02_Blockchain - create a cryptocurrency

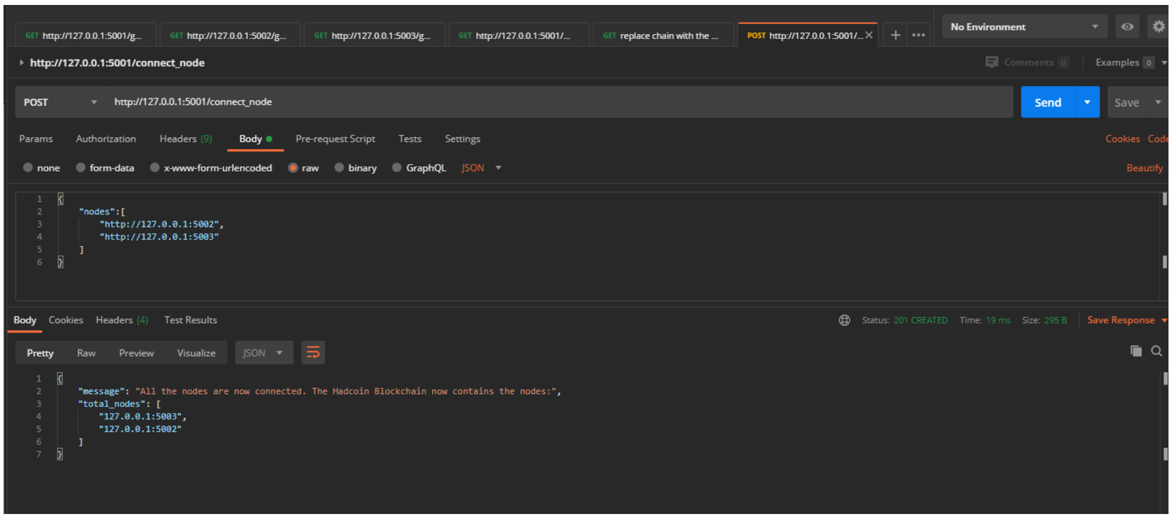
Code:

```
# module 2 - create a Cryptocurrency

# 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

# Part 1 - Building a Blockchain

class Blockchain:

    def __init__(self):
        # chain in which blocks will be added
        self.chain=[]
        # temp list of transactions before they are added to a mined block
        self.transactions=[]
        # initialising the first block
        self.create_block(proof=1, previous_hash="0")
        # nodes participating in the distributing system
        self.nodes=set()

    def create_block(self,proof,previous_hash):
        # structure of a block
        block={
                "index":len(self.chain)+1,
                "timestamp":str(datetime.datetime.now()),
                "proof":proof,
                "previous_hash":previous_hash,
                "transactions":self.transactions
            }
        # transactions are added to the block, so empty the temp transactions list
        self.transactions=[]
        # adding the block to the chain
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self,previous_proof):
        # finding the nonce, through brute force
        new_proof=1
        check_proof=False
        while check_proof is False:
            puzzle=new_proof**2 - previous_proof**2
            hash_operation = hashlib.sha256(str(puzzle).encode()).hexdigest()
            if(hash_operation[:4]=="0000"):
                check_proof=True
            else:
                new_proof+=1
        return new_proof

    def hash(self,block):
        # To get the hash of the 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] # Genesis block
        block_index=1
        while block_index<len(chain):
            # Get the current block
            block=chain[block_index]

            # check if the previous hash of the current block doesn't match the hash of the previous block, return False
            if(block["previous_hash"]!=self.hash(previous_block)):
                return False

            # check if the current block's nonce/proof is valid
            proof=block["proof"]
            previous_proof=previous_block["proof"]
            puzzle=proof**2 - previous_proof**2
            hash_operation = hashlib.sha256(str(puzzle).encode()).hexdigest()
            if(hash_operation[:4]!="0000"):
                return False

            # make current block as the previous block for the next iteration
            previous_block=block

            # increment the value of block_index
            block_index+=1
        return True

    def add_transaction(self,sender,receiver,amount):
        # appending a new transaction object to the list of transactions
        self.transactions.append({
            "sender":sender,
            "receiver":receiver,
            "amount":amount
            })
        # returning the index of the next block in which transactions need to be added
        previous_block=self.get_previous_block()
        return previous_block["index"]+1

    def add_node(self,address):
        # parse the url to get the parsed address
        parsed_url=urlparse(address)
        # adding the parsed url to the set of nodes
        self.nodes.add(parsed_url.netloc)

    # establishing consesus
    def replace_chain(self):
        # network i.e., all the nodes
        network=self.nodes

        # initializing the longest chain
        longest_chain=None

        # initializing the max length of the longest chain with the value of the current Node's blockchain's length
        max_length=len(self.chain)

        # iterating over the nodes
        for node in network:
            # GET request to get the blockchain of the node
            response=requests.get(f"http://{node}/get_chain")
            node_blockchain=response.json()
            # check if the response was 200(success) and
            #if the length of the node's blockchain > max_length and if the node's blockchain is valid
            if response.status_code == 200 and node_blockchain["length"] > max_length and self.is_chain_valid(node_blockchain["
                max_length=node_blockchain["length"]
                longest_chain=node_blockchain["chain"]

        if longest_chain:
            # replace the chain
            self.chain=longest_chain
            return True
        # no replacement made, i.e., the current node's chain is the longest chain
        return False


# Part 2 - Mining our blockchain

# Creating a web app
app=Flask(__name__)

# Creating an address for the node on PORT: 5000
node_address=str(uuid4()).replace("-", "")

# Creating a blockchain
blockchain=Blockchain()

# mining a new block
@app.route("/mine_block",methods=["GET"])
def mine_block():
    # Get the last block as we need its proof/nonce
    previous_block=blockchain.get_previous_block()
    previous_proof=previous_block["proof"];

    # Get the proof/nonce
    proof=blockchain.proof_of_work(previous_proof)

    # Get the previous_hash
    previous_hash=blockchain.hash(previous_block)

    # add the transaction to the blockchain's temp transaction list
    # first transaction to pay the miner for mining the block, who is assumed to have mined a block for the node with address n
    blockchain.add_transaction(sender=node_address, receiver="Varun", amount=1)

    # Create the new block and add it to the chain
    block=blockchain.create_block(proof, previous_hash)
    response={
        **block,
        "message":"Congrats, you just mined a block!",
        }

    return jsonify(response), 200

# Get the full Blockchain
@app.route("/get_chain",methods=["GET"])
def get_chain():
    response={
            "chain": blockchain.chain,
            "length":len(blockchain.chain)
        }
    return jsonify(response), 200

# Get a check result if Blockchain is still valid
@app.route("/is_valid",methods=["GET"])
def is_valid():
    is_valid=blockchain.is_chain_valid(blockchain.chain)
    response={"message":"The Blockchain is NOT valid anymore."}
    if is_valid:
        response={"message":"The Blockchain is valid."}
    return jsonify(response), 200

# Add a new transaction to the Blockchain
@app.route("/add_transaction",methods=["POST"])
def add_transaction():
    # Get the json object send over the POST request's body
    json = request.get_json()
    # The fields necessary to be a part of add_transaction's POST request's body object
    transaction_keys=['sender','receiver','amount']
    # Check if all the keys are present
    if not all(key in json for key in transaction_keys):
        # return a Bad Request as response
        return "Some elements of the transaction are missing", 400
    # Add transaction to the Blockchain
    index=blockchain.add_transaction(sender=json["sender"], receiver=json["receiver"], amount=json["amount"])
    # Create the response
    response = {
            "message":f"This transaction will be added to Block {index}"
        }
    # Return the response
    return jsonify(response), 201

# Part 3 - Decentralizing our Blockchain

# Connecting new nodes
@app.route("/connect_node",methods=["POST"])
def connect_node():
    # Get the POST request's body
    json=request.get_json()
    # Extract the nodes from the body object
    nodes=json.get("nodes")
    # checking if nodes are missing
    if nodes is None:
        return "No node", 400
    # iterate over each node
    for node in nodes:
        # adding the node to the blockchain list of nodes
        blockchain.add_node(node)
    # Creating the response
    response={
        "message":"All the nodes are now connected. The Hadcoin Blockchain now contains the 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():
    # call the blockchain's method to verify and if required the longest chain
    is_chain_replaced=blockchain.replace_chain()

    response={}

    # if is_chain_replaced is True then the blockchain's chain has been replaced with the longest chain among the nodes
    if is_chain_replaced:
        response={
            "message":"The nodes had different chain so the chain were replaced by the longest chain",
            "new_chain":blockchain.chain
            }
    else:
        response={
            "message":"All good, the chain is the longest chain",
            "actual_chain":blockchain.chain
            }
    return jsonify(response), 200
# Running the app
app.run(host="0.0.0.0",port=5000)
```

Instructions to run:

1. Create three different files and copy the above code. Each file should have a different Port on which the flask server will be hosted. These programs running at different port in different consoles represents different nodes
2. We need to connect the nodes, to establish a distributed environment, using the **connect_node** POST method:

node 1:



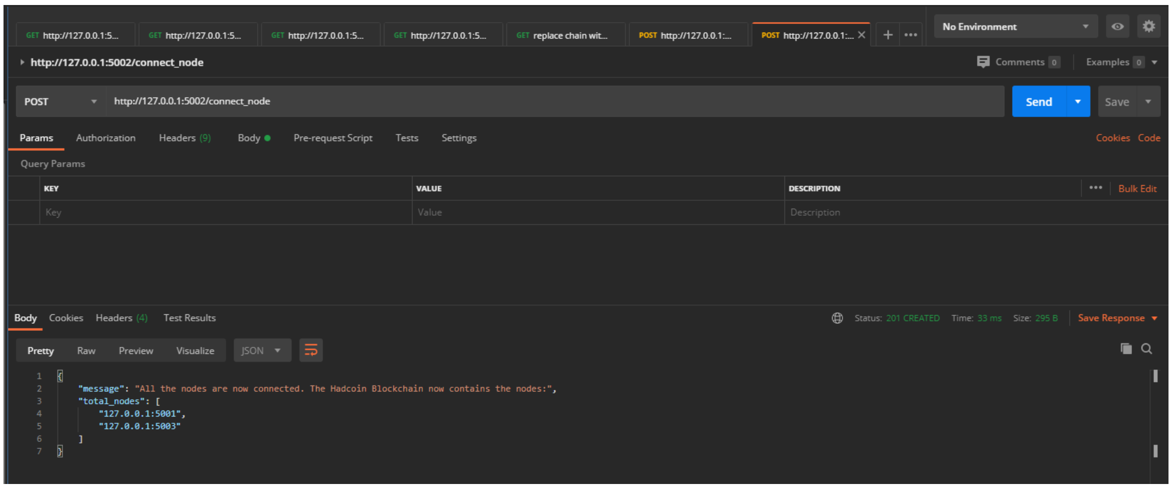
POST http://127.0.0.1:5001/connect_node

Status: 201 CREATED Time: 19 ms Size: 295 B

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

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

node 2:

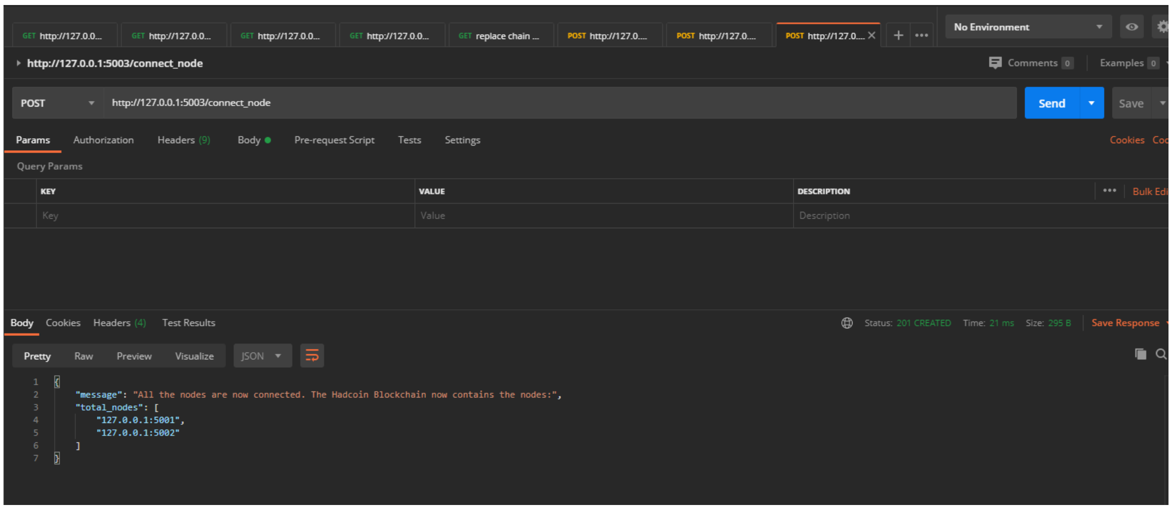


POST http://127.0.0.1:5002/connect_node

Status: 201 CREATED Time: 33 ms Size: 295 B

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

node 3:

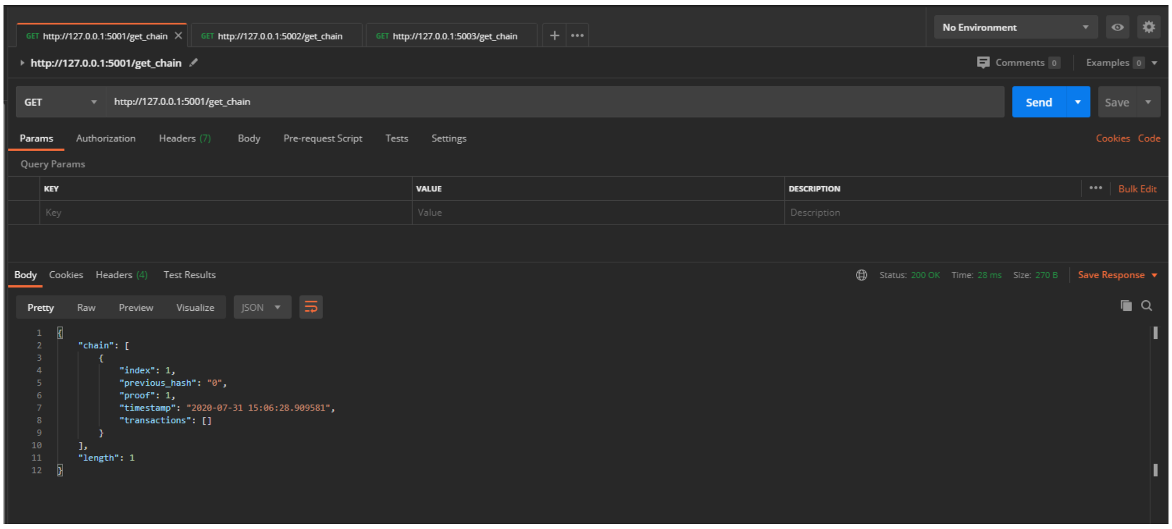


POST http://127.0.0.1:5003/connect_node

Status: 201 CREATED Time: 21 ms Size: 295 B

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

3. For each node, create a new blockchain, using the "**get_chain**" GET request

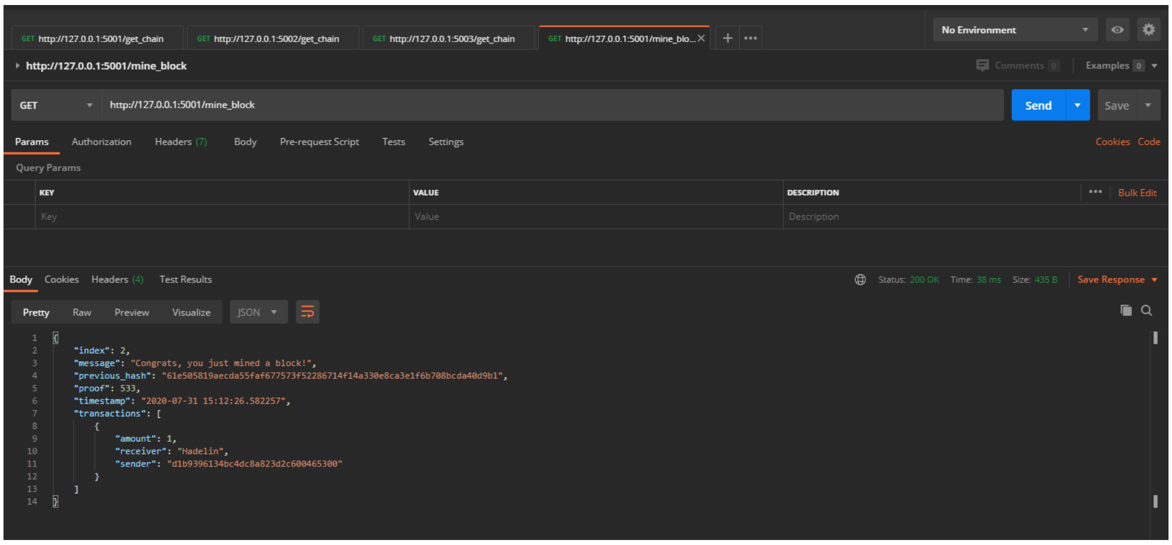


GET http://127.0.0.1:5001/get_chain

Status: 200 OK Time: 28 ms Size: 270 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2020-07-31 15:06:28.909581",
            "transactions": []
        }
    ],
    "length": 1
}
```

4. Once, the initial blockchain is created for each node, we have a problem. Each node's chain was created at a different timestamp so they are already different. So, we will fix that using the **replace_chain** GET method to achieve a consensus amount them, but in order to do that one of the node's blockchain needs to be the longest. So, using one of the nodes, mine a block, using **mine_block** GET method.



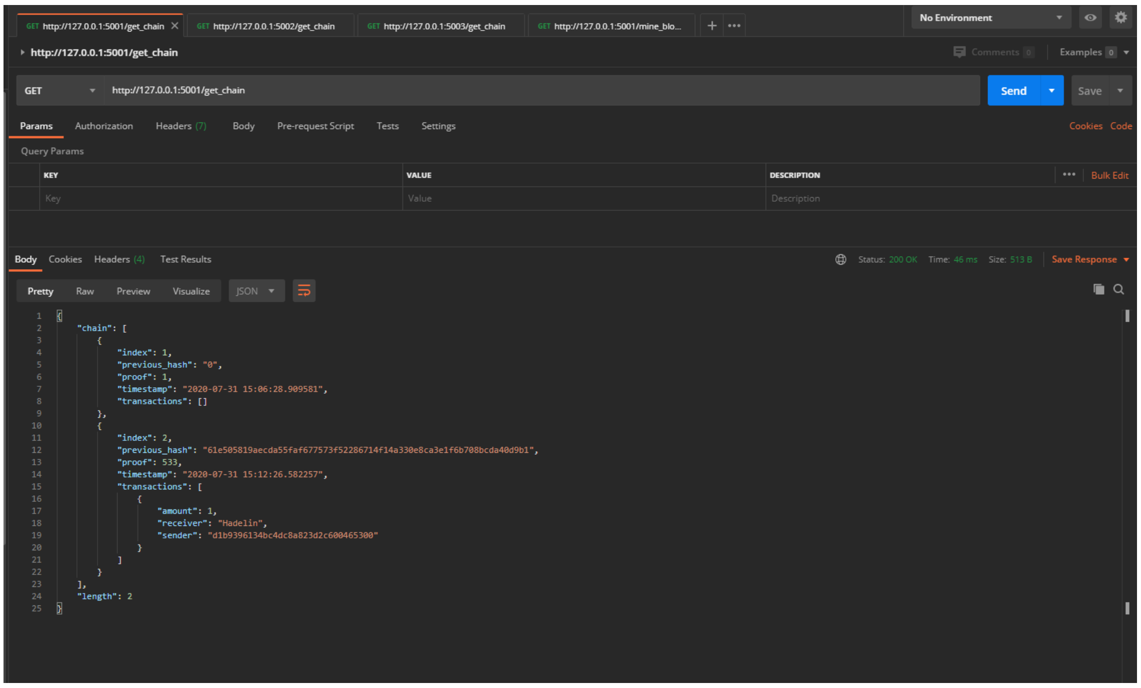
GET http://127.0.0.1:5001/mine_block

Status: 200 OK Time: 38 ms Size: 435 B

```
{
    "index": 2,
    "message": "Congrats, you just mined a block!",
    "previous_hash": "61e505819aecda55faf677573f52286714f14a330e8ca3e1f6b708bcda40d9b1",
    "proof": 533,
    "timestamp": "2020-07-31 15:12:26.582257",
    "transactions": [
        {
            "amount": 1,
            "receiver": "Hadelin",
            "sender": "d1b9396134bc4dc8a823d2c600465300"
        }
    ]
}
```

5. If we check the block chain of the node who mined the block, we will get the blockchain with newly added block(without any transactions yet). Again using the **get_chain** GET method for node who mined the block and the one of the nodes which didn't.

   Miner node:

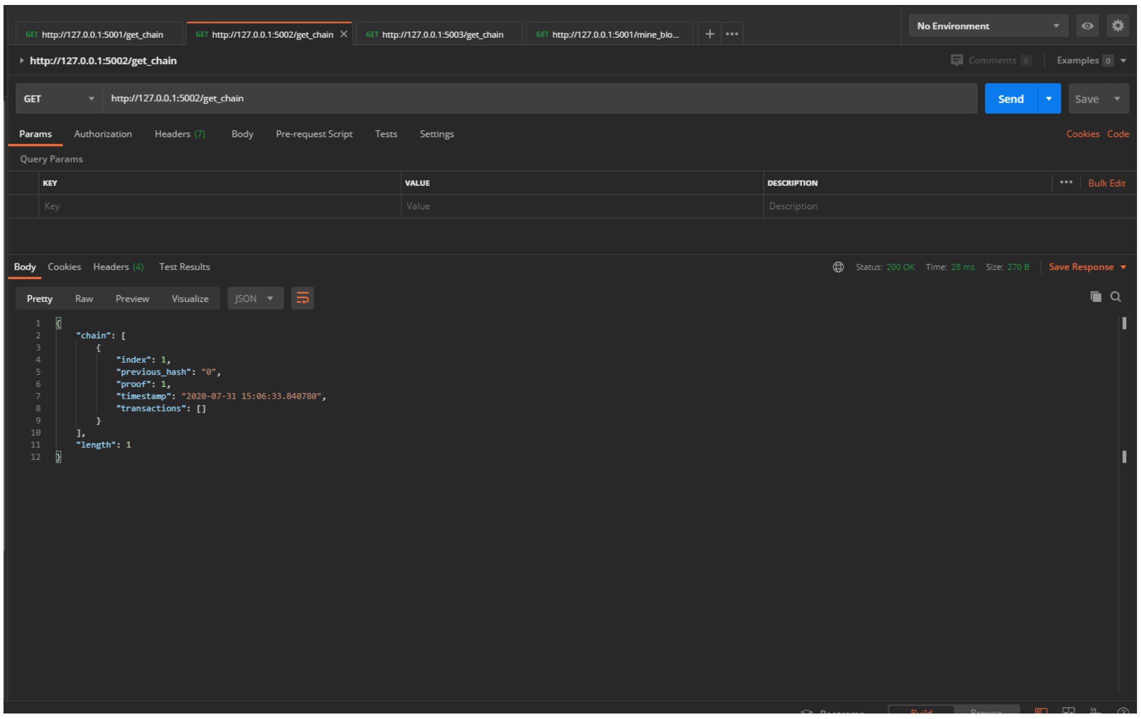


GET http://127.0.0.1:5001/get_chain

Status: 200 OK Time: 46 ms Size: 513 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2020-07-31 15:06:28.909581",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "61e505819aecda55faf677573f52286714f14a330e8ca3e1f6b708bcda40d9b1",
            "proof": 533,
            "timestamp": "2020-07-31 15:12:26.582257",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        }
    ],
    "length": 2
}
```

   Other nodes:

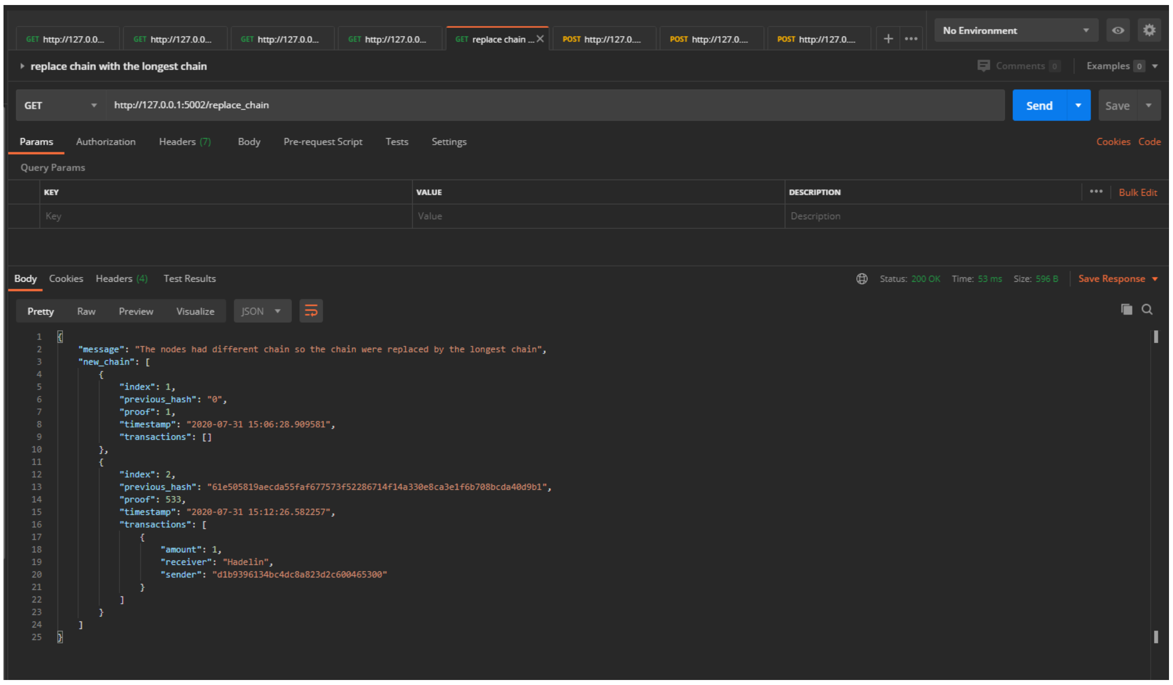


GET http://127.0.0.1:5002/get_chain

Status: 200 OK Time: 28 ms Size: 270 B

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2020-07-31 15:06:33.840780",
            "transactions": []
        }
    ],
    "length": 1
}
```

6. Clearly, all the nodes are out of sync, but we have a blockchain which is the longest. Hence, we will establish consensus by using the **replace_chain** GET method.



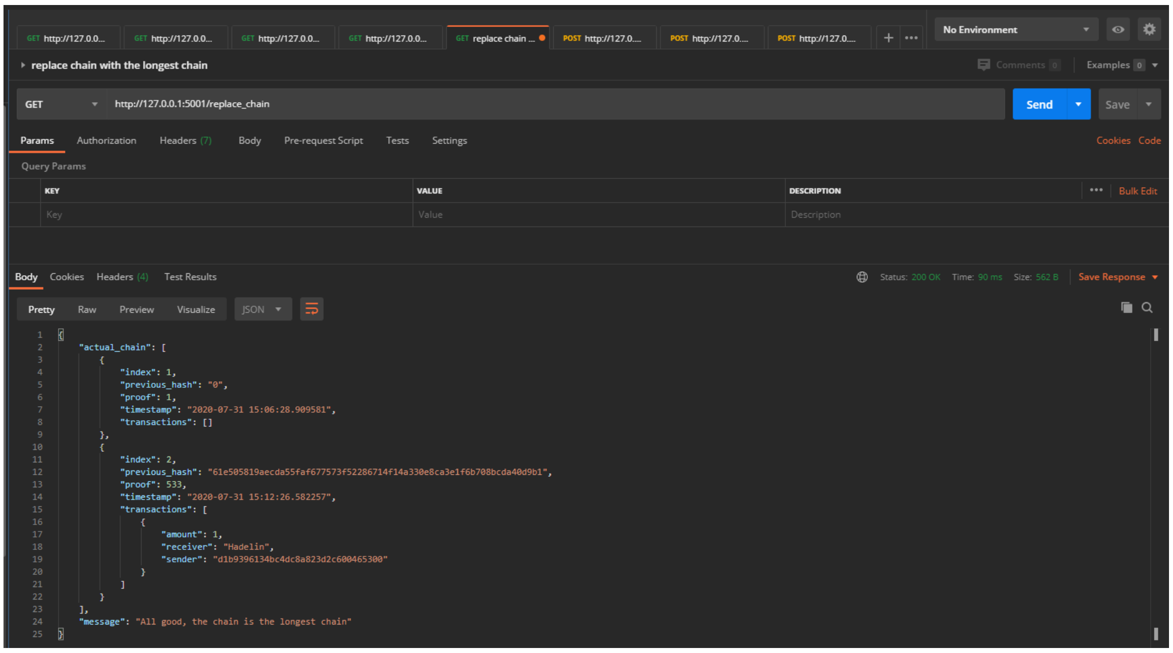
```
GET http://127.0.0.1:5002/replace_chain
```

```
Status: 200 OK   Time: 53 ms   Size: 596 B
```

```json
{
    "message": "The nodes had different chain so the chain were replaced by the longest chain",
    "new_chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2020-07-31 15:06:28.909581",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "61e505819aecda55faf677573f52286714f14a330e8ca3e1f6b708bcda40d9b1",
            "proof": 533,
            "timestamp": "2020-07-31 15:12:26.582257",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        }
    ]
}
```

Do the same for all other node.

If a node already had the longest chain, then the response would have been:

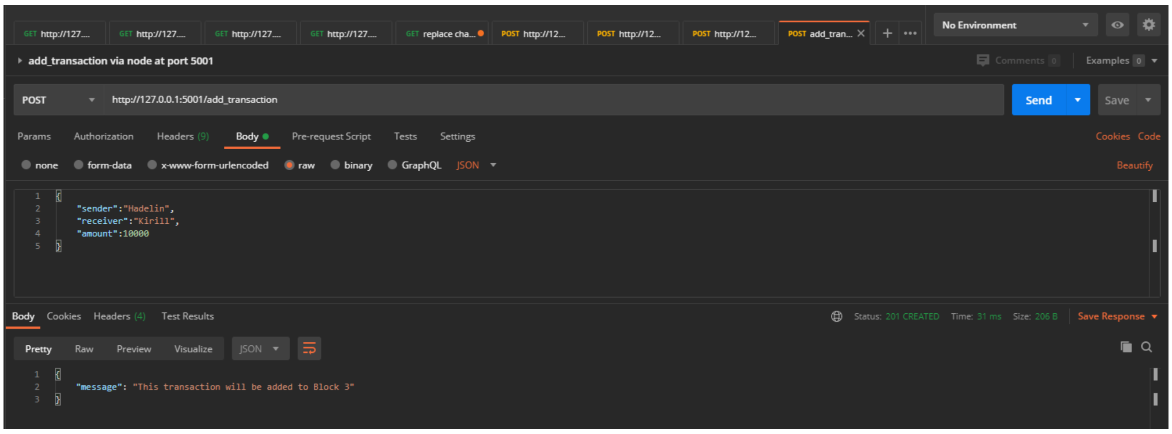


```
GET http://127.0.0.1:5001/replace_chain
```

```
Status: 200 OK   Time: 90 ms   Size: 562 B
```

```json
{
    "actual_chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2020-07-31 15:06:28.909581",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "61e505819aecda55faf677573f52286714f14a330e8ca3e1f6b708bcda40d9b1",
            "proof": 533,
            "timestamp": "2020-07-31 15:12:26.582257",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        }
    ],
    "message": "All good, the chain is the longest chain"
}
```

7. Now that we have a peer-to-peer block chain network setup, we can add transactions using the **add_transaction** POST method. Note, all the transactions will be added to a node's blockchain until a new block has been mined and transactions are stored in them:



add_transaction via node at port 5001

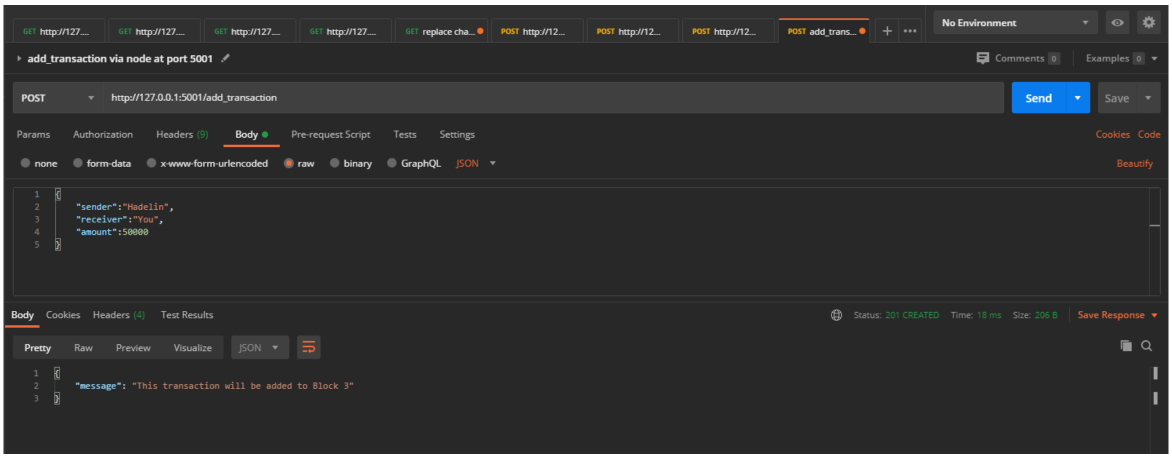
POST http://127.0.0.1:5001/add_transaction

Body: raw JSON

```
{
    "sender":"Hadelin",
    "receiver":"Kirill",
    "amount":10000
}
```

Status: 201 CREATED Time: 31 ms Size: 206 B

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



add_transaction via node at port 5001

POST http://127.0.0.1:5001/add_transaction

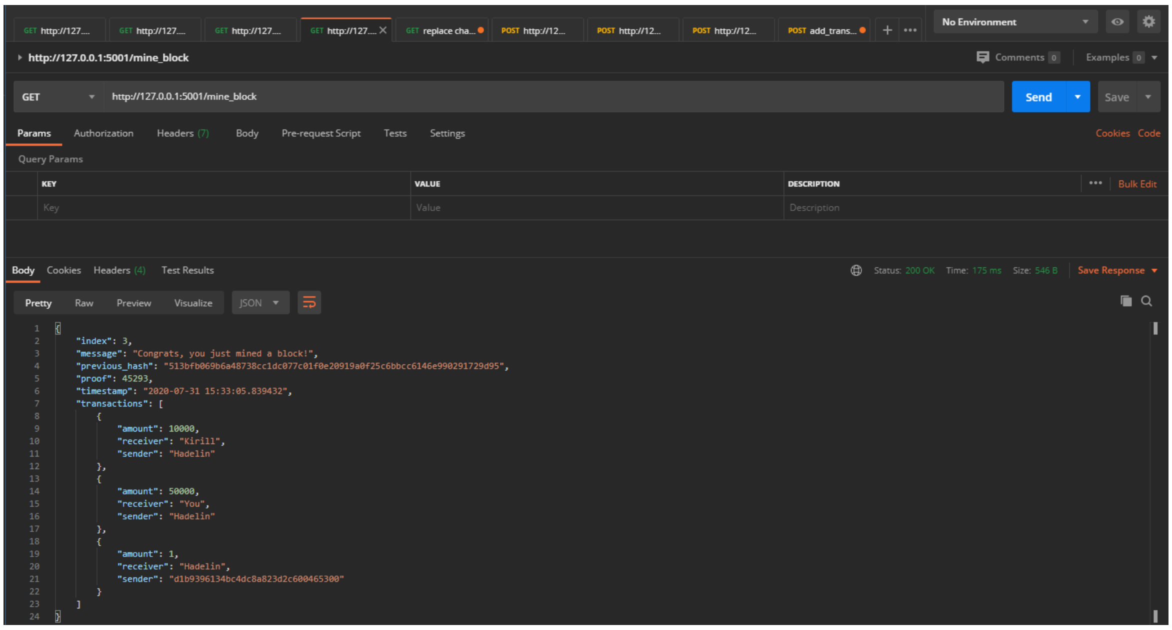
Body: raw JSON

```
{
    "sender":"Hadelin",
    "receiver":"You",
    "amount":50000
}
```

Status: 201 CREATED Time: 18 ms Size: 206 B

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

8. Now if we mine a block using which we added the transactions



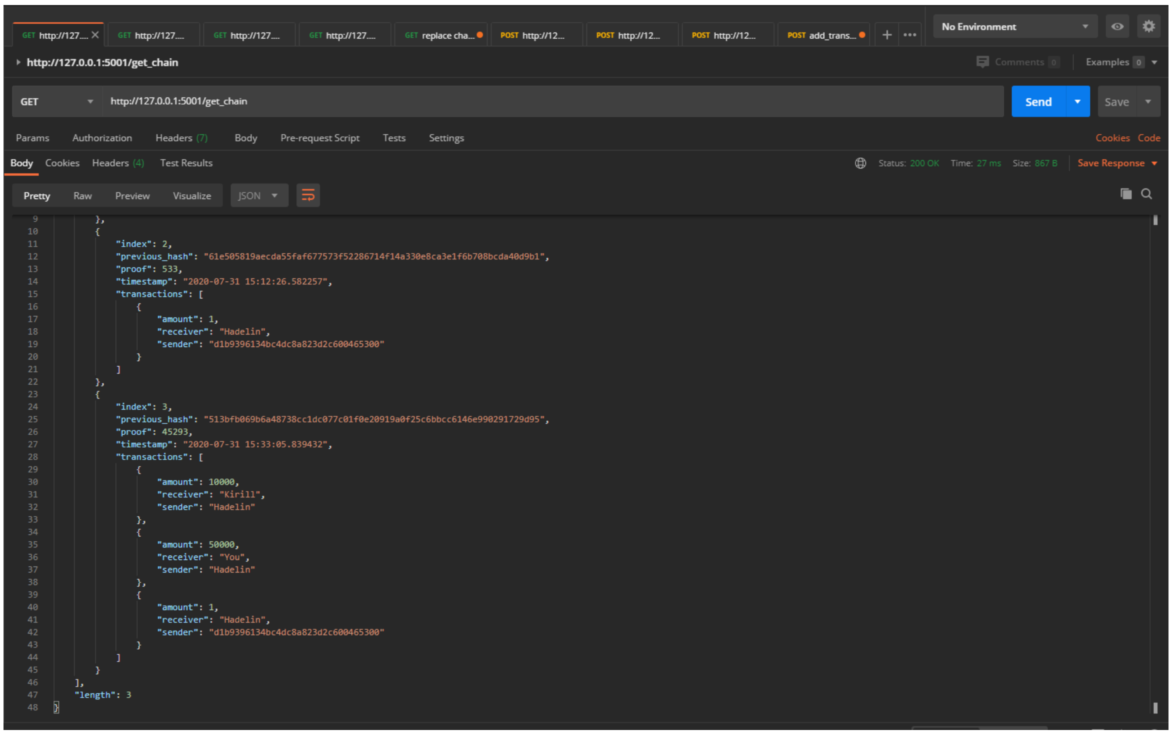
http://127.0.0.1:5001/mine_block

GET http://127.0.0.1:5001/mine_block

Status: 200 OK Time: 175 ms Size: 546 B

```
{
    "index": 3,
    "message": "Congrats, you just mined a block!",
    "previous_hash": "513bfb069b6a48738cc1dc077c01f0e20919a0f25c6bbcc6146e990291729d95",
    "proof": 45293,
    "timestamp": "2020-07-31 15:33:05.839432",
    "transactions": [
        {
            "amount": 10000,
            "receiver": "Kirill",
            "sender": "Hadelin"
        },
        {
            "amount": 50000,
            "receiver": "You",
            "sender": "Hadelin"
        },
        {
            "amount": 1,
            "receiver": "Hadelin",
            "sender": "d1b9396134bc4dc8a823d2c600465300"
        }
    ]
}
```

9. Check the blockchain of the miner node:



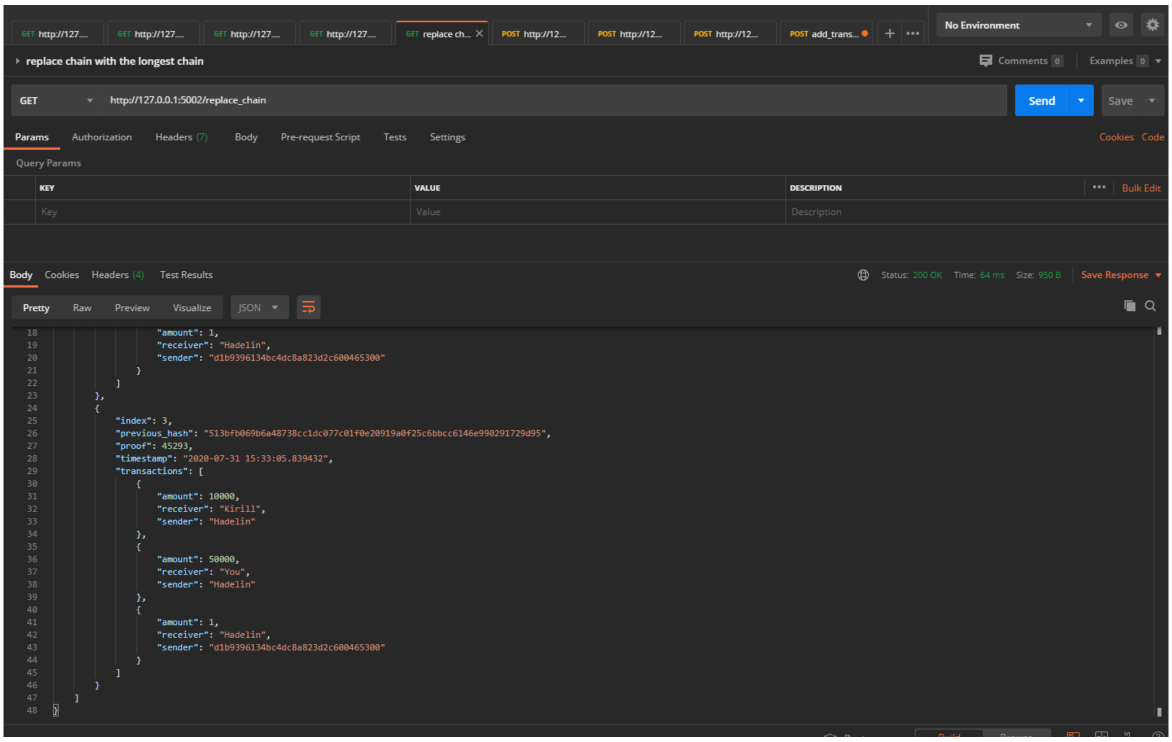
http://127.0.0.1:5001/get_chain

GET http://127.0.0.1:5001/get_chain

Status: 200 OK Time: 27 ms Size: 867 B

```
        },
        {
            "index": 2,
            "previous_hash": "61e505819aecda55faf677573f52286714f14a330e8ca3e1f6b708bcda40d9b1",
            "proof": 533,
            "timestamp": "2020-07-31 15:12:26.582257",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        },
        {
            "index": 3,
            "previous_hash": "513bfb069b6a48738cc1dc077c01f0e20919a0f25c6bbcc6146e990291729d95",
            "proof": 45293,
            "timestamp": "2020-07-31 15:33:05.839432",
            "transactions": [
                {
                    "amount": 10000,
                    "receiver": "Kirill",
                    "sender": "Hadelin"
                },
                {
                    "amount": 50000,
                    "receiver": "You",
                    "sender": "Hadelin"
                },
                {
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        }
    ],
    "length": 3
}
```

10. Run the **replace_chain** GET method for other nodes to achieve consensus.



replace chain with the longest chain

GET http://127.0.0.1:5002/replace_chain

Status: 200 OK Time: 64 ms Size: 950 B

```
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        },
        {
            "index": 3,
            "previous_hash": "513bfb069b6a48738cc1dc077c01f0e20919a0f25c6bbcc6146e990291729d95",
            "proof": 45293,
            "timestamp": "2020-07-31 15:33:05.839432",
            "transactions": [
                {
                    "amount": 10000,
                    "receiver": "Kirill",
                    "sender": "Hadelin"
                },
                {
                    "amount": 50000,
                    "receiver": "You",
                    "sender": "Hadelin"
                },
                {
                    "amount": 1,
                    "receiver": "Hadelin",
                    "sender": "d1b9396134bc4dc8a823d2c600465300"
                }
            ]
        }
    ]
}
```