            "transactions": [
```



http://127.0.0.1:5002/get_chain

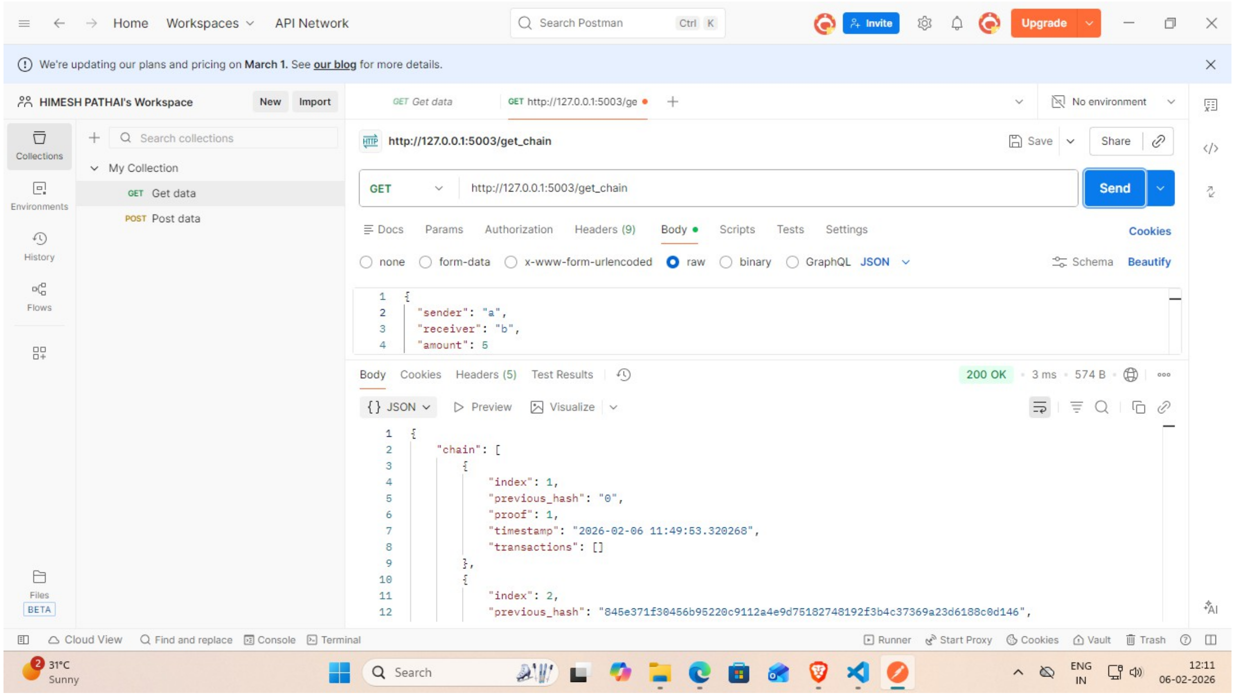
GET http://127.0.0.1:5002/get_chain

Body: raw JSON

```
{
 "sender": "a",
 "receiver": "b",
 "amount": 5
```

200 OK · 5 ms · 574 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 11:49:53.320268",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "845e371f30456b95220c9112a4e9d75182748192f3b4c37369a23d6188c0d146",
```



http://127.0.0.1:5003/get_chain

GET http://127.0.0.1:5003/get_chain

Body: raw JSON

```
{
 "sender": "a",
 "receiver": "b",
 "amount": 5
```

200 OK · 3 ms · 574 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-06 11:49:53.320268",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "845e371f30456b95220c9112a4e9d75182748192f3b4c37369a23d6188c0d146",
```

Conclusion:

We developed a cryptocurrency using Python and implemented mining in a simulated three-node blockchain network. Each node maintained its own ledger and synchronized with peers using the Longest Chain Rule to ensure consistency. Mining was performed through Proof-of-Work, securely adding transactions and rewarding miners. This demonstrated key blockchain concepts, including decentralized consensus, transaction validation, and network synchronization.