Build a Blockchain & Cryptocurrency

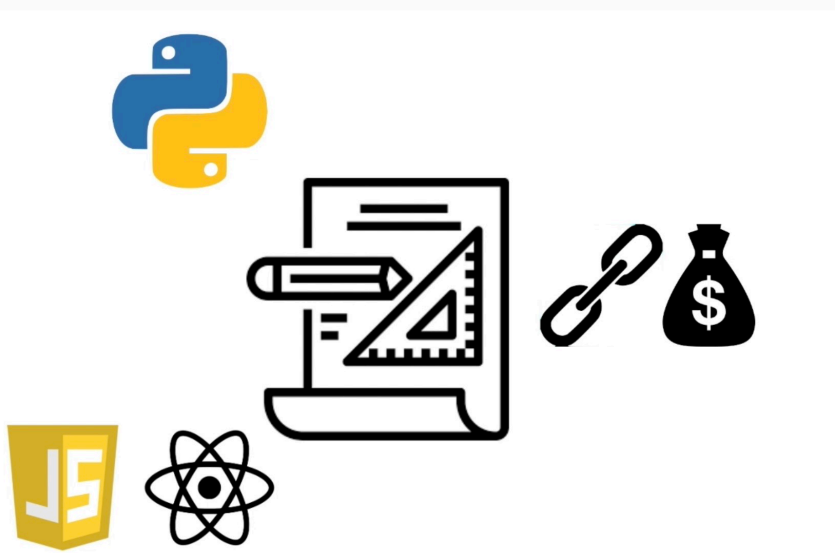
Tech Stack: Python & Flask, JS & React

Github repo: https://github.com/SoochowRobin/python-blockchain

Guobin Chen
12/06/2022

Project Goal

1. Learn Python Fundamentals and Flask framework
2. Build a Blockchain and Cryptocurrency
3. Learn Fronted Web Development, JavaScript, and React



Python Fundamentals

Familiar with: variable, functions, while-loop, for-loop, lambda, built-in data structures: lists, sets, dictionaries, classes, modules

```
class Block:
    """
    Block: a unit of storage.
    Store transactions in a blockchain that supports a cryptocurrency.
    """
    def __init__(self, timestamp, last_hash, hash, data, difficulty, nonce):
        self.timestamp = timestamp
        self.last_hash = last_hash
        self.hash = hash
        self.data = data
        self.difficulty = difficulty
        self.nonce = nonce
```

Blockchain: a list of blocks where each block represents a unit of storage for data. The list is called a chain because each block references the block before it

Genesis block is the first block in the blockchain. And it serves as a hardcoded starter block for the chain

A hashing algorithm generates a unique output for every unique output. I used sha-256 algorithm, which produces a unique 256 characters hash in binary

```
def crypto_hash(*args):
    """
    Return a sha-256 hash of the given arguments.
    """
    stringified_args = sorted(map(lambda data: json.dumps(data), args))
    joined_data = ''.join(stringified_args)

    return hashlib.sha256(joined_data.encode('utf-8')).hexdigest()
```

Proof of Work

Proof of work: it is a mechanism that requires miners to solve a computational puzzle in order to create valid blocks. Solving the puzzle requires a brute-force algorithm that demands CPU power(standard: leading 0's requirement)

Dynamic difficulty: It is a mechanism that increases or decreases the difficulty of the next block based on how long it is taking to mine the new block

```python
@staticmethod
def adjust_difficulty(last_block, new_timestamp):
    """
    Calculate the adjusted difficulty according to the MINE_RATE.
    Increase the difficulty for quickly mined blocks.
    Decrease the difficulty for slowly mined blocks.
    """
    if (new_timestamp - last_block.timestamp) < MINE_RATE:
        return last_block.difficulty + 1

    if (last_block.difficulty - 1) > 0:
        return last_block.difficulty - 1

    return 1
```

```python
@staticmethod
def mine_block(last_block, data):
    """
    Mine a block based on the given last_block and data, until a block hash
    is found that meets the leading 0's proof of work requirement.
    """
    timestamp = time.time_ns()
    last_hash = last_block.hash
    difficulty = Block.adjust_difficulty(last_block, timestamp)
    nonce = 0
    hash = crypto_hash(timestamp, last_hash, data, difficulty, nonce)

    while hex_to_binary(hash)[0:difficulty] != '0' * difficulty:
        nonce += 1
        timestamp = time.time_ns()
        difficulty = Block.adjust_difficulty(last_block, timestamp)
        hash = crypto_hash(timestamp, last_hash, data, difficulty, nonce)

    return Block(timestamp, last_hash, hash, data, difficulty, nonce)
```

Test

The general approach to tests is to create a series of assert statements that verify whether or not a value is equal to some other value

```python
def test_is_valid_chain_bad_genesis(blockchain_three_blocks):
    blockchain_three_blocks.chain[0].hash = 'evil_hash'

    with pytest.raises(Exception, match='genesis block must be valid'):
        Blockchain.is_valid_chain(blockchain_three_blocks.chain)

def test_replace_chain(blockchain_three_blocks):
    blockchain = Blockchain()
    blockchain.replace_chain(blockchain_three_blocks.chain)

    assert blockchain.chain == blockchain_three_blocks.chain

def test_replace_chain_not_longer(blockchain_three_blocks):
    blockchain = Blockchain()

    with pytest.raises(Exception, match='The incoming chain must be longer'):
        blockchain_three_blocks.replace_chain(blockchain.chain)
```

40 tests, all pass

```
blockchain-env ❯ python3 -m pytest backend/tests
============================= test session starts ==============================
platform darwin -- Python 3.9.13, pytest-5.1.2, py-1.11.0, pluggy-0.13.1
rootdir: /Users/robinchen/Desktop/python-blockchain
collected 40 items

backend/tests/blockchain/test_block.py ..........
backend/tests/blockchain/test_blockchain.py ............
backend/tests/util/test_crypto_hash.py .
backend/tests/util/test_hex_to_binary.py .
backend/tests/wallet/test_transaction.py ..........
backend/tests/wallet/test_transaction_pool.py ..
backend/tests/wallet/test_wallet.py ...

============================== 40 passed in 8.25s ==============================
```

Cryptocurrency

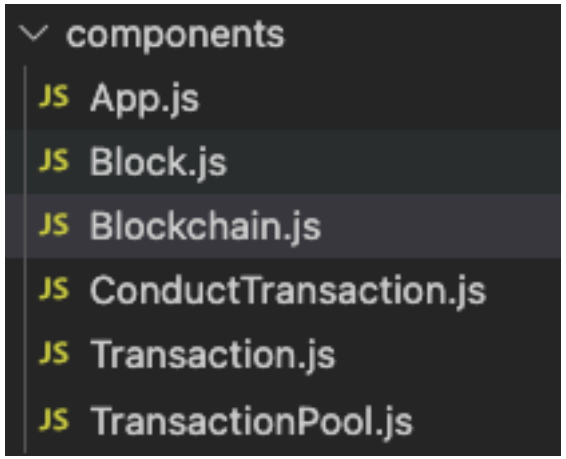
Wallet: A wallet keeps track of an individual's amount of currency. Each wallet has an address, and a pair of keys (private and public)

Signature: It is unique data object created using the private key of keypair and an original data object to sign. With the signature, public key, and the original data object can verify if the signature was generated by the true owner of the public key.

Transaction pool: It is an object that collects transactions that have been broadcasted across the network. It stores transaction according to their id.

```
class Wallet:
    """
    An individual wallet for a miner.
    Keeps track of the miner's balance.
    Allows a miner to authorize transactions.
    """
    def __init__(self, blockchain=None):
        self.blockchain = blockchain
        self.address = str(uuid.uuid4())[0:8]
        self.private_key = ec.generate_private_key(
            ec.SECP256K1(),
            default_backend()
        )
        self.public_key = self.private_key.public_key()
        self.serialize_public_key()
```

Fronted end

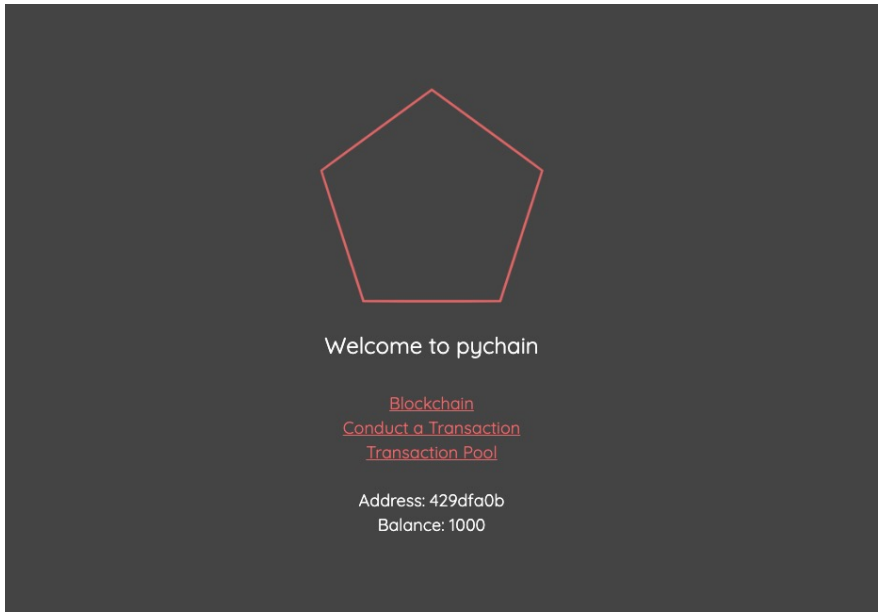


React is a JavaScript framework for web applications. React optimizes how JavaScript is used on the browser to make the web app dynamic and efficient.

Components represent reusable pieces of the UI within the React application.

Hooks could apply React functionality to components – like state, side effects.

Pagination is an approach to display long lists of data in a fronted web application.



Thank you for your time!