网络与信息安全期末实验报告

——区块链

### 本次实验目的

〉 1. 了解区块链是个什么东西。

〉 2. 搭建一个自己的区块链。

### 实验平台

Ubuntu20.04虚拟机

### 实验原理

###### 1.区块链

区块链本质上是一个去中心化数据库。是一种分布式数据存储，点对点传输，共识机制，加密算法等计算机技术的新型应用模式，具有数据难以篡改和去中心化的两大核心特点。

区块链中的“区块”是指信息块，用于记录一定时间内发生的所有交易或事件，并且每个区块都被赋予了一个唯一的标识——时间戳和区块体。时间戳确保了区块按时间顺序排列，而区块体则包含了交易或事件的具体信息。每个区块还包含了一个指向其前一个区块的指针，这个指针实际上是一个哈希值，用于确保数据的完整性。通过这种方式，所有的区块都被串联起来，形成了一个链条，即“区块链”。

随着比特币的成功，人们开始意识到区块链技术的潜力不仅仅局限于数字货币领域，而是可以应用于更多的领域。

###### 2.比特币

比特币是一种基于区块链技术的加密货币，它使用去中心化的支付系统，允许用户在没有中央管理机构的情况下进行点对点的交易。

比特币的交易记录被存储在被称为“区块链”的公共账本上，这个账本是由网络中的所有参与者共同维护的。每个比特币交易都需要经过一个被称为“挖矿”的复杂计算过程来验证，这个过程确保了交易的合法性和真实性。一旦交易被验证并添加到区块链上，它就无法被更改或删除，从而保证了交易的安全性和可靠性。

###### 3.创造一个自己的区块链

import hashlib

import json

from time import time

from uuid import uuid4

class Blockchain(object):

    def \_\_init\_\_(self):

        self.current\_transactions = []

        self.chain = []

        # 创建创世区块

        self.new\_block(previous\_hash=1, proof=100)

    def new\_block(self, proof, previous\_hash=None):

        """

        创建一个新的区块到区块链中

        :param proof: <int> 由工作证明算法生成的证明

        :param previous\_hash: (Optional) <str> 前一个区块的 hash 值

        :return: <dict> 新区块

        """

        block = {

            'index': len(self.chain) + 1,

            'timestamp': time(),

            'transactions': self.current\_transactions,

            'proof': proof,

            'previous\_hash': previous\_hash or self.hash(self.chain[-1]),

        }

        # 重置当前交易记录

        self.current\_transactions = []

        self.chain.append(block)

        return block

    def new\_transaction(self, sender, recipient, amount):

        """

        创建一笔新的交易到下一个被挖掘的区块中

        :param sender: <str> 发送人的地址

        :param recipient: <str> 接收人的地址

        :param amount: <int> 金额

        :return: <int> 持有本次交易的区块索引

        """

        self.current\_transactions.append({

            'sender': sender,

            'recipient': recipient,

            'amount': amount,

        })

        return self.last\_block['index'] + 1

    @staticmethod

    def hash(block):

        """

        给一个区块生成 SHA-256 值

        :param block: <dict> Block

        :return: <str>

        """

        # 我们必须确保这个字典（区块）是经过排序的，否则我们将会得到不一致的散列

        block\_string = json.dumps(block, sort\_keys=True).encode()

        return hashlib.sha256(block\_string).hexdigest()

    @property

    def last\_block(self):

        return self.chain[-1]

def proof\_of\_work(self, last\_proof, difficulty=4):

        """

        Simple Proof of Work Algorithm:

        - Find a number p' such that hash(pp') contains leading `difficulty` zeroes,

          where p is the previous proof and p' is the new proof.

        :param last\_proof: <int> Previous Proof

        :param difficulty: <int> Number of leading zeroes required

        :return: <int> New Proof

        """

        proof = 0

        while not self.valid\_proof(last\_proof, proof, difficulty):

            proof += 1

        return proof

    @staticmethod

    def valid\_proof(last\_proof, proof, difficulty):

        """

        Validates the Proof: Does hash(last\_proof, proof) contain `difficulty` leading zeroes?

        :param last\_proof: <int> Previous Proof

        :param proof: <int> Current Proof

        :param difficulty: <int> Number of leading zeroes required

        :return: <bool> True if correct, False otherwise.

        """

        guess = f'{last\_proof}{proof}'.encode()

        guess\_hash = hashlib.sha256(guess).hexdigest()

        # Dynamically check the required number of leading zeroes

        return guess\_hash[:difficulty] == "0" \* difficulty

###### 4.Blockchain 作为 API 接口

创建三个接口：

/transactions/new 创建一个交易并添加到区块

/mine 告诉服务器去挖掘新的区块

/chain 返回整个区块链

@app.route('/mine', methods=['GET'])

def mine():

    # We run the proof of work algorithm to get the next proof...

    last\_block = blockchain.last\_block

    proof = blockchain.proof\_of\_work(last\_block)

    # We must receive a reward for finding the proof.

    # The sender is "0" to signify that this node has mined a new coin.

    blockchain.new\_transaction(

        sender="0",

        recipient=node\_identifier,

        amount=1,

    )

    # Forge the new Block by adding it to the chain

    previous\_hash = blockchain.hash(last\_block)

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

    response = {

        'message': "New Block Forged",

        'index': block['index'],

        'transactions': block['transactions'],

        'proof': block['proof'],

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

    }

    return jsonify(response), 200

@app.route('/transactions/new', methods=['POST'])

def new\_transaction():

    values = request.get\_json()

    # Check that the required fields are in the POST'ed data

    required = ['sender', 'recipient', 'amount']

    if not all(k in values for k in required):

        return 'Missing values', 400

    # Create a new Transaction

    index = blockchain.new\_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}

    return jsonify(response), 201

@app.route('/chain', methods=['GET'])

def full\_chain():

    response = {

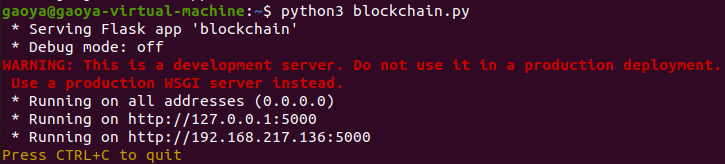
        'chain': blockchain.chain,

        'length': len(blockchain.chain),

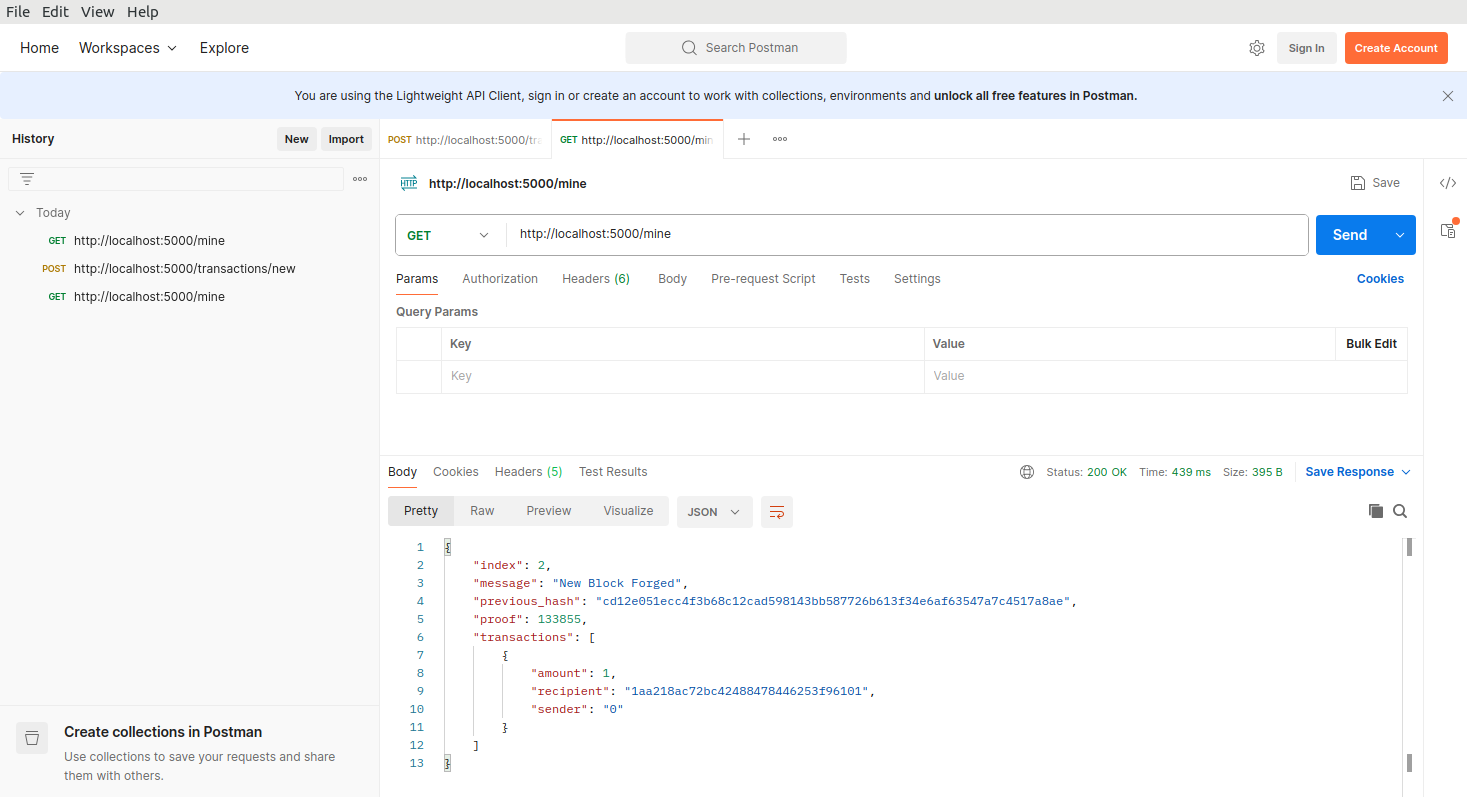
    }

    return jsonify(response), 200

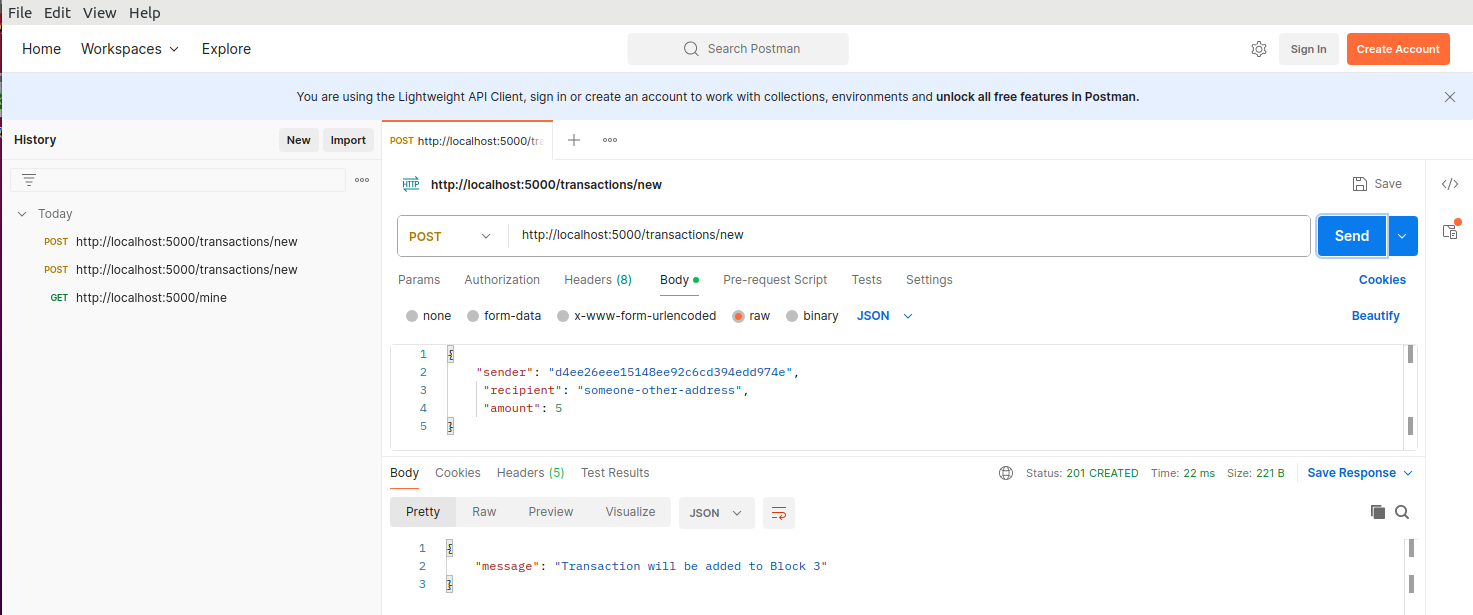
###### 5.运行区块链



请求 http://localhost:5000/mine （ GET ）来进行挖矿



请求 http://localhost:5000/transactions/new （POST）



###### 6.一致性（共识）

在区块链这种去中心化的分布式账本系统中，要确保所有节点都能达成共识，就必须实现一个一致性的算法，否则由于网络延迟、节点故障等因素，不同节点上的数据可能会出现不一致的情况。

注册节点：新增两个接口

/nodes/register

/nodes/resolve

    def register\_node(self, address):

        """

        Add a new node to the list of nodes

        :param address: Address of node. Eg. 'http://192.168.0.5:5000'

        """

        parsed\_url = urlparse(address)

        if parsed\_url.netloc:

            self.nodes.add(parsed\_url.netloc)

        elif parsed\_url.path:

            # Accepts an URL without scheme like '192.168.0.5:5000'.

            self.nodes.add(parsed\_url.path)

        else:

            raise ValueError('Invalid URL')

实现共识算法

    def valid\_chain(self, chain):

        """

        Determine if a given blockchain is valid

        :param chain: <list> A blockchain

        :return: <bool> True if valid, False if not

        """

        last\_block = chain[0]

        current\_index = 1

        while current\_index < len(chain):

            block = chain[current\_index]

            print(f'{last\_block}')

            print(f'{block}')

            print("\n-----------\n")

            # Check that the hash of the block is correct

            if block['previous\_hash'] != self.hash(last\_block):

                return False

            # Check that the Proof of Work is correct

            if not self.valid\_proof(last\_block['proof'], block['proof']):

                return False

            last\_block = block

            current\_index += 1

        return True

    def resolve\_conflicts(self):

        """

        This is our Consensus Algorithm, it resolves conflicts

        by replacing our chain with the longest one in the network.

        :return: <bool> True if our chain was replaced, False if not

        """

        neighbours = self.nodes

        new\_chain = None

        # We're only looking for chains longer than ours

        max\_length = len(self.chain)

        # Grab and verify the chains from all the nodes in our network

        for node in neighbours:

            response = requests.get(f'http://{node}/chain')

            if response.status\_code == 200:

                length = response.json()['length']

                chain = response.json()['chain']

                # Check if the length is longer and the chain is valid

                if length > max\_length and self.valid\_chain(chain):

                    max\_length = length

                    new\_chain = chain

        # Replace our chain if we discovered a new, valid chain longer than ours

        if new\_chain:

            self.chain = new\_chain

            return True

        return False

将两个端点注册到我们的API中

@app.route('/nodes/register', methods=['POST'])

def register\_nodes():

    values = request.get\_json()

    nodes = values.get('nodes')

    if nodes is None:

        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:

        blockchain.register\_node(node)

    response = {

        'message': 'New nodes have been added',

        'total\_nodes': list(blockchain.nodes),

    }

    return jsonify(response), 201

@app.route('/nodes/resolve', methods=['GET'])

def consensus():

    replaced = blockchain.resolve\_conflicts()

    if replaced:

        response = {

            'message': 'Our chain was replaced',

            'new\_chain': blockchain.chain

        }

    else:

        response = {

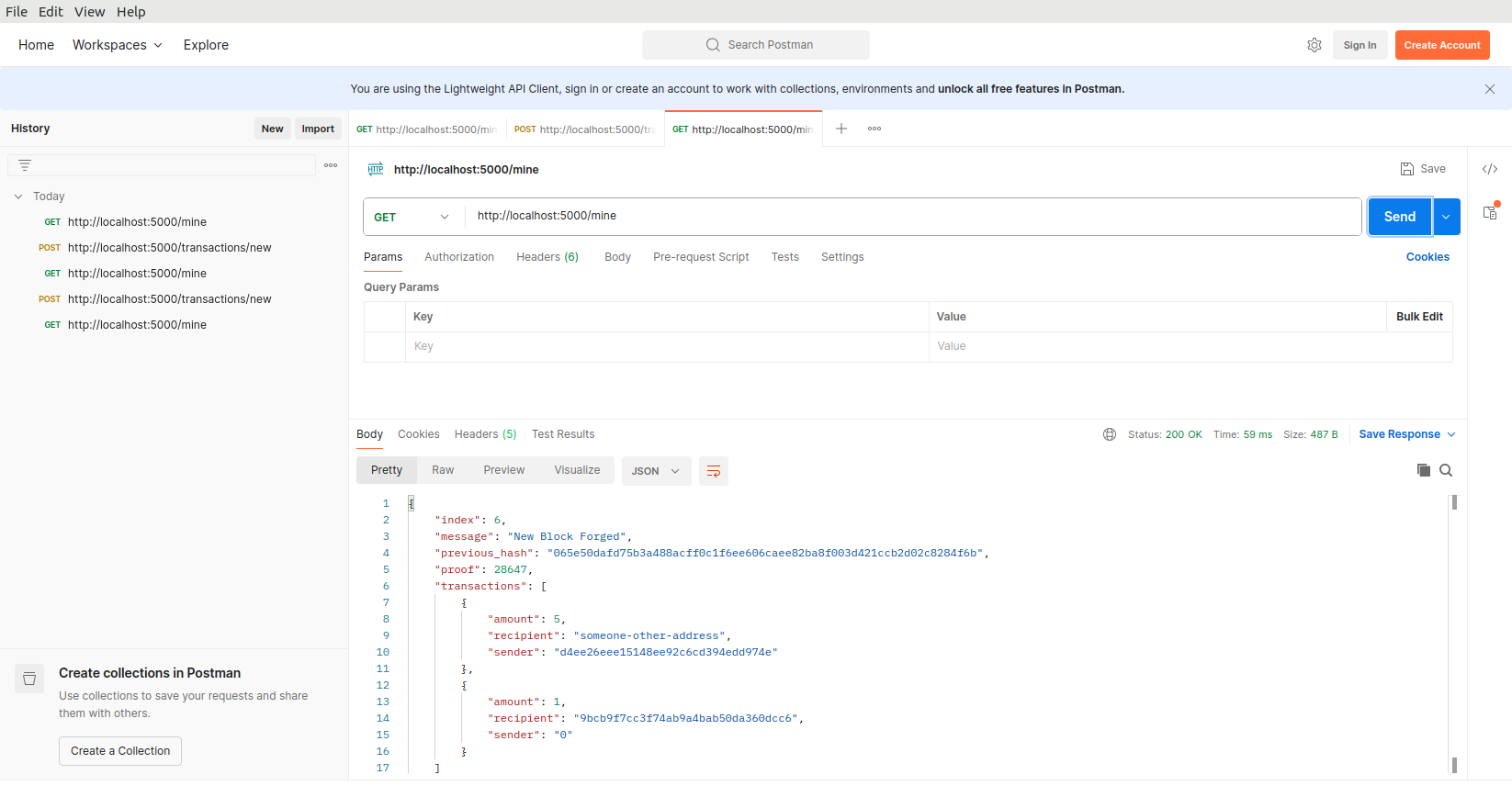
            'message': 'Our chain is authoritative',

            'chain': blockchain.chain

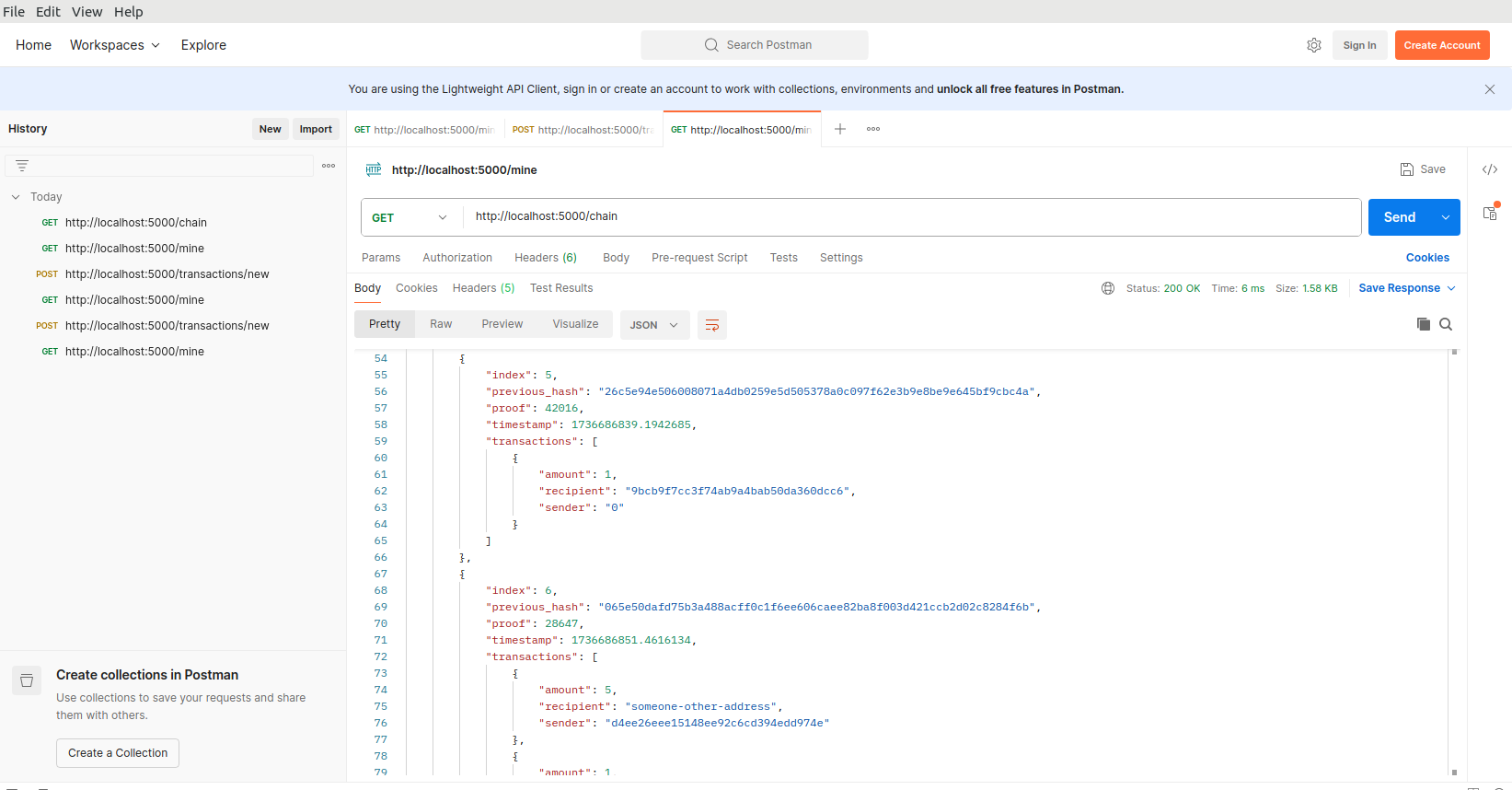
        }

    return jsonify(response), 200

测试



请求 http://localhost:5000/chain 得到所有的块信息



###### 问题

1.

index：区块的索引号，用于快速定位块timestamp：记录区块被创建的时间，用于确保块的顺序和验证数据的时效性

transactions：包含区块中所有的交易记录，是区块链的主要信息负载，用于价值或信息的转移

proof：工作量证明，一种共识机制，用于验证区块的创建是否经过了足够的计算工作，确保了区块链的安全性和不可篡改性，因为恶意攻击者很难重新编写区块链历史

previous\_hash：前一个块的哈希值，用于确保链的完整性和连续性，任何对链中块的修改都会破坏这一连续性

2.

proof\_of\_work 使用暴力法，通过不断递增 proof 来找到合适的值。由于原本代码中目标难度是固定的前4个字符，valid\_proof中只检查哈希的前4个字符是否为零，这个难度较低，当目标难度增加时，计算时间会显著增加，效率降低，对于较大规模或高难度的区块链网络，这种方式可能会导致性能瓶颈。

修改：

默认难度仍然是4，添加difficulty参数，可以动态调整目标难度

    def proof\_of\_work(self, last\_proof, difficulty=4):

        """

        Simple Proof of Work Algorithm:

        - Find a number p' such that hash(pp') contains leading `difficulty` zeroes,

          where p is the previous proof and p' is the new proof.

        :param last\_proof: <int> Previous Proof

        :param difficulty: <int> Number of leading zeroes required

        :return: <int> New Proof

        """

        proof = 0

        while not self.valid\_proof(last\_proof, proof, difficulty):

            proof += 1

        return proof

    @staticmethod

    def valid\_proof(last\_proof, proof, difficulty):

        """

        Validates the Proof: Does hash(last\_proof, proof) contain `difficulty` leading zeroes?

        :param last\_proof: <int> Previous Proof

        :param proof: <int> Current Proof

        :param difficulty: <int> Number of leading zeroes required

        :return: <bool> True if correct, False otherwise.

        """

        guess = f'{last\_proof}{proof}'.encode()

        guess\_hash = hashlib.sha256(guess).hexdigest()

        # Dynamically check the required number of leading zeroes

        return guess\_hash[:difficulty] == "0" \* difficulty

3.

实验中的区块链网络通常是单节点或少数节点进行模拟，缺乏真实区块链网络中去中心化和节点之间的信任机制。在实际区块链中，区块链节点分布更加广泛。

4.

区块链技术带来了数据去中心化存储的革命，使得数据更加透明、安全、不可篡改。它不仅能在金融领域实现去中心化交易，还能在供应链、身份认证等多个行业得到应用。

虽然我们搭建的是一个非常基础的区块链，但它已经具备了区块链的核心特性，去中心化、数据不可篡改和链式结构。我已经初步理解了区块链的内部机制。