

Blockchain Workshop

Learn Blockchain by Building One

Presenter: Samaneh Miri

July 31, 2020

Documentation

Create a Blockchain

Genesis Block



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

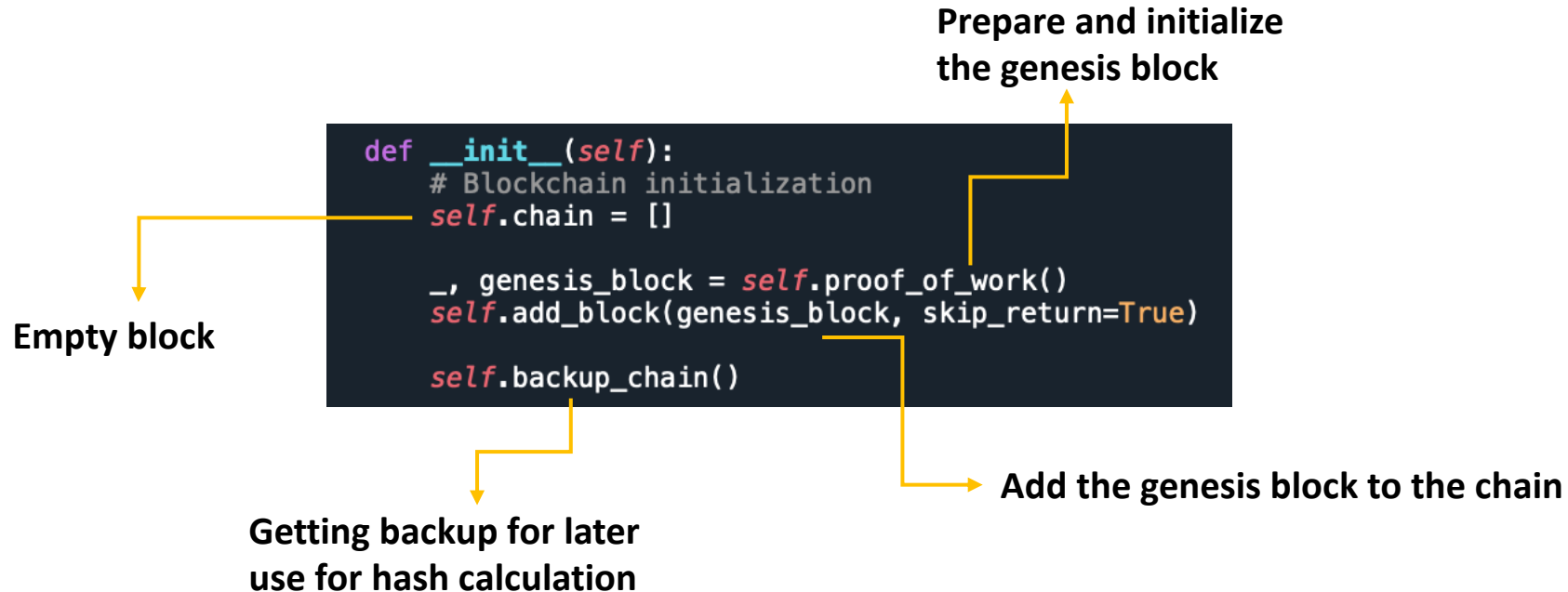
Building a Blockchain/Libraries

- **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__()`



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.

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

Serializing the chain
into chain_bk file

```
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

```
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`

1 ∞ 2 ∞ 3 ∞ 4 ∞ ...

Current Block

```
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
```

First check: hash

Second check: proof of work

Mining a Block

```
@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
```

Telling Flask what URL should trigger the function

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

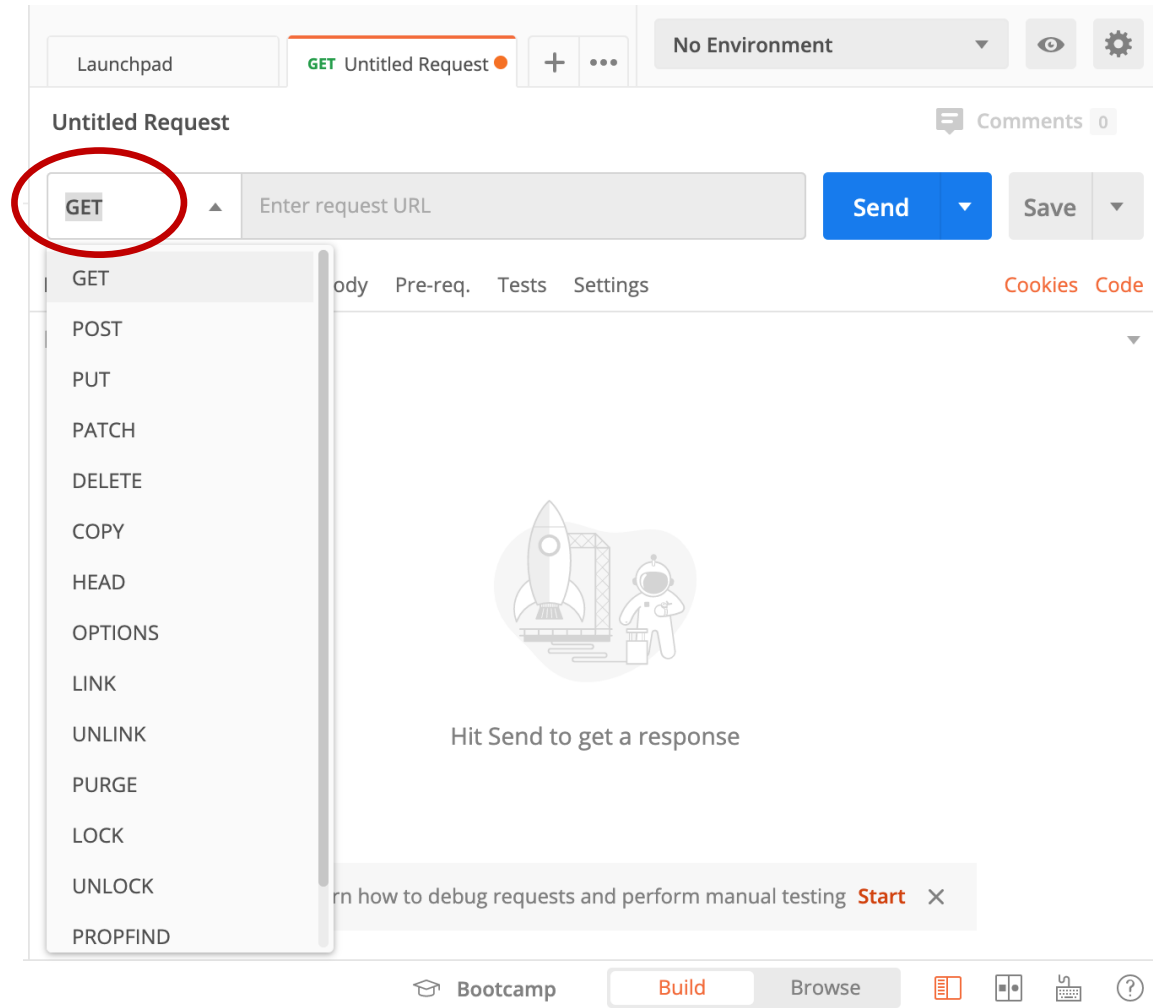
```
@app.route('/get_chain', methods=['GET'])
def get_chain():
    response = {'chain': blockchain.get_chain(),
               'depth': len(blockchain.chain)}
    return jsonify(response), 200
```

Checking if the block is valid

```
@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/`



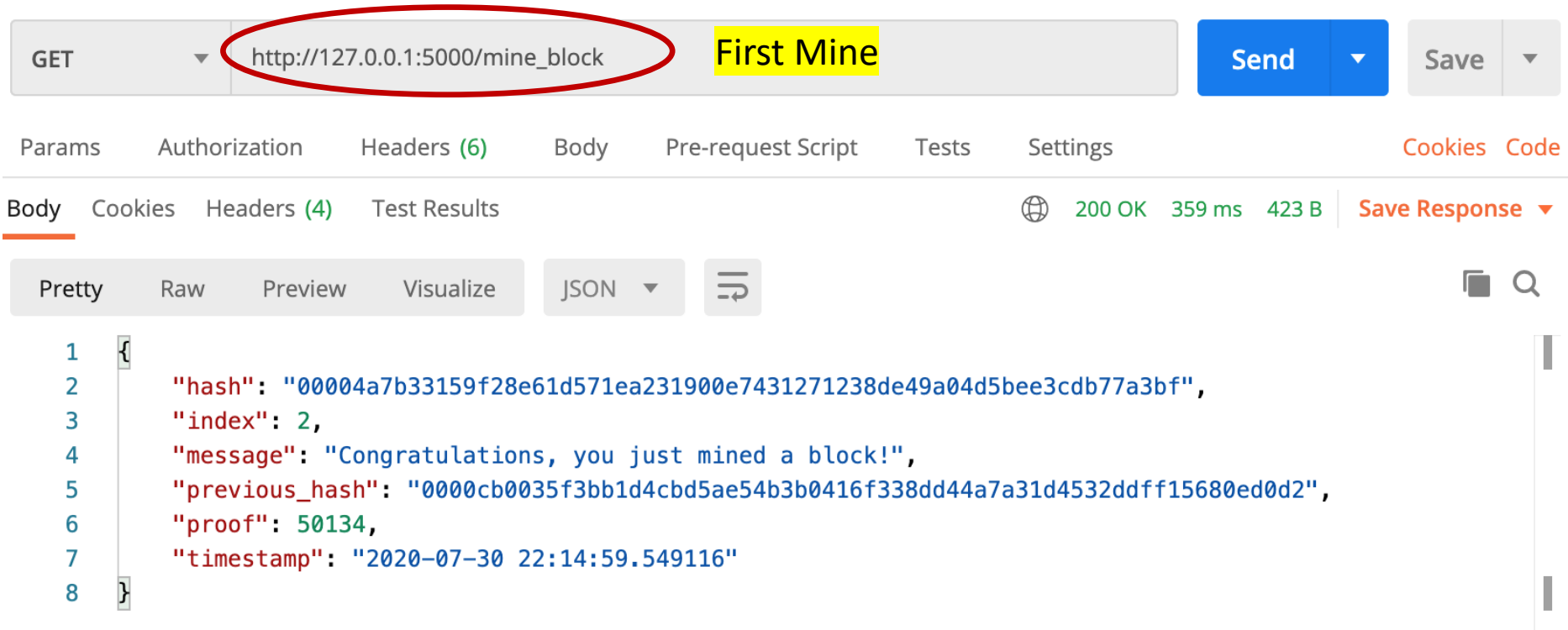
Test

The screenshot displays a web client interface with a GET request to `http://127.0.0.1:5000/get_chain`. The response is a JSON object representing a blockchain chain. The first block in the chain is the Genesis Block, identified by `"index": 1` and `"previous_hash": "0"`.

```
1 {
2   "chain": [
3     {
4       "hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
5       "index": 1,
6       "previous_hash": "0",
7       "proof": 73213,
8       "timestamp": "2020-07-30 22:13:21.246083"
9     }
10  ],
11  "depth": 1
12 }
```

Genesis Block

Test



The screenshot displays a web client interface. At the top, a request bar shows the method 'GET', the URL 'http://127.0.0.1:5000/mine_block' (circled in red), and the title 'First Mine'. To the right are 'Send' and 'Save' buttons. Below this is a tabbed interface with 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected, showing a response status of '200 OK', a time of '359 ms', and a size of '423 B'. A 'Save Response' button is also present. The response body is displayed in 'Pretty' JSON format, showing a successful block mining result.

```
{
  "hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
  "index": 2,
  "message": "Congratulations, you just mined a block!",
  "previous_hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddfff15680ed0d2",
  "proof": 50134,
  "timestamp": "2020-07-30 22:14:59.549116"
}
```

Test

The screenshot shows a REST client interface. At the top, a GET request is configured with the URL `http://127.0.0.1:5000/mine_block` (circled in red) and the name `Second Mine` (highlighted in yellow). The `Send` button is blue, and the `Save` button is grey. Below the request bar, tabs for `Params`, `Authorization`, `Headers (6)`, `Body`, `Pre-request Script`, `Tests`, and `Settings` are visible. The `Body` tab is selected, showing a JSON response. The response status is `200 OK` with a response time of `724 ms` and a size of `424 B`. The response body is displayed in `JSON` format, showing a successful block mining result.

GET `http://127.0.0.1:5000/mine_block` `Second Mine` `Send` `Save`

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies Code

Body Cookies Headers (4) Test Results `200 OK` `724 ms` `424 B` `Save Response`

Pretty Raw Preview Visualize JSON

```
1 {  
2   "hash": "0000f92f8c723f66387e2c0574b1fc290d46c315ea21a44bc8748e39d89bc313",  
3   "index": 3,  
4   "message": "Congratulations, you just mined a block!",  
5   "previous_hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",  
6   "proof": 100990,  
7   "timestamp": "2020-07-30 22:15:47.158162"  
8 }
```

Test

The screenshot displays a REST client interface. At the top, a request bar shows the method 'GET' and the URL 'http://127.0.0.1:5000/is_valid', which is circled in red. To the right of the URL bar are 'Send' and 'Save' buttons. Below the request bar is a tabbed interface with 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected and underlined. To the right of the tabs are links for 'Cookies' and 'Code'. Below the tabs, the response status is shown as 'Status: 200 OK', 'Time: 8 ms', and 'Size: 184 B', followed by a 'Save Response' button. The response body is displayed in a code editor with a 'Pretty' button and a 'JSON' dropdown. The response content is a JSON object:

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

Test

GET http://127.0.0.1:5000/get_chain Send Save

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies Code

Body Cookies Headers (4) Test Results Status: 200 OK Time: 5 ms Size: 778 B Save Response

Pretty Raw Preview Visualize JSON ⌵

```
1  [
2    {
3      "hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
4      "index": 1,
5      "previous_hash": "0",
6      "proof": 73213,
7      "timestamp": "2020-07-30 22:13:21.246083"
8    },
9    {
10     "hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
11     "index": 2,
12     "previous_hash": "0000cb0035f3bb1d4cbd5ae54b3b0416f338dd44a7a31d4532ddff15680ed0d2",
13     "proof": 50134,
14     "timestamp": "2020-07-30 22:14:59.549116"
15   },
16   {
17     "hash": "0000f92f8c723f66387e2c0574b1fc290d46c315ea21a44bc8748e39d89bc313",
18     "index": 3,
19     "previous_hash": "00004a7b33159f28e61d571ea231900e7431271238de49a04d5bee3cdb77a3bf",
20     "proof": 100990,
21     "timestamp": "2020-07-30 22:15:47.158162"
22   }
23 ]
24
```