專案:用 sqlite 實作區塊鏈並利用 flask 操作用戶端

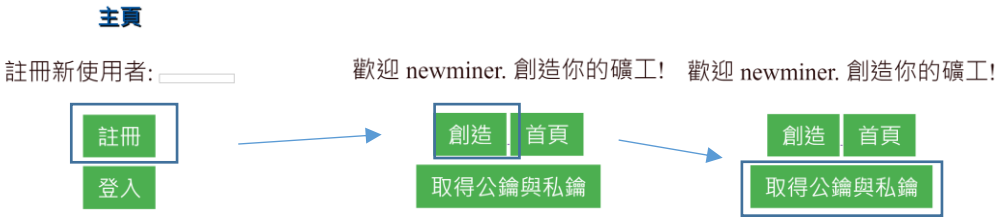
資工三 A 吳尚明

(1) 題目發想:

- ▶ 區塊鏈的概念前幾年就開始紅起來，慢慢地到現在，包括虛擬幣交易,智能合約等等…的用途也越來越多，故有興趣去了解所謂區塊鏈是如何實現的，當中就出現了一個問題，這些資料儲存在哪裡?，和一般的資料庫有何差別?
- ▶ 我想嘗試用在課堂上學習到的 flask 與 sqlite 去實作出一個區塊鏈，有關區塊鏈的技術我也是透過這次作業才開始學，所以對我來說是個既有挑戰又有趣的專案。

(2) 功能說明:

1.註冊新礦工: (可註冊多個礦工到相同區塊鏈裡) 新礦工註冊後，他會收到與其他礦工同步的區塊鏈，並與其他礦工建立通訊

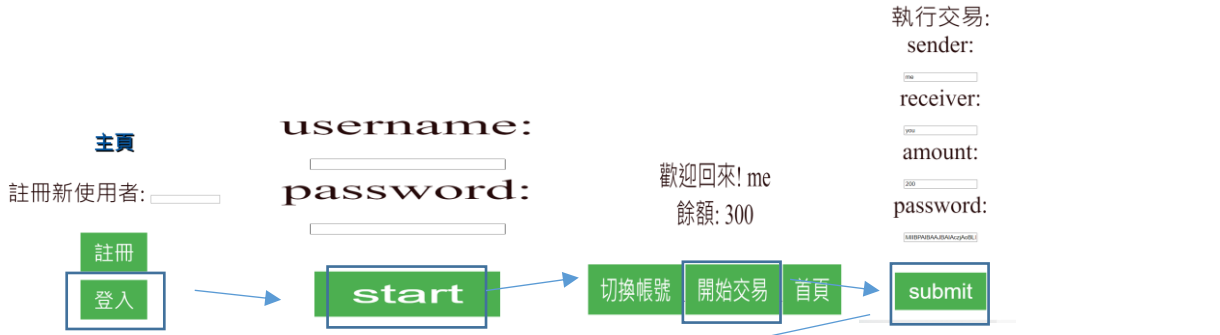


輸入想要的暱稱後，按下註冊。

按下創造，此時系統會創造一個礦工在區塊鏈裡不停運作。

使用者按下按鈕，取得礦工的公鑰與私鑰。

2.登入礦工，執行交易:



輸入用戶名，以及系統產生的私鑰

若錢足夠，私鑰正確，則交易成功

需等待交易被傳達到區塊鏈裡，以及區塊鏈新增新的區塊，交易才會生效

3.用戶的資料以及歷史交易紀錄都會被存在 sqlite 裡，以供 client 和 server 使用

	username	address	password	balance	number
	過濾	過濾	過濾	過濾	過濾
1	alex	MEgCQQCZ5/…	MIIBPQIBAAJBAJnn+FCfJ/…	600	1111
2	ming	MEgCQQCuJemLTw5tU/UOEWtKuGj/…	MIIBOwIBAAJBAK4l6YtPDm1T9Q4Ra…	620	1112
3	123	MEgCQQDEf8iAufEbIr4JJfGka9261yW…	MIIBPAIBAAJBAMR/…	0	1113
4	123	MEgCQQDEf8iAufEbIr4JJfGka9261yW…	MIIBPAIBAAJBAMR/…	0	1113

	sender	receiver	amounts	fee	count
	過濾	過濾	過濾	…	過濾
1	alex	ming	800	0	0

4.在程式內部執行的功能:

Client.py: (1.創立礦工，2.礦工各自的區塊鏈之間建立連線，3.傳入合法交易紀錄)

```
if request.method == 'POST':
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
        cur1 = con.cursor()
        cur1.execute("Insert into userdata Values(?,?,?,?,?)",[request.form.get('username'),"n","n",0,0])
    con.commit()
    return redirect(url_for('usermode',username=request.form.get('username')))

return render_template('home.html')
```

此時只是先存入此使用者名稱，並無產生新的礦工

```
def register():
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=0.5) as con:
            cur1 = con.cursor()
            cur1.execute("SELECT MAX(cast(number as int)) from userdata")
            a=cur1.fetchone()
    con.commit()
    if (a[0]==0):
        os.system('python blockchain_server.py '+"1111")
    else:
        b=a[0]+1
        strb=str(b)    #  利用request取得表單欄位值
        os.system('python blockchain_server.py '+strb+" 127.0.0.1:1111")
    return render_template('register.html')
```

產生區塊鏈的第一個礦工

產生區塊鏈的其他礦工，並指定 port:1111 通訊。

```
@app.route('/packet')
def packet():
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=0.5) as con:
            cur1 = con.cursor()
            cur1.execute("SELECT MAX(cast(number as int)) from userdata")
            port=cur1.fetchone()
            cur2 = con.cursor()
            cur2.execute("SELECT username, address, password, balance from userdata where number=?",[port[0]])
            data=cur2.fetchall()
    con.commit()
    return render_template('register.html',username=data[0][0],address=data[0][1],password=data[0][2],balance=data[0][3])
```

資料庫裡的資料是在 server 端存入的，這邊只是依照最新創造的 port number，讓使用者取得他所新建的礦工資料。

```
def tran():
    if request.method == 'POST':
        with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
            cur1 = con.cursor()
            cur1.execute("SELECT password, balance from userdata where username =?",[request.form.get('sender')])
            send=cur1.fetchone()
        con.commit()
        if send[0]==request.form.get('password') and send[1]>=int(request.form.get('amount')):
            with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
                cur1 = con.cursor()
                cur1.execute("Insert into usertransaction Values(?,?,?,?,?)",
                [request.form.get('sender'),request.form.get('receiver'),int(request.form.get('amount')),0,0])
            con.commit()
            amou=send[1]-int(request.form.get('amount'))
            return render_template('receipt.html',sender=request.form.get('sender'),receiver=request.form.get('receiver'),amount=request.form.get('amount'),balance=amou)
        elif send[0]!=request.form.get('password'):
            result = "密碼錯誤!"
            return render_template('transaction.html',交易結果=result)
        elif send[1]<request.form.get('amount'):
            result = "餘額不足!"
            return render_template('transaction.html',交易結果=result)
```

交易失敗

核對密碼是否正確，以及餘額是否充足，若都滿足，把交易紀錄存進 sqlite 裡。

Server.py:(1.通過加密方法取得公鑰私鑰,2.礦工不停的挖礦,3.透過區塊鏈紀錄推算礦工餘額,4.廣播

```
def start(self):
    address, private = self.generate_address()

    print(f"Miner address: {address}")
    print(f"Miner private: {private}")
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
        cur = con.cursor()
        cur.execute("update userdata SET address = ? , password=?, number=? where address = ?",[address,private,self.socket_port,"n"])
    con.commit()
```

```
def generate_address(self):
    public, private = rsa.newkeys(512)
    public_key = public.save_pkcs1()
    private_key = private.save_pkcs1()
    return self.get_address_from_public(public_key), \
        self.extract_from_private(private_key)
```

在 server 裡隨機產生地址以及私鑰。

傳入 sqlite 以供 client 和 server 使用。

礦工會不斷地挖掘新區塊，並將區塊與自身 address 連結，之後在翻閱區塊鏈時就能知道該礦工挖了多少區塊

Server 可以取用 client 端存入的歷史交易紀錄來更新使用者的餘額

Get_balance 的功能是查閱區塊鏈裡的紀錄，推算出礦工有多少餘額。

礦工之間可以互相廣播

```
while(True):
    self.mine_block(address)
    self.adjust_difficulty()
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
        cur = con.cursor()
        cur.execute("update userdata SET balance=? where address = ?",[self.get_balance(address),address])
    con.commit()
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
        cur = con.cursor()
        cur.execute("SELECT * from usertransaction where count<?",[1])
        rows=cur.fetchall()
    con.commit()
    final=int(self.get_balance(address))
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
        cur = con.cursor()
        for row in rows:
            cur.execute("SELECT address from userdata where username=?",[row[0]])
            s=cur.fetchone()
            cur.execute("SELECT address from userdata where username=?",[row[1]])
            r=cur.fetchone()
            if address==s[0]:
                final-=row[2]
            elif address==r[0]:
                final+=row[2]
    con.commit()
    with sqlite3.connect(r'C:\Users\User\Desktop\blockchain_data.db',timeout=3) as con:
        cur = con.cursor()
        #for row in rows:
            #if address==s[0]:
        cur.execute("update userdata SET balance=? where address = ?",[final,address])
            #elif address==r[0]:
        #cur.execute("update userdata SET balance=? where address = ?",[self.get_balance(address)+row[2],ad
    con.commit()
```

```
elif parsed_message["request"] == "clone_blockchain":
    print(f"[*] Receive blockchain clone request by {address}...")
elif parsed_message["request"] == "broadcast_block":
    print(f"[*] Receive block broadcast by {address}...")
elif parsed_message["request"] == "broadcast_transaction":
    print(f"[*] Receive transaction broadcast by {address}...")
elif parsed_message["request"] == "add_node":
    print(f"[*] Receive add_node broadcast by {address}...")
```

(3) 實作技術: 1.socket 2.thread 3.rsa 加密法 4.哈希函數 5. 完整區塊鏈網路(節點可互相溝通且更新彼此的區塊鏈) 6. Sqlite 資料儲存傳遞 7. Flash 網頁 8. Css 修飾網頁

學習過程有花時間實作但最後運用失敗的技術: 1. 非對稱式加密 2.產生簽章用於交易 3. 客戶端與區塊鏈的直接溝通

(4) 自我評估:

最初學習的時候是先從網路上的教程去實作區塊鏈，剛開始對區塊鏈的概念完全不了解，也不懂資料是甚麼方式存在著的，更不懂他怎麼實現他的特別之處(去中心化, 不可竄改, 同步廣播)，所以光是了解區塊鏈並且思考怎麼套用課堂所學就花費了我很多心神，更別提網路上的教程運用到的每個技術都是我從未實做過的，逐字讀懂每個 function 以及系統是如何運作的，更是讓我心力憔悴的同時，開始懷疑實作的可行性，但我很慶幸我沒有半途而廢，才最終做出了這個我有信心和別人不同的專案。

在到了實作專案這個階段時，我發現當我 client 端通訊到 flask 頁面時，client 就沒辦法和礦工運行的 server 通訊了，屢試屢挫，花了大半時間學習的技術最終無法實現，讓我很受挫，甚至白白坐在 K 中一整天 12 個小時沒進展，結果隔天才思考出其他解決方案。

雖說過程艱難，但這次的結果對我來說是很豐盛的，我憑藉著---如果怕失敗就沒去做，那這一生其他困難我也會放棄---這樣的心理建設，硬是把這些我完全不懂的技術學起來，了解其中原理，並實作出來，雖然後來設計出的畫面沒有時間再做美化，但我這次的實作是研究到很深度的東西(至少我自己是很陌生的)，這樣的成就感與知道自己也能做成功的自信，雖然有點晚，但在這段時間我終於體會到了，我認為我付出的努力相當足夠。