CW2

PROGRAMMING AND ALGORITHMS 2

Toqa Mahmoud

CU1900305

TABLE OF CONTENTS

Toqa Mahmoud CU1900305 03/01/2022

ABSTRACT

A blockchain is a growing list of records, called blocks, that are linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. The timestamp proves that the transaction data existed when the block was published in order to get into its hash. As blocks each contain information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks. Blockchains are typically managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks.

INTRODUCTION

In this coursework, I built my own cryptocurrency which is known as Arcave. I developed an Arcave transfer application using object oriented programming (OOP), in python. The program was precisely designed by optimal use of data types and data structures, proper use of control statements, efficient modular design using classes and methods, and coherent flow sequence. This report aims to provide detailed analysis of this program.

EXPLANATION OF THE CODE

A blockchain is essentially a series of blocks linked together. A block, on the other hand, is a data structure that stores information, in our case it is transactions. Each block contains its own hash, as well as the preceding block's hash. This is how the blockchain will be created and stored. So, in order to create a blockchain, I created a class for each of these: blockchain, chain, and transaction.

```
class Block():…

class Blockchain():…

class Transactions():…
```

Screenshot 1 Classes

BLOCK

In the block class, necessary information (block number, previous hash, transaction, nonce, timestamp) and a hashing mechanism was added. The hash is calculated using the SHA256 cryptographic hash technique, which is the same encoder used by Bitcoin. Because the hash is determined by all of the data in each block or transaction, if anything is tampered with or modified, such as the amount in a transaction, the hash will change as well, alerting us to the fact that something is wrong.

```
class Block():

    def __init__(self, number=0, prev_hash="0" * 64, transaction=None, nonce=0,
    timestamp=time.strftime('%Y-%m-%d %H:%M:%S')):…

    # returns the hashed block
    def hash(self):…

    # convert object to a string to make it readable
    def __str__(self):…
```

Screenshot 2 Block Class

BLOCKCHAIN

In the blockchain class, I created a function to add blocks to the Blockchain, keep in mind that if someone modifies data in the blockchain, the hash of the tampered block will change as well, causing the chain to break. As a result, if someone alters the value of a transaction to give them more money, the chain will be broken. However, if this individual recalculates the hashes of every block in the chain, they will be able to make it a genuine blockchain once more, one that includes their modified data. With a machine that can recalculate the hash values of the current and prior blocks in seconds, this is extremely plausible.

To prevent this from happening, blockchain technology employs a proof-of-work mechanism. Mining is the process of solving this proof-of-work in order to create a new block for our crypocurrency. I built a proof-of-work method for Arcave that solves for a variable nonce which makes the block hash begin with '000'.

The most significant feature of cryptocurrencies and blockchain technology is decentralisation. This means that it is controlled via a peer-to-peer network rather than a central point of control. When a new block has to be added to the blockchain, each participant double-checks that it's alright to do so before adding it to their own version. We have consensus, or majority agreement, if everything checks out and more than half of the network agrees on this new version of the blockchain. When a new version of the blockchain has gained widespread acceptance, it becomes the accepted blockchain. If someone modifies data on the blockchain, their version will differ from what the majority of the network accepts, and they will be unable to get the alteration onto the actual, accepted blockchain. To do this, we'll create a node for each user of our platform.

```python
class Blockchain():

    # first 3 digits of the hash must be 0
    difficulty = 3

    def __init__(self):...

    # peer to peer function
    def register_node(self, address):...

    # adds blocks to the chain
    def add(self, block):...

    # mines blocks
    def mining(self, block):...

    # adds transactions to the pending transaction list
    def addTrans(self, sender, receiver, amt, keyString, senderKey):...

    # generate keys
    def generateKeys(self):...

    # validates that prev hash and hash are equal and difficulty is implemented

    def valid(self):...
```

Screenshot 3 Blockchain Class

TRANSACTIONS

In the transactions class, we will validate the validity of each transaction that has been made. We'll do this in the same manner that individuals sign cheques to make them verified in real life. Crypocurrencies do this by assigning a set of unique keys to each "wallet" or user which are the private key and the public key. Users will be able to sign their transactions with these keys to ensure that they are legitimate. We're going to utilise PyCryptoDome, a Python library, for the encryption method that we'll employ to produce these keys, which is known as RSA. (Rivest–Shamir–Adleman).

```
class Transactions():

    def __init__(self, sender="", receiver="", amount=int,
    timestamp=time.strftime('%Y-%m-%d %H:%M:%S')):…

    # validates transactions
    def validTrans(self):…

    # signs the transaction
    def signTrans(self, key, senderKey):…

    # returns the hashed transaction

    def hash(self):…
```

Screenshot 4 Transactions Class

DATA STORAGE

SQL is Structured Query Language is a domain-specific language used in programming and designed for handling data in a database management system. I stored the databases in to SQL server. The blockchain data was stored in the blockchain database and the users data was stored in the users database, as shown in the screenshot below.

```
mysql> select * from blockchain;
+--------+------------------------------------------------------------------+------------------------------------------------------------------+------------------------+-------+---------------------+
| number | hash                                                             | previous                                                         | transaction            | nonce | timestamp           |
+--------+------------------------------------------------------------------+------------------------------------------------------------------+------------------------+-------+---------------------+
| 1      | 0006f90b2c23e90031e95419b4d92ae73fb1d99f53b95b401a8017238f13d95f | 0000000000000000000000000000000000000000000000000000000000000000 | BANK-->user1-->100.0   | 12411 | 2021-12-30 18:25:45 |
| 2      | 000e9be4323bb0474b5d88d382f2f48bd8e344704708a6ca5921573509a04b68 | 0006f90b2c23e90031e95419b4d92ae73fb1d99f53b95b401a8017238f13d95f | BANK-->user3-->100.0   | 6589  | 2021-12-30 18:25:45 |
| 3      | 00006c0770529a76789f8ee47a5969c4f2608d6d0adad48e2a0c22891a46f6fd | 000e9be4323bb0474b5d88d382f2f48bd8e344704708a6ca5921573509a04b68 | user3-->user2-->50.0   | 3670  | 2021-12-30 18:25:45 |
| 4      | 000e5e6d0b64325591edb19ecb1f6811d6ab9b33d8a857b38d1d8e12b2edd33e | 00006c0770529a76789f8ee47a5969c4f2608d6d0adad48e2a0c22891a46f6fd | user2-->user1-->10.0   | 8312  | 2021-12-30 18:25:45 |
| 5      | 0006965df434af76eac1cc8d927fa1ad9283c9880c33f94c27b9c1dab2f22280 | 000e5e6d0b64325591edb19ecb1f6811d6ab9b33d8a857b38d1d8e12b2edd33e | user1-->user3-->11.25  | 9244  | 2021-12-30 18:25:45 |
| 6      | 00003532940944c74cccf06585ffb0539dcbafb3d0ed1b741c6d517dffb69050 | 0006965df434af76eac1cc8d927fa1ad9283c9880c33f94c27b9c1dab2f22280 | BANK-->user1-->100.0   | 13936 | 2021-12-30 18:25:45 |
| 7      | 000931b9660e9927fafb1e7df86dad99e0a39511064aec9c5d0e86834eae19e5 | 00003532940944c74cccf06585ffb0539dcbafb3d0ed1b741c6d517dffb69050 | user1-->user3-->50.0   | 908   | 2021-12-30 20:35:42 |
| 8      | 000bf9767275aa44c90d0e03c314af4d027170ab725b1a98a1939fee5d2f9a56 | 000931b9660e9927fafb1e7df86dad99e0a39511064aec9c5d0e86834eae19e5 | user1-->user2-->8.75   | 7405  | 2022-01-02 22:15:10 |
| 9      | 00063d16e8aa87aeb59b109255cd400a28fa66dff78dc85a884cf93d21f70932 | 000bf9767275aa44c90d0e03c314af4d027170ab725b1a98a1939fee5d2f9a56 | BANK-->user4-->90.0    | 4308  | 2022-01-03 01:51:37 |
+--------+------------------------------------------------------------------+------------------------------------------------------------------+------------------------+-------+---------------------+
9 rows in set (0.00 sec)

mysql> select * from users;
+---------+-----------------+----------+-----------------------------------------------------------------------------+
| name    | email           | username | password                                                                    |
+---------+-----------------+----------+-----------------------------------------------------------------------------+
| toqaaa  | toqaaa          | toqaaa   | $5$rounds=535000$UxjRtnJbFeHH9SwI$zOUuFlLbGcYt7Nfnh65cWhX9bpC1iPVEuO97eDUqAL2 |
| testing | testing         | testing  | $5$rounds=535000$TVvGALoxAxDSmVf9$eHfTGxHDs0r3P3GlCpvzFdoNHEOsmOR.sJQzTtWerxD |
| ahmed   | ahmedd          | ahmed    | $5$rounds=535000$Cp44HMlPSJNvI38Z$z/pS8dohPyDs6aw0UnMu2eMkAQ8QzLLcvPhoFPcIY14 |
| user1   | user1@gmail.com | user1    | $5$rounds=535000$oqYY1cMKjdChjmoG$m7mNLw5smzzS0K2vjysQVhyfYfMhWUxQdaAMNTkq52A |
| user2   | user2@gmail.com | user2    | $5$rounds=535000$qZdDw0IhuCVSR0rk$q8tfDk3hpp/wMAaApqJVF0JEk.31GplGC1CW19oqWwC |
| user3   | user3@gmail.com | user3    | $5$rounds=535000$AvzuQtIW1aCqd/Ni$STHlVQ9QxW/zFOzDdCHnP0/vBCFRfhykHRB8Nrw3Hr7 |
| user4   | user4@gmail.com | user4    | $5$rounds=535000$69Xe6eDcRXFmdavk$EFrRGZTBwt4YiLfRNiCz9HXSo5zUCz90Kd2T99u36t/ |
+---------+-----------------+----------+-----------------------------------------------------------------------------+
7 rows in set (0.00 sec)
```

Screenshot 5 Data Storage

THREADING IN FLASK

Flask server is multi-threaded by default as of Flask 1.0. A new thread is created for each new request.

INSTRUCTIONS FOR USING THE CODE

REPISITORY'S LINK

https://github.com/Toqahassib/CW2_Blockchain.git

REPOSITORY'S ORGANIZATION

1. README.md
   - Contains information for the user about the code.
2. app.py
   - this is the main front-end file, which contains all data that will be represented to the user
3. blockchain.py
   - This is the main Blockchain file, which contains all data about the blockchain
4. sql.py
   - this is the main back-end file, which contains all data related to the database and how the user-interface functions
5. templates & static
   - these are the styles folder, which contains all data related to the styling of the user-interface.

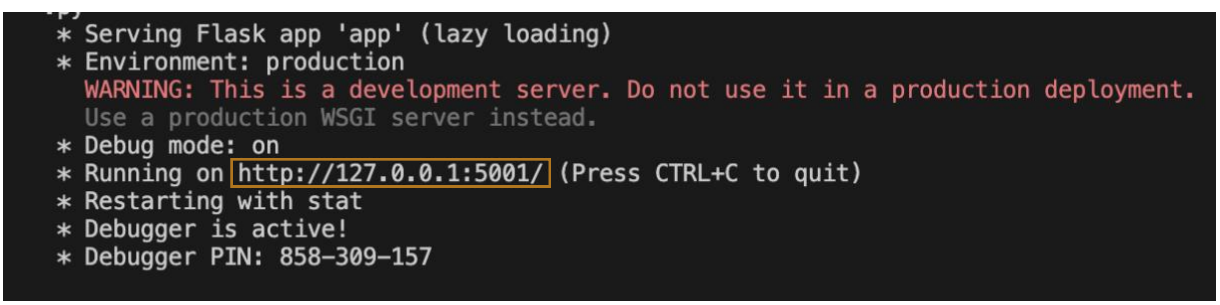
PRE-REQUISITES

Programming Language: Python 3.9.7

Modules: flask, hashlib, Crypto, time, urllib, wtforms, passlib, flask_mysqldb, functools.

HOW TO USE THE CODE

To start the program, open your terminal and change the current directory to where you want to clone the repository. Then clone it using the repository link and finally run the "app.py" file. After running the file, go to the following URL in your browser.



Screenshot 6 running app.py

DISCUSSION OF THE CODE

At the beginning of implementing the code, I stored the block and transaction data in a list of dictionaries, which use *keys* to associate with each value, this means that the same keys will be stored for all blocks in the blockchain. Taking all this space for the same keys is extremely inefficient, so I decided to store it in a list of data objects in the blockchain class. Moreover, The nodes of the users where stored in a set because they're immutable.

CODING STYLE

The code is designed in a systemized way by conducting OOP. Working with OOP did not only give me the flexibility to clearly structure a clean code, but also eased its modification and maintenance. The Blockchain is made up of 3 main parts – the blockchain, the block, and the transaction. Therefore, my code is divided into these 3 main parts as classes. The following illustrates the 3 classes and objects used in the code:

- ❖ The Blockchain class contains 3 objects – 1 to store the blocks, 1 to store the pending transactions, and 1 to store the nodes.
- ❖ The Block class contains 5 objects – 1 to store the number of the block, 1 to store the previous block's hash, 1 to store the transaction, 1 to store the nonce, and 1 to store the timestamp.
- ❖ The Transaction class contains 5 objects – 1 to store the sender of the transaction, 1 to store the receiver of the transaction, 1 to store the amount sent, 1 to store the timestamp, and 1 to store the hash of the transaction.

I've styled my code in that particular way for various reasons including:

- **ORGANIZATION**: data and procedure were stored at varying levels – storing blockchain data in the blockchain class, storing block data in the block class, and storing transaction data in the transaction class. Later, this will help me and others understand, edit, and use my code.

- **MAINTAINABILITY**: this approach made my code more maintainable. Identifying the source of errors was easier because each object is self-contained in its respective class.

SECURE PROGRAMMING PRINCIPLES

Writing secure code is an essential part of secure software development. The following points where implemented, taking in consideration the scale of the software.

- The system default is to deny access, not to grant it.
  - All users cannot access the web pages (except the index), if they are not registered.
- All access is controlled.
  - All users cannot access the web pages (except the index), if they are not registered.
  - Once the user logs out, the session gets cleared.
- Things are kept modular.
  - A wide range of modules have been used, check Pre-requisites section.
- Input Validation
  - In all forms, maximum and minimum input validations were implemented.
- The security mechanisms do not disrupt the users' work of flow.

CLASS DIAGRAM

Block Class
+ number: int + prev_hash: str + transaction: None + nonce: int + timestamp: str
- __init__(self, number=int, prev_hash=int, transaction=None, nonce=int, timestamp=str) - hash() - __str__()

Blockchain Class
+ chian: list + pendingTrans: list + nodes: set
- __init__(self) - register_node(address:str) - add() - remove() - mining() - addTrans(sender:str, receiver:str, amt:int, keyString:str, senderKey:str) - generateKeys() - valid()

Transactions Class
+ sender: str + receiver: str + amount: int + timestamp: str
- __init__(self, sender=str, receiver=str, amount=int, timestamp=str) - validTrans() - signTrans(key:int, senderKey:int) - hash()

Figure 1 Class Diagram

FUNCTIONS

Table 1 Functions

Class	Name	Arguments	Returns	Explanation
	new_hash()	*args	hash.hexdigest()	Hashes using sha256
Block()	hash()	none	new_hash()	Hashes the block
	__str__()	none	str()	Converts objects to a string
Blockchain()	register_node()	address	None	Registers addresses as nodes to implement peer to peer
	add()	Block	None	Adds blocks to the blockchain
	mining()	block	none	Mines the blocks
	addTrans()	sender, receiver, amt, keyString, senderKey	len(self.chain) + 1	Adds transactions to the block
	generateKeys()	none	key.publickey().export_key().decode('ASCII')	Generates private and publics keys
	valid()	none	True	validates that the previous block's hash is equal to the previous block's hash and ensures that difficulty is met
Transactions()	validTrans()	none	True	Validates that transactions have been signed
	signTrans()	key, senderKey	True	Signs the transactions
	hash()	none	new_hash()	Hashes the transaction

WEBAPP FUNCTIONALITIES

USER FUNCTIONALITIES

- The user can create an account by registering and inputting their data such as name, username, and password.
- After creating an account, the user can login the website.
- After logging in, the user can buy Arcave and mine transactions.
- The user can logout.

RESTRICTIONS

- Without logging in first, the user cannot access any website page other than the index.

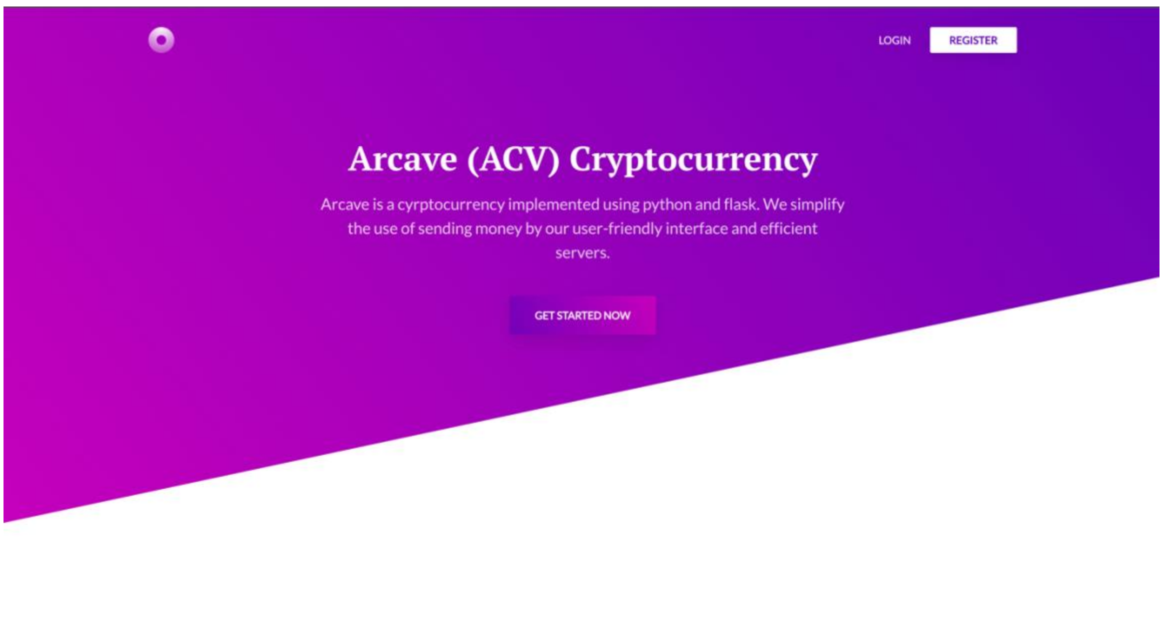
USER INTERFACE

The web application was precisely designed to be as user friendly as possible for the users and to develop or improve the utility effectiveness and usability of the system. As a general rule of thumb, there are '4 E's' of good UI design that I made sure to accomplish:

1. Easy to use
2. Easy to understand
3. Error-free
4. Effective for end-goal

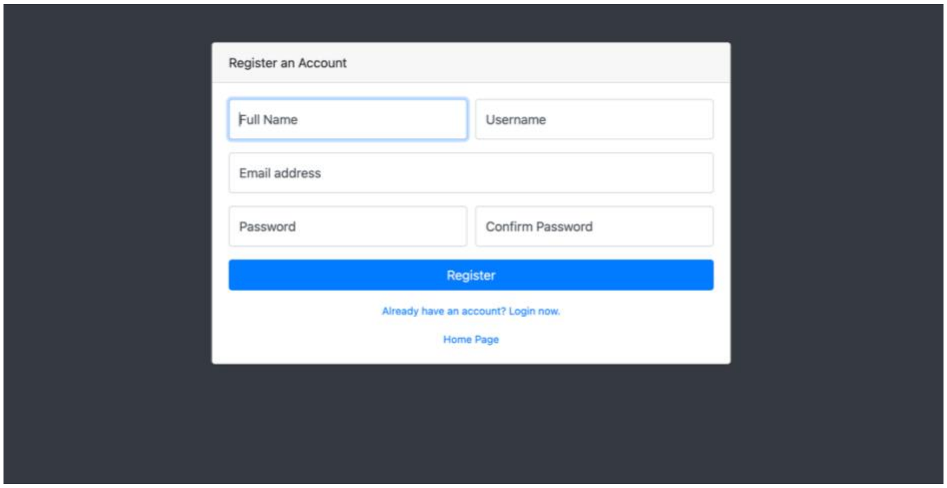
In the following screenshot, the website will be clearly explained.

Index page: calls the index page, which is shown in the screenshot below. In this section, the get started button will direct the user to the registration page. Moreover, The header includes the Login and register button.



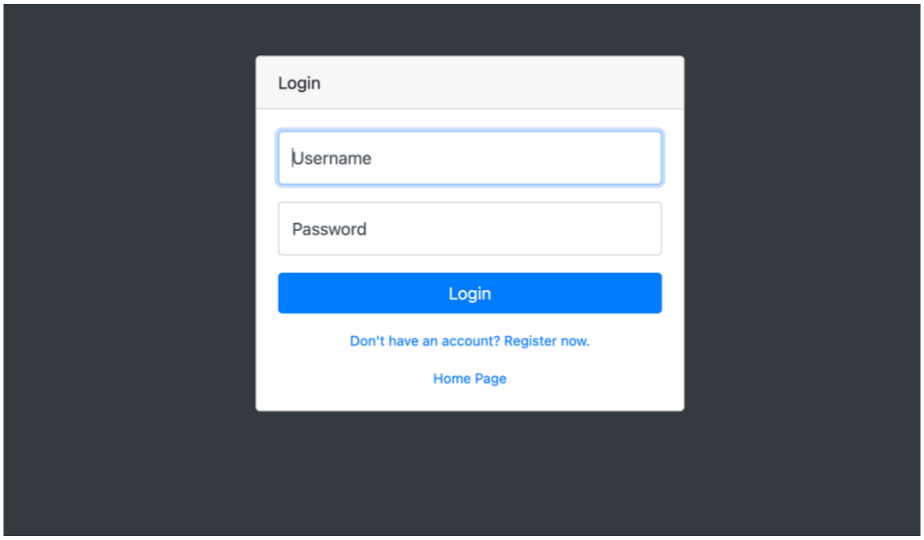
Screenshot 7 Index Page

Register page: when a client will click on the get started button, the register page will come up. If the user is new, they will have to fill the form in order to continue with the procedure. If the user is already registered they can click on the Already have an account link. The user should fill all the fields and make sure the confirmation password matches the password in order to complete the registration procedure. If the procedure is not carried out in the right way, the registration will not be completed. After that, by clicking the register button, your account will be registered and only then will you be able to login to the system. After the procedure is done, the user is logged in.



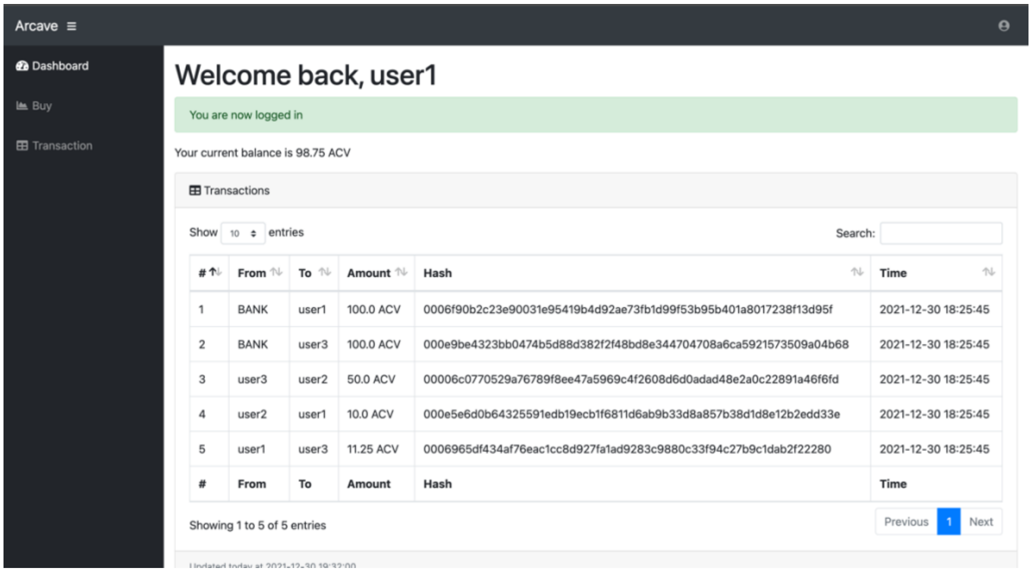
Screenshot 8 Register Page

Login page: In the login page, the user will simply enter the username and password they registered with. If the user is still not registered, they can click on the register button. After the completion of this step, the user will be allowed to access the rest of the website and a logout button in the header will be displayed for the user to log out anytime.



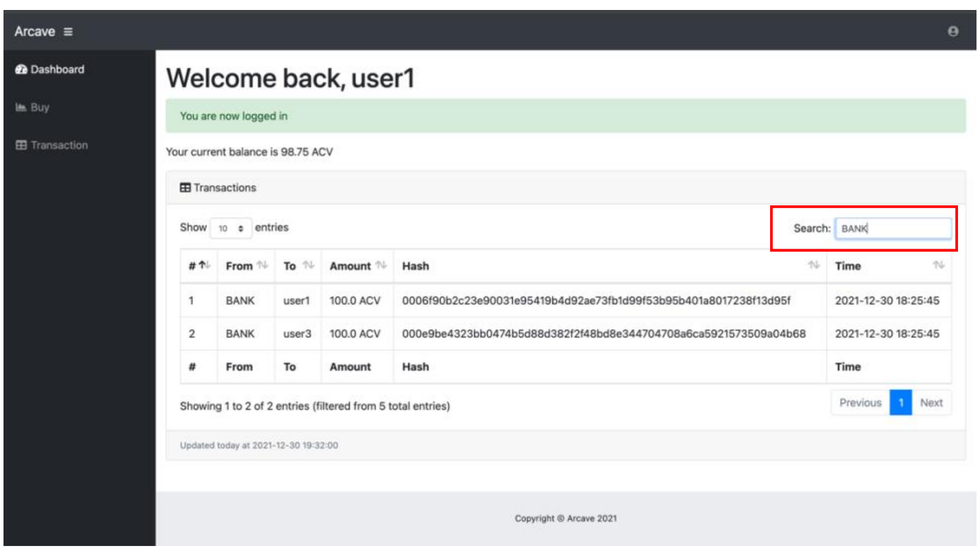
Screenshot 9 Login Page

Dashboard page: After the user logins successfully, they will be directed to the dashboard page, where all transactions are recorded and presented in a table.



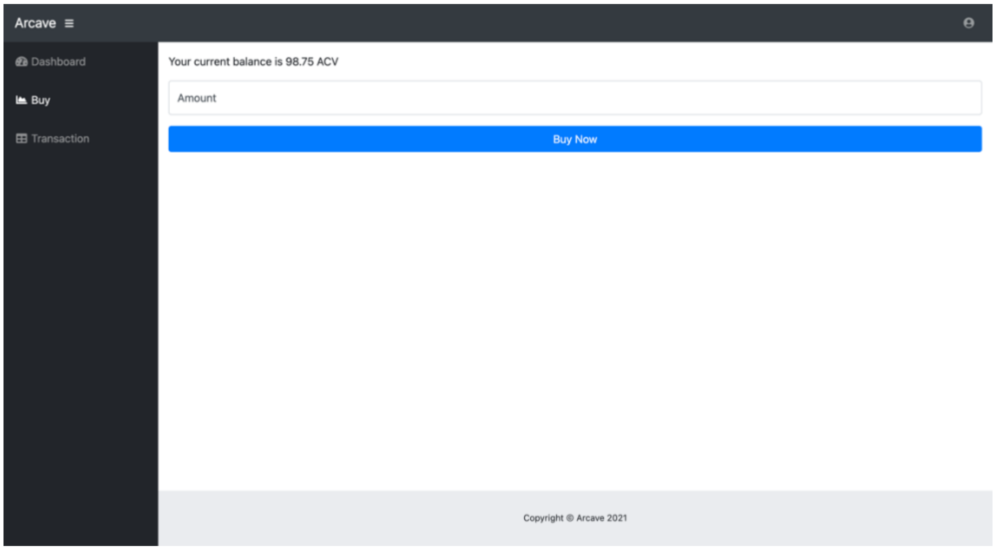
Screenshot 10 Dashboard Page

The search section: In this section the user will be able to search for transaction data. When the user searches for anything such as the name of the sender, the table will filter out what was searched for, as shown below.



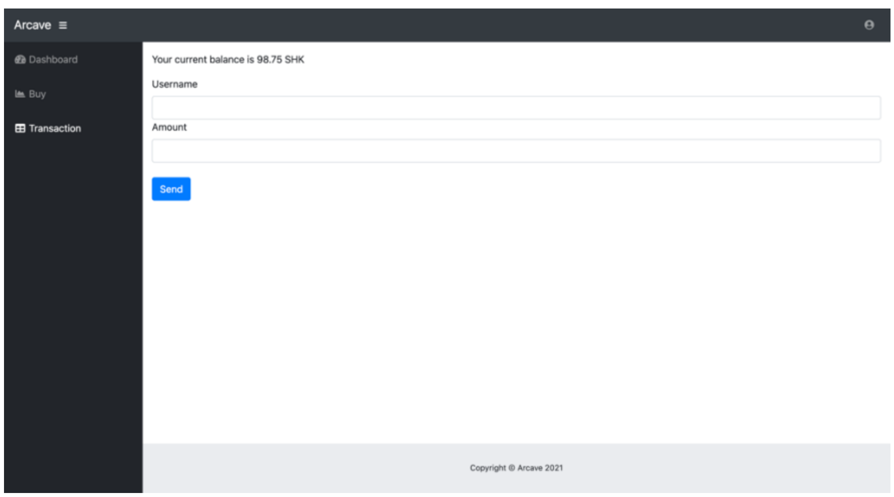
Screenshot 11 Search Section

Buy page: this is where the user can buy Arcave, it is as simple as entering the desired amount and clicking on the buy now button.



Screenshot 12 Buy Page

Transaction page: This is where the user can send any amount of Arcave, as long as it is equal or less to the user's balance.



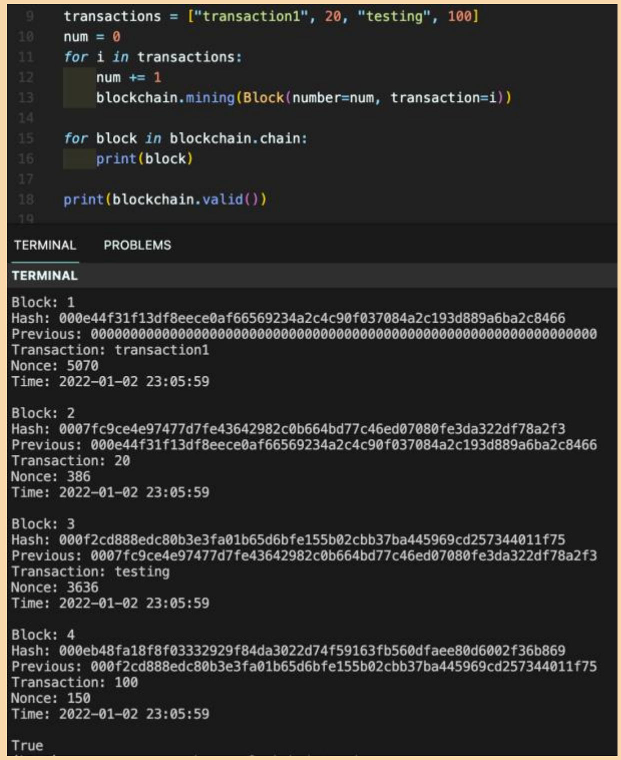
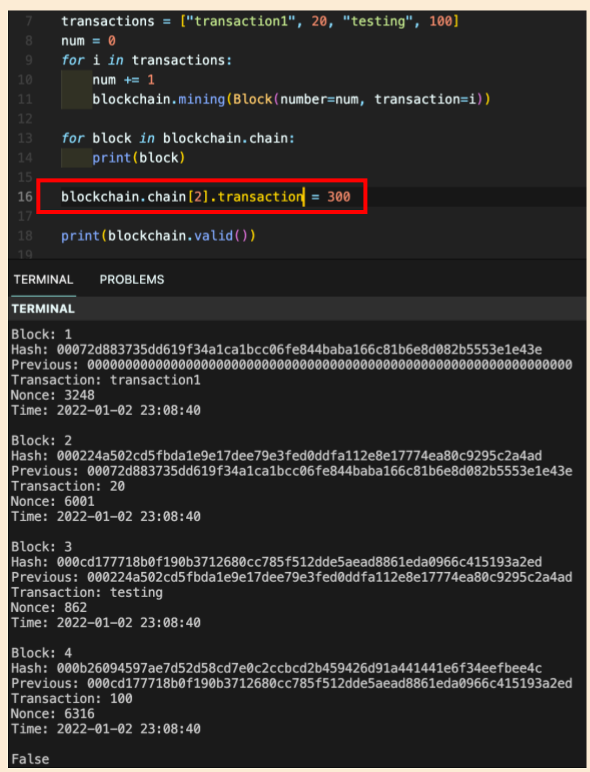
Screenshot 13 Transaction Page

To conclude, it is safe to say that all the followings were met regarding the user interface:

- Simple
- Use the user language
- clearly marked exit
- Good error messages
- Documentation

TESTING

Table 2 Functions Testing

Functions Testing	
Testing the mining and validation functions, the screenshot shows the blocks as well as true indicating that the blockchain has NOT been tampered.	
Testing the mining and validation functions, however in this scenario I edited the blockchain. The screenshot shows the blocks as well as false indicating that the blockchain has been tampered.	

```
transactions = ["transaction1", 20, "testing", 100]
num = 0
for i in transactions:
    num += 1
    blockchain.mining(Block(number=num, transaction=i))

for block in blockchain.chain:
    print(block)

print(blockchain.valid())
```

TERMINAL PROBLEMS

TERMINAL

```
Block: 1
Hash: 000e44f31f13df8eece0af66569234a2c4c90f037084a2c193d889a6ba2c8466
Previous: 0000000000000000000000000000000000000000000000000000000000000000
Transaction: transaction1
Nonce: 5070
Time: 2022-01-02 23:05:59

Block: 2
Hash: 0007fc9ce4e97477d7fe43642982c0b664bd77c46ed07080fe3da322df78a2f3
Previous: 000e44f31f13df8eece0af66569234a2c4c90f037084a2c193d889a6ba2c8466
Transaction: 20
Nonce: 386
Time: 2022-01-02 23:05:59

Block: 3
Hash: 000f2cd888edc80b3e3fa01b65d6bfe155b02cbb37ba445969cd257344011f75
Previous: 0007fc9ce4e97477d7fe43642982c0b664bd77c46ed07080fe3da322df78a2f3
Transaction: testing
Nonce: 3636
Time: 2022-01-02 23:05:59

Block: 4
Hash: 000eb48fa18f8f03332929f84da3022d74f59163fb560dfaee80d6002f36b869
Previous: 000f2cd888edc80b3e3fa01b65d6bfe155b02cbb37ba445969cd257344011f75
Transaction: 100
Nonce: 150
Time: 2022-01-02 23:05:59

True
```

```
transactions = ["transaction1", 20, "testing", 100]
num = 0
for i in transactions:
    num += 1
    blockchain.mining(Block(number=num, transaction=i))

for block in blockchain.chain:
    print(block)

blockchain.chain[2].transaction = 300

print(blockchain.valid())
```

TERMINAL PROBLEMS

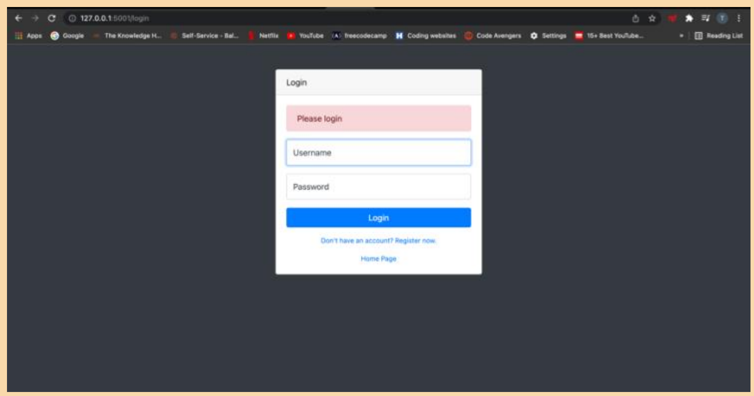
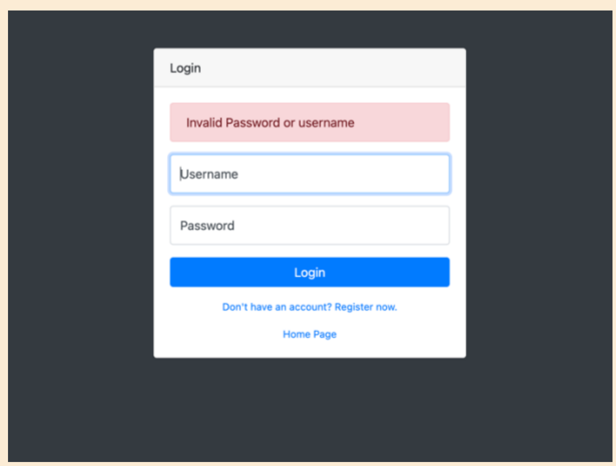
TERMINAL

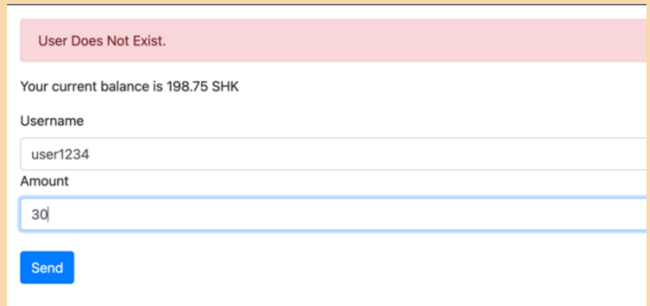
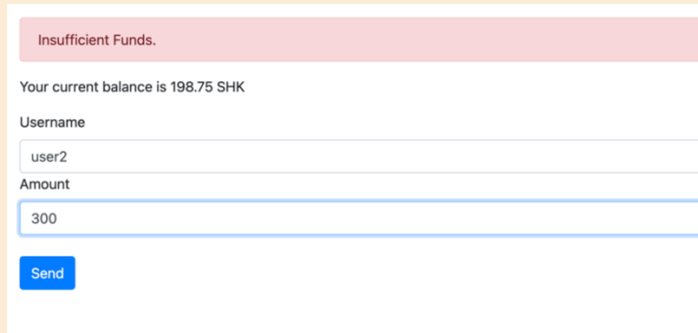
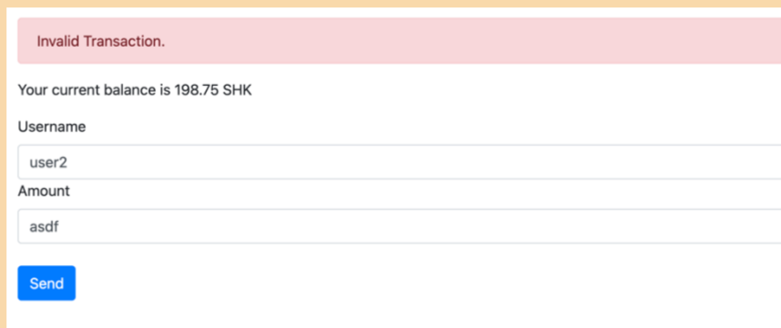
```
Block: 1
Hash: 00072d883735dd619f34a1ca1bcc06fe844baba166c81b6e8d082b5553e1e43e
Previous: 0000000000000000000000000000000000000000000000000000000000000000
Transaction: transaction1
Nonce: 3248
Time: 2022-01-02 23:08:40

Block: 2
Hash: 000224a502cd5fbda1e9e17dee79e3fed0ddfa112e8e17774ea80c9295c2a4ad
Previous: 00072d883735dd619f34a1ca1bcc06fe844baba166c81b6e8d082b5553e1e43e
Transaction: 20
Nonce: 6001
Time: 2022-01-02 23:08:40

Block: 3
Hash: 000cd177718b0f190b3712680cc785f512dde5aead8861eda0966c415193a2ed
Previous: 000224a502cd5fbda1e9e17dee79e3fed0ddfa112e8e17774ea80c9295c2a4ad
Transaction: testing
Nonce: 862
Time: 2022-01-02 23:08:40

Block: 4
Hash: 000b26094597ae7d52d58cd7e0c2ccbcd2b459426d91a441441e6f34eefbee4c
Previous: 000cd177718b0f190b3712680cc785f512dde5aead8861eda0966c415193a2ed
Transaction: 100
Nonce: 6316
Time: 2022-01-02 23:08:40

False
```

<table>
<tr><td>Testing the generateKeys and addTrans functions, the screenshot shows the generated key as well as signed successfully indicating that the transaction has been signed.</td><td>
</td></tr>
</table>

All inputs handle errors to avoid the program from crashing. The table below shows how these errors are handled.

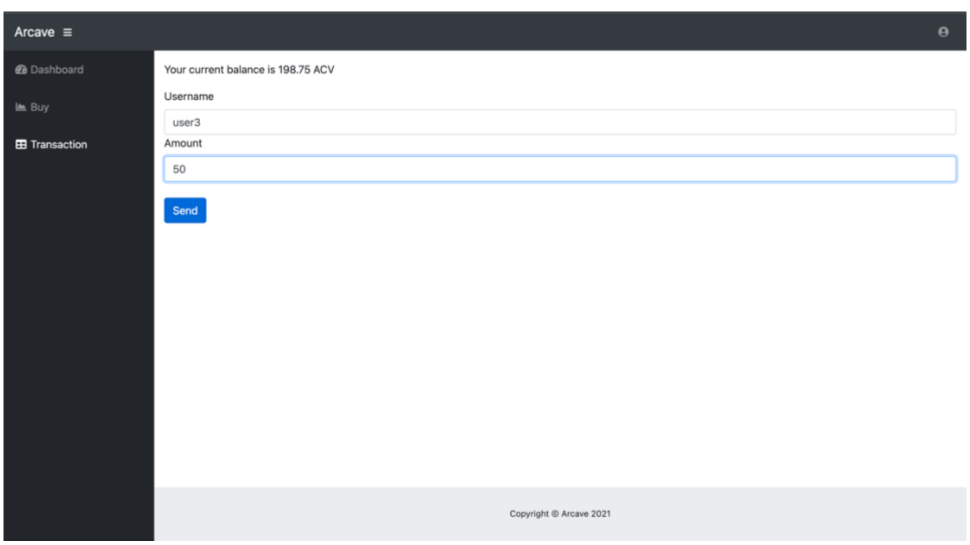
Table 3 Error Handling

<table>
<tr><th colspan="2">Testing Error Handling</th></tr>
<tr><td>The user cannot access any page other than the index, if they are not logged in</td><td>
</td></tr>
<tr><td>The user must register before logging in</td><td>
</td></tr>
</table>

<table>
<tr><td>Users must send transactions to registered users only.</td><td>
</td></tr>
<tr><td>Users must send transactions equal to or less than their balance</td><td>
</td></tr>
<tr><td>Amount sent should be an integer</td><td>
</td></tr>
</table>

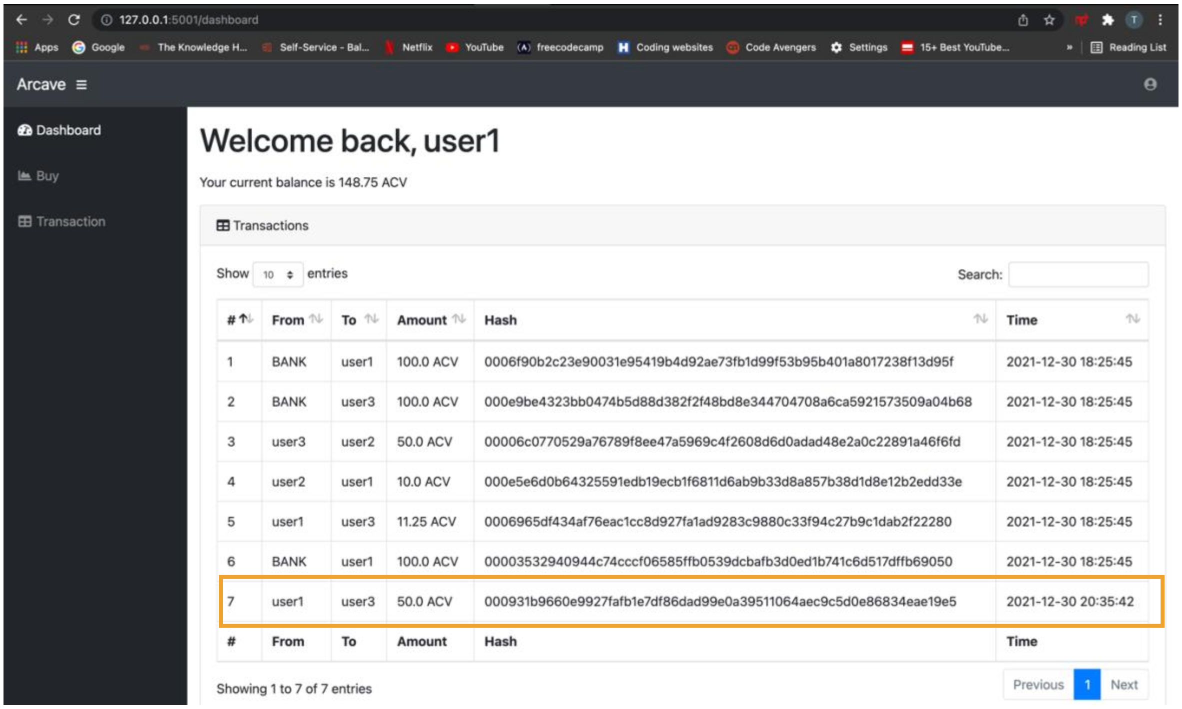
TEST CASE SCENARIO

Sending 50 ACV from user1’s account to user3 should show on all user’s side.



Screenshot 14 50 ACV Transaction

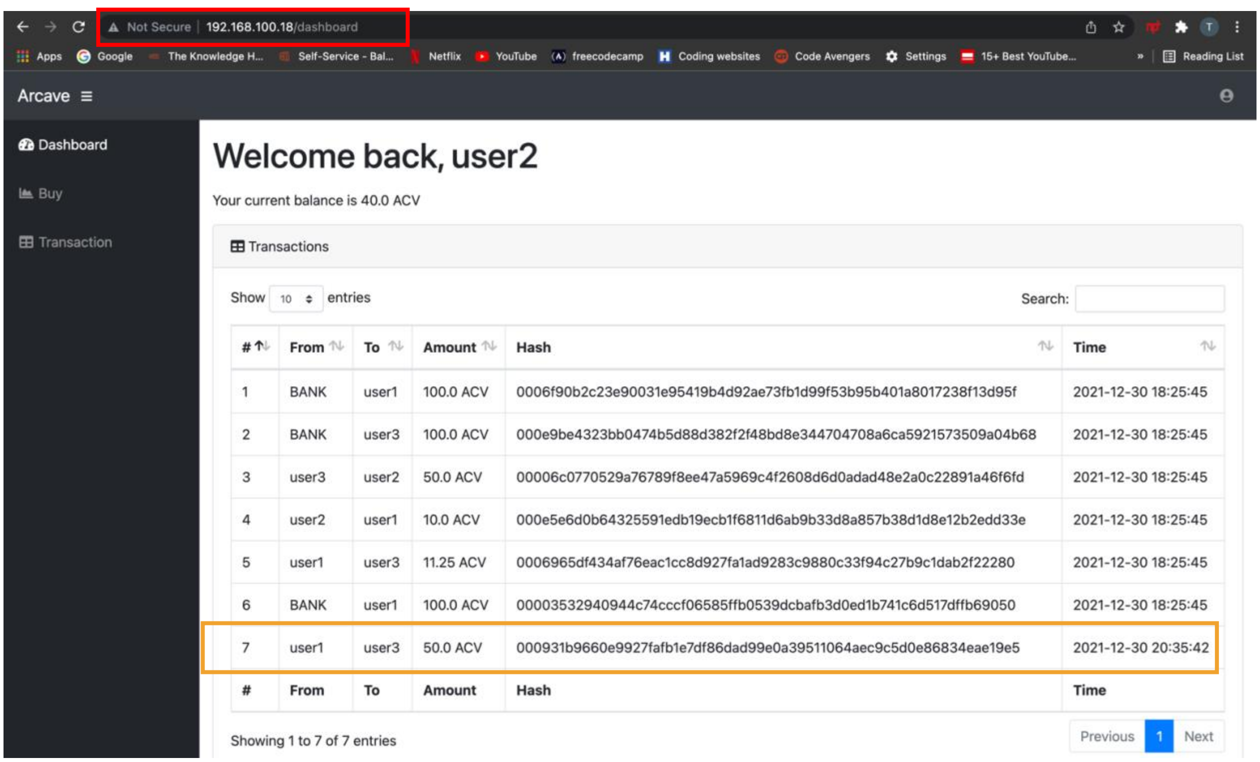
Below is user1’s dashboard, which successfully shows the transaction made.



#	From	To	Amount	Hash	Time
1	BANK	user1	100.0 ACV	0006f90b2c23e90031e95419b4d92ae73fb1d99f53b95b401a8017238f13d95f	2021-12-30 18:25:45
2	BANK	user3	100.0 ACV	000e9be4323bb0474b5d88d382f2f48bd8e344704708a6ca5921573509a04b68	2021-12-30 18:25:45
3	user3	user2	50.0 ACV	00006c0770529a76789f8ee47a5969c4f2608d6d0adad48e2a0c22891a46f6fd	2021-12-30 18:25:45
4	user2	user1	10.0 ACV	000e5e6d0b64325591edb19ecb1f6811d6ab9b33d8a857b38d1d8e12b2edd33e	2021-12-30 18:25:45
5	user1	user3	11.25 ACV	0006965df434af76eac1cc8d927fa1ad9283c9880c33f94c27b9c1dab2f22280	2021-12-30 18:25:45
6	BANK	user1	100.0 ACV	00003532940944c74cccf06585ffb0539dcbafb3d0ed1b741c6d517dffb69050	2021-12-30 18:25:45
7	user1	user3	50.0 ACV	000931b9660e9927fafb1e7df86dad99e0a39511064aec9c5d0e86834eae19e5	2021-12-30 20:35:42

Screenshot 15 User1's Dashboard

Below is user2’s dashboard, which successfully shows the transaction made, even though user2 was not a part of it. (notice that the host is a different IP)



#	From	To	Amount	Hash	Time
1	BANK	user1	100.0 ACV	0006f90b2c23e90031e95419b4d92ae73fb1d99f53b95b401a8017238f13d95f	2021-12-30 18:25:45
2	BANK	user3	100.0 ACV	000e9be4323bb0474b5d88d382f2f48bd8e344704708a6ca5921573509a04b68	2021-12-30 18:25:45
3	user3	user2	50.0 ACV	00006c0770529a76789f8ee47a5969c4f2608d6d0adad48e2a0c22891a46f6fd	2021-12-30 18:25:45
4	user2	user1	10.0 ACV	000e5e6d0b64325591edb19ecb1f6811d6ab9b33d8a857b38d1d8e12b2edd33e	2021-12-30 18:25:45
5	user1	user3	11.25 ACV	0006965df434af76eac1cc8d927fa1ad9283c9880c33f94c27b9c1dab2f22280	2021-12-30 18:25:45
6	BANK	user1	100.0 ACV	00003532940944c74cccf06585ffb0539dcbafb3d0ed1b741c6d517dffb69050	2021-12-30 18:25:45
7	user1	user3	50.0 ACV	000931b9660e9927fafb1e7df86dad99e0a39511064aec9c5d0e86834eae19e5	2021-12-30 20:35:42

Screenshot 16 User2's Dashboard

It was also recorded successfully in the blockchain database.

```
mysql> select * from blockchain;
+--------+------------------------------------------------------------------+------------------------------------------------------------------+-------------------------+-------+---------------------+
| number | hash                                                             | previous                                                         | transaction             | nonce | timestamp           |
+--------+------------------------------------------------------------------+------------------------------------------------------------------+-------------------------+-------+---------------------+
| 1      | 0006f90b2c23e90031e95419b4d92ae73fb1d99f53b95b401a8017238f13d95f | 0000000000000000000000000000000000000000000000000000000000000000 | BANK-->user1-->100.0    | 12411 | 2021-12-30 18:25:45 |
| 2      | 000e9be4323bb0474b5d88d382f2f48bd8e344704708a6ca5921573509a04b68 | 0006f90b2c23e90031e95419b4d92ae73fb1d99f53b95b401a8017238f13d95f | BANK-->user3-->100.0    | 6589  | 2021-12-30 18:25:45 |
| 3      | 00006c0770529a76789f8ee47a5969c4f2608d6d0adad48e2a0c22891a46f6fd | 000e9be4323bb0474b5d88d382f2f48bd8e344704708a6ca5921573509a04b68 | user3-->user2-->50.0    | 3670  | 2021-12-30 18:25:45 |
| 4      | 000e5e6d0b64325591edb19ecb1f6811d6ab9b33d8a857b38d1d8e12b2edd33e | 00006c0770529a76789f8ee47a5969c4f2608d6d0adad48e2a0c22891a46f6fd | user2-->user1-->10.0    | 8312  | 2021-12-30 18:25:45 |
| 5      | 0006965df434af76eac1cc8d927fa1ad9283c9880c33f94c27b9c1dab2f22280 | 000e5e6d0b64325591edb19ecb1f6811d6ab9b33d8a857b38d1d8e12b2edd33e | user1-->user3-->11.25   | 9244  | 2021-12-30 18:25:45 |
| 6      | 00003532940944c74cccf06585ffb0539dcbafb3d0ed1b741c6d517dffb69050 | 0006965df434af76eac1cc8d927fa1ad9283c9880c33f94c27b9c1dab2f22280 | BANK-->user1-->100.0    | 13936 | 2021-12-30 18:25:45 |
| 7      | 000931b9660e9927fafb1e7df86dad99e0a39511064aec9c5d0e86834eae19e5 | 00003532940944c74cccf06585ffb0539dcbafb3d0ed1b741c6d517dffb69050 | user1-->user3-->50.0    | 908   | 2021-12-30 20:35:42 |
+--------+------------------------------------------------------------------+------------------------------------------------------------------+-------------------------+-------+---------------------+
7 rows in set (0.00 sec)
```

Screenshot 17 Blockchain Database

SOURCE CODE

```python
from hashlib import sha256
import time
from Crypto.Signature import pkcs1_15
from Crypto.PublicKey import RSA
from urllib.parse import urlparse


class Block():

    def __init__(self, number=0, prev_hash="0" * 64, transaction=None, nonce=0, timestamp=time.strftime('%Y-%m-%d %H:%M:%S')):
        self.transaction = transaction
        self.number = number
        self.nonce = nonce
        self.prev_hash = prev_hash
        self.timestamp = timestamp

    def hash(self):
        return new_hash(self.number, self.prev_hash, self.transaction, self.nonce, self.timestamp)

    def __str__(self):
        return str("Block: {}\nHash: {}\nPrevious: {}\nTransaction: {}\nNonce: {}\nTime: {}\n".format(self.number, self.hash(), self.prev_hash, self.transaction, self.nonce, self.timestamp))

class Blockchain():

    difficulty = 3

    def __init__(self):
        self.chain = []
        self.pendingTrans = []
        self.nodes = set()

    def register_node(self, address):
        parsedUrl = urlparse(address)
        self.nodes.add(parsedUrl.netloc)

    def add(self, block):
        self.chain.append(block)

    def remove(self, block):
        self.chain.remove(block)

    def mining(self, block):

        try:
            block.prev_hash = self.chain[-1].hash()
        except IndexError:
            pass

        while True:
            if block.hash()[:self.difficulty] == "0" * self.difficulty:
                self.add(block)
                break
            else:
                block.nonce += 1
```

```python
    def addTrans(self, sender, receiver, amt, keyString, senderKey):
        keyByte = keyString.encode("ASCII")
        senderKeyByte = senderKey.encode('ASCII')

        key = RSA.import_key(keyByte)
        senderKey = RSA.import_key(senderKeyByte)

        if not sender or not receiver or not amt:
            return False

        transaction = Transactions(sender, receiver, amt)

        transaction.signTrans(key, senderKey)

        if not transaction.validTrans():
            return False

        self.pendingTrans.append(transaction)
        return len(self.chain) + 1

    def generateKeys(self):
        key = RSA.generate(2048)
        private_key = key.export_key()
        file_out = open("private.pem", "wb")
        file_out.write(private_key)

        public_key = key.publickey().export_key()
        file_out = open("receiver.pem", "wb")
        file_out.write(public_key)

        return key.publickey().export_key().decode('ASCII')

    def valid(self):
        for i in range(1, len(self.chain)):
            previous = self.chain[i].prev_hash
            current = self.chain[i-1].hash()
            if previous != current or current[:self.difficulty] != "0"*self.difficulty:
                return False
        return True

class Transactions():
    def __init__(self, sender="", receiver="", amount=int, timestamp=time.strftime('%Y-%m-%d %H:%M:%S')):
        self.sender = sender
        self.receiver = receiver
        self.amount = amount
        self.timestamp = timestamp
        self.hashed = self.hash()

    def validTrans(self):
        if self.hashed != self.hash():
            return False
        if self.sender == self.receiver:
            return False
        if not self.signature or len(self.signature) == 0:
            return False
        return True
```

```python
    def signTrans(self, key, senderKey):

        if self.hashed != self.hash():
            return False
        if str(key.publickey().export_key()) != str(senderKey.publickey().export_key()):
            return False

        pkcs1_15.new(key)

        self.signature = "made"
        return True


    def hash(self):
        return new_hash(self.sender, self.receiver, self.amount, self.timestamp)

def new_hash(*args):
    text = ""
    hash = sha256()
    for i in args:
        text += str(i)

    hash.update(text.encode('utf-8'))

    return hash.hexdigest()
```

Screenshot 18 Source Code

CONCLUSION

To conclude, this coursework implemented the a cryptocurrency transfer application. Working with classes/OOP made the program very organized which makes it reusable. This program is tremendously user-friendly, not only is the blockchain resistant to modification of the transaction but also is managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks.

A large portion of time was dedicated to research, understanding and testing, as well as looking at alternative implementations to figure out what was most efficient way. Finally, this made me improve my coding/programming skills and it also enhanced my problem solving skills.

LIST OF FIGURES

LIST OF TABLES

LIST OF SCREENSHOTS

REFERENCES

Ang, N. (2019, September 7). *Making My Own Cryptocurrency From Scratch*. Medium. https://medium.com/@nathan_149/making-my-own-cryptocurrency-from-scratch-42e05d4460c2

Blockchain - an overview | ScienceDirect Topics. (n.d.). Www.sciencedirect.com. https://www.sciencedirect.com/topics/engineering/blockchain

Blockchain Tutorial for Beginners: Learn Blockchain Technology. (2019, October 6). Guru99.com. https://www.guru99.com/blockchain-tutorial.html

Blockchain.com Explorer | BTC | ETH | BCH. (n.d.). Www.blockchain.com. Retrieved January 2, 2022, from https://www.blockchain.com/eth/unconfirmed-transactions

BuiltIn. (2013). *What Is Blockchain Technology? How Does Blockchain Work? | Built In*. Builtin.com. https://builtin.com/blockchain

Create simple Blockchain using Python. (2020, July 25). GeeksforGeeks. https://www.geeksforgeeks.org/create-simple-blockchain-using-python/

Flask Tutorial - Tutorialspoint. (2019). Tutorialspoint.com. https://www.tutorialspoint.com/flask/index.htm

How to Create a Blockchain with Python? (2021, November 3). Geekflare. https://geekflare.com/create-a-blockchain-with-python/

OWASP Secure Coding Practices Quick Reference Guide. (2010). https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf

PricewaterhouseCoopers. (2013). *Making sense of bitcoin and blockchain: PwC*. PwC. https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html

Sblendorio, D. (2020, February 6). *How to Build a Blockchain in Python (Pre-built Runtime)*. ActiveState. https://www.activestate.com/blog/how-to-build-a-blockchain-in-python/