第十七屆旺宏科學獎

成果報告書

參賽編號：SA17-494

作品名稱：以區塊鏈概念
開發現行勞基法適用之智能合約

關 鍵 詞：區塊鏈、勞資問題、智能合約

目錄

摘要 ……1

壹、 研究動機……1

貳、 研究目的……1

參、 研究設備……2

肆、 研究過程……3

伍、 結論……15

陸、 討論……16

柒、 應用……18

捌、 參考資料……24

摘要

隨著科技的發展，網路逐漸引領世界，而如何在網路上取得信任，便成為了眾人注目的難題。

區塊鏈的概念，使我能夠在沒有任何第三方機構的認可下，達到節點間的信任。藉由從底層研究區塊鏈，我得以了解其作為數字貨幣，真正的基礎架構與優缺點；甚至能應用此觀念與技術，透過以太坊及智能合約，打造一平台，來解決勞資糾紛的問題，讓雙方在此合約裡，完全的服從勞基法的規定，技術上達到真正的和諧共處！

壹、研究動機

近年來，人們口口聲聲喊著比特幣，看著節節高漲的價格，各方的聲音與意見在網路上來回穿梭，究竟，比特幣是什麼？區塊鏈又是什麼？

又，區塊鏈的特性除了作為貨幣，難道近期僅能成為一炒作的產品嗎？是否在「去中心化」、「透明公開」、「不可否認」的特色裡，我能夠找到一條不一樣的路，將這門技術發揚光大呢？於此同時，眼看勞資新聞越演越烈，權力、資訊的不對等；政府、勞方、資方三者間的信任關係劣化。區塊鏈與智能合約，是否可以成為其另一出路？

貳、研究目的

依照現行勞基法的規範，將可能出現：資方強迫勞工加班，卻不會給予加班費，而是給予無限延遲之假期；資方給予的假期期限內公司倒閉，而勞方得不到其應得利益；資方給予之加班費不符合約定等等問題。

為了瞭解區塊鏈與改善上述問題，我從根本研究區塊鏈，並希望透過以太坊與智能合約，期望能達到以下目的：

一、 根本了解區塊鏈架構：利用 python 實作一區塊鏈，根本上理解與認識之。

二、 提供勞資一平等平台：提供勞資雙方一公平、公正、公開之平台與介面。

三、 提供去中心合約架構：利用以太坊與智能合約，撰一份不可否認的合約。

四、 推廣區塊鏈與其價值：藉由真正應用區塊鏈，讓大眾認識與學習應用之。

參、研究設備

一、 硬體設備：

紙、筆、電腦、伺服器

二、 軟體與其配置：

(一) Anconda3：python3、jupyter notebook、json、textwrap、time、uuid、flask

(二) visual studio code：python 0.9.1、solidity 0.0.31

(三) Mist：3 Accounts

(四) Ethereum Remix:https：//remix.ethereum.org/

(五) Geth：https://geth.ethereum.org/

三、 環境截圖：

圖 3.1：Anconda3 & jupyter notebook

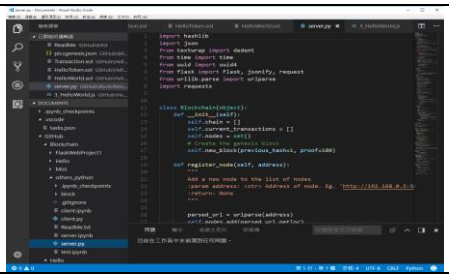


圖 3.2：visual studio code with python 0.9.1

圖 3.3：Mist

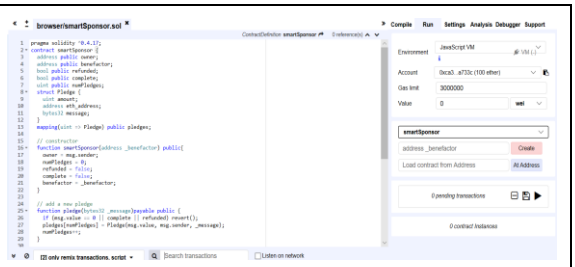


圖 3.4：Ethereum Remix

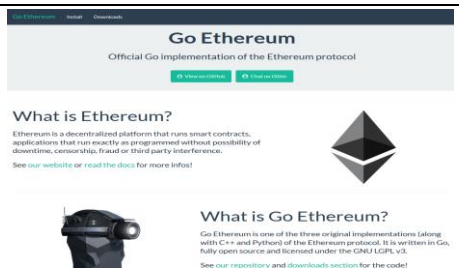


圖 3.5：Geth

肆、研究過程或方法

一、 基本名詞介紹：

(一) 區塊鏈（BlockChain）：

區塊鏈(BlockChain)原本只是比特幣網路的一種記帳技術，近幾年來卻在金融、知、數據交易、電子證照、慈善機構、新能源等領域引起了廣泛的關注。其不外乎有兩大特色：

1. 區塊鏈具有去中心化的特徵,不以參與交易任何一方為中心,這帶來了效率的提升與成本的降低,直接增加了企業的利潤。
2. 區塊鏈具有去信任的特徵，也就是假定交易中的任何一方都是不可信任的。透過記錄不可抵賴的交易的信息，來使交易各方遵守誠信。

區塊鏈（BlockChain）是透過「去中心化」和「去信任化」的方式來共同維護資料庫的技術，其將資料庫分別放在不同的節點裡保存，並互相監控數據，資料更動須其餘節點的共識同意。它讓交易過程中每個節點的每一筆交易，都能透明、安全地被紀錄下來。而各交易被廣播到全網後一段時間，會被收進一個區塊（Block），和全部的節點複製共享；隨著時間區塊一個一個生成形成一條鏈（Chain）。而所有節點都會擁有所有交易紀錄，共同驗證鏈上資料的正確性，故極難利用少數節點竄改歷史數據。而後續我將對此進行研究與討論。

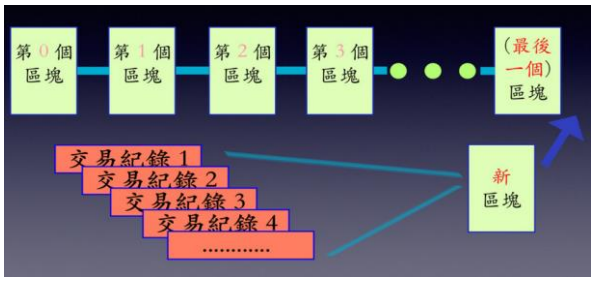


圖 4.1：區塊鏈基礎架構

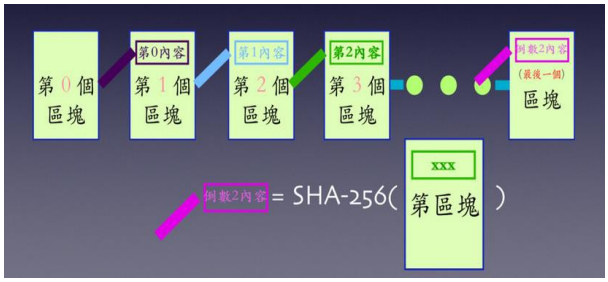


圖 4.2：區塊鏈實現方式

(二) Python：

Python，是一種廣泛使用的高階程式語言，屬於通用型程式語言，能讓開發者能夠用更少的代碼表達想法。

(三) Anconda3：

Anaconda 是一種 Python 語言的免費增值開源發行版，用於進行大規模數據處理，預測分析，和科學計算，致力於簡化包的管理和部署。

二、 區塊鏈代碼實作：

(一) 區塊架構：

以上述之比特幣架構、比特幣白皮書及其原文翻譯為主要研究對象。文章連結標示於:捌、參考資料及其他之 四、比特幣白皮書。

(二) 代碼實做：

在此階段，我將實現除了白皮書中，「網路」篇以外的大部分特色，而「網路」篇則將交由下個部分的以太坊做實作與解釋。

1. 引入需要使用的套件。

```
import hashlib
import json
from textwrap import dedent
from time import time
from uuid import uuid4
from flask import Flask, jsonify, request
from urllib.parse import urlparse
import requests
```

2. 初始化類別。

```
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.nodes = set()
        # 創建創世塊
        self.new_block(previous_hash=1, proof=100)
```

3. 註冊節點。

```
    def register_node(self, address):
        """
        在清單中加入新的節點
        :param address: <str> Address of node. Eg. 'http://192.168.0.5:5000'
        :return: None
        """

        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)
```

4. 新增一個新區塊。

```
    def new_block(self, proof, previous_hash=None):
        # 創建一個新的塊並將其添加到鏈中
        """
        生成新塊
        :param proof: <int> The proof given by the Proof of Work algorithm
        :param previous_hash: (Optional) <str> Hash of previous Block
        :return: <dict> New Block
        """

        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

        # 重置當前的交易列表
        self.current_transactions = []

        self.chain.append(block)
        return block
```

5. 新增一筆新交易。

```
def new_transaction(self, sender, recipient, amount):
    """
    生成新交易信息，信息將加入到下一個待挖的區塊中
    :param sender: <str> Address of the Sender
    :param recipient: <str> Address of the Recipient
    :param amount: <int> Amount
    :return: <int> The index of the Block that will hold this transaction
    """

    self.current_transactions.append({
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
    })

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

6. 得到一個一個區塊鏈的 Hash 值。

```python
@staticmethod
# 從外部非實體化調用函數
def hash(block):
    """
    生成塊的 SHA-256 hash值
    :param block: <dict> Block
    :return: <str>
    """

    # 我們必須確保字典是有序的，否則我們將有不一致的哈希值
    block_string = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_string).hexdigest()
```

7. 回傳最後一個區塊，驗證 proof 的結果是否正確。

```python
@property
# 檢查參數後再傳入
def last_block(self):
    return self.chain[-1]

@staticmethod
def valid_proof(last_proof, proof):
    """
    驗證證明：是否hash(last_proof, proof)以4個0開頭？
    :param last_proof: <int> Previous Proof
    :param proof: <int> Current Proof
    :return: <bool> True if correct, False if not.
    """

    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:4] == "0000"
```

8. 驗證整條區塊鍊是否合法。

```python
def valid_chain(self, chain):
    """
    Determine if a given blockchain is valid
    :param chain: <list> A blockchain
    :return: <bool> True if valid, False if not
    """

    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        print(f'{last_block}')
        print(f'{block}')
        print("\n-----------\n")
        # 檢查塊的hash是否正確
        if block['previous_hash'] != self.hash(last_block):
            return False

        # 檢查工作證明是否正確
        if not self.valid_proof(last_block['proof'], block['proof']):
            return False

        last_block = block
        current_index += 1

    return True
```

9. 尋找更長的鏈，並解決共識衝突。

```python
    def resolve_conflicts(self):
        """
        共識算法解決衝突
        使用網路中最長的鏈.
        :return: <bool> True 如果鏈被取代, 否則為False
        """

        neighbours = self.nodes
        new_chain = None

        # 只尋找比我們更長的鏈條
        max_length = len(self.chain)

        # 抓取並驗證我們網絡中所有節點的鏈
        for node in neighbours:
            response = requests.get(f'http://{node}/chain')

            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                # 如果我們發現一個比我們的更長的、新的有效鏈條，就取代我們的鏈條
                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain

        # 如果我們發現一個比我們的更長的新的有效鏈條，就取代我們的鏈條
        if new_chain:
            self.chain = new_chain
            return True

        return False
```

10. 藉由 Flask 部署各種方法於 http 協議上。

```python
# 實體化我們的節點
app = Flask(__name__)

# 建立一個node節點
node_identifier = str(uuid4()).replace('-', '')

# 實體化區塊
blockchain = Blockchain()
```

11. 透過驗證 proof 建立區塊，並將交易包含於其中。

```python
@app.route('/mine', methods=['GET'])
def mine():
    # 使用 proof of work 演算法來得到下一個 proof...
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)

    # 给工作量證明的節點提供獎勵.
    # 發送者為 "0" 表明是新挖出的幣
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
    )

    # 通過將其添加到鏈中來創造新的塊
    block = blockchain.new_block(proof)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
```

12. 傳送一筆新的交易，並置於交易清單中等待包覆。

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # 確定是一個可實行的交易（資料充足與完備）
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # 創建一新的交易
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201


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

13. 傳送已身節點位置，使節點彼此串連。

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()
    print(values)
    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

14. 透過遍歷尋找最長的鏈並取代之。

```
@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

14. 執行於區網之中。

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

三、 區塊鏈運行結果：

```
127.0.0.1 - - [22/Jan/2018 22:14:29] "POST /transactions/new HTTP/1.1" 500 -
127.0.0.1 - - [22/Jan/2018 22:14:40] "POST /transactions/new HTTP/1.1" 201 -
127.0.0.1 - - [22/Jan/2018 22:18:15] "POST /nodes/register HTTP/1.1" 201 -
{'nodes': ['http://0.0.0.0:5001']}
{'0.0.0.0:5001'}
127.0.0.1 - - [22/Jan/2018 22:22:37] "GET /nodes/resolve HTTP/1.1" 200 -
{'index': 1, 'previous_hash': 1, 'proof': 100, 'timestamp': 1516629852.086288, 'transactions': []}
{'index': 2, 'previous_hash': 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', 'proof': 35293, 'ti
mestamp': 1516630877.7717729, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender
': '0'}]}

-----------

{'index': 2, 'previous_hash': 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', 'proof': 35293, 'ti
mestamp': 1516630877.7717729, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender
': '0'}]}
{'index': 3, 'previous_hash': 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', 'proof': 35089, 'ti
mestamp': 1516630879.711094, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender'
: '0'}]}

-----------

{'index': 3, 'previous_hash': 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', 'proof': 35089, 'ti
mestamp': 1516630879.711094, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender'
: '0'}]}
{'index': 4, 'previous_hash': '229728353739f40ac78711de0ffec649539ec5cbe3587c66eb434a2e67992fea', 'proof': 119678, 't
imestamp': 1516630881.634832, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender
': '0'}]}
```

四、 以太坊名詞介紹：

(一) 以太坊：

以太坊的目的是基於腳本、競爭幣和鏈上元協議(on-chain meta-protocol)概念進行整合和提高,使得開發者能夠創建任意的基於共識的、可擴展的、標準化的、特性完備的、易於開發的和協同的應用。以太坊通過建立終極的抽象的基礎層-內置有圖靈完備編程語言的區塊鏈-使得任何人都能夠創建合約和去中心化應用並在其中設立他們自由定義的所有權規則、交易方式和狀態轉換函數。域名幣的主體框架只需要兩行代碼就可以實現,諸如貨幣和信譽系統等其它協議只需要不到二十行代碼就可以實現。智能合約-包含價值而且只有滿足某些條件才能打開的加密箱子-也能在我的平台上創建,並且因為圖靈完備性、價值知曉(value-awareness)、區塊鏈知(blockchain-awareness)和多狀態所增加的力量而比比特幣腳本所能提供的智能合約強大得多。

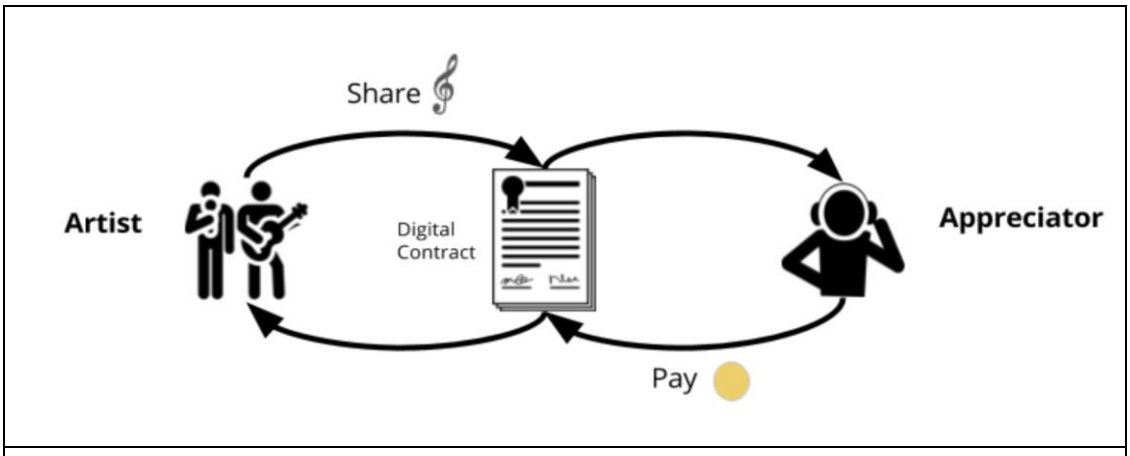


圖 4.3:智能合約示意圖

(二) 智能合約：

智能合約程序不只是一個可以自動執行的計算機程序:它自己就是一個系統參與者。它對接收到的信息進行回應,它可以接收和儲存價值,也可以向外發送信息和價值。這個程序就像一個可以被信任的人,可以臨時保管資產,總是按照事先的規則執行操作。

下面這個示意圖就是一個智能合約模型:一段代碼(智能合約),被部署在分享的、複製的賬本上,它可以維持自己的狀態,控制自己的資產和對接收到的外界信息或者資產進行回應。

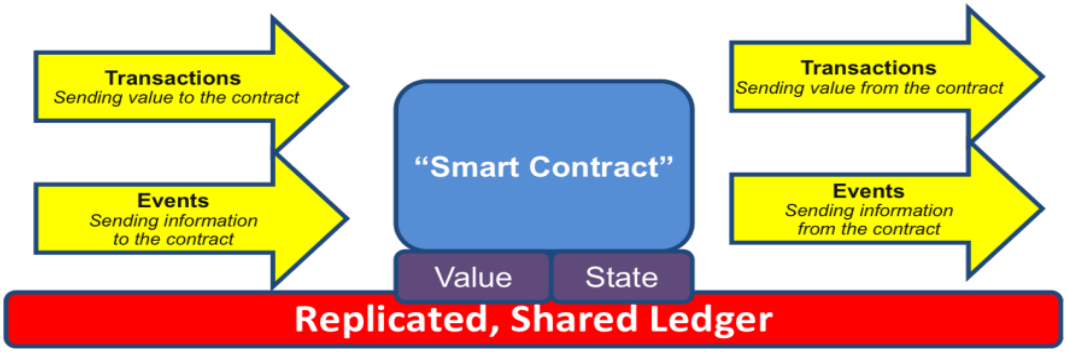


圖 4.4:智能合約流程圖

(三) Solidity 語言：

Solidity 是一種語法類似 JavaScript 的高級語言。它被設計成以編譯的方式生成以太坊虛擬機代碼。使用它很容易創建各式的合約。

(四) Geth 系統：

Geth 又名Go Ethereum. 是以太坊協議的三種實現之一,由Go語言開發,完全開源的項目。通過 Geth 的一些基本命令,可以很方便的創建出一個以太坊的私有鏈條。

五、 智能合約建構：

(一) 合約設計：

1. 設計理念：

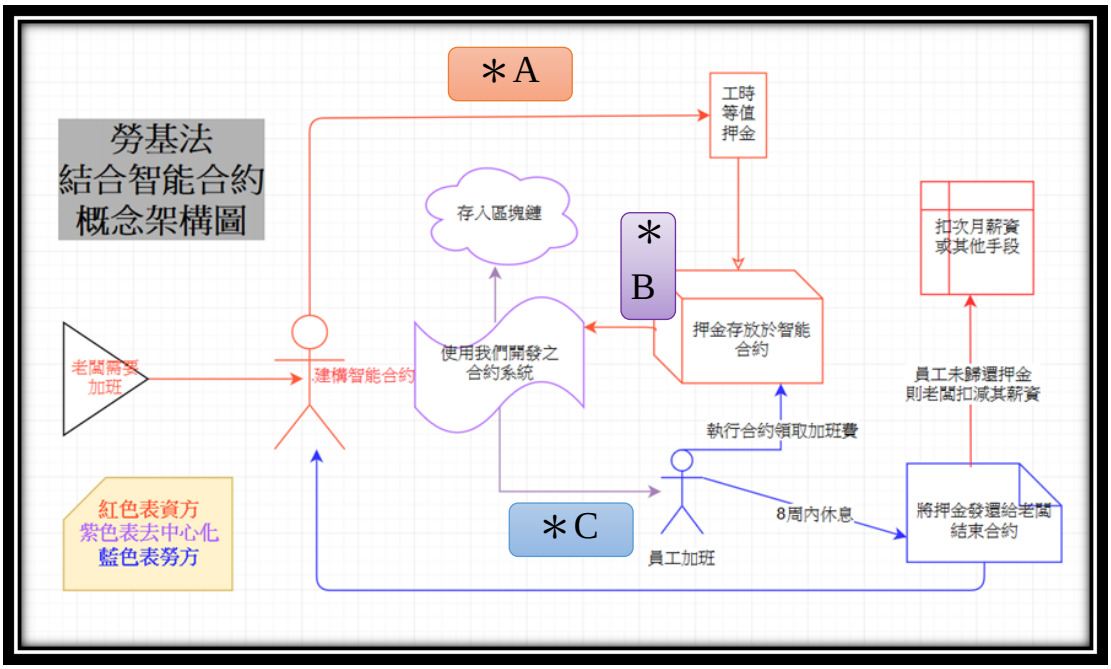
(1) 勞資雙方需互相平等或得以互相牽制。

(2) 不需仰賴任何政府或其他中心化機構。

(3) 法律所規定須嚴格設定於合約代碼中。

(4) 法律未規定之盡可能化為可設定參數。

2. 製作原型：



3. 模型說明：

＊A：為了避免發不出工資等情形,先將錢存入不可更改之智能合約中。

＊B：連同已經寫好的代碼與參數,一同放入 Mist 錢包中編譯並上傳。

＊C：員工確認合約。

(二) 代碼實做：

1. 定義各項參數以便後續使用

```
pragma solidity ^0.4.17;

contract Transaction {
    //定義勞方，資方
    address employee;
    address employer;
    //定義合約創建時間
    uint create_time;
    //定義基本工資
    uint base_wage;
    //定義應付工資
    uint payables;
    //定義補修期限
    uint expire = 8;
    //定義工作時數
    uint working_hours;
    //定義每wei對台幣之轉換
    uint ETH_to_TWD_100 = 300000000000000;
    //定義補修期限是否結束
    bool ended = false;
    //定義合約是否合格
    bool usaged = false;
```

2. 建置初始化方法,用來實現基礎功能

```
    //合約啟動時之函數（勞方地址,工作時數，基礎薪資）[設定為可傳入以太幣][設定為公開函數]
    function Transaction(address new_employee, uint new_working_hours ,uint new_base_wage)payable public{
        //導入變數
        employee = new_employee;
        employer = msg.sender;
        working_hours = new_working_hours;
        base_wage=new_base_wage;

        //依勞基法判斷加班薪資
        if (working_hours >= 2) payables = (working_hours-2)*base_wage*266+2*233*base_wage;
        else payables = working_hours*233*base_wage;

        //判斷薪資是否合格
        if (msg.value>=payables/ETH_to_TWD_100) usaged=true;
        else return;
```

3. 建立補充方法,補充初始化時誤輸的資料

```
//合約修改之函數（新增工作時數）[設定為可傳入以太幣][設定為公開函數]
function add_more(uint new_working_hours )payable public{
    //如果合約已結束，則回傳以太幣
    if (ended == true) revert();

    //否則導入變數
    working_hours+=new_working_hours;

    //依勞基法判斷加班薪資
    if (working_hours >= 2) payables = (working_hours-2)*base_wage*266+working_hours*233*base_wage;
    else payables = working_hours*233*base_wage;

    //判斷薪資是否合格
    if (msg.value>=payables/ETH_to_TWD_100) usaged=true;
    else return;

    //合格及建立合約
    create_time=now;
}
```

4. 建立勞工領錢方法,與超越八周未休假時使用

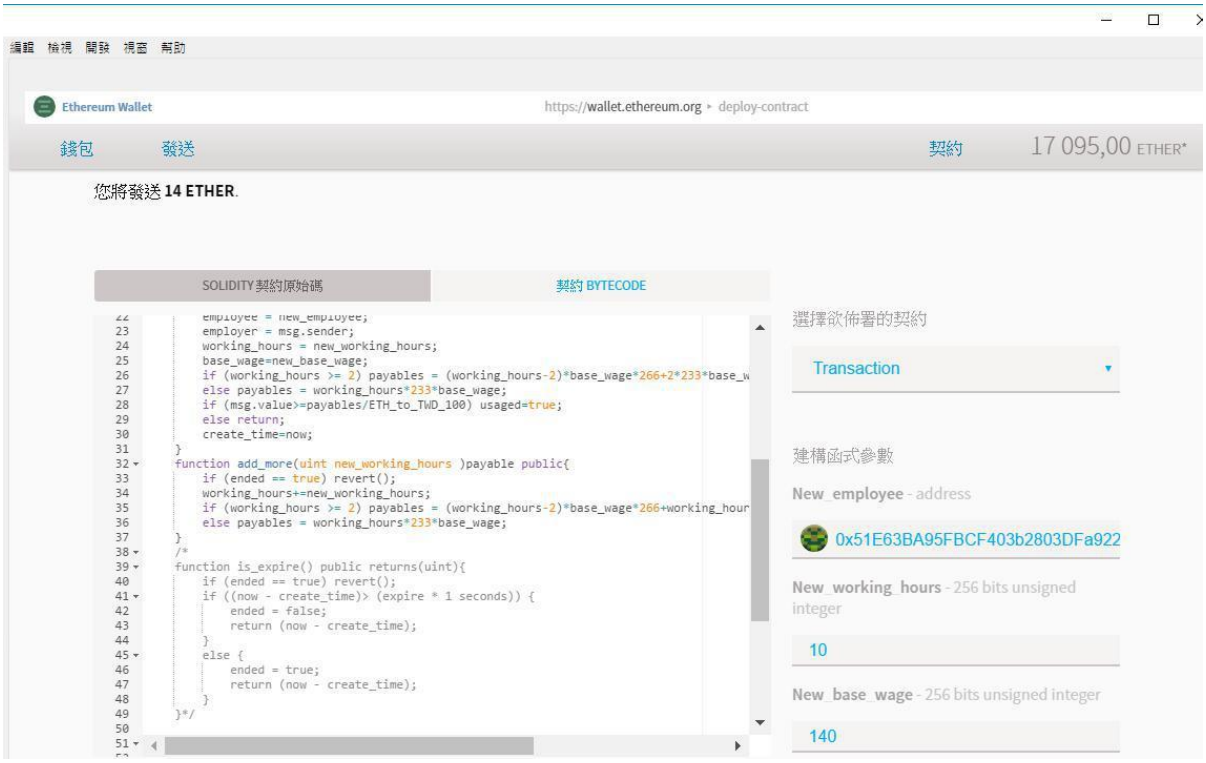
```
// 勞工依法取回加班薪資[設定為公開函數]
function collectMoney() public {
    //如果時間已超過8周
    if ((now - create_time) > (expire * 1 weeks)) {
        //設定合約以結束
        ended = true;
    }
    //否則設定合約繼續
    else {
        ended = false;
    }

    //若申請者非勞方，或合約未結束，或合約不可使用，則不執行
    if (msg.sender!=employee || ended == false || usaged == false) revert();
    //否則執行函數transfer傳送押金(薪資)給勞方
    employee.transfer(this.balance);
}
```

5. 建立返回方法,使勞方有權力能控制薪資。

```
// 若已休假，勞方回傳押金給資方[設定為公開函數]
function returnMoney () public {
    //若申請者非勞方，或合約未結束，則不執行
    if (msg.sender!=employee || ended == true ) revert();
    //否則執行函數transfer傳送押金給資方
    employer.transfer(this.balance);
}
}
```

六、 智能合約結果展示：


編輯 檢視 開發 視窗 幫助
Ethereum Wallet
https://wallet.ethereum.org ▸ deploy-contract
錢包
發送
契約
17 095,00 ETHER*
您將發送 14 ETHER.
SOLIDITY 契約原始碼
契約 BYTECODE
employee = new_employee;
employer = msg.sender;
working_hours = new_working_hours;
base_wage=new_base_wage;
if (working_hours >= 2) payables = (working_hours-2)*base_wage*266+2*233*base_w
else payables = working_hours*233*base_wage;
if (msg.value>=payables/ETH_to_TWD_100) usaged=true;
else return;
create_time=now;
}
function add_more(uint new_working_hours)payable public{
if (ended == true) revert();
working_hours+=new_working_hours;
if (working_hours >= 2) payables = (working_hours-2)*base_wage*266+working_hour
else payables = working_hours*233*base_wage;
}
/*
function is_expire() public returns(uint){
if (ended == true) revert();
if ((now - create_time)> (expire * 1 seconds)) {
ended = false;
return (now - create_time);
}
else {
ended = true;
return (now - create_time);
}
}*/
選擇欲佈署的契約
Transaction
建構函式參數
New_employee - address
0x51E63BA95FBCF403b2803DFa922
New_working_hours - 256 bits unsigned integer
10
New_base_wage - 256 bits unsigned integer
140

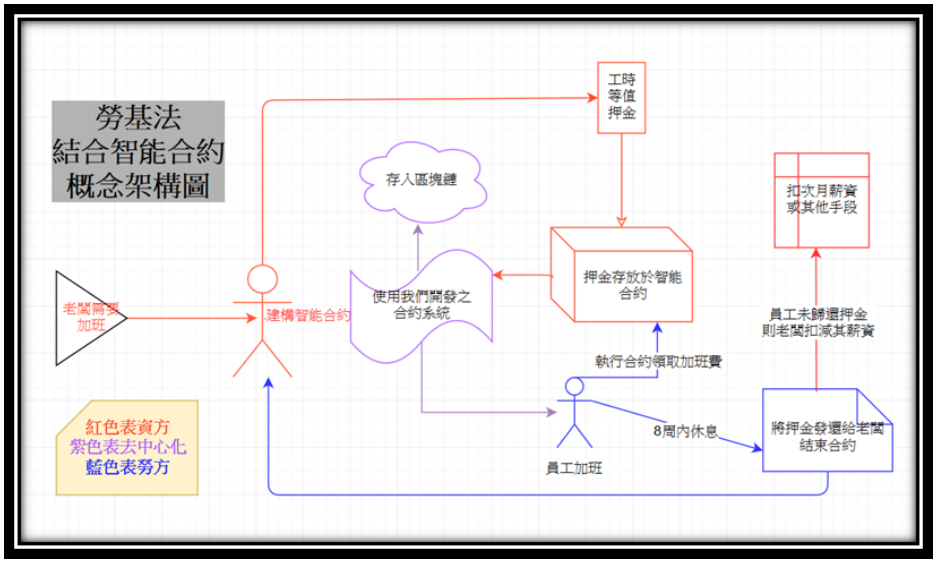
伍、結論

本研究總共利用了 BlockChain、Python、Ethereum、Solidity、Smart Contract、Geth、Mist 等科技，開發出對勞資雙方皆公平可靠的智能合約與使用平台。

透過研究 BlockChain 並實作之，我得以理解一種嶄新的資料儲存與分享方式，使一切訊息都變得公開透明、一切記錄都不可否認，除此之外，更得以確保了資料的安全性；而在研究區塊鏈的同時，有鑑於國內現況——因一例一休等相關之立法，造成社會諸多問題，例如：嘩眾取寵、勞資糾紛等，導致勞工並未得到政府設想的「假期」與「工資」。那究竟有無良方，能讓勞、資、國家均滿意呢？

我便突發奇想，以區塊鏈為基礎概念，掌握其公開透明、安全、不可否認等特性，嘗試開發適用於勞基法之智能合約（如下圖），使未來勞資雙方的合約不再是由任何一方控制；金流更不僅只掌握於資方之手，重新賦予了勞資雙方平等的權力！透過我設計的智能合約，給予了勞基法一個適當的歸所，使勞資方得以在一個更適合的環境中，制定其專屬的智能合約。

此研究除了提供政府一種新的思維，得以用不同的角度去正視勞資問題，尋找新的解決方案；更期望在廣泛應用此觀念後，能使整個勞資生態與區塊鏈技術，以至於其他相關用途之場所，共同成長、欣欣向榮。



而此研究之未來展望如下：

一、 因使用以太幣作為交易貨幣，可能因貨幣價值不穩定，導致價格差異。期望能引進即時價格，將差異減至最小，提供更穩定的合約。

二、 合約內容可能含有漏洞，需要公布或請駭客審查之，才能夠正式使用。

三、 盼望與政府合作，推廣此合約與技術，確實應用於勞資市場，實際達成目的。

四、 於部分公司嘗試使用合約，蒐集資料與數據，更加完善系統。

五、 考慮更多勞資方需求與實務細節，使系統更貼近生活與使用習慣。

隨著科技的發展，網路逐漸引領世界，而如何在網路上取得信任，便成為了眾人注目的難題。區塊鏈的概念，使我能夠在沒有任何第三方機構的認可下，達到節點間的信任。藉由從底層研究區塊鏈，我得以了解其作為數字貨幣，真正的基礎架構與優缺點；甚至能應用此觀念與技術，透過以太坊及智能合約，打造一平台，來解決勞資糾紛的問題，讓雙方在此合約裡，完全的服從勞基法的規定，技術上達到真正的和諧共處！

此研究除了提供一種新的思維，得以用不同的角度去正視勞資問題，尋找新的解決方案；更期望在廣泛應用此觀念後，能使整個勞資生態與區塊鏈技術，以至於其他相關用途之場所，共同成長、欣欣向榮。

陸、討論

Q：為何不使用 python 區塊鏈製作合約而採用以太坊？
A：因為 python 區塊鏈製作合約時，我無一統一而有價值的代幣；若要於 python 區塊鏈實現，則必須交由政府發行，那便失去去中心化的意義，且若政府非可信的一方，那勞工的資產將有危險。
Q：Solidity 語言中的 now 不一定準確，以太坊的挖礦也可能造成時間的延遲，請問如何解決時間差的問題？
A：因為區塊鏈必須由礦工共同確認，故在安全的基礎上時間延遲是不可避免的，但因合約時間單位較久（數週），不因此影響此操作。
Q：為何不採用紙本合約？紙本合約具有法律效力及法律的保障。
A：有鑒於現今社會勞方常遇到打壓，勞方多無法得到一個有力的回覆。透過法律途徑卻往往勞民傷財，最後又回歸集體罷工或上街抗議，造成兩敗俱傷的結局，故採用智能合約，以不信任任何一個節點為出發點，那麼將不存在毀約的問題，也能促進社會和諧。
Q：為何最後是由勞方返回押金，而不是由資方主動贖回？
A：原因同上，社會勞方常處於不利的立場，故若由資方贖回，可能會造成紛爭，例如勞方未確實休假等等。故選擇由勞方贖回，給予權力的平等。

Q：若勞方為返回押金，資方權利是否受損？

A：資方因掌有較多的權力，可藉由扣減次月薪水，扣減福利或其他手段避免此類事情發生。

Q：若程式設定錯誤，是否押金將永存於合約中不得取出？

A：是的，因為是去中心化的系統，將沒有任何客服可以協助，技術上更是不允許，否則會造成區塊鏈的崩毀。但為防範此狀況，設立 add 方法，避免資方在輸入過少金額時，造成的合約不成立（usaged == false），藉此給資方一個更改合約的機會。

Q：勞方如何得知資方已確實創立合約，而非只是隨口胡言？

A：資方可傳送合約地址及其 json 檔案，供勞方於 Mist 錢包中查看合約概況。

Q：資方如何得知勞方已確實返還押金，而非只是隨口胡言？

A：資方可查看合約地址或己身錢包紀錄，於 Mist 錢包中確認金流。

Q：紙本合約與智能合約的比較。

A：

	現今紙本合約	此研究之智能合約
公正第三方存在	需要	不需要
合約建立是否平等	不平等，多利資方	平等
合約用途與效益	往往說一套做一套	強制遵守，不得抗命
合約不成立時的影響	法律訴訟、上街抗議	呼叫方法，強制履行
社會普及與應用程度	成熟，但有紛爭與疑慮	發展中
若實際引入職場，能否符合勞基法規範？	不容易，因工資常遭到資方控制，而無法完全實施	可以，倚賴不可竄改特性，得以輕易實行合約內容

柒、研究結果

一、 成功用 Python 建造一個區塊鏈架構，並測試之。詳情如圖 5.1.1~5.1.4

```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [22/Jan/2018 22:05:43] "GET /mine HTTP/1.1" 200 -
```

JSON 原始資料 檔頭

儲存 複製

```
index:          3
message:        "New Block Forged"
previous_hash:  "0a7e9ec1d4d1b81decc44016d85ecf81e538c8014f5940d4dd323778c0463e19"
proof:          35089
transactions:
  0:
    amount:     1
    recipient:  "3e34a3f9511148ca95bdc1e518a46fdf"
    sender:     "0"
```

圖 7.1.1 藉由/mine 挖礦：http://localhost:5000/mine

```
127.0.0.1 - - [22/Jan/2018 22:14:40] "POST /transactions/new HTTP/1.1" 201 -
```

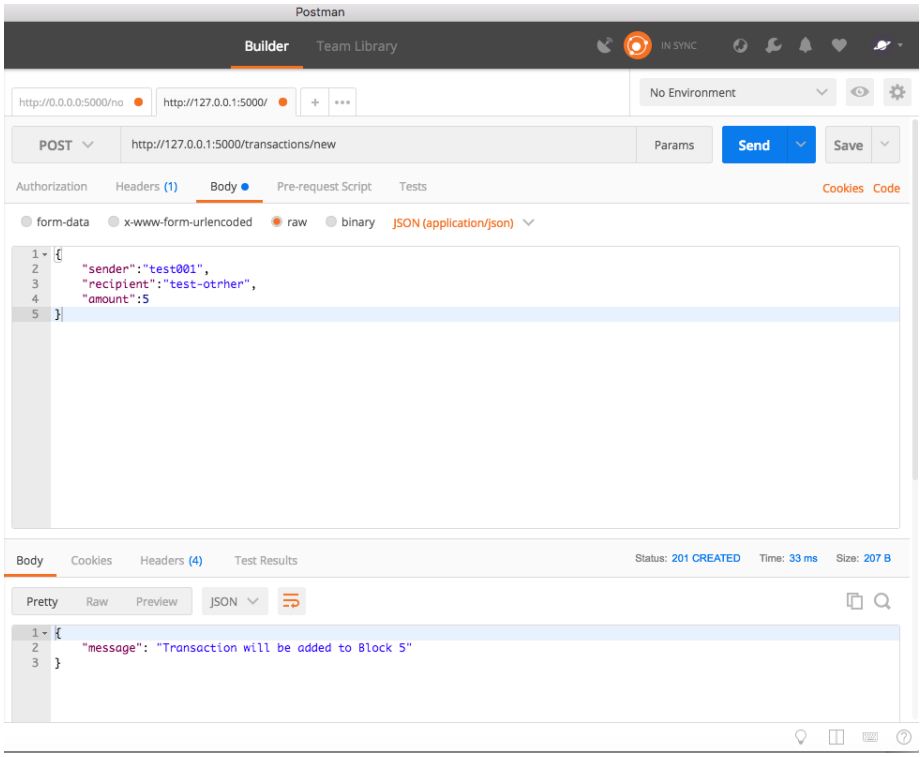


圖 7.1.2 藉由/transactions/new 新增交易：http://127.0.0.1:5000/transactions/new

```
127.0.0.1 - - [22/Jan/2018 22:18:15] "POST /nodes/register HTTP/1.1" 201 -
```

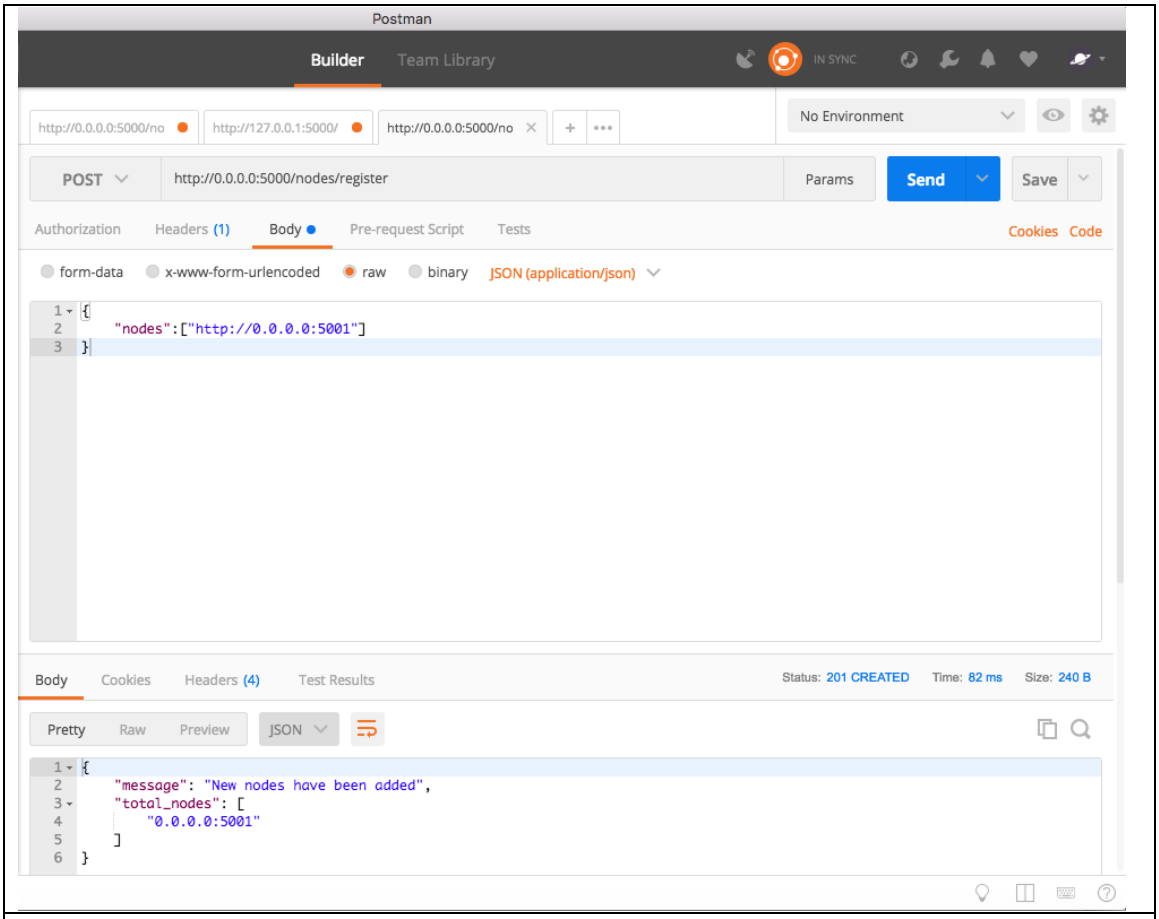


圖 7.1.3 藉由/nodes/register 新增交易：http://0.0.0.0:5000/nodes/register

```
127.0.0.1 - - [22/Jan/2018 22:22:37] "GET /nodes/resolve HTTP/1.1" 200 -
{'index': 1, 'previous_hash': 1, 'proof': 100, 'timestamp': 1516629852.086288, 'transactions': []}
{'index': 2, 'previous_hash': 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', 'proof': 35293, 'ti
mestamp': 1516630877.7717729, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender
': '0'}]}

-----------

{'index': 2, 'previous_hash': 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', 'proof': 35293, 'ti
mestamp': 1516630877.7717729, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender
': '0'}]}
{'index': 3, 'previous_hash': 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', 'proof': 35089, 'ti
mestamp': 1516630879.711094, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender'
: '0'}]}

-----------

{'index': 3, 'previous_hash': 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', 'proof': 35089, 'ti
mestamp': 1516630879.711094, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender'
: '0'}]}
{'index': 4, 'previous_hash': '229728353739f40ac78711de0ffec649539ec5cbe3587c66eb434a2e67992fea', 'proof': 119678, 't
imestamp': 1516630881.634832, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender
': '0'}]}
```

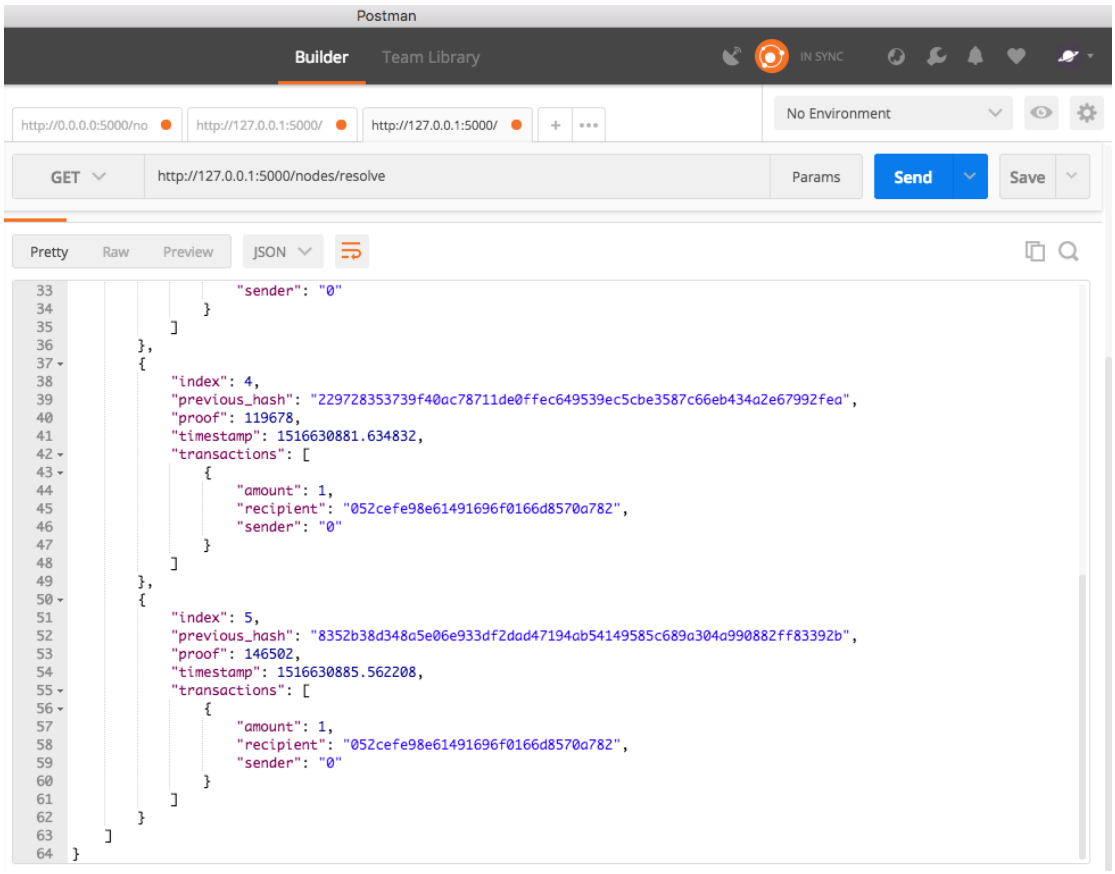
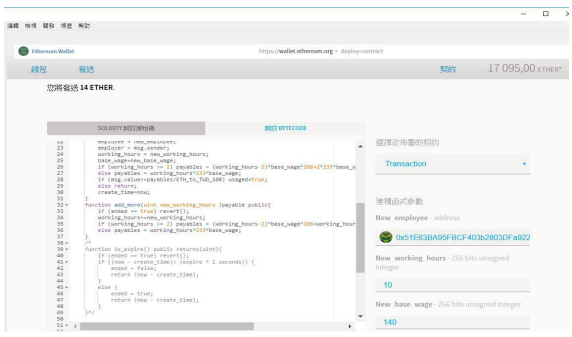


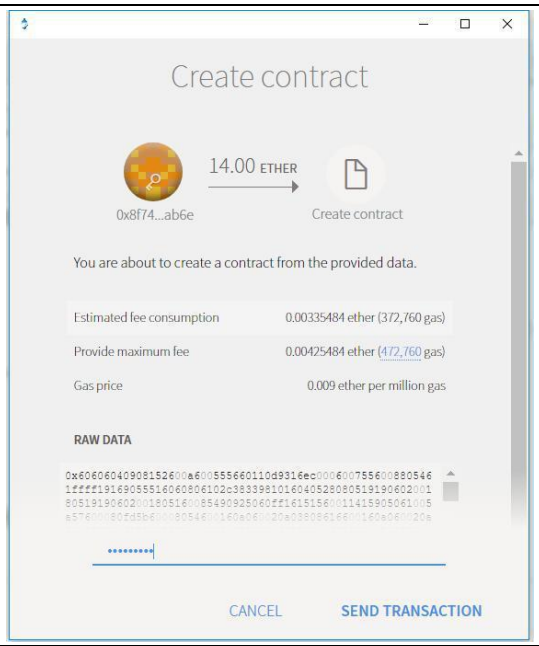
圖 7.1.4 藉由/nodes/resolve 新增交易：http://127.0.0.1:5000/nodes/resolve

二、 成功建構一智能合約，得已應用於現行法律之中。詳情如圖 5.2.1~5.2.10

7.2.1 起初帳戶總覽

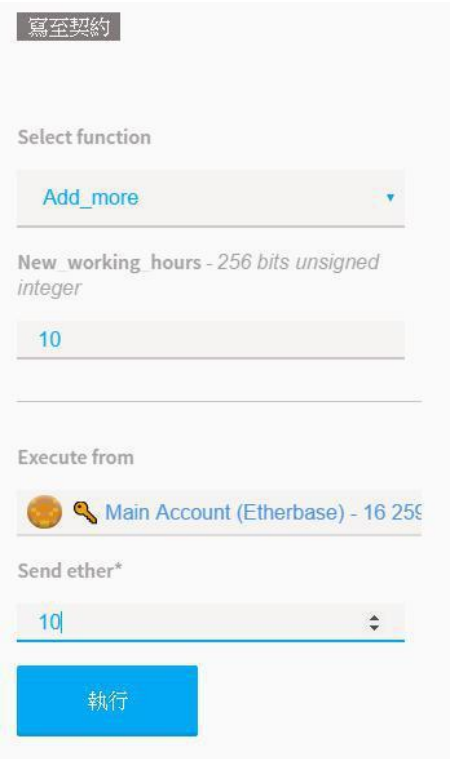


7.2.2 建設智能合約

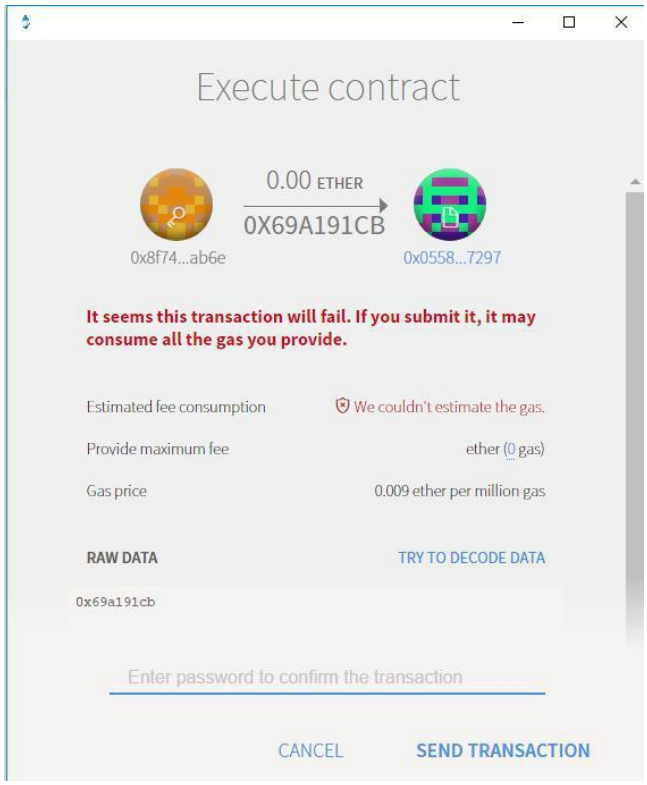


7.2.3 建設合約認證

7.2.4 合約寫入區塊鏈

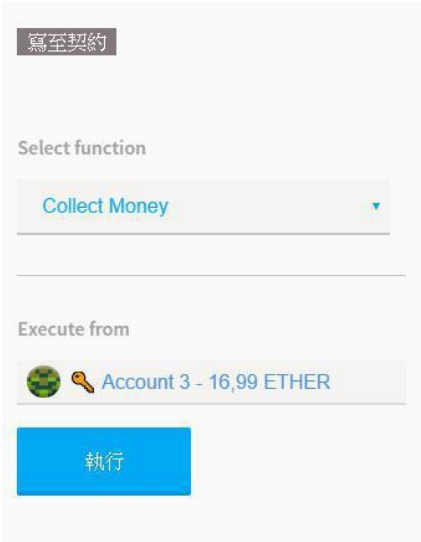


7.2.5 Add 方法新增押金



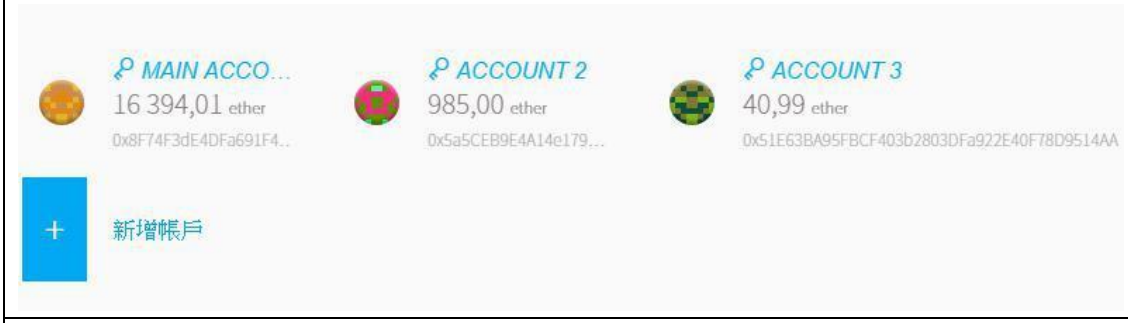
7.2.6 合約驗證

7.2.7 合約寫入區塊鏈

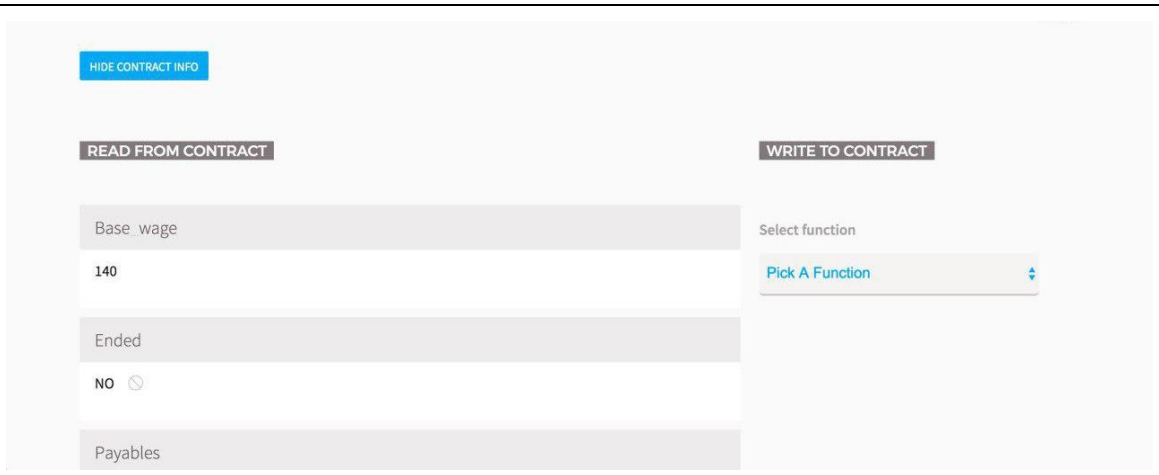


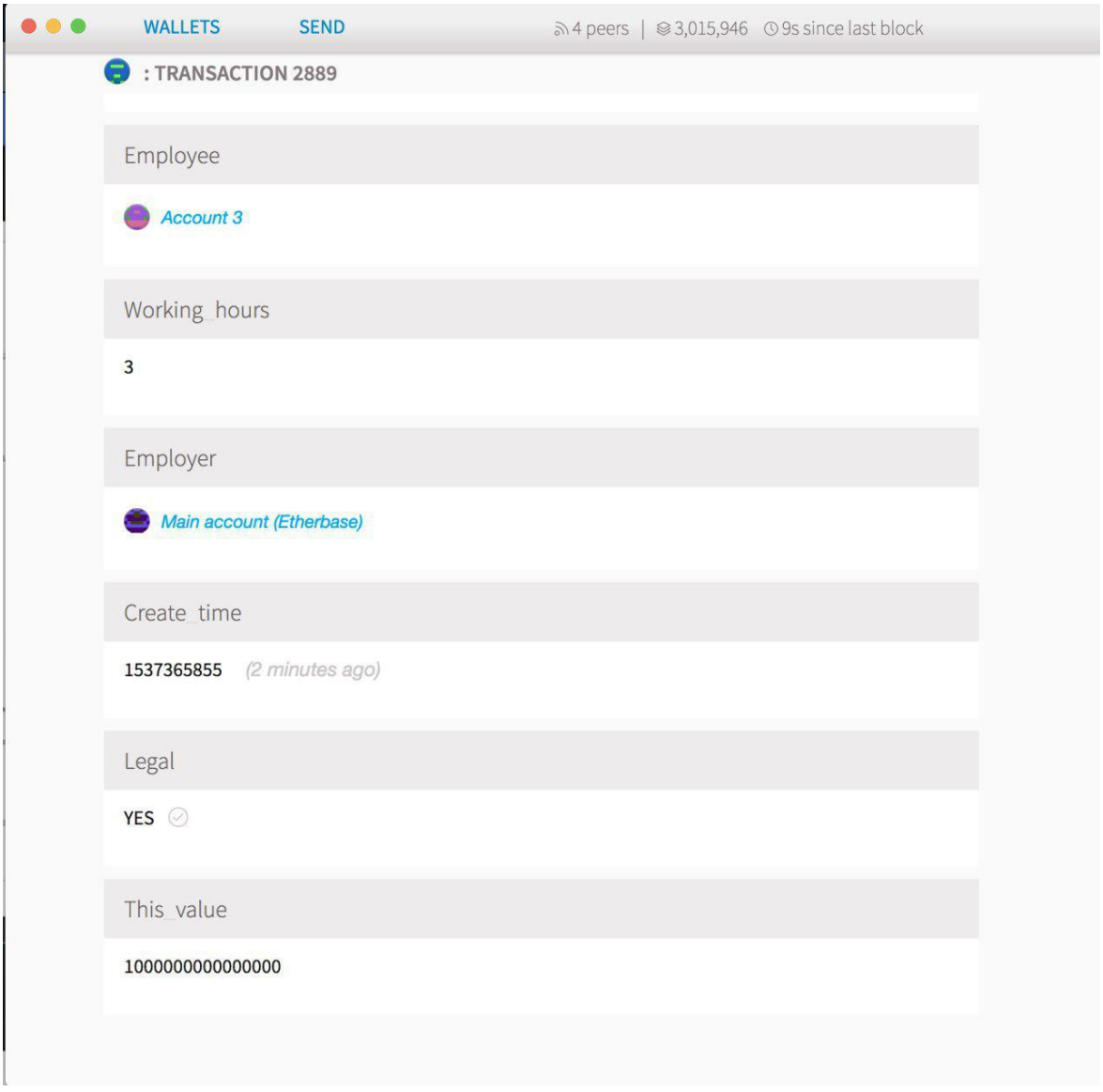
7.2.8 勞方使用 Collect 方法，拿到工資

7.2.9 合約寫入區塊鏈



7.2.10 勞方獲得工資（ACCOUNT3）


HIDE CONTRACT INFO
READ FROM CONTRACT
WRITE TO CONTRACT
Base_wage
140
Ended
NO
Payables
Select function
Pick A Function


WALLETS
SEND
4 peers | 3,015,946 9s since last block
: TRANSACTION 2889
Employee
Account 3
Working_hours
3
Employer
Main account (Etherbase)
Create_time
1537365855 (2 minutes ago)
Legal
YES
This_value
1000000000000000

7.2.11 合約內容顯示

捌、參考資料及其他

一、 科展作品

(一) 57 屆科展：使用區塊鏈開發可監督捐款之公開募捐平台。取自：http://activity.ntsec.gov.tw/activity/race-1/57/pdf/052505.pdf

二、 觀念建構

(一) 區塊鏈技術概觀（一）：讓我們從歷史文本說起。取自：https://finance.technews.tw/2017/08/01/block-chain-technology-overview/

(二) 區塊鏈技術概觀（二）：資料分割與密碼學。取自：https://technews.tw/2017/08/09/blockchain-principle-part-two/

(三) 區塊鏈技術概觀（三）：多數決的運作結構。取自：https://technews.tw/2017/08/16/blockchain-principle-part-three/

(四) 區塊鏈技術指南。取自：https://yeasy.gitbooks.io/blockchain_guide/content/

三、 Python 語法

(一) "@"裝飾器概念。取自：https://foofish.net/python-decorator.html

(二) Python & Flask。取自：http://blog.techbridge.cc/2017/06/03/python-web-flask101-tutorial-introduction-and-environment-setup/

四、 比特幣白皮書

(一) 原文。取自：https://bitcoin.org/bitcoin.pdf

(二) 譯文。取自：https://medium.com/taipei-ethereum-meetup/%E6%AF%94%E7%89%B9%E5%B9%A3-%E7%AB%AF%E5%B0%8D%E7%AB%AF%E9%9B%BB%E5%AD%90%E7%8F%BE%E9%87%91%E7%B3%BB%E7%B5%B1-bitcoin-a-peer-to-peer-electronic-cash-system-i-8a52de003c9

五、 python 實作區塊鏈

(一) 用 Python 從零開始創建區塊鏈。取自：http://www.jianshu.com/p/cc6663cbbc41

(二) 50 行 Python 代碼構建一個區塊鏈。取自：http://blog.csdn.net/simple_the_best/article/details/75448617

六、 乙太訪運作與實作：

(一) 以太坊白皮書。取自：https://github.com/ethereum/wiki/wiki/%5B%E4%B8%AD%E6%96%87%5D-%E4%BB%A5%E5%A4%AA%E5%9D%8A%E7%99%BD%E7%9A%AE%E4%B9%A6

(二) 智能合約。取自：https://blog.gasolin.idv.tw/2017/08/13/got-my-ens-domain/

(三) 以太坊錢包 Parity。取自：https://medium.com/taipei-ethereum-meetup/%E5%9C%A8parity%E5%89%B5%E5%BB

%BA%E4%BB%A5%E5%A4%AA%E5%9D%8A%E9%8C%A2%E5%8C%85-b2c2977378fc

(四) 教練，我”只”想學 Solidity。取自：https://medium.com/taipei-ethereum-meetup/%E6%95%99%E7%B7%B4-%E6%88%91-%E5%8F%AA-%E6%83%B3%E5%AD%B8solidity-92b7ba8054f5

(五) 收到我的 ENS 網域啦 gasolin.eth。取自：https://blog.gasolin.idv.tw/2017/08/13/got-my-ens-domain/

(六) 如何撰寫智能合約 SmartContract I。取自：https://blog.gasolin.idv.tw/2017/09/06/howto-write-a-smart-contract/

(七) 如何撰寫智能合約 SmartContract II 建立加密代幣。取自：https://blog.gasolin.idv.tw/2017/09/11/howto-write-a-simple-token/

(八) 如何撰寫智能合約 SmartContract III 建立標準代幣。取自：https://blog.gasolin.idv.tw/2017/09/16/howto-write-an-erc20-compatible-token/

(九) 如何撰寫智能合約 SmartContract IV 加入單元測試。取自：https://blog.gasolin.idv.tw/2018/01/02/howto-write-a-contract-test/

(十) 區塊鏈開發（一）搭建基於以太坊的私有鏈環境。取自：http://blog.csdn.net/sportshark/article/details/51855007

(十一) 區塊鏈開發（二）部署和運行第一個以太坊智能合約。取自：http://blog.csdn.net/sportshark/article/details/52249607

(十二) 區塊鏈開發（三）編寫調試第一個以太坊智能合約。取自：http://blog.csdn.net/fidelhl/article/details/52524434

(十三) Solisity 文檔。取自：http://solidity.readthedocs.io/en/latest/

(十七) 台北乙太坊社群專欄。取自：https://medium.com/taipei-ethereum-meetup

(十八) Solidity 撰寫智能合約與注意事項(一)。取自：https://medium.com/taipei-ethereum-meetup/solidity%E6%92%B0%E5%AF%AB%E6%99%BA%E8%83%BD%E5%90%88%E7%B4%84%E8%88%87%E6%B3%A8%E6%84%8F%E4%BA%8B%E9%A0%85-%E4%B8%80-6c9eacc00168

(十九) Solidity 撰寫智能合約與注意事項(二)。取自：https://medium.com/taipei-ethereum-meetup/solidity%E6%92%B0%E5%AF%AB%E6%99%BA%E8%83%BD%E5%90%88%E7%B4%84%E8%88%87%E6%B3%A8%E6%84%8F%E4%BA%8B%E9%A0%85-%E4%BA%8C-dd915bdeafa0

(二十) 合約與以太接收。取自：https://ethereum.stackexchange.com/questions/27052/how-to-create-a-smart-contract-to-send-an-eth-tx-on-time