**Blockchain**

**Practical 7**

Aim

Program to implement Blockchain DAPP

Create two Python files

blockchain1.py

port – 5001

blockchain2.py

port – 5002

Code

**blockchain1.py**

import datetime

import hashlib

import json

from flask import Flask, jsonify, request

import requests

from uuid import uuid4

from urllib.parse import urlparse

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,

                 'timestamp':str(datetime.datetime.now()),

                 'proof':proof,

                 'previous\_hash':previous\_hash,

                 'transaction':*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 check\_proof is False:

            hash\_value=hashlib.sha256(str(new\_proof\*\*2-previous\_proof\*\*2).encode()).hexdigest()

            if hash\_value[:5] == '00000':

                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']

            new\_proof = block['proof']

            hash\_value=hashlib.sha256(str(new\_proof\*\*2-previous\_proof\*\*2).encode()).hexdigest()

            if hash\_value[:5] != '00000':

                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

        max\_length = len(*self*.chain)

        longest\_chain = None

        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\_\_)

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)

    block = blockchain.create\_block(proof,previous\_hash)

    response = {'message':'block is mined',

                'index':block['index'],

                'timestamp':block['timestamp'],

                'proof':block['proof'],

                'previous\_hash':block['previous\_hash']}

    return jsonify(response), 200

@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':'Your chain is validated'}

    else:

        response = {'message':'Your chain is not valid'}

    return jsonify(response), 200

@app.route('/add\_transaction',methods=['POST'])

def add\_transaction():

    json = request.get\_json()

    transaction\_keys = ['sender','receiver','amount']

    if not all(key in json for key in transaction\_keys):

        return 'Some elements are missing', 400

    index = blockchain.add\_transaction(json['sender'],

                                       json['receiver'],

                                       json['amount'])

    response = {'Message':f'The transaction is added to block {index}'}

    return jsonify(response), 201

@app.route('/connect\_nodes',methods=['POST'])

def connect\_nodes():

    json = request.get\_json()

    nodes = json.get('nodes')

    print(nodes)

    if nodes is None:

        return 'No Nodes', 400

    for node in nodes:

        blockchain.add\_node(node)

    response = {'message': 'all nodes are now connected',

                'Total nodes': list(blockchain.nodes)}

    return jsonify(response), 201

@app.route('/replace\_chain',methods=['GET'])

def replace\_chain():

    is\_chain\_replace = blockchain.replace\_chain()

    if is\_chain\_replace:

        response = {'message': 'chain replaced',

                    'new\_chain': blockchain.chain}

    else:

        response = {'message': 'chain not replaced',

                    'new\_chain': blockchain.chain}

    return jsonify(response), 201

app.run(host='0.0.0.0',port = 5001)

**blockchain2.py**

import datetime

import hashlib

import json

from flask import Flask, jsonify, request

import requests

from uuid import uuid4

from urllib.parse import urlparse

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,

                 'timestamp':str(datetime.datetime.now()),

                 'proof':proof,

                 'previous\_hash':previous\_hash,

                 'transaction':*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 check\_proof is False:

            hash\_value=hashlib.sha256(str(new\_proof\*\*2-previous\_proof\*\*2).encode()).hexdigest()

            if hash\_value[:5] == '00000':

                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']

            new\_proof = block['proof']

            hash\_value=hashlib.sha256(str(new\_proof\*\*2-previous\_proof\*\*2).encode()).hexdigest()

            if hash\_value[:5] != '00000':

                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

        max\_length = len(*self*.chain)

        longest\_chain = None

        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\_\_)

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)

    block = blockchain.create\_block(proof,previous\_hash)

    response = {'message':'block is mined',

                'index':block['index'],

                'timestamp':block['timestamp'],

                'proof':block['proof'],

                'previous\_hash':block['previous\_hash']}

    return jsonify(response), 200

@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':'Your chain is validated'}

    else:

        response = {'message':'Your chain is not valid'}

    return jsonify(response), 200

@app.route('/add\_transaction',methods=['POST'])

def add\_transaction():

    json = request.get\_json()

    transaction\_keys = ['sender','receiver','amount']

    if not all(key in json for key in transaction\_keys):

        return 'Some elements are missing', 400

    index = blockchain.add\_transaction(json['sender'],

                                       json['receiver'],

                                       json['amount'])

    response = {'Message':f'The transaction is added to block {index}'}

    return jsonify(response), 201

@app.route('/connect\_nodes',methods=['POST'])

def connect\_nodes():

    json = request.get\_json()

    nodes = json.get('nodes')

    print(nodes)

    if nodes is None:

        return 'No Nodes', 400

    for node in nodes:

        blockchain.add\_node(node)

    response = {'message': 'all nodes are now connected',

                'Total nodes': list(blockchain.nodes)}

    return jsonify(response), 201

@app.route('/replace\_chain',methods=['GET'])

def replace\_chain():

    is\_chain\_replace = blockchain.replace\_chain()

    if is\_chain\_replace:

        response = {'message': 'chain replaced',

                    'new\_chain': blockchain.chain}

    else:

        response = {'message': 'chain not replaced',

                    'new\_chain': blockchain.chain}

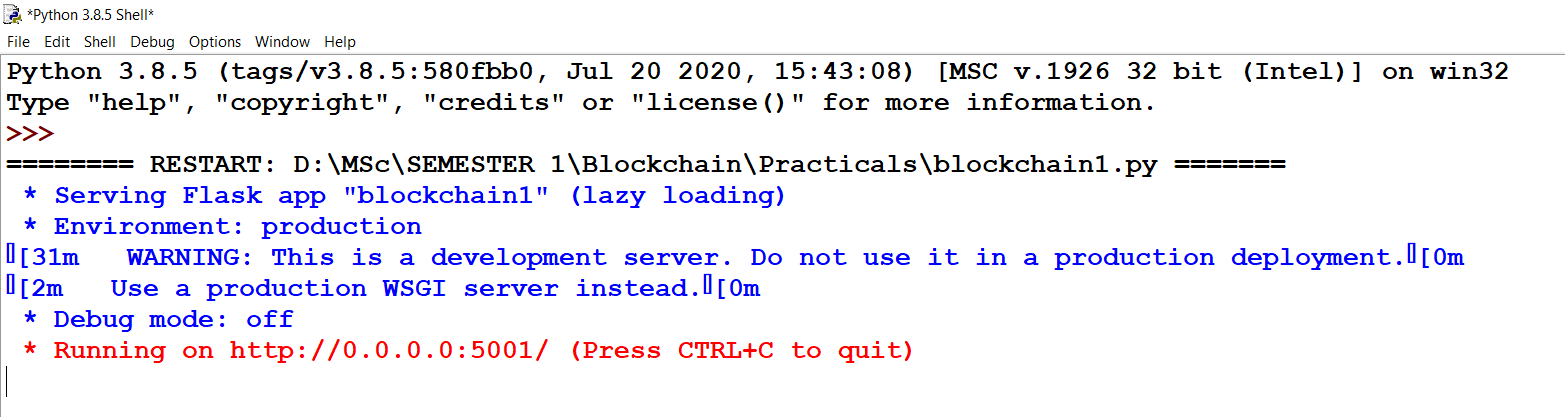
    return jsonify(response), 201

app.run(host='0.0.0.0',port = 5002)

Output

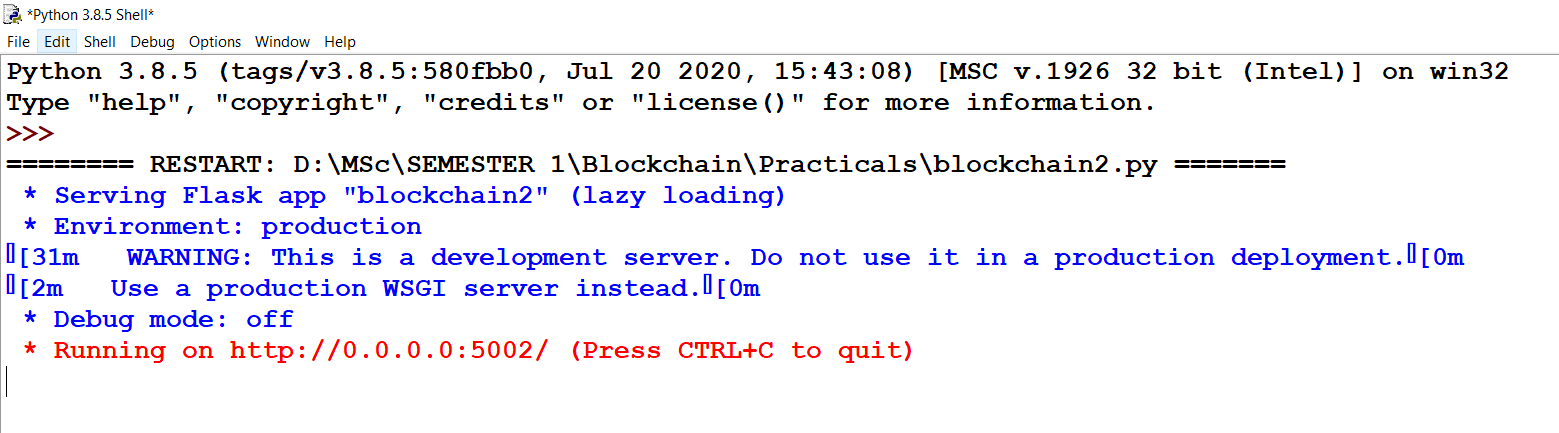
Run blockchain1.py

Port - 5001

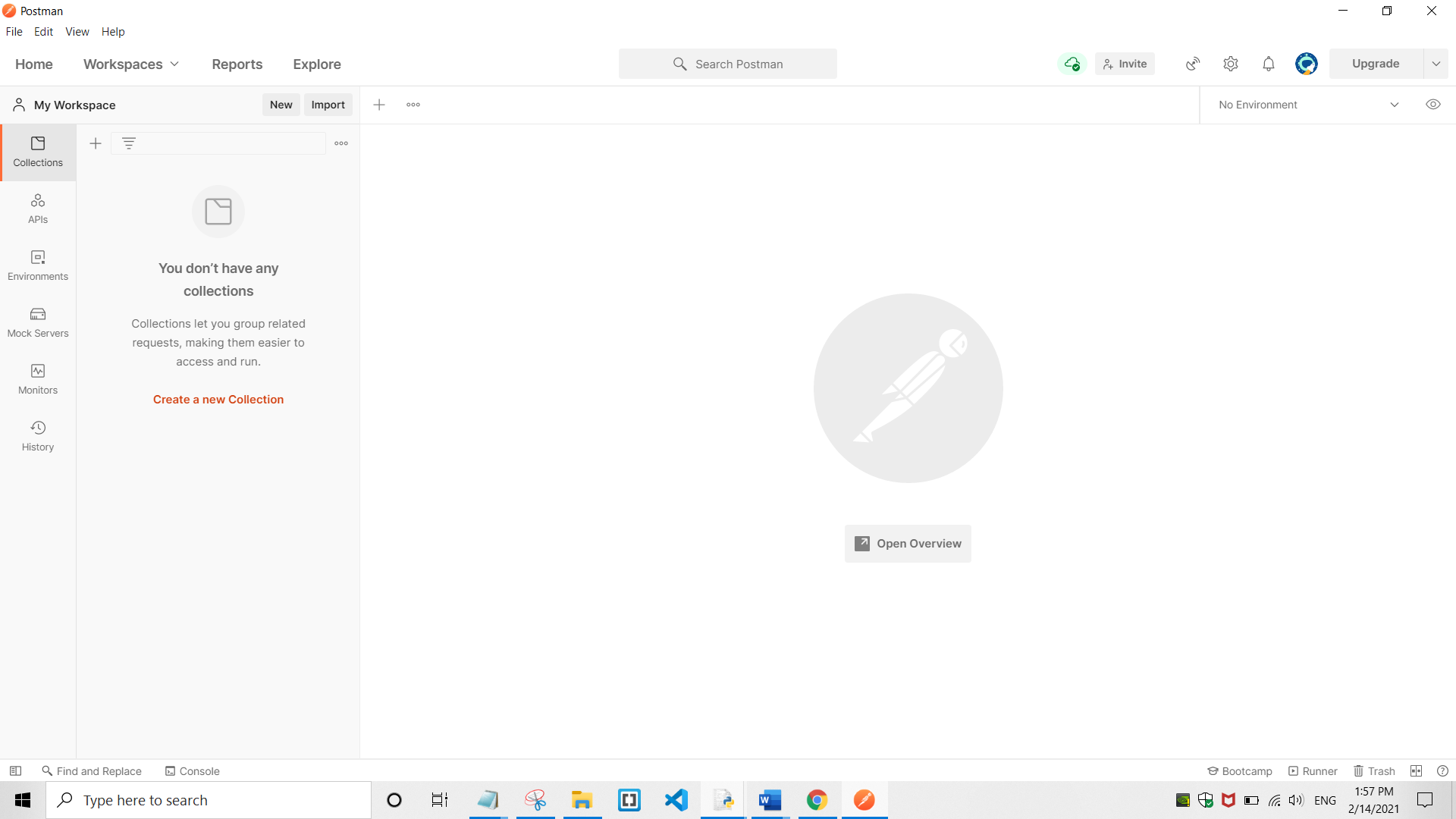


Run blockchain2.py

Port - 5002



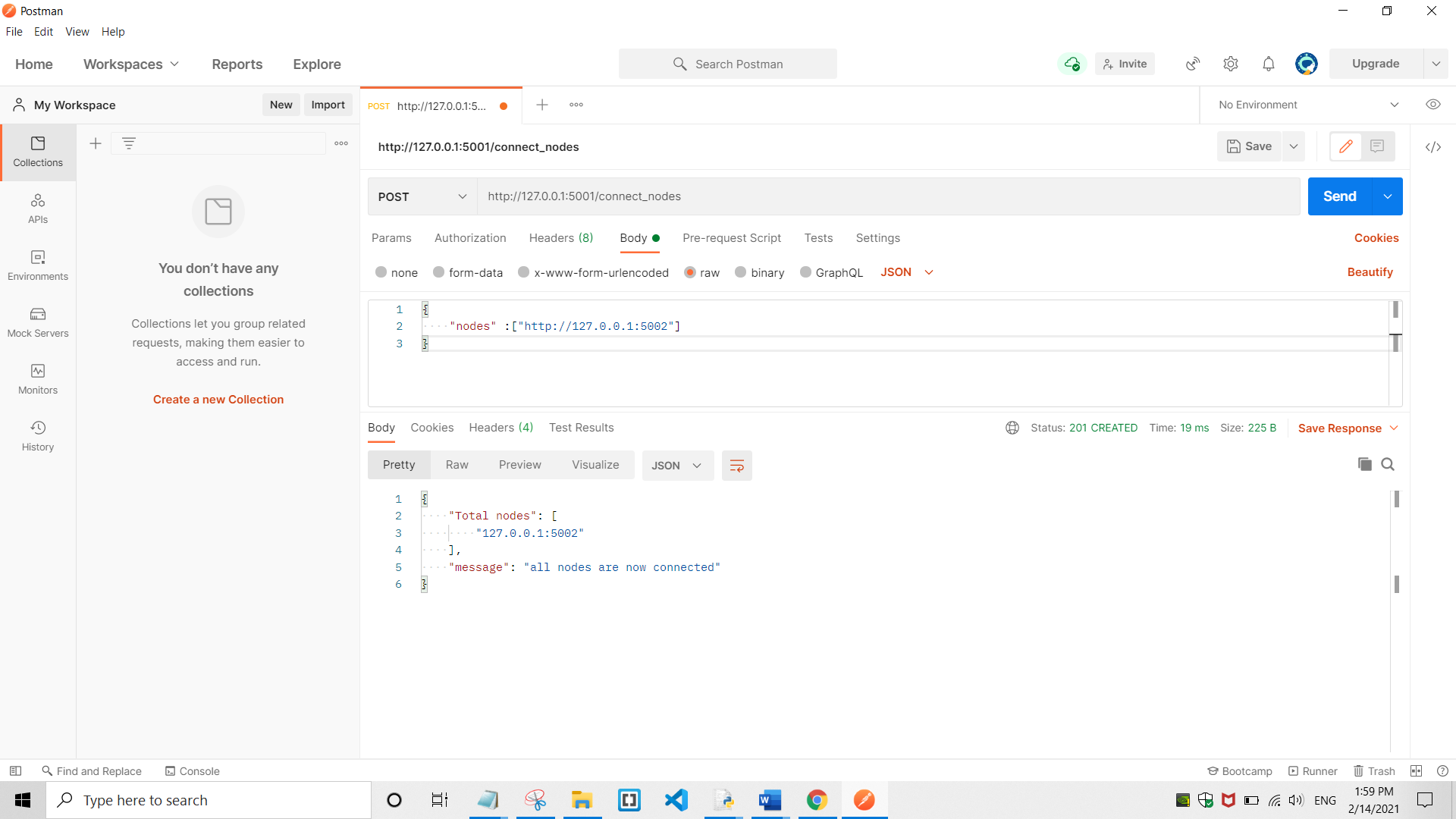
Start Postman



1) Connect node(5001) to node(5002)

Method - POST

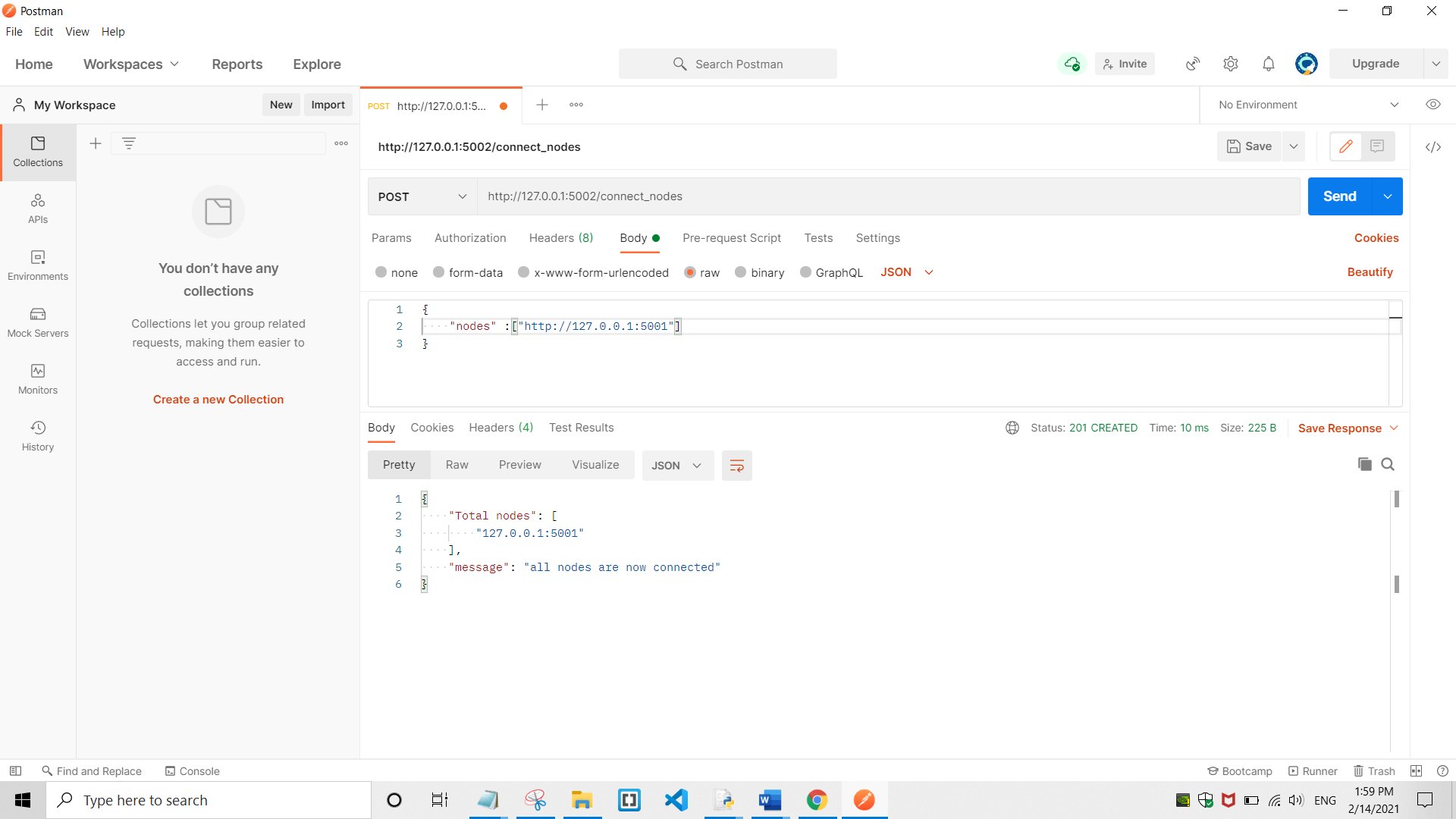
Request URL - http://127.0.0.1:5001/connect\_nodes



2) Connect node(5002) to node(5001)

Method - POST

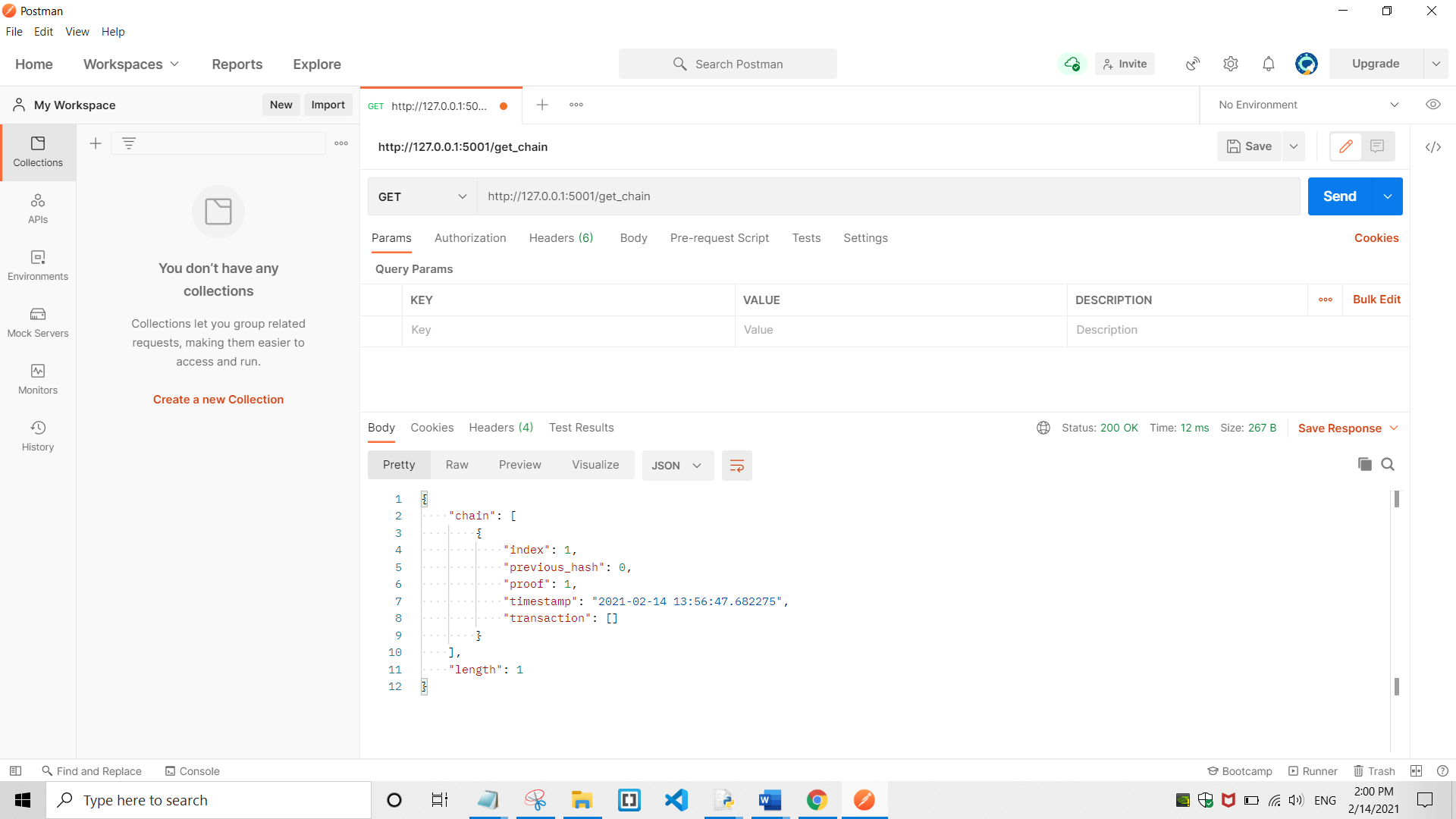
Request URL - http://127.0.0.1:5002/connect\_nodes



3) Get chain of node(5001)

Method - GET

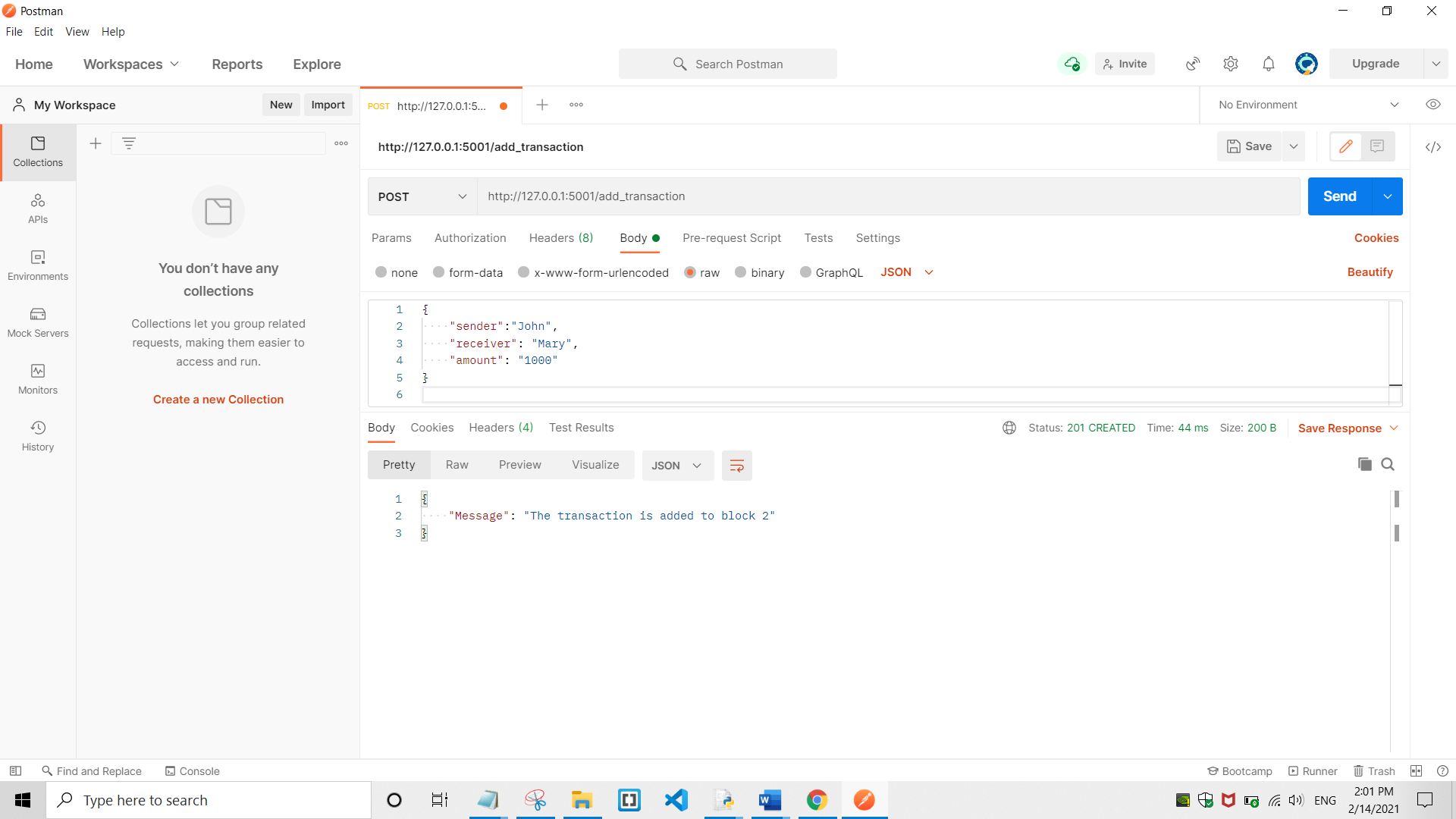
Request URL - http://127.0.0.1:5001/get\_chain



4) Add transaction on node(5001)

Method - POST

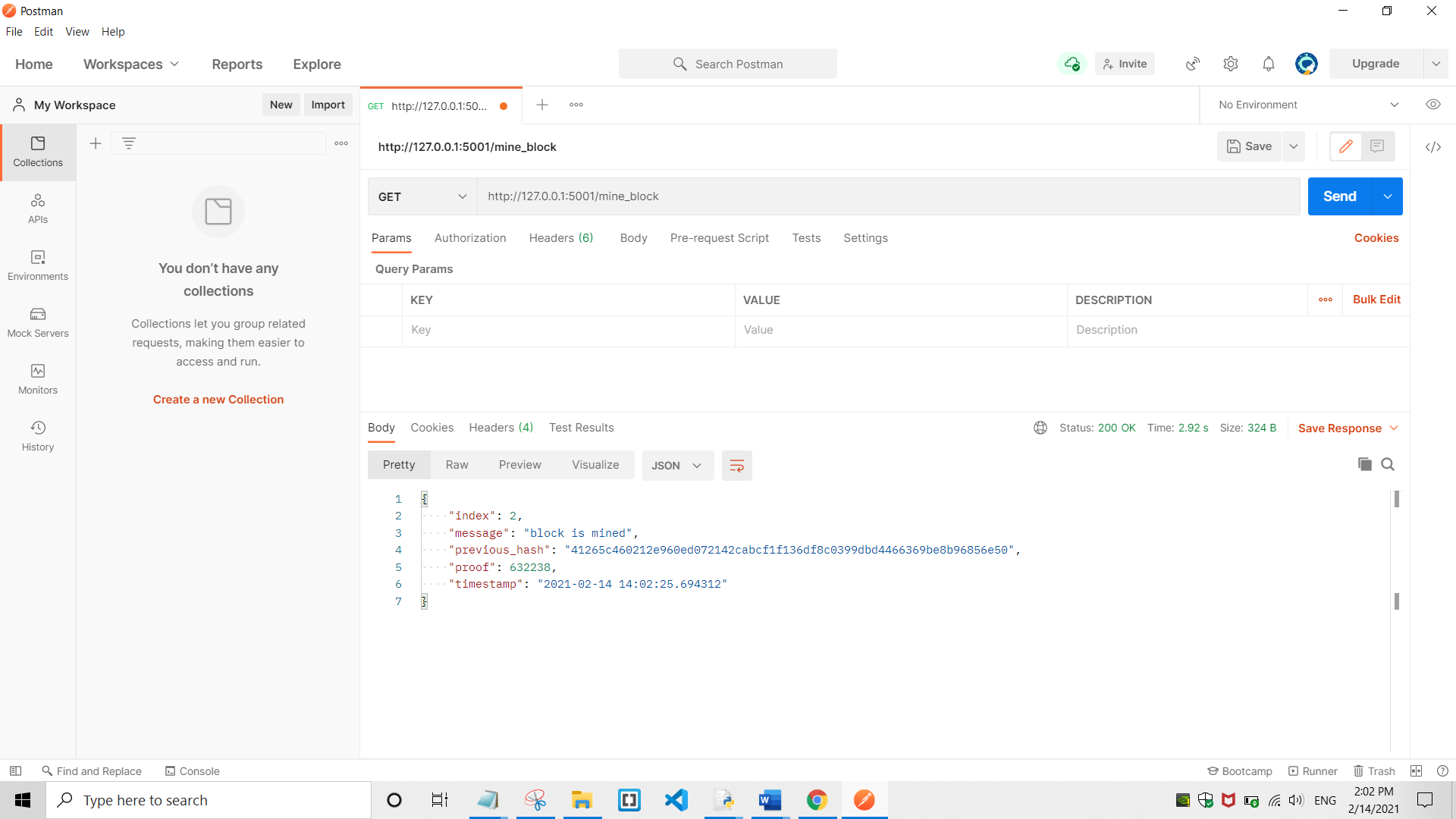
Request URL - http://127.0.0.1:5001/add\_transaction



5) Mine block on node(5001)

Method - GET

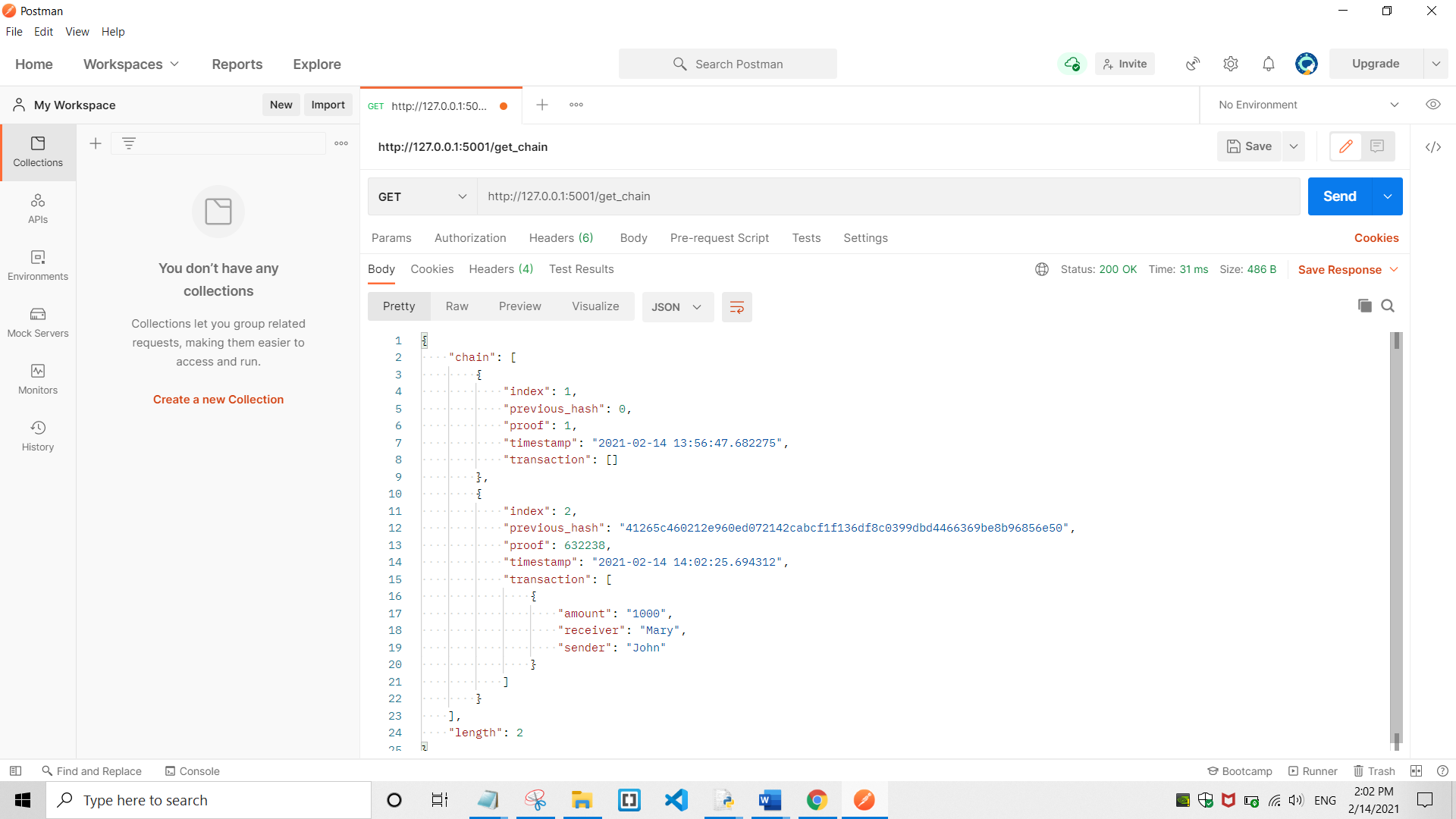
Request URL - http://127.0.0.1:5001/mine\_block



6) Get chain of node(5001)

Method - GET

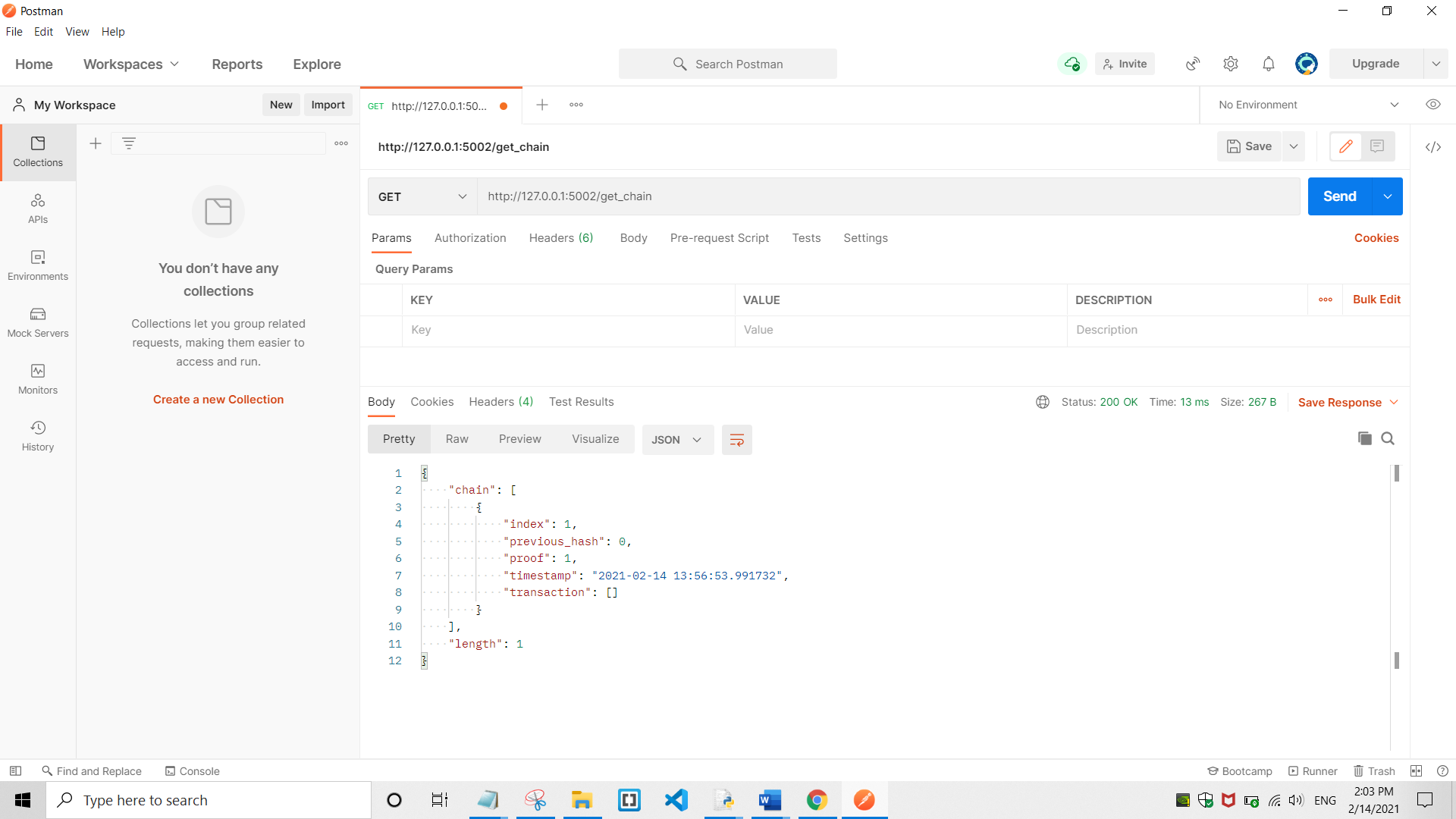
Request URL - http://127.0.0.1:5001/get\_chain



7) Get chain of node(5002)

Method - GET

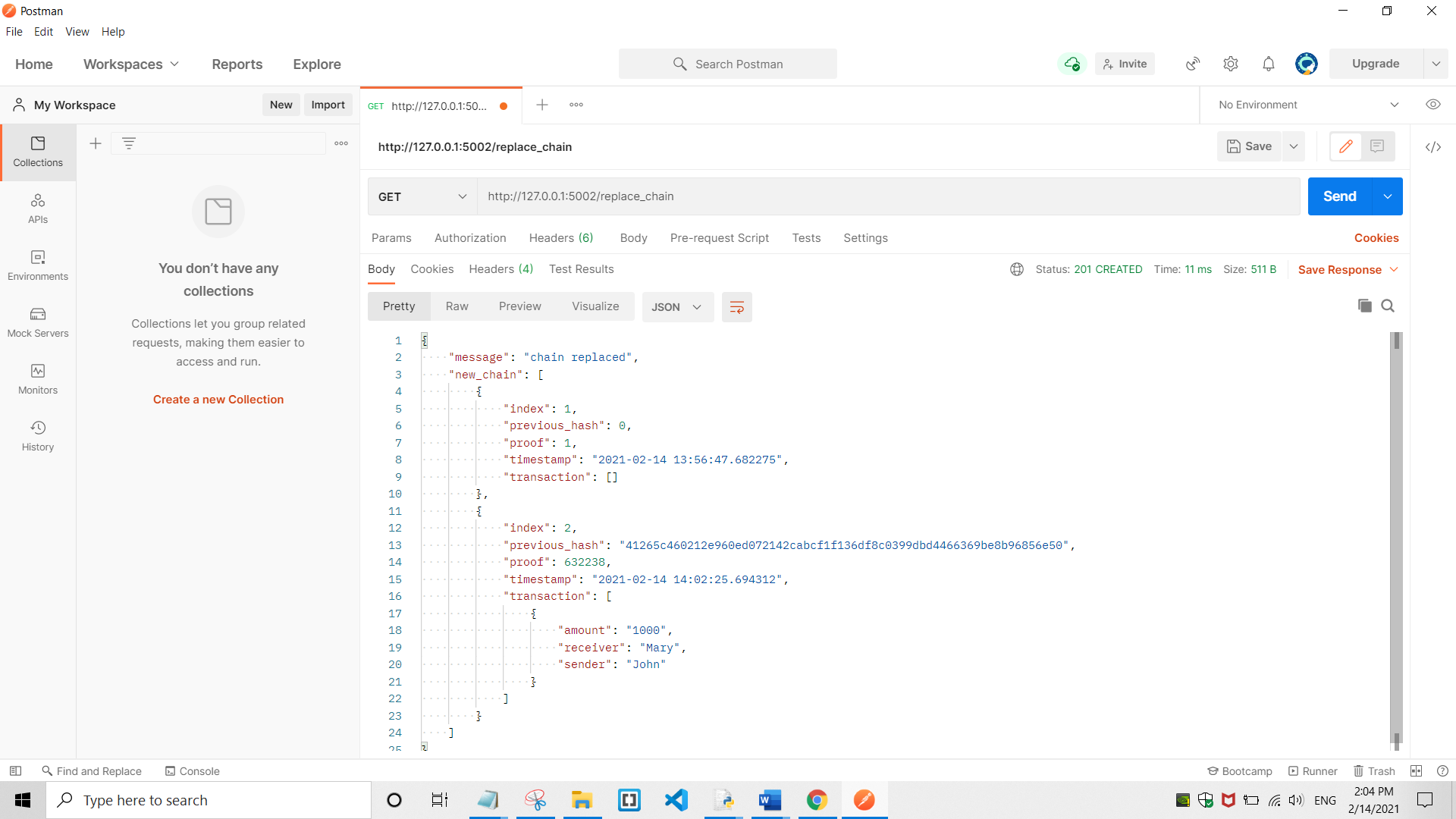
Request URL - http://127.0.0.1:5002/get\_chain



8) Replace chain on node(5002)

Method - GET

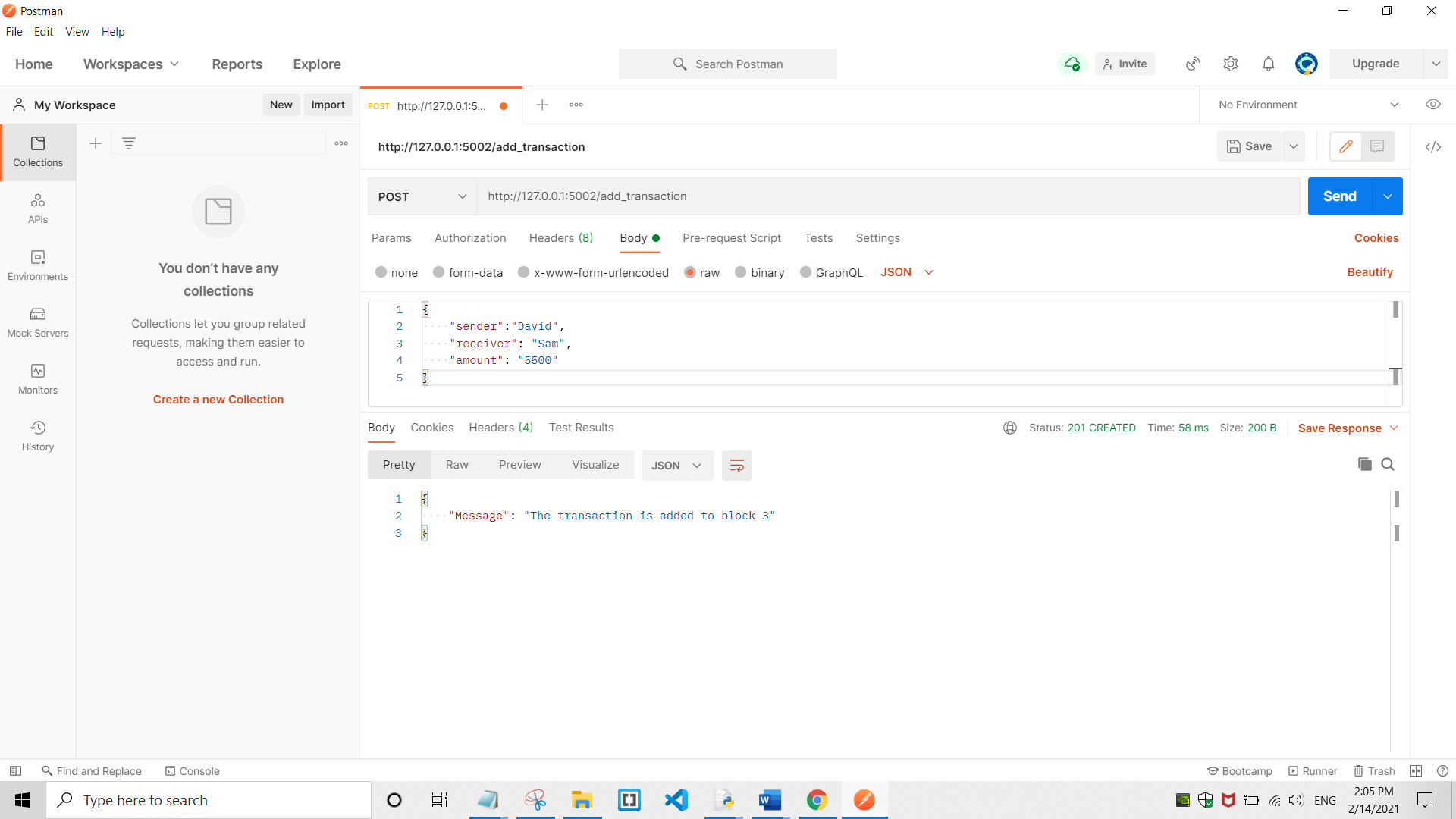
Request URL - http://127.0.0.1:5002/replace\_chain



9) Add Transaction on node(5002)

Method - POST

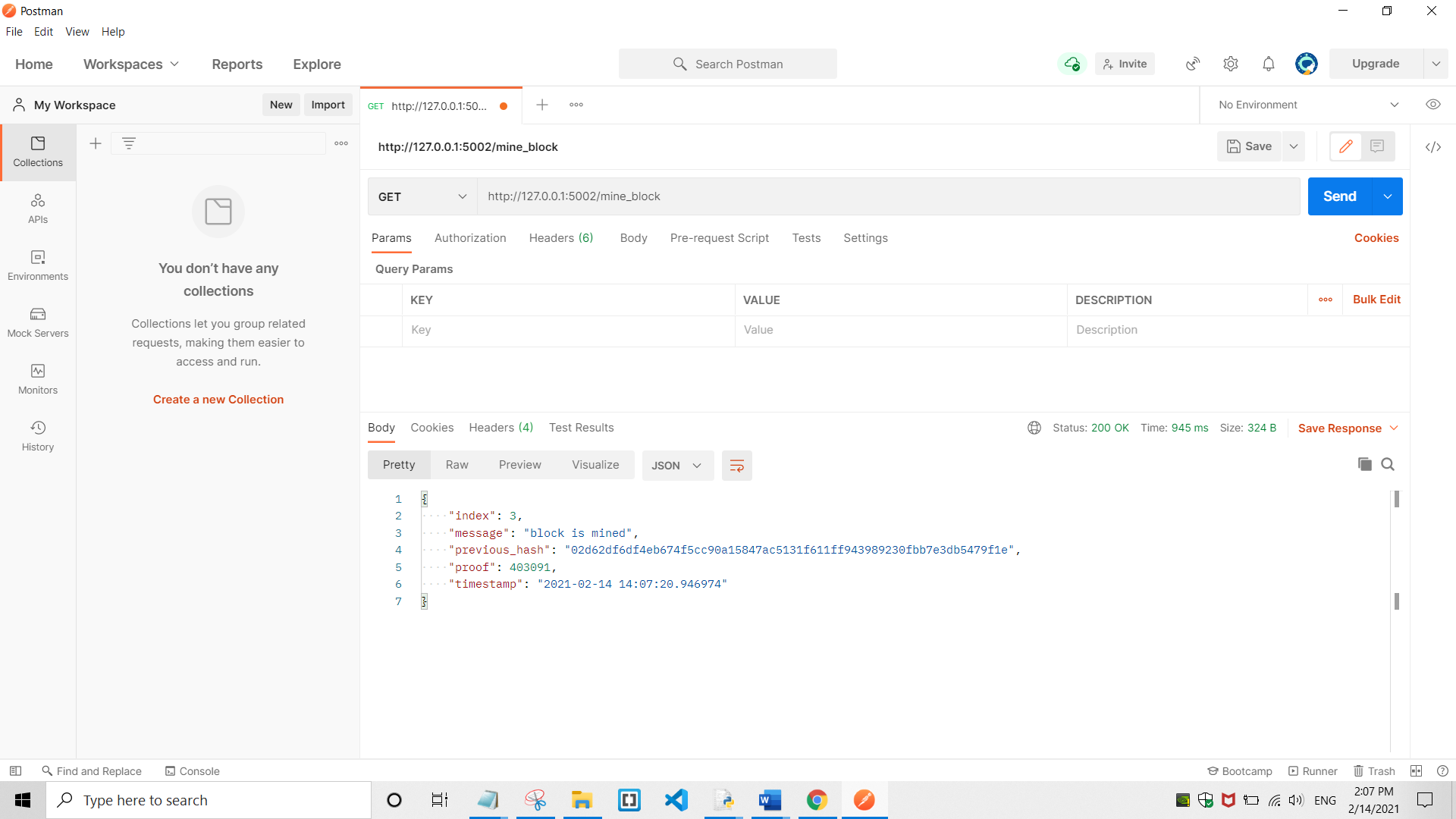
Request URL - http://127.0.0.1:5002/add\_transaction



10) Mine block on node(5002)

Method - GET

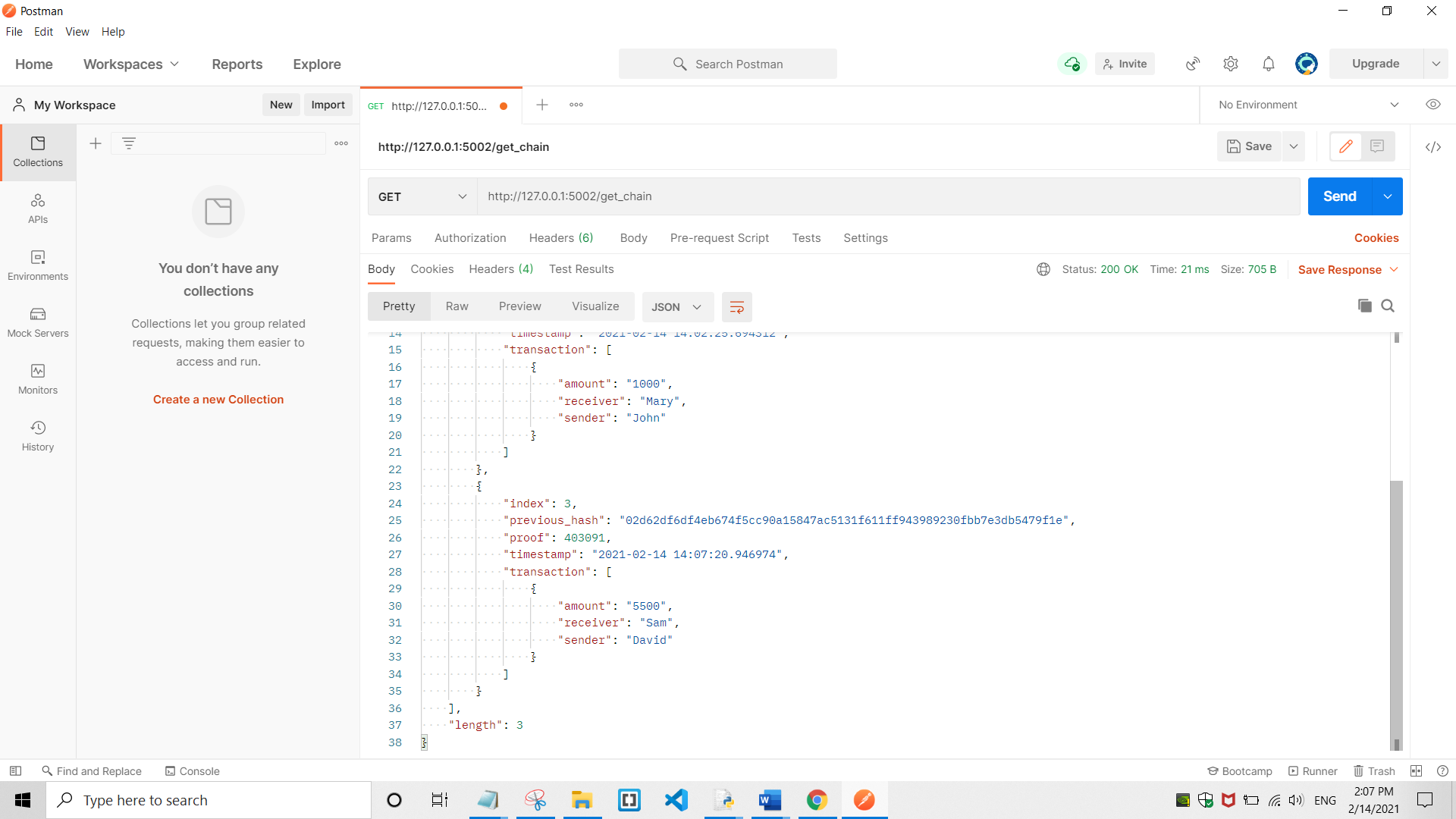
Request URL - http://127.0.0.1:5002/mine\_block



11) Get chain on node(5002)

Method - GET

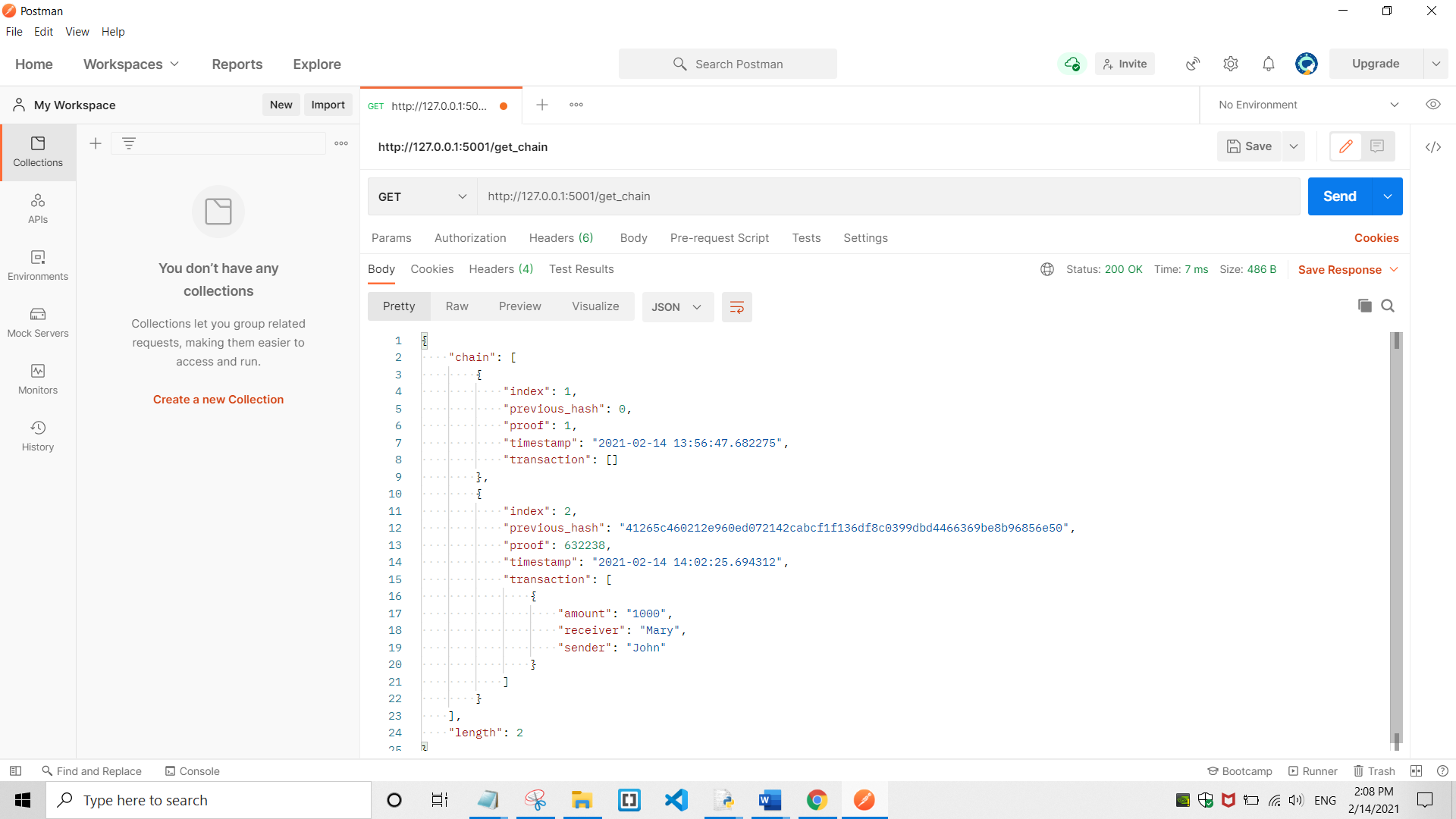
Request URL - http://127.0.0.1:5002/get\_chain



12) Get chain on node(5001)

Method - GET

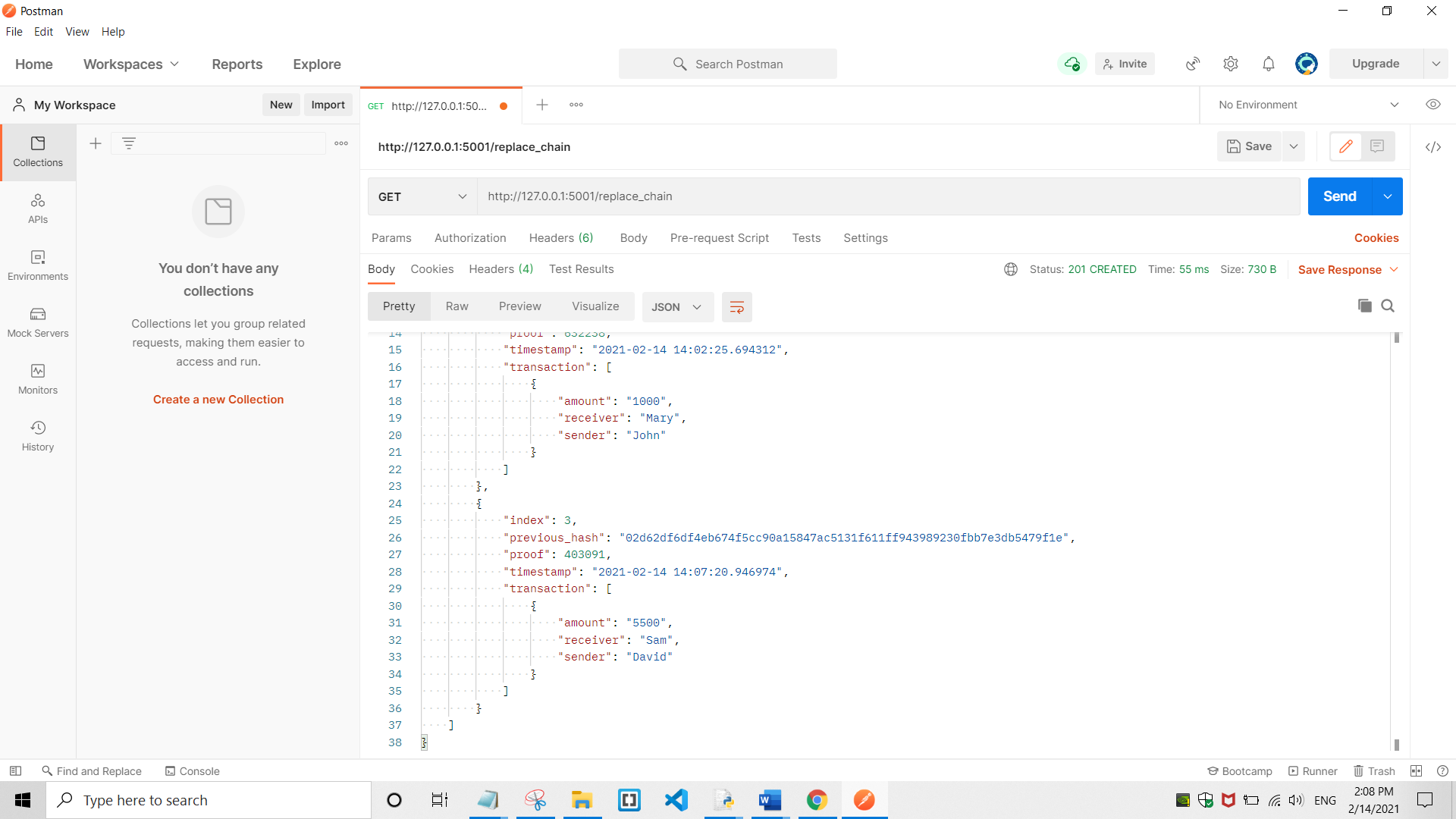
Request URL - http://127.0.0.1:5001/get\_chain



13) Replace chain on node(5001)

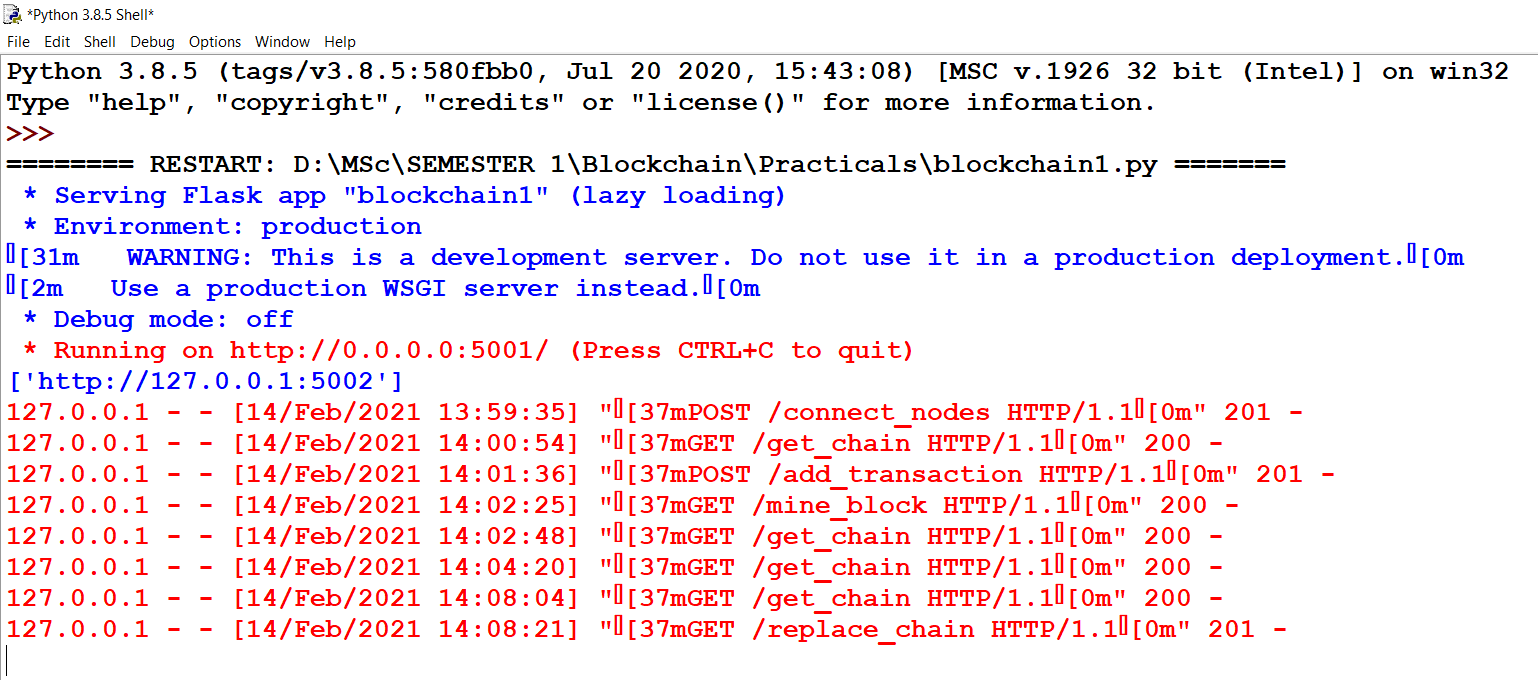
Method - GET

Request URL - http://127.0.0.1:5001/replace\_chain



Requests are displayed

Port – 5001



Port – 5002

