

Final Report

Blockchain is a technology that achieves the Point-to-Point transport, every node in the network share the same information, which builds a highly-decentralized system, based on cryptography.

Modern supply chains are inherently complex, consists of disjointed modules follows a specific logic and geographical distribution, these entities competing each other to serve consumers. Diverse nation cultures, policys and emotional human behaviours make it impossible to prevent or hold back ithe likely accident occurs in supply-chain in time. Inefficient transactions, fraud and poorly-performing supply chains, lead to greater trust shortage. So, something that can be traceable is becoming an increasingly urgent requirement, and Blockchain take the role.

The ID of the products will be recorded on the blockchain network at the moment being put into the production, since that the whole process from industry to market will be transparent to you. It helps us discover the accident quickly, and once problems arised, you can prevent the product from market in time.

The commonly used computer language inculdes C/C++, Java, python, Ruby and Go. I prefer Go in dividually. Since python style code is also a pratical language and easier to understand, I will explain the Blockchain building process in python style code.

As we see, BlockChain itself is a kind of data structure, and though there is still a long way to go in before BlockChain can be truly used in the service industry, the core algorithm of BlockChain is not very complex, but what makes it become one of the hot topic in our informational era?(the other one is machine learning). BlockChain is truly not the innovation of technology, but a product of the progress of people's ideas that pursuing a better individual life and the new need of the era

Now, I will show the building process of blockchain to you in code.

In this case, I define a Block class to build the data structure of Blockchain. (Nevermind, Blockchain it self is a kind of data structure.)

```
import hashlib as hasher#The hashlib is a package that helps encrypt the public key and private key.

class Block:
    def __init__(self, index, timestamp, data, previous_hash):
        self.index = index
        self.timestamp = timestamp#时间戳
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.hash_block()

    def hash_block(self):
        sha = hasher.sha256()
        sha.update(str(self.index) +
                  str(self.timestamp) +
                  str(self.data) +
                  str(self.previous_hash))
        return sha.hexdigest()
```

We'll create a function that simply returns a genesis block to make things easy. This block is of index 0, and it has an arbitrary data value and an arbitrary value in the "previous hash" paramter.

Look, that's the function that create the genesis block, the details of the function just return a Block which has been defined well above. Do you know what is the genesis block. The genesis block is the head of the Blockchain, **the first Block**. The hash_block encrypt the index ,timestamp,data and previous_hash.

```
import datetime as date
def create_genesis_block():
    # Manually construct a block with
    # index zero and arbitrary previous hash
    return Block(0, date.datetime.now(), "Genesis Block", "0")
```

Now that we're able to create a genesis block, we need a function that will generate succeeding blocks in the blockchain. This function will take the previous block in the chain as a parameter, create the data for the block to be generated, and return the new block with its appropriate data. When new blocks hash information from previous blocks, the integrity of the blockchain increases with each new block. If we didn't do this, it would be easier for an outside party to "change the past" and replace our chain with an entirely new one of their own. This chain of hashes acts as cryptographic proof and helps ensure that once a block is added to the blockchain it cannot be replaced or removed. Can you see the data? The data is "Hey I'm block". That is a kind of linear structure look like this : 1,2,.... The index symbols the position of the block in corresponding blockchain. Every time I create a new block, I will transmit the last_block to the "next_block" function.

```

def next_block(last_block):
    this_index = last_block.index + 1
    this_timestamp = date.datetime.now()
    this_data = "Hey! I'm block " + str(this_index)
    this_hash = last_block.hash
    return Block(this_index, this_timestamp, this_data, this_hash)

```

That's the majority of the hard work. Now, we can create our blockchain. In our case, the blockchain itself is a simple python list, which is similar to the "array" in C. The first element of the list is the genesis block. And of course, we need to add the succeeding blocks. Because SnakeCoin is the tiniest blockchain, we'll only add 20 new blocks. We can do this with a "for" loop.

```

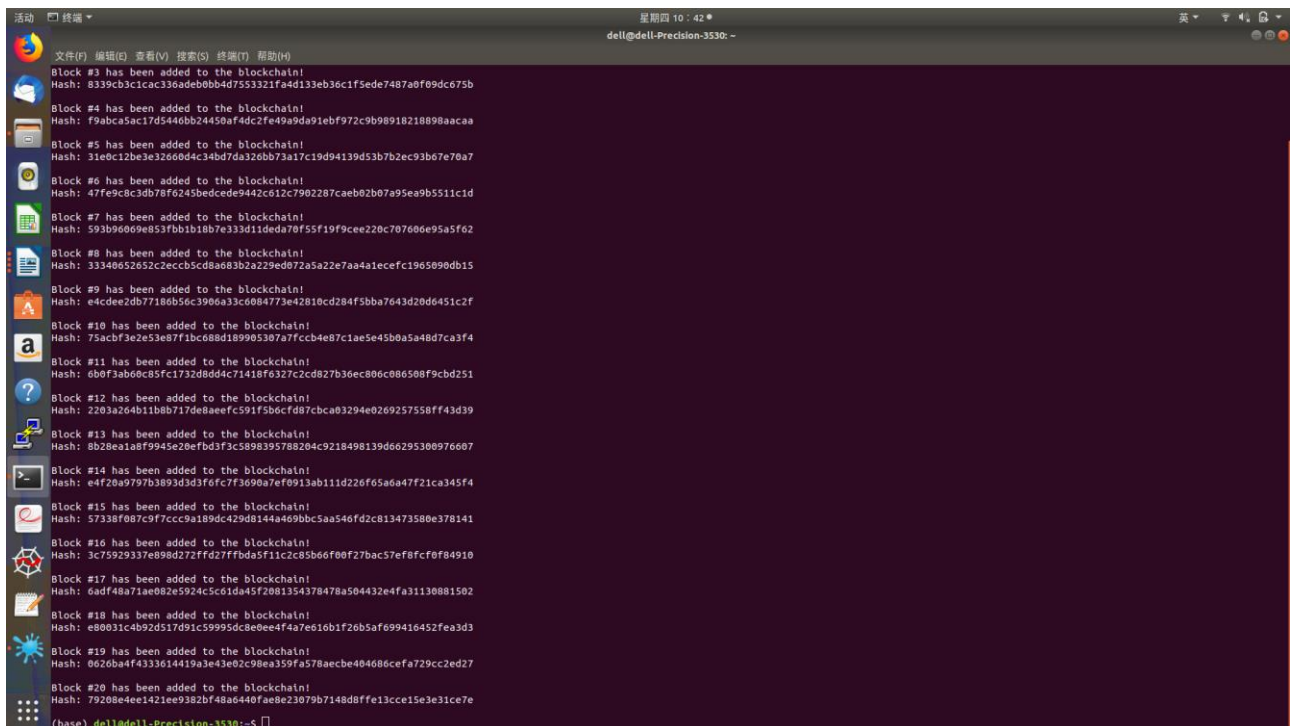
# Create the blockchain and add the genesis block
blockchain = [create_genesis_block()] #similar to the "array" in C.
previous_block = blockchain[0]

# How many blocks should we add to the chain
# after the genesis block
num_of_blocks_to_add = 20

# Add blocks to the chain
for i in range(0, num_of_blocks_to_add):
    block_to_add = next_block(previous_block)
    blockchain.append(block_to_add)#The core code add the new block to the list.
    previous_block = block_to_add
    # Tell everyone about it!
    print "Block #{} has been added to the blockchain!".format(block_to_add.index)
    print "Hash: {} \n".format(block_to_add.hash)

```

Running Screenshot:



```
Block #3 has been added to the blockchain!  
Hash: 8339cb3c1cac336adeb0bb4d7553321fa4d133eb36c1f5ede7487a0f09dc675b  
Block #4 has been added to the blockchain!  
Hash: f9abca5ac17d5440bb24450af4dc2fe49a9da91ebf972c9b98918218898aaaca  
Block #5 has been added to the blockchain!  
Hash: 31e0c12be3e326e0d4c34bd7da326bb73a17c19d94139d53b7b2ec93b67e70a7  
Block #6 has been added to the blockchain!  
Hash: 47fe9c8c3db78f6245bedced9442c612c7902287caeb2b07a95ea9b5511c1d  
Block #7 has been added to the blockchain!  
Hash: 593b96069e853fbb1b87e333d1deda70f55f19f9cee220c707606e95a5f62  
Block #8 has been added to the blockchain!  
Hash: 33340652652c2eccb5cd8a083b2a229ed0725a22e70a41ecef1965090db15  
Block #9 has been added to the blockchain!  
Hash: e4cdee20b77186b5ec396ea33c0084773e42810cd284f5ba7643d20d6451c2f  
Block #10 has been added to the blockchain!  
Hash: 75acbf3e2e53e87f1bc688d189905307a7fcc4e07c1ae5e45b0a5a48d7ca3f4  
Block #11 has been added to the blockchain!  
Hash: 6b0f3ab68c85fc1732d8dd4c71418f6327c2d827b36ec806c086508f9cbd251  
Block #12 has been added to the blockchain!  
Hash: 2203a264b11b8b717de8aeefc591f5b0cf087c0a03294e0269257550ff43d39  
Block #13 has been added to the blockchain!  
Hash: 0b2eaa1a8f9945e2e0efbd3f3c5098395788204c9218498139d66295300976607  
Block #14 has been added to the blockchain!  
Hash: e4f20a9797b3893d3d3f6c7f3690a7ef0913ab111d226f65a6a47f21ca345f4  
Block #15 has been added to the blockchain!  
Hash: 57338f087c9f7cc9a189dc429d8144a469bbc5aa546fd2c813473580e378141  
Block #16 has been added to the blockchain!  
Hash: 3c75929337e898d272ffdz7ffbd95f11c2c85b66f0f27bac57ef8fcfef84910  
Block #17 has been added to the blockchain!  
Hash: 6adf48a71ae082e5924c5c1da45f2081354378478a504432e4fa31130801502  
Block #18 has been added to the blockchain!  
Hash: e80031c4b92d517d91c59995dc8e0ee4f4a7e510b1f26b5af699416452fea3d3  
Block #19 has been added to the blockchain!  
Hash: 0626ba4f4333614419a3e43e02c98ea359fa578aecbe404686cefa729cc2ed27  
Block #20 has been added to the blockchain!  
Hash: 79208e4ee1421ee9382bf48a6440fae8e23079b7148d8ffe13cce15e3e31ce7e  
(base) dell@dell-Precision-3530:~$
```

Following is the most common design of Blockchain in python style code.

```
from time import time  
import json  
import hashlib  
from flask import Flask, jsonify ,request  
from uuid import uuid4  
from urllib.parse import urlparse  
import requests  
  
class Blockchain():  
    def __init__(self):  
        #Initializes the instance variables of the Blockchain class.  
        self.chain=[]  
        -----  
        -----[1]  
        self.current_transactions={}  
        self.nodes=set()  
        self.to_be_mined={}  
        self.new_block(proof=100,previous_hash=1)  
        -----  
        -----[2]  
  
    def register_node(self,address):  
        #Registers the address of a new node.  
        parsed_url=urlparse(address)  
        self.nodes.add(parsed_url.netloc)
```

```
def register_node(self, address):
    #Registers the address of a new node.
    parsed_url=urlparse(address)
    self.nodes.add(parsed_url.netloc)
```

```
def new_block(self, proof, previous_hash=None):
```

```
-----[3]
```

```
#Adds new block to existing chain.
block={
    'index':len(self.chain),
    'timestamp':time(),
    'transactions':self.to_be_mined,
    'proof':proof,
    'previous_hash':previous_hash
}
```

```
self.to_be_mined={}
self.chain.append(block)
```

```
-----[4]
```

```
return block
```

```
def
```

```
new_transaction(self, supplier_manu, requested_hash=None, sent_hash=None, received_hash=None, requested_payment=None, sent_payment=None, received_payment=None):
```

```
#Adds a transaction to the list of current transactions to be
#added to the next block .Returns the index of the block the
#transaction is going to be added to.
```

```
key=supplier_manu
```

```
if(key not in self.current_transactions ):
```

```
    self.current_transactions[key]={}
```

```
if(requested_hash):
```

```
    self.current_transactions[key]['requested_hash']=requested_hash
```

```
if(sent_hash):
```

```
    self.current_transactions[key]['sent_hash']=sent_hash
```

```
if(received_hash):
```

```
    self.current_transactions[key]['received_hash']=received_hash
```

```
if(requested_payment):
```

```
    self.current_transactions[key]['requested_payment']=requested_payment
```

```
if(sent_payment):
```

```
    self.current_transactions[key]['sent_payment']=sent_payment
```

```
if(received_payment):
```

```
    self.current_transactions[key]['received_payment']=received_payment
```

```
return self.last_block['index']+1
```

```
def proof_of_work(self, last_proof):
```

```
#calculates the valid proof using our Proof Of Work Algorithm
```

```

proof=0
while not self.valid_proof(proof,last_proof) :
    proof=proof+1

return proof

def valid_chain(self,chain):
    #Function to check if a chain is valid
    #(checks the hashes and the proof of works of all the blocks of a chain)
    #Returns True if valid chain else returns False
    curr_block=chain[0]
    last_proof=curr_block['proof']
    last_hash=self.hash(curr_block)
    for i in range(1,len(chain)):
        curr_block=chain[i]
        if(curr_block['previous_hash']!=last_hash):
            return False
        if(not self.valid_proof(curr_block['proof'],last_proof)):
            return False
        last_proof=curr_block['proof']
        last_hash=self.hash(curr_block)

    return True

def mine(self):
    last_block=self.last_block
    last_proof=last_block['proof']
    proof=self.proof_of_work(last_proof)
    #blockchain.new_transaction(sender="0",reciever=node_id,amount=1)
    prev_hash=self.hash(last_block)
    new_block=self.new_block(proof,prev_hash)

def consensus(self):
    #Consensus algorithm implementation
    #Gets the longest valid chain among all the nodes.
    #Returns True if chain is changed and False if not.
    all_nodes=self.nodes
    max_chain_length=len(self.chain)
    new_chain=None
    for node in all_nodes:
        response=requests.get(f'http://{node}/chain')
        if response.status_code==200:
            length=response.json()['length']
            chain=response.json()['chain']
            if length>max_chain_length and self.valid_chain(chain) :
                max_chain_length=length
                new_chain=chain

    if new_chain:

```

```

        self.chain=new_chain
        return True

    return False

def check_transactions(self,supplier_manu):
    temp=self.current_transactions.pop(supplier_manu,None)
    if(temp['requested_hash']==temp['sent_hash']
        and temp['sent_hash']==temp['received_hash']
        and temp['requested_payment']==temp['sent_payment']
        and temp['sent_payment']==temp['received_payment']):
        self.to_be_mined[supplier_manu]=temp
        self.mine()
        return True
    return False

    @staticmethod
    def valid_proof(proof,last_proof):
        #Returns True if a proof of work is valid else returns False.
        #For proof to be valid , when hashed with the last proof ,
        #the result should have four zeros at the end (0000) .
        temp=f'{last_proof}{proof}'.encode()
        temp=hashlib.sha256(temp).hexdigest()
        if(temp[:4]=="0000"):
            return True
        else :
            return False

    @staticmethod
    def hash(block):
        #Calculates the hash of a block using the SHA256 algorithm.
        #Returns the hash in hexadecimal.
        block=json.dumps(block,sort_keys=True).encode()#needs to be encoded because
        hashlib.sha256() can only hash bytes .
        return hashlib.sha256(block).hexdigest()

    @property
    def last_block(self):
        #returns the last block of the chain.
        return self.chain[-1]

app=Flask(__name__)#Making blockchain API using Flask microframework

node_id=str(uuid4()).replace('-','')

blockchain=Blockchain()

```

```

#Whenever we want to register new nodes , we send a post request
#to the server at /nodes/register relative URL with the address
#of the node in the POST request's body.
@app.route('/nodes/register',methods=['POST'])
def register():
    values=request.get_json()
    nodes=values.get("nodes")
    if nodes is None:
        return "Error",400
    for node in nodes:
        blockchain.register_node(node)
    response={
        'message':'New nodes have been added',
        'nodes_list':list(blockchain.nodes)
    }
    return jsonify(response),201

#If we want to create a consensus about the validity of the current
#blockchain , we send a GET request at the /nodes/resolve relative
#URL . It uses the Blockchain.consensus() method to download the chains
#of all the nodes in the network and sets the longest valid chain as the
#correct chain.
@app.route('/nodes/resolve',methods=['GET'])
def resolve():
    temp=blockchain.consensus()
    response={}
    if temp:
        response={
            'message':'Chain was replaced',
            'chain':blockchain.chain
        }
    else :
        response={
            'message':'Chain not replaced',
            'chain':blockchain.chain
        }

    return jsonify(response),200

#Simply retrieves the current copy of the blockchain at the node by sending
#a GET request at /chain relative URL.
@app.route('/chain',methods=['GET'])
def full_chain():
    response={
        'chain':blockchain.chain,
        'length':len(blockchain.chain)
    }
    return jsonify(response),200

```



```

@app.route('/Manu/Request',methods=['POST'])
def manu_request():
    values=request.form
    required=['supplier_manu','requested_hash']
    if not all(k in values for k in required):
        return "Missing Data",400

index=blockchain.new_transaction(values['supplier_manu'],requested_hash=values['request
ed_hash'])
    response={'message':f'Manufacturer has sent request'}
    return jsonify(response),201

@app.route('/Supplier/Sent',methods=['POST'])
def supplier_sent():
    values=request.form
    required=['supplier_manu','sent_hash','requested_payment']
    if not all(k in values for k in required):
        return "Missing Data",400

index=blockchain.new_transaction(values['supplier_manu'],sent_hash=values['sent_hash'],
requested_payment=values['requested_payment'])
    response={'message':f'Supplier has sent requested goods and payment request to
manufacturer'}
    return jsonify(response),201

@app.route('/Manu/Received',methods=['POST'])
def manu_received():
    values=request.form
    required=['supplier_manu','received_hash']
    if not all(k in values for k in required):
        return "Missing Data",400

index=blockchain.new_transaction(values['supplier_manu'],received_hash=values['received
_hash'])
    response={'message':f'Manufacturer has received the sent goods'}
    return jsonify(response),201

@app.route('/Manu/Sent',methods=['POST'])
def manu_sent():
    values=request.form
    required=['supplier_manu','sent_payment']
    if not all(k in values for k in required):
        return "Missing Data",400

index=blockchain.new_transaction(values['supplier_manu'],sent_payment=values['sent_paym
ent'])
    response={'message':f'Manufacturer has sent the requested payment to the supplier'}
    return jsonify(response),201

@app.route('/Supplier/Received',methods=['POST'])
def Supplier_received():

```

```

values=request.form
required=['supplier_manu','received_payment']
if not all(k in values for k in required):
    return "Missing Data",400

index=blockchain.new_transaction(values['supplier_manu'],received_payment=values['received_payment'])
temp=blockchain.check_transactions(values['supplier_manu'])
if temp :
    response={'message':f'block has beeb mined'}
    return jsonify(response),200
else:
    response={'message':f'transaction has been cancelled'}
    return jsonify(response),200

@app.route('/test',methods=['GET'])
def test():
    response={
        'current_transactions':blockchain.current_transactions
    }
    return jsonify(response),200

if __name__=='__main__':
    app.run(host='0.0.0.0',port=5000)

```

- [1] define the python `list` of BlockChain's chain.
- [2] create the genesis block.
- [3] define the function that adds new block to BlockChain
- [4] the core code sentence that add new block to BlockChain(python `list`)

SupplyChain combined with Block-chain

If you want to combine the Blockchain and supply-chain, all you need to do is just define the data of genesisBlock, and obtain the information of product you want to save. Then you can begin blockchain with function “makeBlock” happily!

Core Code

```
name=str("bolatoglouAE")
state={u'Company':name}
genesis_Block_state=state
genesis_Block_Contents={u'blockNumber':0,u'parent_Hash':None,u'entire_Count':1,u'entry':genesis_Block}
genesis_Hash=hashMe(genesi s_Block_Contents)
genesis_Block={u'hash':genesis_Hash,u'contents':genesis_Block_Contents}
genesis_Block_Str=json.dumps(genesi s_Block,sort_keys=True)
chain=[genesisBlock]
new = makeBlock(entry1,chain)
```

The “chain” is the genesisBlock, and entry1 is the information of block you want to save. Then a supply-chain node is successfully added to the Blockchain.

Will blockchain technology revolutionize excipient supply chain management?

1.Summarize the reasons why Blockchain is considered an ideal solution for supply chain operations in pharmaceutical bussiness.

First, Blockchain is based on asymmetric encryption modulo mathematics, which is hard to decrypt. Second, Blockchain is a simple way of passing information, which constructs a direct access (P2P) between every two nodes in the supply-chain, and outlaws the intermediaries.

Finally, the block is verified by multiple computers distributed around the net, which makes it impossible to alter the record individually.

2.Why was the paper rejecting the idea that Blockchain will replace traditional quality and auditing processes?

Because traditional quality and auditing processes includes the physical goods, and Blockchain only being responsible for the electronic record. In fact, as each of the supply chain can verify that appropriate audits have been undertaken by appropriately credentialed authorities and can hereby ‘trust’ the whole transactional record.

In other words, a transacted record is computerized and 'blockchained' does not necessarily imply that its physical world counterpart material of commerce has not been tampered with; all it implies is that the transaction record cannot, and has not, been tampered with.

All in all, the blockchain in supply-chain itself is a product of traditional process such as appropriate audit process. Blockchain ensures the information can't be falsified or altered after the event, but can not make sure that the physical goods is qualified before recording.

3.What processes/methods/techniques, along with Blockchain, can you think of to enforce quality audit in excipient supply chain?

When it comes to processes/methods/techniques along with Blockchain that enforce quality audit in excipient supply chain, we can talk about iot sensors and smart contracts.

First, according to the data obtained by iot sensors and machine tools, we can monitor production process and product quality in real time. In this way, suppliers and manufacturers can discover errors in time. Second, the data which has been audited will be uploaded to the blockchain. Finally, we can use smart contract to evaluate the quality of data, process and product, and send back the results to suppliers, manufacturers and retailers.

References:

《Will blockchain technology revolutionize excipient supply chain management?》

《A Blockchain-based Supply Chain Quality Management Framework》