

Simple Blockchain Implementation for Blockchain Global Master M8 End Project

1. Introduction

Through the M8 program, I learned the basic components of blockchain(i.e., crypto graphy, blocks, wallets, utxo, mining) and implemented them in a simple way. In this project, I added Flask to simulate interaction between blockchain and wallet interface.

2. Main Files: Project Structure

```
.
|__ templates
| |__ index.html          ... wallet UI
|__ Block.py
|__ Blockchain.py
|__ BlockchainServer.py   ... (New) blockchain server to handle a
request from a wallet
|__ CONFIG.py
|__ crypto.py
|__ Genesis.py
|__ hashing.py
|__ Main.py
|__ Mempool.py
|__ Miner.py
|__ Networking.py
|__ private_key.json      ... store the creator's wallet private and
password
|__ requirements.txt
|__ Transaction.py
|__ UTX0.py
|__ Wallet.py
|__ WalletServer.py       ... (New) wallet server to accept a request
from a user.
```

3. How to run & send a transaction

1. Install the dependencies:

```
pip3 install -r requirements.txt
```

2. Run the BlockchainServer.py:

```
python3 BlockchainServer.py
```

3. Run the WalletServer.py:

```
python3 WalletServer.py
```

4. Create a creator's wallet:

Create a creator's wallet having the initial coinbase rewards.

Open 127.0.0.1:8080/creator on your browser

You will see the following page and hit the "Get Balance" button on the screen

5. Create a recipient's wallet:

Create another wallet so that you can transfer a value from the creator's wallet.

Open 127.0.0.1:8080/<any wallet id (i.e., wallet_1)> on your browser

6. Transfer a value from the creator's wallet to the one created in step 5:

Copy the public key of the recipient's wallet and Paster it in the creator's

Enter 50 in Amount.

And hit *Send* button.

7. Run mining:

Open the recipient's wallet and hit *Send* button Open 127.0.0.1:5000/mine on your browser

Then, See the following message returned

```
{
  "message": "success"
}
```

8. Check the balance of recipient's wallet:

Open the recipient's wallet and hit *Get Balance* button

4. Changes I made

BlockchainServer.py

```
'''
Serve requests from a wallet server
'''

app = Flask(__name__)

cache = {} # a cache to keep a blockchain, mempool, and miner instance
def get_chain():
    #create Blockchain instance if not instantiated
    blockchain = cache.get('blockchain')
    if not blockchain:
        cache['blockchain'] = Blockchain()
    return cache['blockchain']
```

```

def get_mempool():
    #create Mempool instance if not instantiated
    mempool = cache.get('mempool')
    if not mempool:
        cache['mempool'] = Mempool()
    return cache['mempool']

@app.route('/mine', methods=['GET'])
def mine():
    """
    Call this method after sending money
    """
    #create Miner instance if not instantiated
    miner = cache.get('miner')
    if not miner:
        # get the creator's pulic key through blockchain server
        response = requests.get(
            urllib.parse.urljoin(app.config['gw'], 'wallet'),
            {
                'wallet_id': 'creator',
            },
            timeout=3)
        if response.status_code == 400:
            return jsonify({'message': 'fail', 'error': 'Please create the
creator wallet first by opening the wallet page'}), 400
        # set the creator's key so that the creator receive mining rewards
        cache['miner'] = Miner(response.json()['public_key'])

    cache['miner'].mine(get_chain(), get_mempool()) # do mining
    return jsonify({'message': 'success'}), 200

@app.route('/balance', methods=['GET'])
def get_balance():
    """
    Get balance. private key and password are required as params
    """
    private_key = request.args['private_key']
    password = request.args['password']
    blockchain = get_chain()
    utxos =
    blockchain.get_utxos(crypto.generate_public_pem_string(private_key,
password))
    assert isinstance(utxos, list)
    total_amount = 0
    for i in utxos:
        assert isinstance(i, UTXO)
        if get_chain().is_valid_UTXO(i):
            total_amount += float(i.get_dict()['message'])

    return jsonify({ 'amount': total_amount }), 200

@app.route('/transfer', methods=['POST'])
def transfer():
    """

```

```

    Send money.
    Receivers' public keys, messages, sender's private key and password
    are required as params
    """
    request_json = request.json
    required = (
        'receiver_pks',
        'msgs',
        'private_key',
        'password')
    if not all(k in request_json for k in required):
        return jsonify({'message': 'missing values'}), 400

    private_key = request_json['private_key']
    password = request_json['password']
    receiver_pks = [request_json['receiver_pks']]
    msgs = [request_json['msgs']]
    money_to_send = 0
    for m in msgs:
        money_to_send = money_to_send + m

    tx = create_transaction(utxos=get_utxos(private_key, password,
    money_to_send), receiver_pks=receiver_pks, msgs=msgs,
    private_key=private_key, password=password)
    return insert_to_mempool(tx)

#moved from Wallet.py
def get_utxos(private_key, password, money):
    blockchain = get_chain()
    utxos =
    blockchain.get_utxos(crypto.generate_public_pem_string(private_key,
    password))
    assert isinstance(utxos, list)
    valid_utxos = []
    for i in utxos:
        assert isinstance(i, UTXO)
        if blockchain.is_valid_UTXO(i):
            valid_utxos.append(i)
    needed_utxos = []
    total_amount = 0
    for i in valid_utxos:
        needed_utxos.append(i)
        if total_amount >= money: #needs to fix?
            break
    return needed_utxos

#moved from Wallet.py
def create_transaction(utxos, receiver_pks, msgs, private_key, password):
    unsigned = UnsignedTransaction(utxos=utxos,
    receiver_public_keys=receiver_pks, messages=msgs)
    tx = Transaction(utxos=utxos, receiver_public_keys=receiver_pks,
    messages=msgs, signature=unsigned.sign(priv_key=private_key,
    password=password))
    return tx

```

```
#moved from Wallet.py
def insert_to_mempool(tx):
    get_mempool().insert_transaction(tx)
    return jsonify({'message': 'success'}), 201

if __name__ == '__main__':
    from argparse import ArgumentParser
    parser = ArgumentParser()
    # set default port
    parser.add_argument('-p', '--port', default=5000, type=int, help='port
to listen on')
    # set default url of a wallet server
    parser.add_argument('-g', '--gw', default='http://127.0.0.1:8080',
type=str, help='wallet gateway')

    args = parser.parse_args()
    port = args.port
    app.config['gw'] = args.gw
    app.config['port'] = port

    app.run(host='127.0.0.1', port=port, debug=True)
```

WalletServer.py

```
app = Flask(__name__, template_folder='./templates')
cache = {}

"""
Serve a request from frontend
"""
@app.route('/<init_id>')
def index(init_id):
    return render_template('./index.html', init_id=init_id)

@app.route('/wallet', methods=['GET', 'POST'])
def create_wallet():
    """
    GET: return a wallet if there's a matching wallet
    wallet id is required
    """
    if request.method == 'GET':
        required = ['wallet_id']
        if not all(k in request.args for k in required):
            return 'Missing values', 400

        wallet_id = request.args.get('wallet_id')
        wallet = cache.get(wallet_id)
        if wallet == None:
            return jsonify({'message': 'not found'}), 400

    response = {
```

```

        'wallet_id': cache[wallet_id].wallet_id,
        'private_key': cache[wallet_id].private_key,
        'public_key': cache[wallet_id].public_key,
        'password': cache[wallet_id].password
    }
    print(cache)
    return jsonify(response), 200

"""
POST: create a new wallet
wallet id is required
"""
if request.method == 'POST':
    request_json = request.json
    required = ['wallet_id']

    if not all(k in request_json for k in required):
        return jsonify({'message': 'missing values'}), 400

    wallet = Wallet(creator=True) if request.json and
    request.json['wallet_id'] == 'creator' else
    Wallet(wallet_id=request_json['wallet_id'])
    id = wallet.wallet_id
    cache[id] = wallet
    response = {
        'wallet_id': cache[id].wallet_id,
        'private_key': cache[id].private_key,
        'public_key': cache[id].public_key,
        'password': cache[id].password
    }
    print(cache)
    return jsonify(response), 201

@app.route('/send_money', methods=['POST'])
def send_money():
    """
    Send money to a recipient public key
    recipient public key, message, and wallet id are required
    """
    request_json = request.json
    required = (
        'recipient_public_key',
        'message',
        'wallet_id')
    if not all(k in request_json for k in required):
        return jsonify({'message': 'missing values'}), 400

    wallet_id = request_json['wallet_id']
    wallet = cache[wallet_id]
    data = {
        'receiver_pks': request_json['recipient_public_key'],
        'msgs': request_json['message'],
        'private_key': wallet.private_key,
        'password': wallet.password
    }

```

```
}

# Call wallet server's transfer api
response = requests.post(
    urllib.parse.urljoin(app.config['gw'], 'transfer'),
    json=data, timeout=3)

if response.status_code == 201:
    return jsonify({'message': 'success'}), 201
return jsonify({'message': 'fail', 'response': response}), 400

@app.route('/get_balance', methods=['POST'])
def get_balance():
    """
    Get balance of a wallet
    wallet id is required
    """
    request_json = request.json
    required = ['wallet_id']
    if not all(k in request_json for k in required):
        return jsonify({'message': 'missing values'}), 400

    wallet_id = request_json['wallet_id']
    wallet = cache[wallet_id]

    # Get a wallet balance through wallet server
    response = requests.get(
        urllib.parse.urljoin(app.config['gw'], 'balance'),
        {
            'private_key': wallet.private_key,
            'password': wallet.password
        },
        timeout=3)
    if response.status_code == 200:
        total = response.json()['amount']
        return jsonify({'message': 'success', 'amount': total}), 200
    return jsonify({'message': 'fail', 'error': response.content}), 400

if __name__ == '__main__':
    from argparse import ArgumentParser
    parser = ArgumentParser()
    # set default port
    parser.add_argument('-p', '--port', default=8080, type=int, help='port
to listen on')
    # set default url of a blockchain server
    parser.add_argument('-g', '--gw', default='http://127.0.0.1:5000',
type=str, help='blockchain gateway')
    args = parser.parse_args()
    port = args.port
    app.config['gw'] = args.gw

    app.run(host='127.0.0.1', port=port, debug=True)
```

```

class Wallet:
    def __init__(self, wallet_id=False, creator=False):
        if creator: # load creator's wallet if creator is true
            self.wallet_id = 'creator'
            self.private_key, self.password = self.load_from_file()
            self.public_key =
crypto.generate_public_pem_string(self.private_key, self.password)
        else: # create a new wallet
            self.wallet_id = wallet_id
            self.password = crypto.generate_password()
            self.private_key =
crypto.generate_private_pem_string(password_string=self.password)
            self.public_key =
crypto.generate_public_pem_string(self.private_key, self.password)

    def save_to_file(self):
        data = {
            "private_key": self.private_key,
            "password": self.password
        }
        with open("private_key.json", "w") as output:
            output.write(json.dumps(data))

    def load_from_file(self):
        with open("private_key.json", "r") as input_file:
            data = json.loads(input_file.read())
            return data["private_key"], data["password"]

```

```

# moved the cache to BlockchainServer.py
class Blockchain:
    def __init__(self):
        self.blocks =
[Block("ZEvMflZDcwQJmarInnYi88px+6HZcv2Uoxw7+/J00Tg=",
        [genesis_coinbase()], 0)]

    def insert_block(self, block):
        if not isinstance(block, Block):
            return False
        for tx in block.transactions:
            if not tx.is_valid():
                return False
            if isinstance(tx, Transaction):
                for utxo in tx.utxos:
                    if not self.is_valid_UTX0(utxo):
                        return False
        if not self.check_against_target(block.get_hash()):
            return False
        self.blocks.append(block)
        return True

```



```
def check_against_target(self, hash_string):
    hex = hashing.string_to_hex(hash_string)
    for i in range(1, mining_target+1):
        if not hex[i] == "0":
            return False
    return True

def get_utxos(self, public_key):
    utxos = []
    for block in self.blocks:
        for tx in block.transactions:
            counter = 0
            for pk in tx.receiver_public_keys:
                if pk in public_key:
                    utxo = UTX0(tx.get_hash(), public_key,
tx.messages[counter])
                    utxos.append(utxo)
                    counter = counter + 1
    return utxos

def get_topmost_block(self):
    return self.blocks[len(self.blocks)-1]

def is_valid_UTX0(self, UTX0):
    valid = False
    #find possible UTX0 on Blockchain
    for block in self.blocks:
        for tx in block.transactions:
            if tx.get_hash() == UTX0.tx_hash:
                counter = 0
                for pk in tx.receiver_public_keys:
                    if pk in UTX0.public_key:
                        if UTX0.message == tx.messages[counter]:
                            valid = True
                            counter = counter + 1
    if valid == False:
        return False
    #check double_spending
    for block in self.blocks:
        for tx in block.transactions:
            if isinstance(tx, Transaction):
                for tx_utxo in tx.utxos:
                    if tx_utxo.get_hash() == UTX0.get_hash():
                        return False
    return True

def get_json(self):
    blocks = []
    for i in self.blocks:
        blocks.append(i.get_dict())
    return json.dumps({
        "blocks": blocks
    })
```

Mempool.py

10 / 14

```
def insert_transaction(self, tx):
    assert isinstance(tx, Transaction)
    assert tx.is_valid()
    self.tx.append(tx)
```

Genesis.py

```
# load the save key for the creator who will receive mining rewards
def load_from_file():
    with open("private_key.json", "r") as input_file:
        data = json.loads(input_file.read())
        return data["private_key"], data["password"]

CREATORS_PUBLIC_KEY = None
if os.path.isfile("private_key.json"):
    private_key, password = load_from_file()
    CREATORS_PUBLIC_KEY = crypto.generate_public_pem_string(private_key,
password)
else:
    CREATORS_PUBLIC_KEY = '''-----BEGIN PUBLIC KEY-----
MIICIJANBgqhkiG9w0BAQEFAAA0CAG8AMIICGKCAgEAl20HC7xKreGy16YVuvNQ
heMJc62hLEs4S6iELs98cx6aHGJI3YAmahiB1uAdyX7unWtlwdeeKpdDtZ9b1XS9
R/kJi1vuJcmwTVAnDUCXWKd681+x3iSxnYT7tfSTzwbo3GWeTxnul3rWkd6E0546
hFSLQRI2zoDz0iEe7emb6gAW3PGHsyDAzDI+v2tThLIpJIaM2We6m0RoFxDgDqX
ORWS/V8i3UhaN1Ha5MMpowP+ajk8D0w0z2DsgtAn2930FmRK4ciJ0pFV4d/1Y+qr
poSWMI0wyu02iZLve/3SXP3ariyhxnd7gMmM00WWX/9qNiQ5T8Fx6H8noLMyfg/k
cm7GCajczxGy8MEUvx40fVxoxL0g2dxa909ZPEjx4M5HEExsk/jM1kCiBThcL04BX
xHjMehdDwnPYDxmHXv5LZveYrobsrqJmHESI3Whp0n9vjUqSo2ugPsJI+DbQF5JI
pLnU1bmTf01W7ynzIw5Ry5Td7o2RhSXyk6zCtYvtrHXQt4pflaWJzrq8h2TeKU/n
G30sfQvpWy5KDLBI/71cZnqslChcQ6tnYSNStXS0o3aWhYAKIPI+ByKNiBKXX82V
vQQ14WELBorYCiGiDginapgUC7uKIVaj0nLEBbyim1jrnCWhsGbBK41tF7bPBRo/
8SHuIBtSTPi+L3MkAEfkklsCAwEAAQ==
-----END PUBLIC KEY-----'''

def genesis_coinbase():
    return Coinbase(CREATORS_PUBLIC_KEY)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Your wallet</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
  <script>
    $(function () {
```

```

let data = { 'wallet_id': '{{ init_id }}' };
$.ajax({
  url: '/wallet',
  type: 'GET',
  contentType: 'application/json; charset=utf-8',
  dataType: 'json',
  data: data,
  success: function (response) {
    $('#wallet_id').html(response['wallet_id']);
    $('#public_key').val(response['public_key']);
    $('#private_key').val(response['private_key']);
    $('#password').val(response['password']);
  },
  error: function (response) {
    $.ajax({
      url: '/wallet',
      type: 'POST',
      contentType: 'application/json',
      dataType: 'json',
      data: JSON.stringify(data),
      success: function (response) {
        $('#wallet_id').html(response['wallet_id']);
        $('#public_key').val(response['public_key']);
        $('#private_key').val(response['private_key']);
        $('#password').val(response['password']);
        console.info(response);
      },
      error: function (error) {
        console.error(error);
      }
    })
  }
})

})

$('#send_money_button').click(function () {
  let confirm_text = 'Are you sure to send?';
  let confirm_result = confirm(confirm_text);
  if (confirm_result !== true) {
    alert('Canceled');
    return
  }

  let data = {
    'wallet_id': document.getElementById('wallet_id').innerHTML,
    'recipient_public_key': $('#recipient_public_key').val(),
    'message': parseFloat($('#send_amount').val())
  };

  $.ajax({
    url: '/send_money',
    type: 'POST',
    contentType: 'application/json; charset=utf-8',
    dataType: 'json',

```

```

        data: JSON.stringify(data),
        success: function (response) {
            alert('Send success')
        },
        error: function (response) {
            alert('Send failed', response)
        }
    })
})

$('#reload_wallet').click(function () {
    let data = {
        'wallet_id': document.getElementById('wallet_id').innerHTML
    };

    $.ajax({
        url: '/get_balance',
        type: 'POST',
        contentType: 'application/json; charset=utf-8',
        dataType: 'json',
        data: JSON.stringify(data),
        success: function (response) {
            $('#wallet_amount').html(response['amount']);
        },
        error: function (response) {
            alert('Failed to get a balance', error)
        }
    })

})

})
</script>
</head>
<body>
    <div>
        <h1>Wallet</h1>
        <p>Wallet ID</p>
        <div id="wallet_id">[wallet id to be displaed]</div>
        <p>Balance</p>
        <div id="wallet_amount">0</div>
        <button id="reload_wallet">Get Balance</button>

        <p>Public Key</p>
        <textarea id="public_key" rows="10" cols="100"></textarea>
        <textarea id="private_key" rows="10" cols="100" hidden></textarea>
    </div>
    <div>
        <h1>Send Money</h1>
        <div>
            <p>Address:</p>
            <textarea id="recipient_public_key" rows="10" cols="100"></textarea>
            <p>Amount: </p>
            <input id="send_amount" type="text"><br>
            <input id="password" type="text" hidden><br>

```

```
        <button id="send_money_button">Send</button>
    </div>
</div>
</body>
</html>
```