Blockchain Workshop

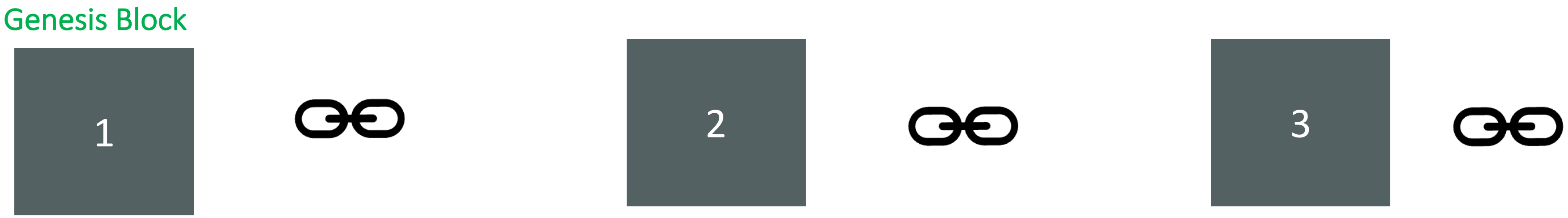
Learn Blockchain by Building One

Presenter: Samaneh Miri

July 31, 2020

Documentation

Create a Blockchain



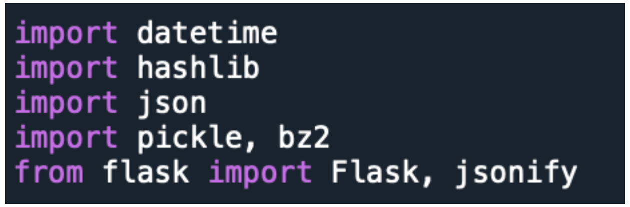
1. Index
2. Timestamp
3. Proof (Nonce)
4. Prev. Hash:
5. Hash:

Create a Blockchain

- Initializing a blockchain
- Mining a block
- Adding the block to the chain
- Checking if the chain is valid
- Getting a full blockchain

Building a Blockchain/Libraries

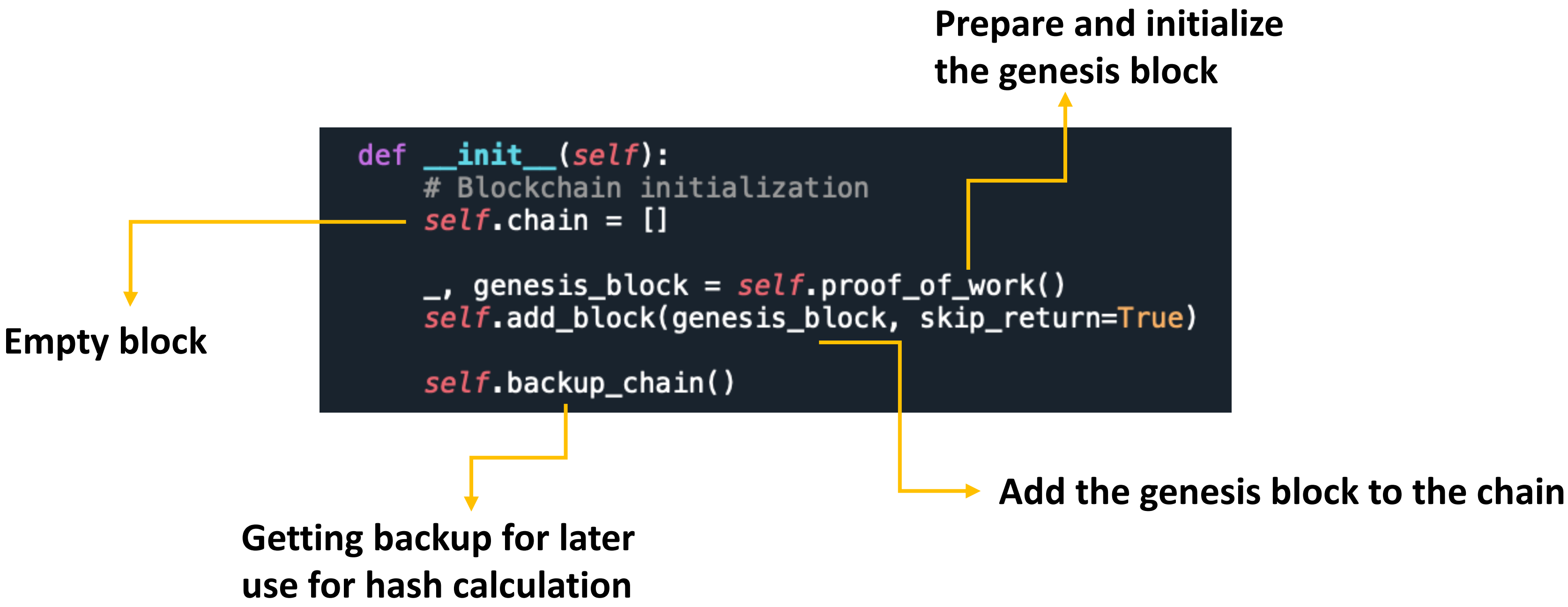
- Datetime
  - Each block needs timestamp which indicates the exact date when the block is created.
- Hashlib
  - It is needed to hash the block.
- Json
  - Json function is used to encode blocks before hashing them.
- Pickle
  - It is used for serializing and de-serializing to use the python object later.
- Bz2
  - It is used for compression.
- Flask, jsonify
  - Help to interact with the web application.



```
import datetime
import hashlib
import json
import pickle, bz2
from flask import Flask, jsonify
```

Building a Blockchain/Methods

- __init__()



```
def __init__(self):
    # Blockchain initialization
    self.chain = []

    _, genesis_block = self.proof_of_work()
    self.add_block(genesis_block, skip_return=True)

    self.backup_chain()
```

Prepare and initialize the genesis block

Empty block

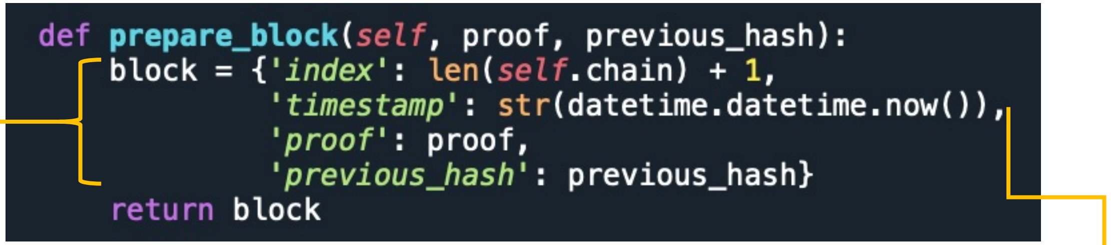
Add the genesis block to the chain

Getting backup for later use for hash calculation

Building a Blockchain/Methods

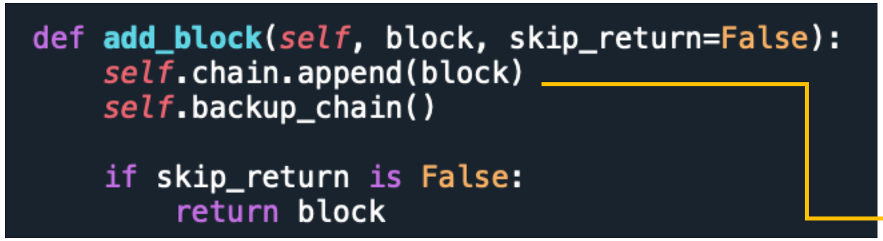
- prepare_block()/add_block()

Block is a dictionary with 4 keys



```
def prepare_block(self, proof, previous_hash):
    block = {'index': len(self.chain) + 1,
             'timestamp': str(datetime.datetime.now()),
             'proof': proof,
             'previous_hash': previous_hash}
    return block
```

Exact time the block was mined



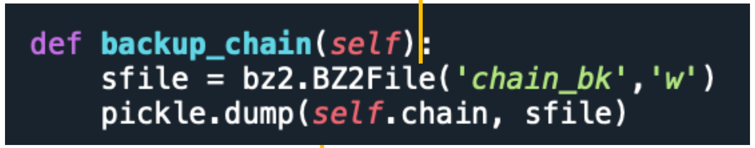
```
def add_block(self, block, skip_return=False):
    self.chain.append(block)
    self.backup_chain()

    if skip_return is False:
        return block
```

Add the mined block to the chain

Building a Blockchain/Methods

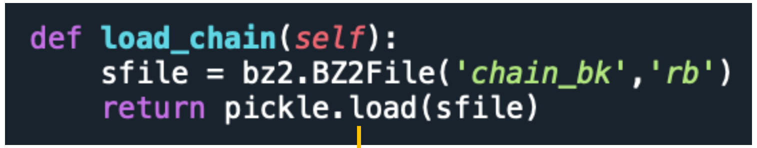
- Pickling/unpickling

BZ2File class for reading and writing compressed files.



```python
def backup_chain(self):
    sfile = bz2.BZ2File('chain_bk','w')
    pickle.dump(self.chain, sfile)
```

Serializing the chain into chain_bk file

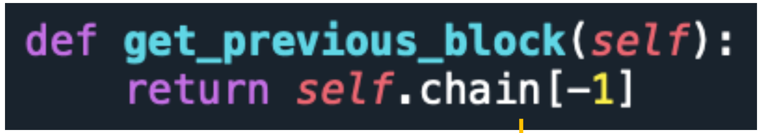


```python
def load_chain(self):
    sfile = bz2.BZ2File('chain_bk','rb')
    return pickle.load(sfile)
```

Deserializing the sfile

Building a Blockchain/Methods

- get_previous_block()



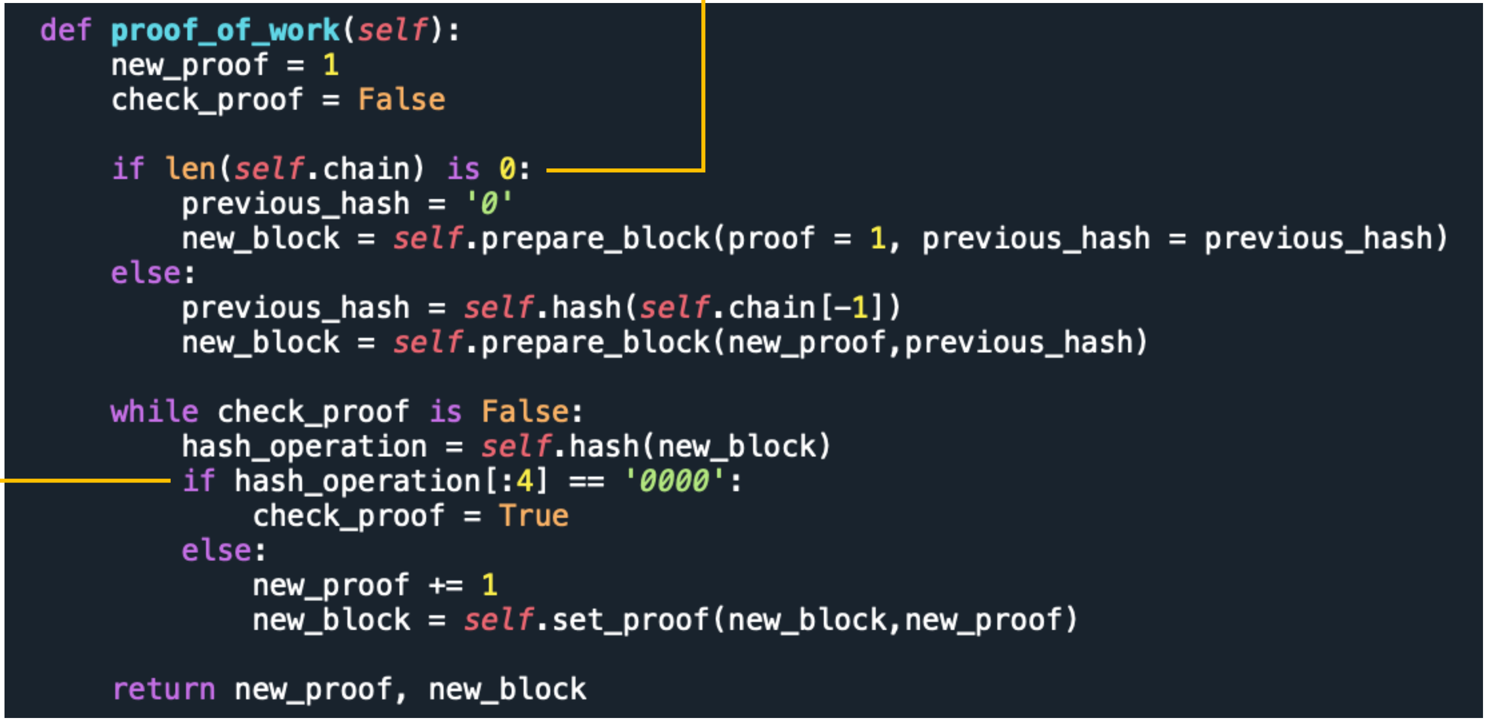
```
def get_previous_block(self):
    return self.chain[-1]
```

returns the last block of the chain

Building a Blockchain/Methods

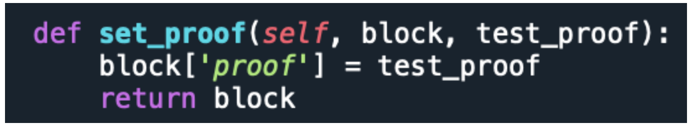
- proof_of_work

Preparing the Genesis Block



```
def proof_of_work(self):
    new_proof = 1
    check_proof = False

    if len(self.chain) is 0:
        previous_hash = '0'
        new_block = self.prepare_block(proof = 1, previous_hash = previous_hash)
    else:
        previous_hash = self.hash(self.chain[-1])
        new_block = self.prepare_block(new_proof,previous_hash)

    while check_proof is False:
        hash_operation = self.hash(new_block)
        if hash_operation[:4] == '0000':
            check_proof = True
        else:
            new_proof += 1
            new_block = self.set_proof(new_block,new_proof)

    return new_proof, new_block
```

Check if the hash is under target value or not (start with 4 zeros)



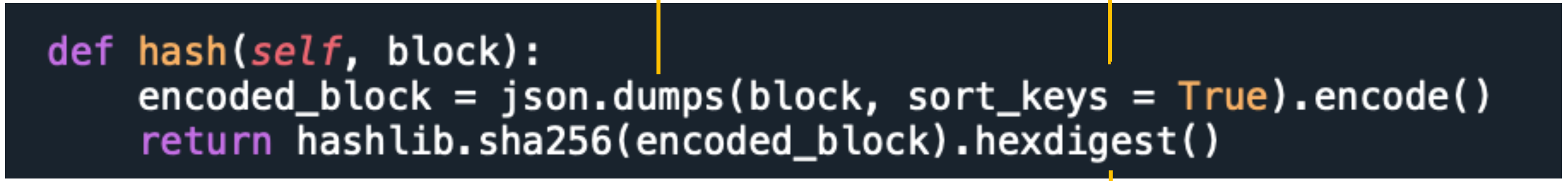
```
def set_proof(self, block, test_proof):
    block['proof'] = test_proof
    return block
```

Building a Blockchain/Methods

- hash

returns a string format of block acceptable for hash sha256

block which is a dictionary is sorted by keys



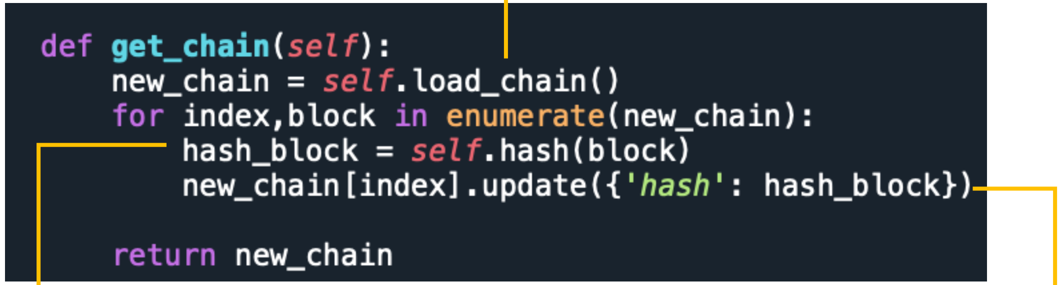
```python
def hash(self, block):
    encoded_block = json.dumps(block, sort_keys = True).encode()
    return hashlib.sha256(encoded_block).hexdigest()
```

returns a string of 64 characters hash

Building a Blockchain/Methods

- get_chain

Load the backup chain



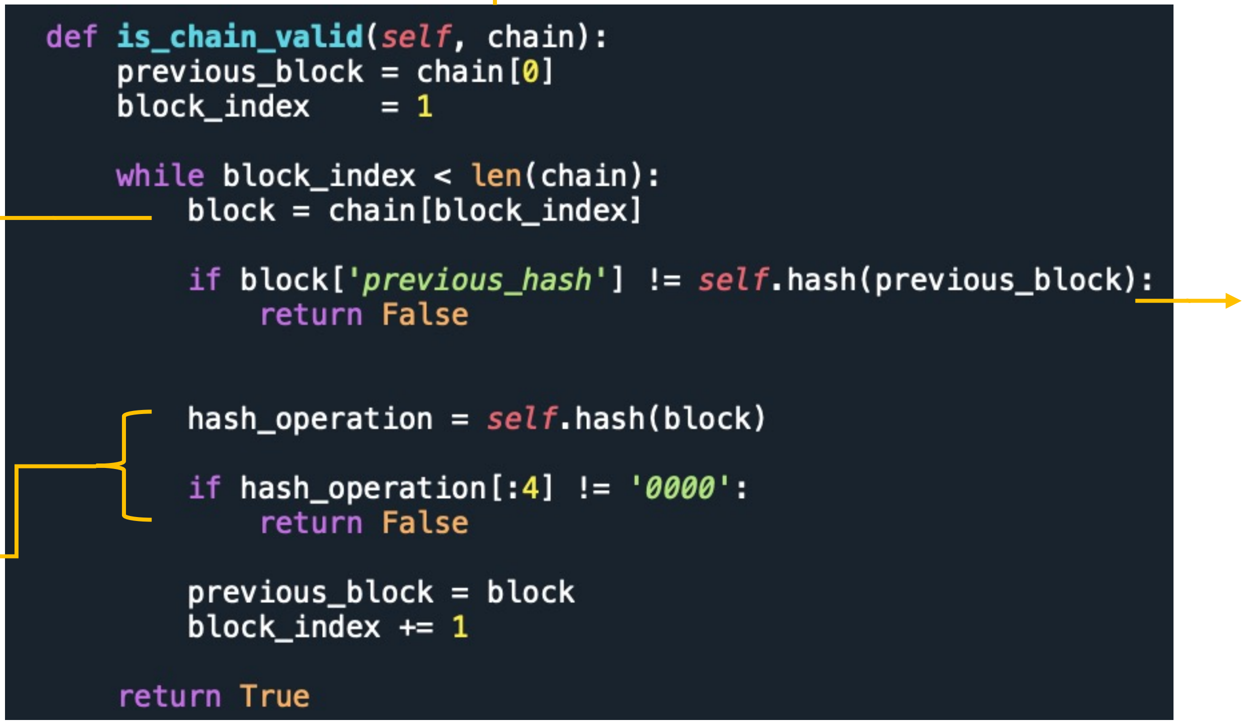
```
def get_chain(self):
    new_chain = self.load_chain()
    for index,block in enumerate(new_chain):
        hash_block = self.hash(block)
        new_chain[index].update({'hash': hash_block})

    return new_chain
```

Calculate hash of blocks

Update the blocks with their hash value

Building a Blockchain/Methods

- is_chain_valid



```
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


        hash_operation = self.hash(block)

        if hash_operation[:4] != '0000':
            return False

        previous_block = block
        block_index += 1

    return True
```

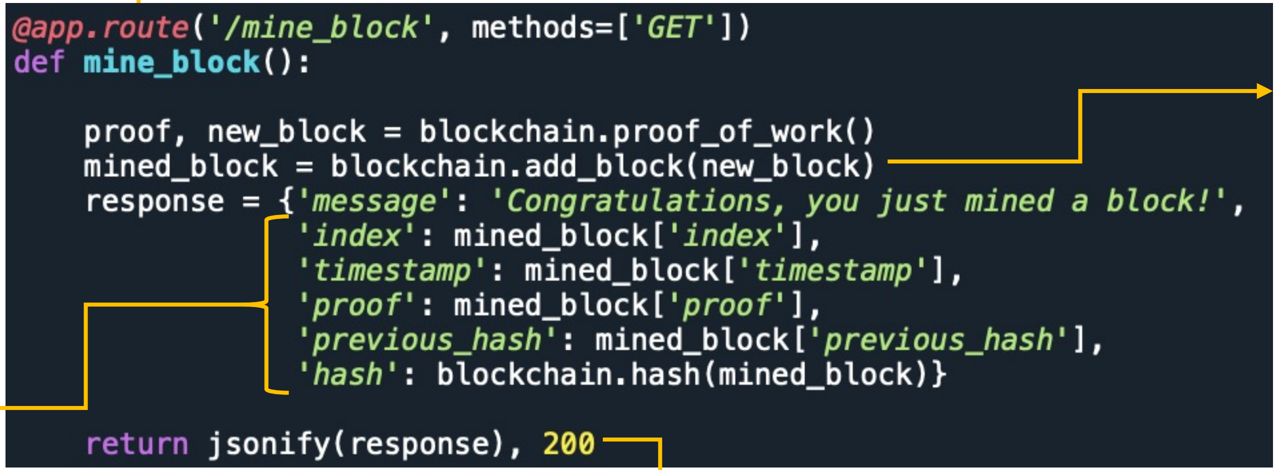
Current Block

First check: hash

Second check: proof of work

Mining a Block

Telling Flask what URL should trigger the function



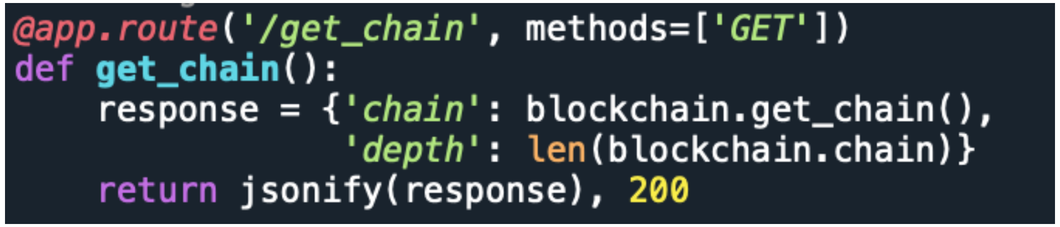
```
@app.route('/mine_block', methods=['GET'])
def mine_block():

    proof, new_block = blockchain.proof_of_work()
    mined_block = blockchain.add_block(new_block)
    response = {'message': 'Congratulations, you just mined a block!',
                'index': mined_block['index'],
                'timestamp': mined_block['timestamp'],
                'proof': mined_block['proof'],
                'previous_hash': mined_block['previous_hash'],
                'hash': blockchain.hash(mined_block)}

    return jsonify(response), 200
```

add_block: append the block to the chain and get a backup of the current change and returns the block

Response display in the postman (in JS format)

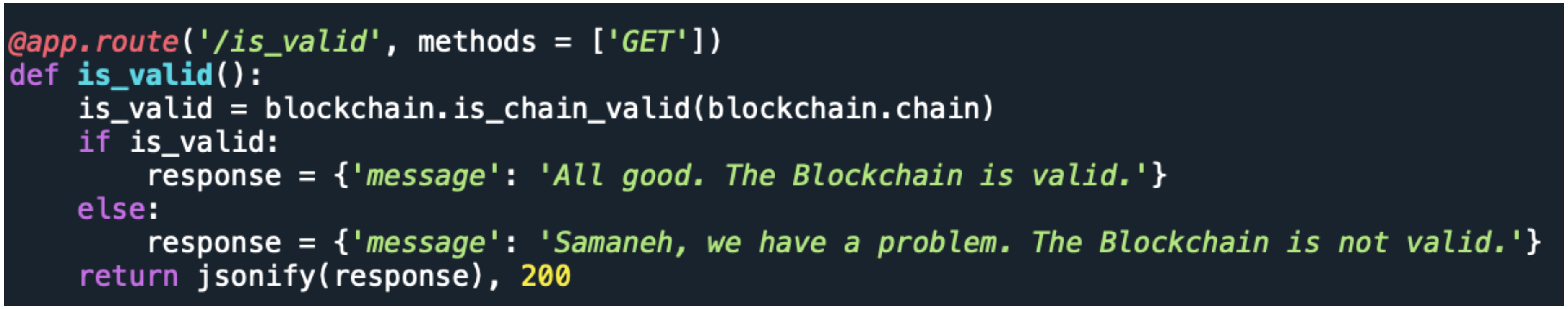
Http OK

Getting the full blockchain



```
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {'chain': blockchain.get_chain(),
                'depth': len(blockchain.chain)}
    return jsonify(response), 200
```

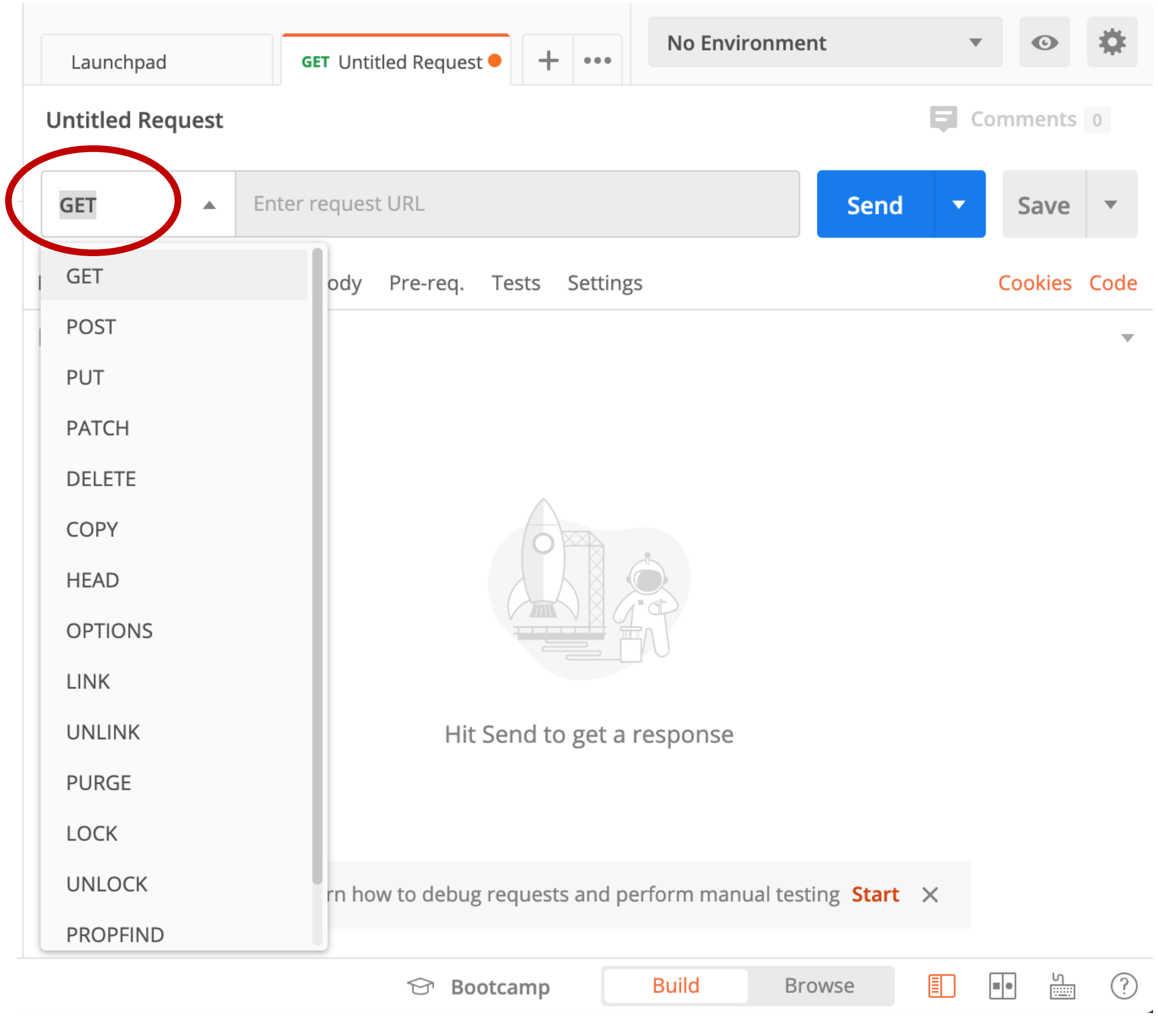
Checking if the block is valid



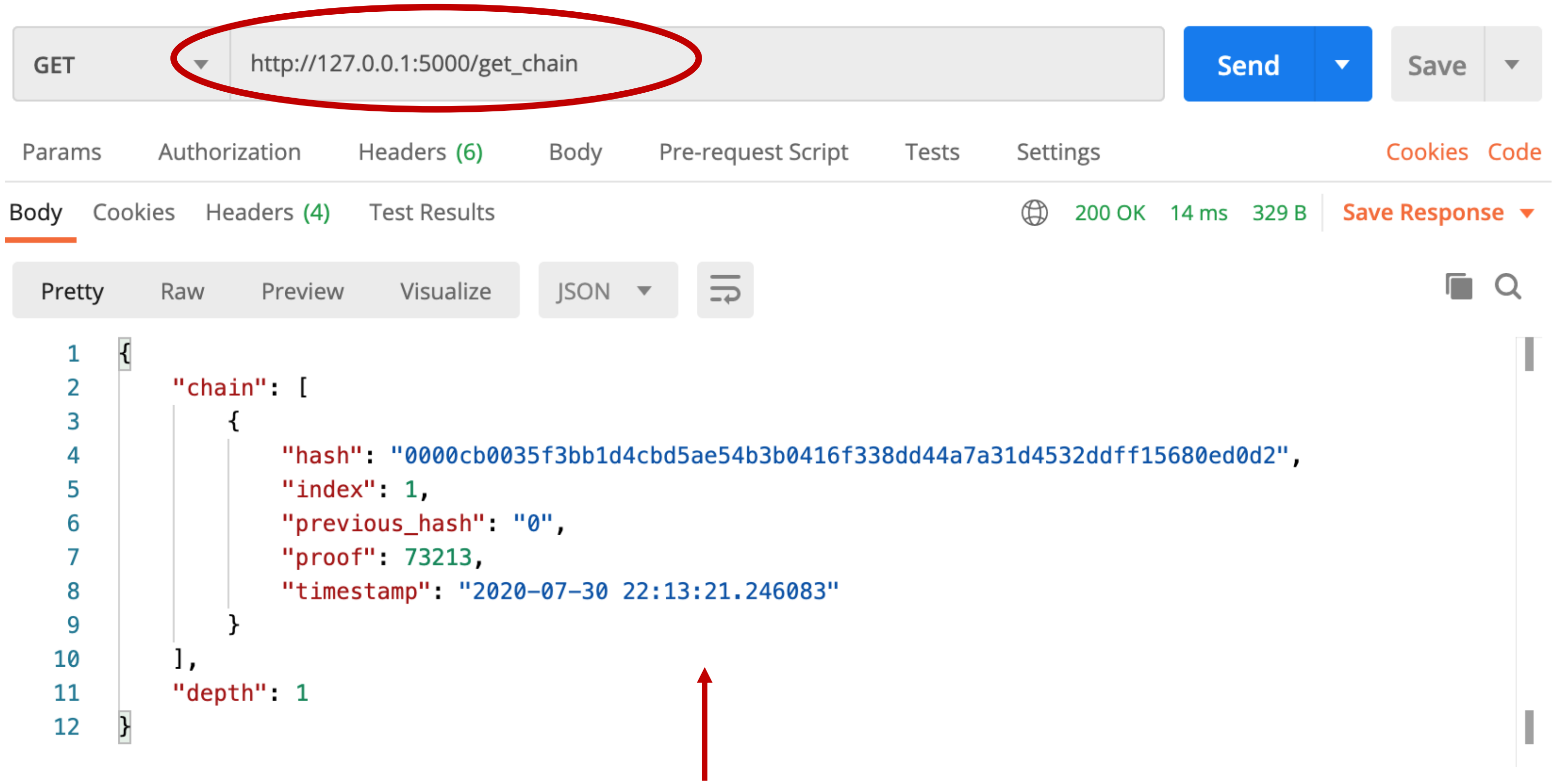
```python
@app.route('/is_valid', methods = ['GET'])
def is_valid():
    is_valid = blockchain.is_chain_valid(blockchain.chain)
    if is_valid:
        response = {'message': 'All good. The Blockchain is valid.'}
    else:
        response = {'message': 'Samaneh, we have a problem. The Blockchain is not valid.'}
    return jsonify(response), 200
```

Test

- Run Code
- Open Postman
- Select HTTP Get Method
- Run on:
  - http://127.0.0.1:5000/



Test



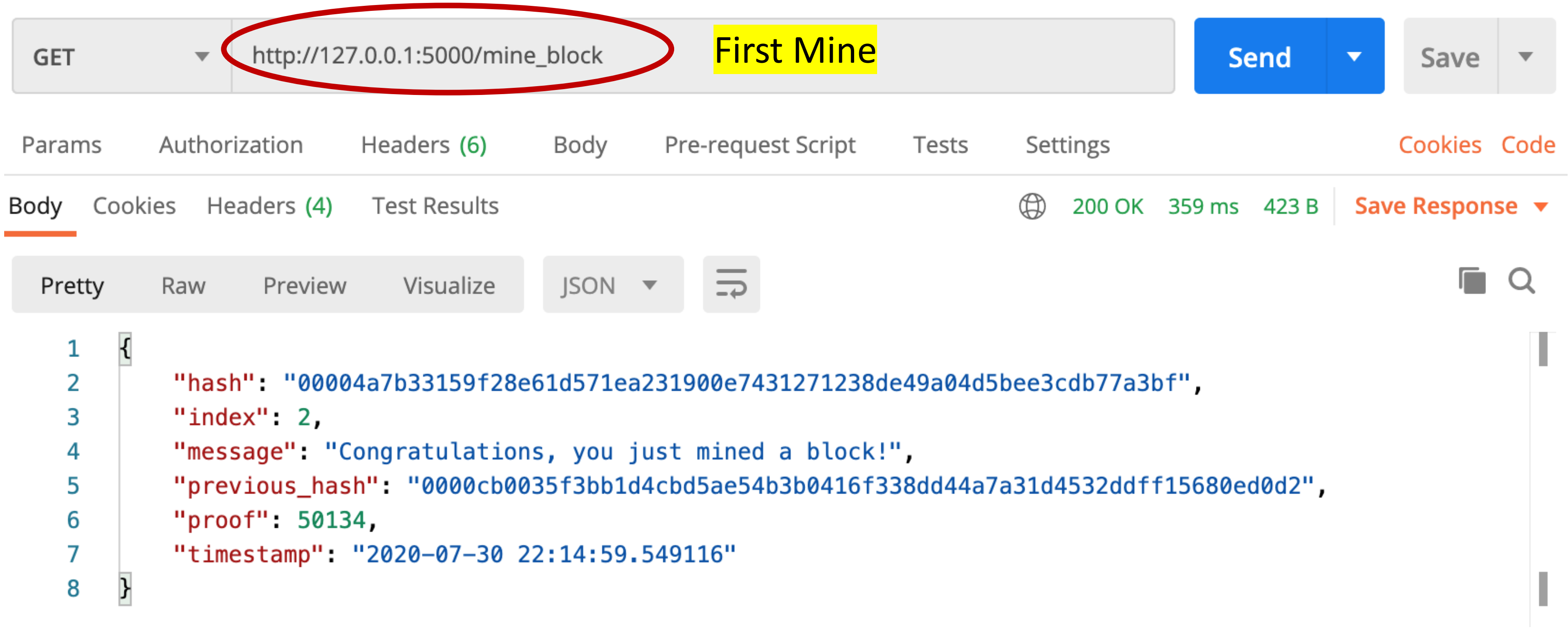
GET http://127.0.0.1:5000/get_chain

200 OK 14 ms 329 B

```
{
    "chain": [
        {
            "hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
            "index": 1,
            "previous_hash": "0",
            "proof": 73213,
            "timestamp": "2020-07-30 22:13:21.246083"
        }
    ],
    "depth": 1
}
```

Genesis Block

Test

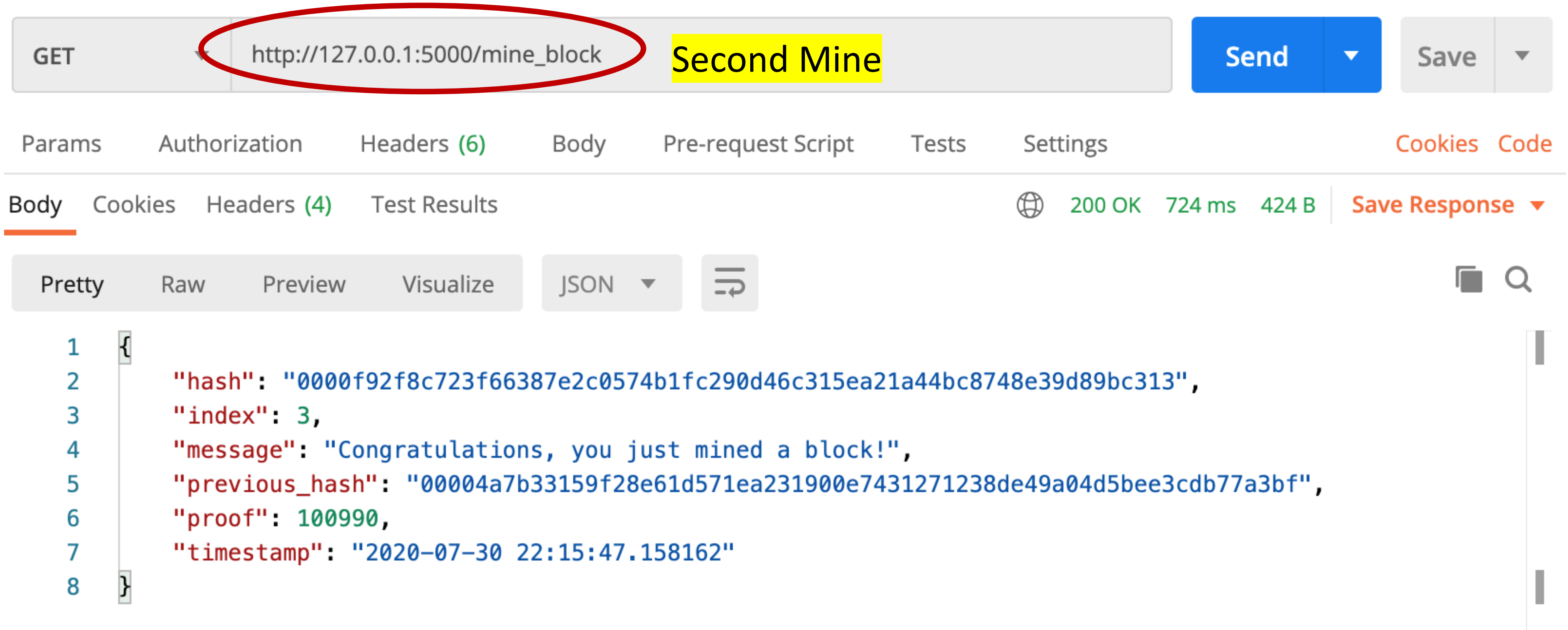


http://127.0.0.1:5000/mine_block

First Mine

```
{
    "hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
    "proof": 50134,
    "timestamp": "2020-07-30 22:14:59.549116"
}
```

Test



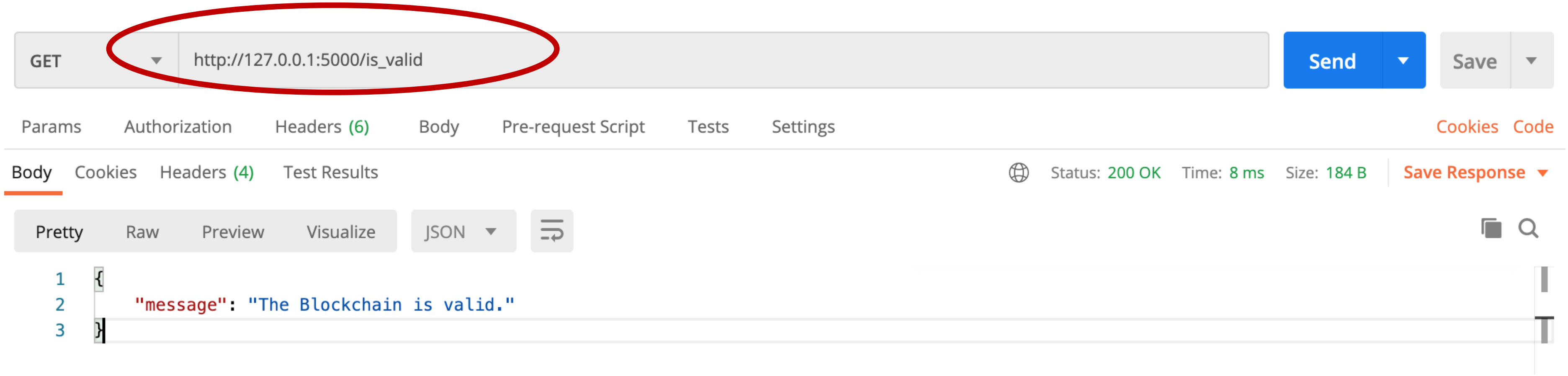
GET http://127.0.0.1:5000/mine_block

Second Mine

200 OK 724 ms 424 B

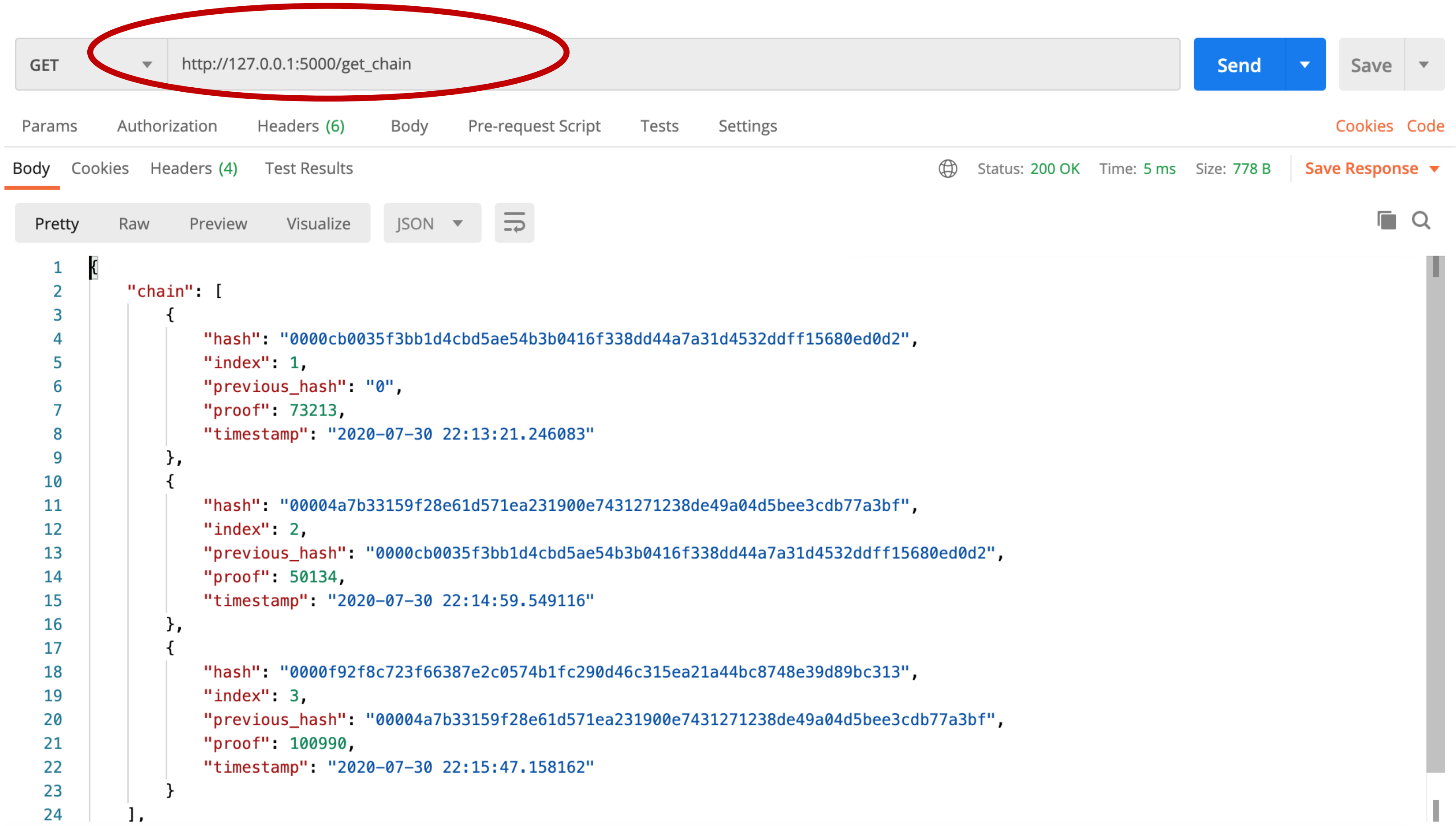
```
{
    "hash": "0000f92f8c723f66387e2c0574b1fc290d46c315ea21a44bc8748e39d89bc313",
    "index": 3,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
    "proof": 100990,
    "timestamp": "2020-07-30 22:15:47.158162"
}
```

Test



```
{
    "message": "The Blockchain is valid."
}
```

Test



http://127.0.0.1:5000/get_chain

Status: 200 OK Time: 5 ms Size: 778 B

```
{
    "chain": [
        {
            "hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
            "index": 1,
            "previous_hash": "0",
            "proof": 73213,
            "timestamp": "2020-07-30 22:13:21.246083"
        },
        {
            "hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
            "index": 2,
            "previous_hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
            "proof": 50134,
            "timestamp": "2020-07-30 22:14:59.549116"
        },
        {
            "hash": "0000f92f8c723f66387e2c0574b1fc290d46c315ea21a44bc8748e39d89bc313",
            "index": 3,
            "previous_hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
            "proof": 100990,
            "timestamp": "2020-07-30 22:15:47.158162"
        }
    ],
```