

FRAMEWORK FOR CRYPTOCURRENCY USING BLOCKCHAIN

A Mini-project report submitted to CUSAT in partial fulfillment of the
requirements for the award of degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE & ENGINEERING

Submitted by,

Vedansh Chandra Dwivedi (12170240)

Ashish Mishra(13170212)

Ayush Kasera(13170213)

Shreya Krishna(12170238)

Ashish Ranjan(12170214)

Under the guidance of

Mrs. Bindu PK,

Assistant Professor, Department of CSE

Mrs. Alice Joseph,

Associate Professor, Department of CSE



Department of Computer Science & Engineering
COCHIN UNIVERSITY COLLEGE OF ENGINEERING KUTTANAD
ALAPPUZHA

APRIL 2019

**COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY
KUTTANADU, ALAPPUZHA**



BONAFIDE CERTIFICATE

This is to certify that the mini project certified **FRAMEWORK FOR CRYPTOCURRENCY USING BLOCKCHAIN** is a bonafide report of mini project is done by **VEDANSH CHANDRA DWIVEDI (12170240), SHREYA KRISHNA (12170238), ASHISH KUMAR MISHRA (13170212), AYUSH KASERA (13170213), ASHISH RANJAN (12170214)** towards the partial fulfillment of the requirements of the degree of B.tech in COMPUTER SCIENCE AND ENGINEERING of COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY.

Mrs. Alice Joseph
Assistant Professor
Dept. of CSE
Project Coordinator

Mrs. Bindu P K
Associate Professor
Dept. of CSE
Project Coordinator

Dr. Preetha Mathew
Head of Dept.,
CSE

Place: Pulincunnu
Date: 11-04-2019

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our project guides Mrs. Alice Joseph and Mrs. Bindu PK for their timely advice and constant encouragement. We would also express our gratitude to our principal Dr Sunil S Kumar who encourages such activities in the curriculum and provided us with all the necessary requirements we might need and giving us the opportunity to make such an innovative project. We would like to extend our sincere gratitude to Dr. Preetha Mathew, head of department (Computer Science and Technology). This project helped us to learn so much about blockchain and cryptocurrency all through the making of this framework.

ABSTRACT

Banking has existed for years as the system of monetary exchange and carrying of economic activities. In this era of Information Technology Banking has evolved over time. And hence the security of the banking system is expected to evolve with every ascending step in the technology of banking. The safe and secure functioning of the system is of paramount concerns.

Attackers are constantly devising new, sophisticated and hidden methods to target the banks. To understand and detect such complex attacks a secure log record service is to be maintained. This project does the same using newer technology blockchaining and cryptocurrency.

Cryptocurrency is a decentralized model of banking which is highly secure. Blockchain technology will help in creating a secure audit log that can't be tampered. It will give a proof of log manipulation and nonrepudiation.

In this project the initiative of implementing the same by creating a Framework for Cryptocurrency using Blockchain which would ensure high security, reliability and will be dependable. This project will be built on Python.

CONTENTS

Sl.no.	Content	Page no.
1.	Introduction	6
1.1	Scope	7
1.2	Outcome	7
1.3	Blockchain Technology	7
2.	Requirement Analysis	9
2.1	Functional Requirements	10
2.2	Non-Functional Requirements	11
2.3	Proposed System	12
3.	Design	13
4.	Program Code	29
5.	System Overview	76
5.	Testing	83
6.	Conclusion	87
7.	References	88

CONTENTS

List of Figures:

Sl.no.	Content	Page no.
1.	Framework to develop Cryptocurrency System	14
2.	Level 1 DFD of framework	14
3.	Level 2 DFD for get balance	15
4.	Level 2 DFD of Wallet	15
5.	Level 2 DFD for Manage nodes	16
6.	Level 2 DFD for Get Chain	17
7.	Level 2 DFD for get open transaction	17
8.	Level 2 DFD for add transaction	18
9.	Level 2 DFD for add coin/confirm transactions	19
10.	Level 2 DFD for resolve conflict	20
11.	Level 2 DFD for verify transaction	21
12.	Level 3 DFD for add node	22
13.	Level 3 DFD for remove node	23
14.	Level 3 DFD for mine blocks	24
15.	Level 4 DFD for Proof of Work	25
16.	Level 4 DFD for broadcast chain	26
17.	Level 4 DFD for validate hash	26
18.	Level 4 DFD for save data	27
19.	Level 4 DFD for hash block	28
20.	Loading UI of the system	77
21.	Creating keys and initializing funds to 0	77
22.	Public and private key stored in local file	78
23.	Loads wallet in user system with balance	78
24.	Return balance of the user	79
25.	Mine and create a new block	79
26.	Checks validity of chain and returns it	80
27.	Create new open transaction	80
28.	Verifies and prints the list of open transactions	81
29.	POST request to '/mine'	81
30.	Storing blockchain, open transaction, peer nodes in text file	82
31.	Loading a wallet	84
32.	Mining a block to get mining rewards as fund	84
33.	The original information that is stored in the file	85
34.	Manipulated the bits of the signature in open transactions	85
35.	Confirming the manipulation	86
36.	Framework identifies the error and detects the invalid chain	86

1. INTRODUCTION

1.1 SCOPE

In our day to day life we humans want to carry less and desire that everything could be done only through clicks on a single device. The framework that we have developed as our project can be used as a foundation to establish a lot of new cryptocurrencies with very less efforts. This framework bridges the gap between implementation efforts and technical knowledge to establish a decentralized model.

1.1 OUTCOME

It opens the door for developing a democratic open and scalable digital economy from a centralized one.

1.2 BLOCKCHAIN TECHNOLOGY

A blockchain is a distributed database of records, or a decentralized system, or a public ledger of all transactions that have been executed and shared among the participants in the network [1]. Each transaction in the public ledger is validated and verified by the system through the Proof of Work (PoW) provided by the cryptocurrency. Once entered, information can never be erased. The blockchain provides a certain and verifiable record of every transaction that ever happened. The major hypothesis is that the blockchain sets up a system of creating a decentralized consensus in the digital online world. Blockchain technology ensures the elimination of the double spend problem, with the help of public key cryptography, whereby each user is assigned a private key (the secret key) and a public key that is shared with all the other users [2]. For ensuring tamper proof transactions digital signature is used that is created by the public key of sender and private key of recipient. RSA algorithm is used to do the same [3].

The first block of the blockchain is called the genesis block. The genesis block has the index number zero (0). Whenever a transaction is requested a trusted third-party process and umpires the transaction. the role of the trusted third party is to validate, safeguard and preserve historic transactions. The entity receiving the digital currency then verifies the digital signature, which implies ownership of the corresponding private key, by using the public key of the sender on the respective transaction. Every transaction is broadcasted to every node present in the network and is then recorded in a public ledger after verification. The verifying node ensures two things before recording any transaction:

1. Spender owns the cryptocurrency, through the digital signature verification on the transaction.
2. Spender has sufficient cryptocurrency in his account, through checking every transaction against the spender's account, through checking every transaction against the spender's account, or public key, that is registered in the ledger. This ensures that there is sufficient balance in his account before finalizing the transaction.

Transactions are ordered by putting them in blocks and linking these blocks and making a blockchain. The transactions recorded in one block are considered to have happened at the same time. These blocks are linked to each-other (like a chain) in a proper linear, chronological order. Each block header contains an index number, proof, timestamp, previous block hash, difficulty level and the new hash. The index number is the block number. The proof is an integer value between 0 to 4,29,49,67,296. Proof is used to validate the transaction by guessing the proof value that satisfies the difficulty level. The difficulty level is set in accordance with the rate at which blocks are mined or generated. Difficulty level is the consecutive number of zeros in the hash. To create a block the hash of the previous block is needed this makes the blockchain completely dependent. Tampering with any of the blocks would result in the distortion of the whole blockchain. If some nodes of the network are unreliable or malicious, the network is able to correctly verify the transactions and protect the ledger from tampering through a mathematical mechanism called Proof-of-Work (PoW), which makes human intervention or controlling authority unnecessary [4]. The rationale for this protocol is the decentralized trust or trust-by-computation and its importance can hardly be overstated: indeed, it represents “a shift from trusting people to trusting math” [5].

When a new transaction is requested it is validated and then added to the open transaction pool. Then the transactions are mined by the miners. The PoW is what the miners actually perform. Mining is a competition among users to approve transactions. A user's chance of winning the competition is proportional to the computing power he controls. Miners are rewarded for contributing to the verification and block construction process [6]. This is the only way to introduce new currency units in the system. To generate a block the miners require to generate a valid hash. Hashing takes the proof, data and the previous block hash as input to generate a fixed size hash for the block. the value of proof is predicted by the miners. For hashing Secure Hash Algorithm (SHA 256) is used [3]. Hashing ensures that the message is not altered by an unauthorized end user. After the successful creation of block the block is appended to the blockchain and the distributed ledger is updated.

2. REQUIREMENT ANALYSIS

2.1 FUNCTIONAL REQUIREMENTS

2.1.1 Wallet

Wallet is used to provide a unique digital signature to each user which is comprised of public key (known to all) and a private key (known to user). Private key is used to generate a corresponding public key using RSA Algorithm. These pair of keys are stored in a txt file locally in the system. The wallet supports the following functions:

- **Create Keys:** Creates a new pair of keys and saves to local file
- **Load Keys:** Loads the pairs of keys from a local file
- **Signature:** Provides a unique digital signature to each user so that each transaction is encrypted and safe

2.1.2 Get Balance

This function gives out the balance of a user. It can take the public key as the parameters. By default, the public key of the user is passed. This function goes through all the blocks in the blockchain and in each block, it accesses the transactions and calculates the amount of coins sent and received by user. The difference of received coins and sent coins is reflected as balance.

2.1.3 Broadcast

Our framework can easily detect any manipulation in the blockchain. But detection is not everything that makes the system not hackable, the ability to broadcast longest valid chains to resolve conflicts and replace corrupt chain in the peer network is ensured by this function.

2.1.4 Mine Block

This feature is responsible for minting new coins in the network as well as to add transactions to a new block. This function is responsible to validate the transactions and then create a block. Also, the miner receives a MINING REWARD (10 coins default).

2.1.5 Add Transaction

Add transaction enables us to add new transactions to the list of open transactions. This function needs a recipient and amount. Also, the transaction is validated before being added to open transactions.

2.1.6 Remove conflicts

This function comes into play whenever the blockchain needs synchronization as it's quite common that there may be a lot of conflicts when the node network is large. This chain ensures that each node has the same copy of the chain.

2.1.7 Get open transactions

This function lists all the open transactions. Open transactions are the transactions that are yet to be confirmed and added to the chain as a block. This function validates the open transactions to ensure no manipulation of data is done.

2.1.8 Get chain

This function gives out the copy of the blockchain which is fetched from the local file stored in the system. Before giving the chain as output, it verifies the integrity of all the blocks in the chain.

2.1.9 Add node

This function is used to add a node to the list of the peer nodes in the network. List of peer nodes is stored in local file in the user's system.

2.1.10 Remove node

This function is used to delete a particular node from the list of peer nodes. List of peer nodes is stored in local file in the user's system.

2.1.11 Get peer nodes

This function lists all the peer nodes that are participating in the network. List of peer nodes is stored in local file in the user's system.

2.2NON-FUNCTIONAL REQUIREMENTS

2.2.1 Hardware Specification:

Compatible on Core 2 Duo and newer platforms having disk space above 500 MB.

2.2.2 Software Specification:

The software requirements of this are as follows:

- Windows OS/ Linux OS.
- Visual Studio Code
- Flask module for frontend
- Pycryptodome package for RSA Algorithm
- Requests package for sending http requests from python

2.3 PROPOSED SYSTEM

Here we propose a cryptocurrency framework that is implemented using blockchain. This framework will evidently show that blockchain technology is potentially the best and secure method to make and record transactions. This framework showcases features like creating wallet, mine block, view blockchain, add transaction etc. Our proposed framework has the self-healing feature of its block. Using the concept of previous hash and proof of work, the chain gets validated and as soon as the chain is seemed manipulated, the long list valid chain is replaced with the corrupted chain. This framework will take the way we transact to a next level as blockchain technology is considered as the future of transaction.

2.2.1 Advantage:

Security is the major advantage of the proposed system. It provides a transaction system that can't be hacked. Hypothetically even if it is hacked it can be easily detected and nullified in no time.

The framework is flexible and provides the user a flexible platform to create his/her own currency with any level of difficulty.

2.2.2 Disadvantages:

The System security is dependent on the user base of the network. As the network is enlarging, the security of the system also increases which means lesser the user base, lesser the security.

3. DESIGN

3.1 Data Flow Diagram

3.1.1 Context Free Diagram:

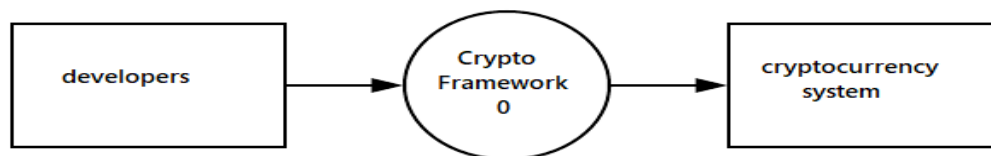


Figure 1: The developer uses the framework to develop Cryptocurrency System.

Level 1:

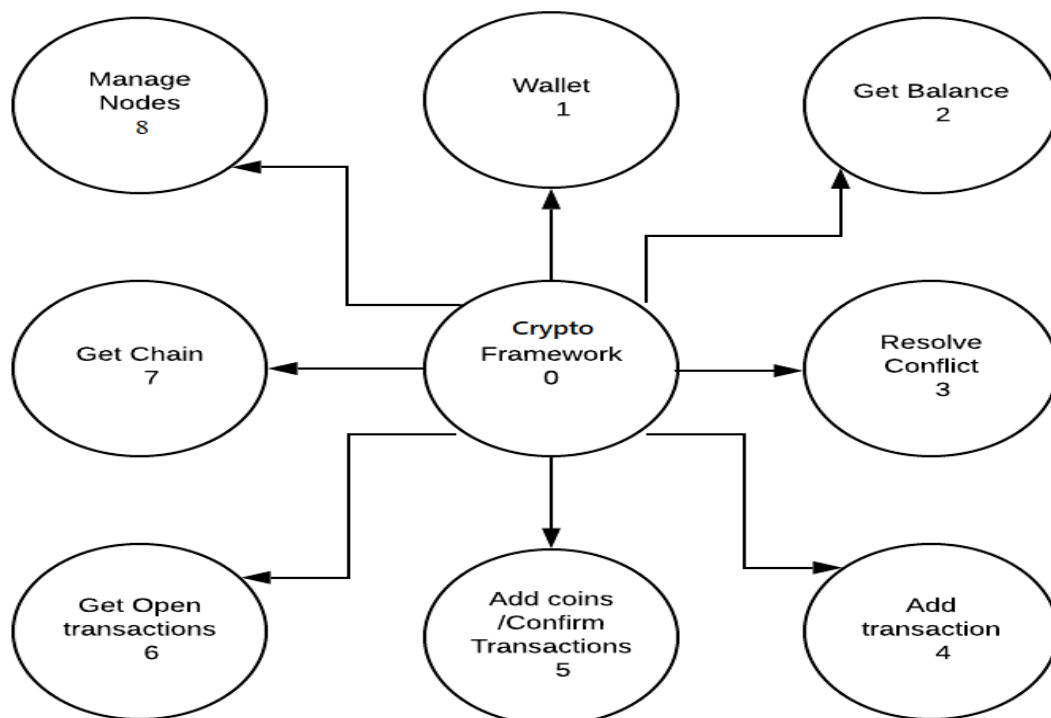
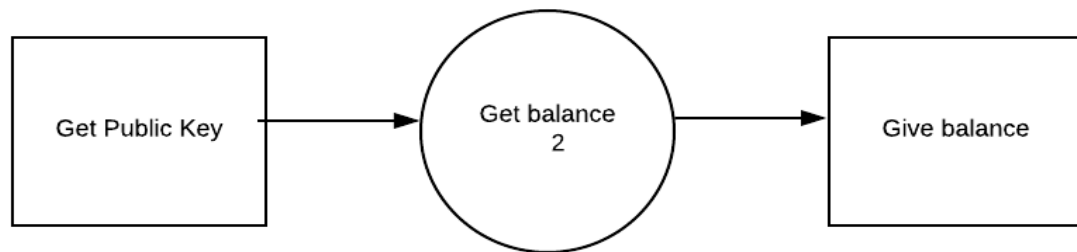
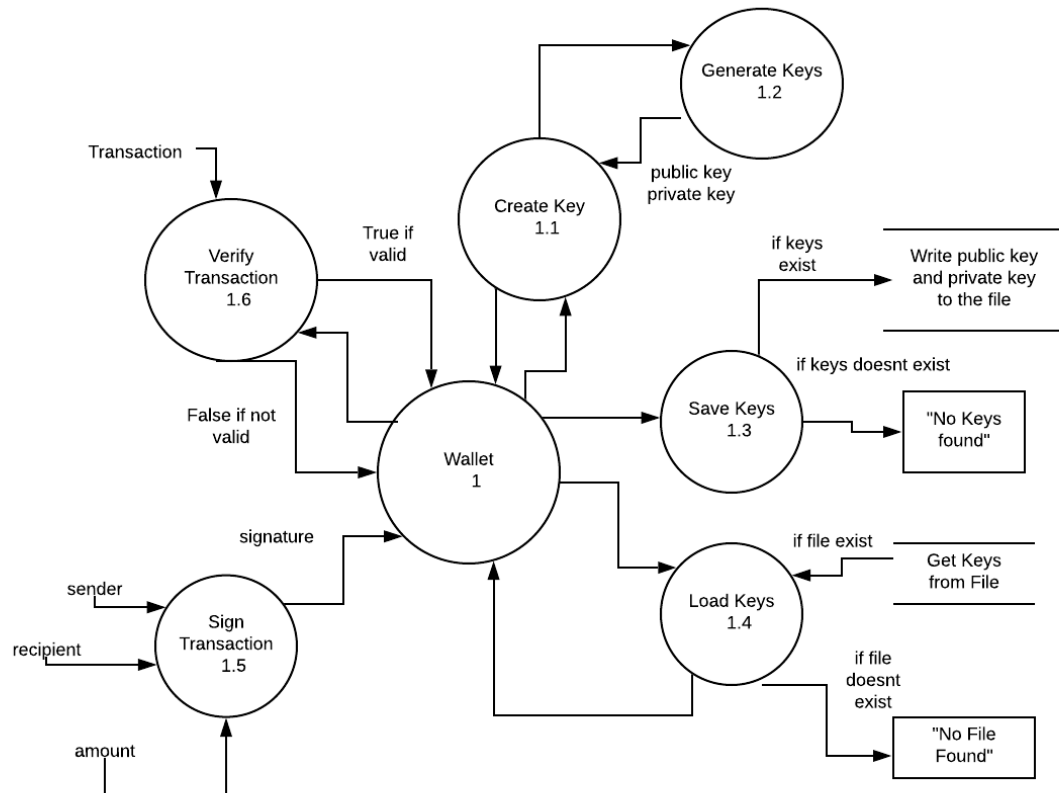


Figure 2: Level 1 DFD of framework

Level 2:**Figure 3: Level 2 DFD for get balance**

The get Balance function needs an optional parameter of 'public key' that is by default set to the user's public key, it gives the balance of the wallet whose public key has been passed as the argument.

**Figure 4: Level 2 DFD of wallet**

Wallet provides the following features

- Create a pair of keys
- Load an existing pair of keys
- Save Keys
- Sign a transaction
- Verify a transaction

The verify transaction verifies that the signature that is there with each transaction. It decrypts the signature using public key corresponding to private key used for signing.

To sign the transaction data is encrypted using the private of the user, and that can only be decrypted by the public key corresponding to private key. So even if one bit is changed, due to avalanche effect, the entire transaction is corrupted.

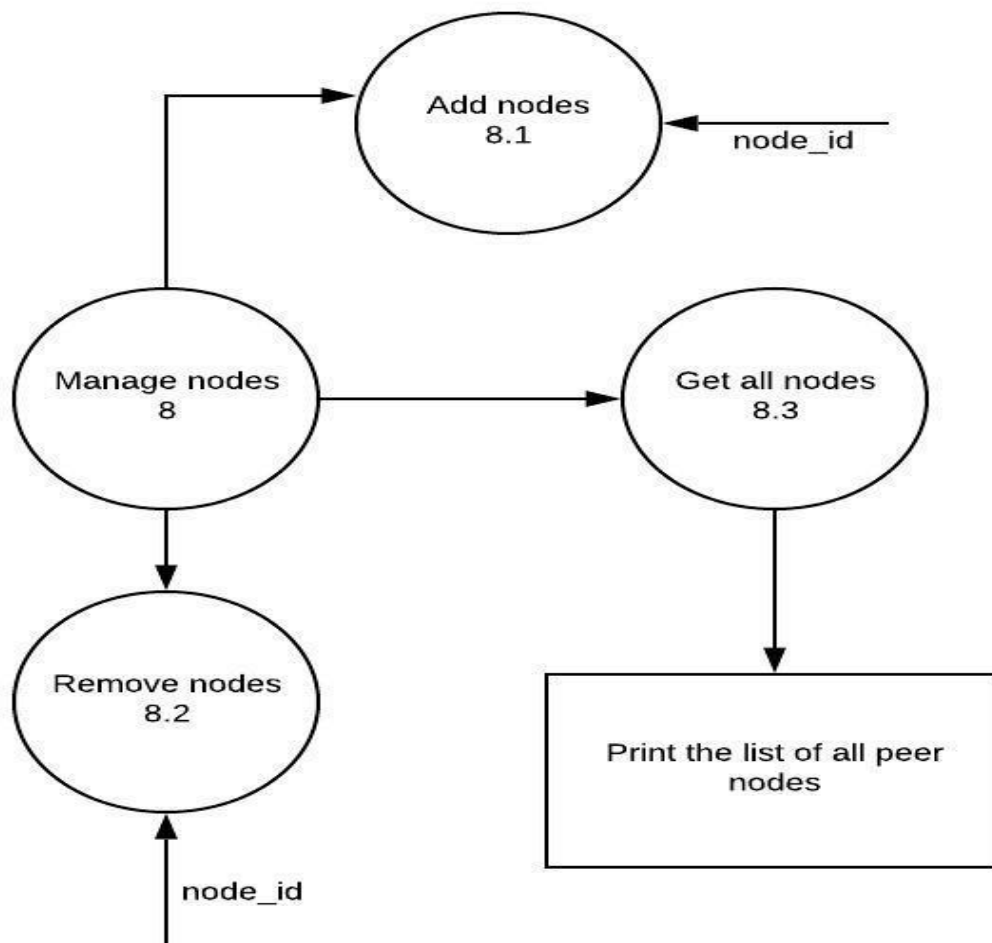


Figure 5: Level 2 DFD for Manage Nodes

The manage node basically does the following tasks:

1. Add a peer node: This feature adds a node to the list of the peer node.
2. Remove a node: This is used to remove a node from the list of peer nodes.
3. Get all nodes: This is used to get the list of all the peer nodes in the network.

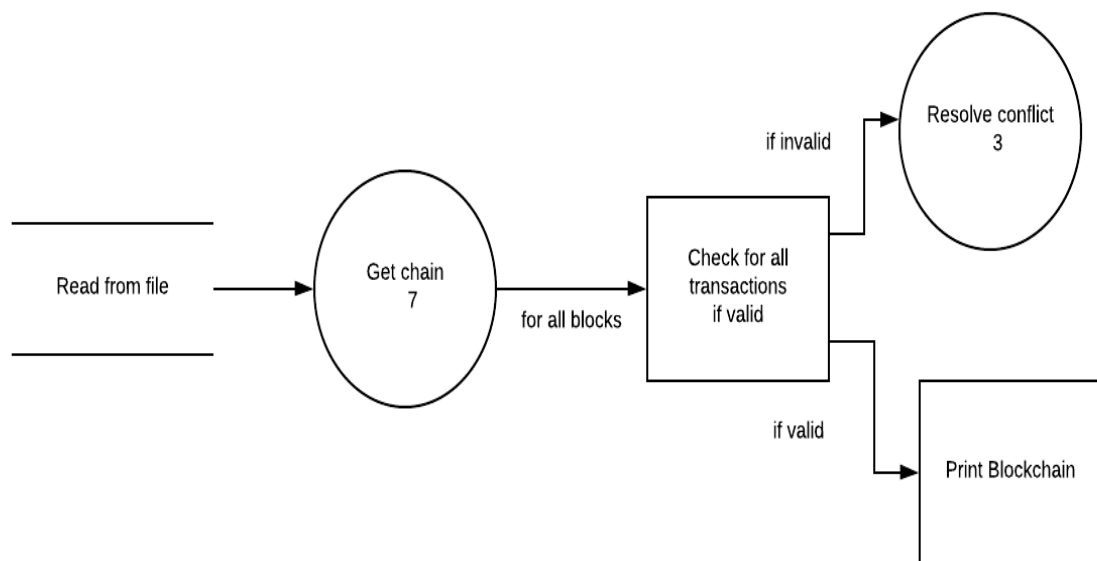


Figure 6: Level 2 DFD for get chain.

The get chain function validates the entire chain and prints the blockchain. In each block for each transaction, it validates the data in the text file and then if valid chain is found, it prints the chain otherwise it calls the resolve-conflict function.

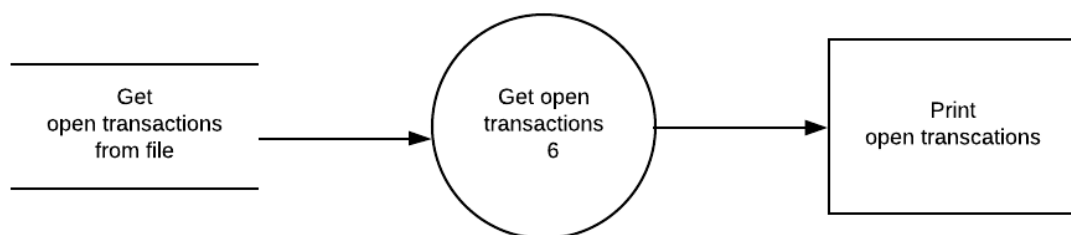


Figure 7: Level 2 DFD for Get Open Transactions

This function fetches the list of open transaction from the local file and prints the list of open transactions.

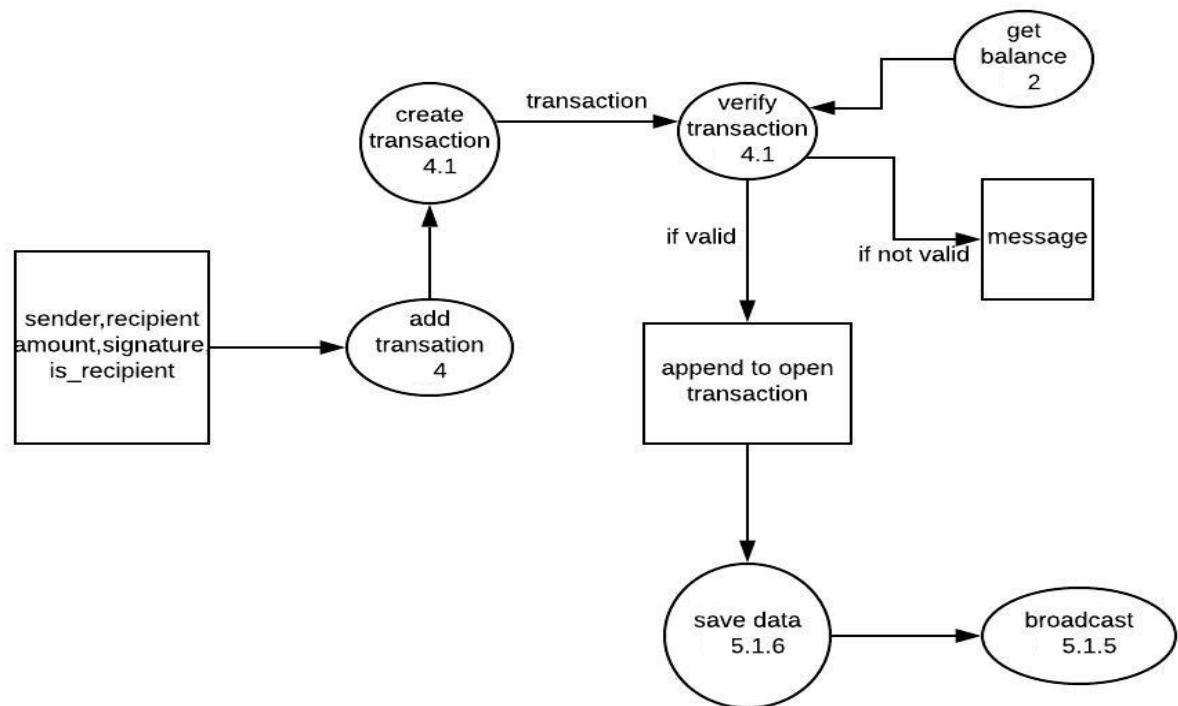


Figure 8: Level 2 DFD for Add Transaction

The 'add transaction' feature allows the user to make a transaction. Initially the transaction is added to the list of unconfirmed transactions. To add a transaction, the user needs to specify the 'recipient' and the 'amount'. After feeding these inputs, the transaction is validated for if the sender has enough funds to make that transaction. Once the validation is done, the transaction is appended to the list of open transactions. After appending the list to the open transactions, we save the data of blockchain and the list of open transactions. After saving, the updates are broadcasted to the network.

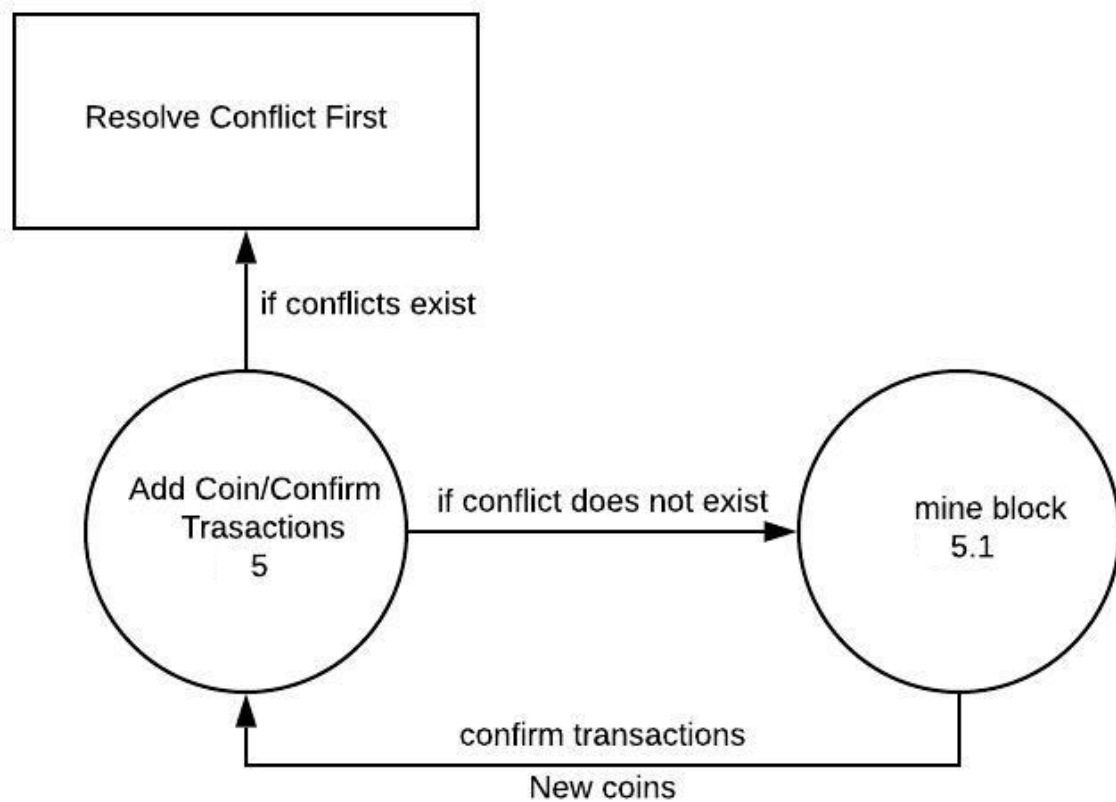


Figure 9: Level 2 DFD for Add Coin/ Confirm Transactions

The add coin/confirm transaction is responsible to mint new coin in the system to mint new coin in the system as well as to confirm the open transactions. Also, if a conflict is present, the resolve conflict function is called.

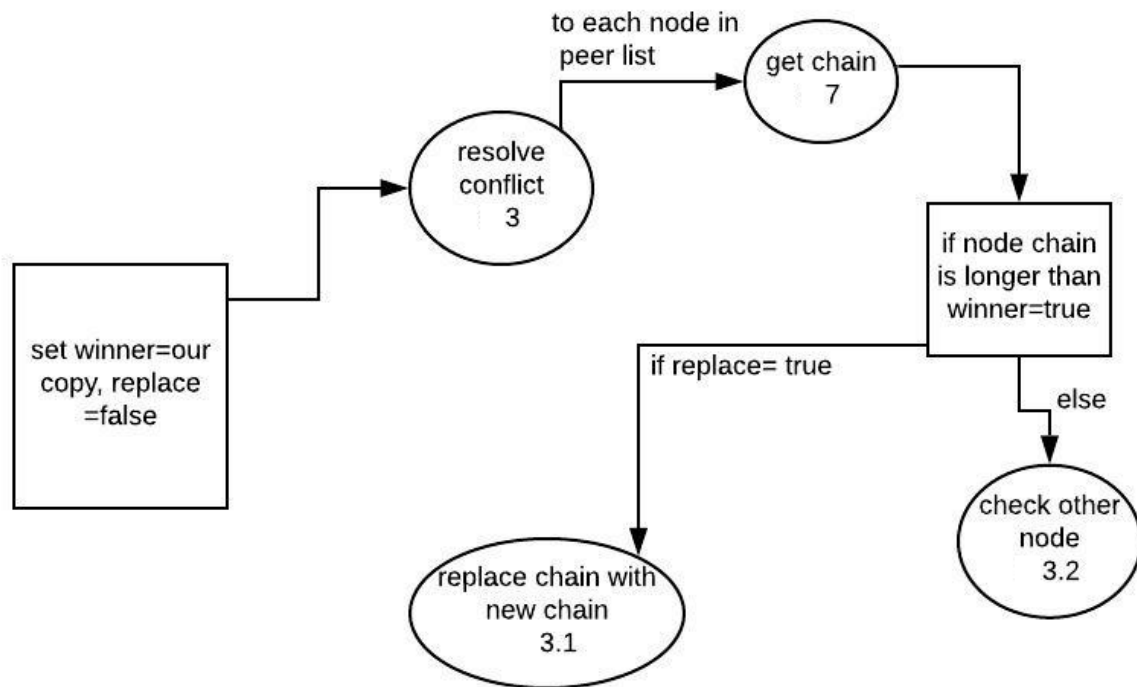


Figure 10: Level 2 DFD for resolve conflict

This feature is used to resolve conflicts between the nodes that is generally caused due to improper synchronization or due to manipulation of the chain by some intruder. It fetches the copy of chain from the peer node and if the length of new chain is compared with local copy and the longer chain is kept. For performing the operation this operation for each node, we get the copy of the longest chain.

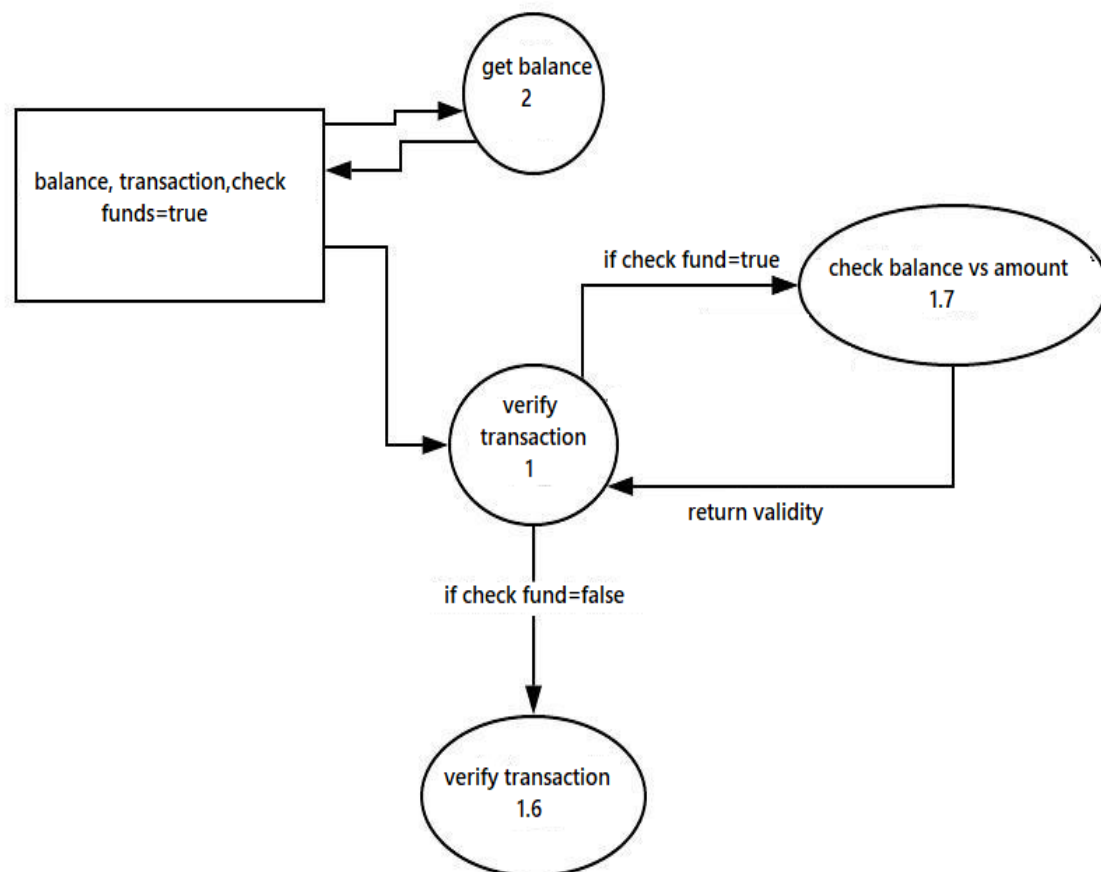


Figure 11: Level 2 DFD for verify transaction

Verify transaction is used to check if a transaction is valid or not, it has an optional parameter which is used to check if amount in transaction is less than balance of the user. Also, it checks the signature.

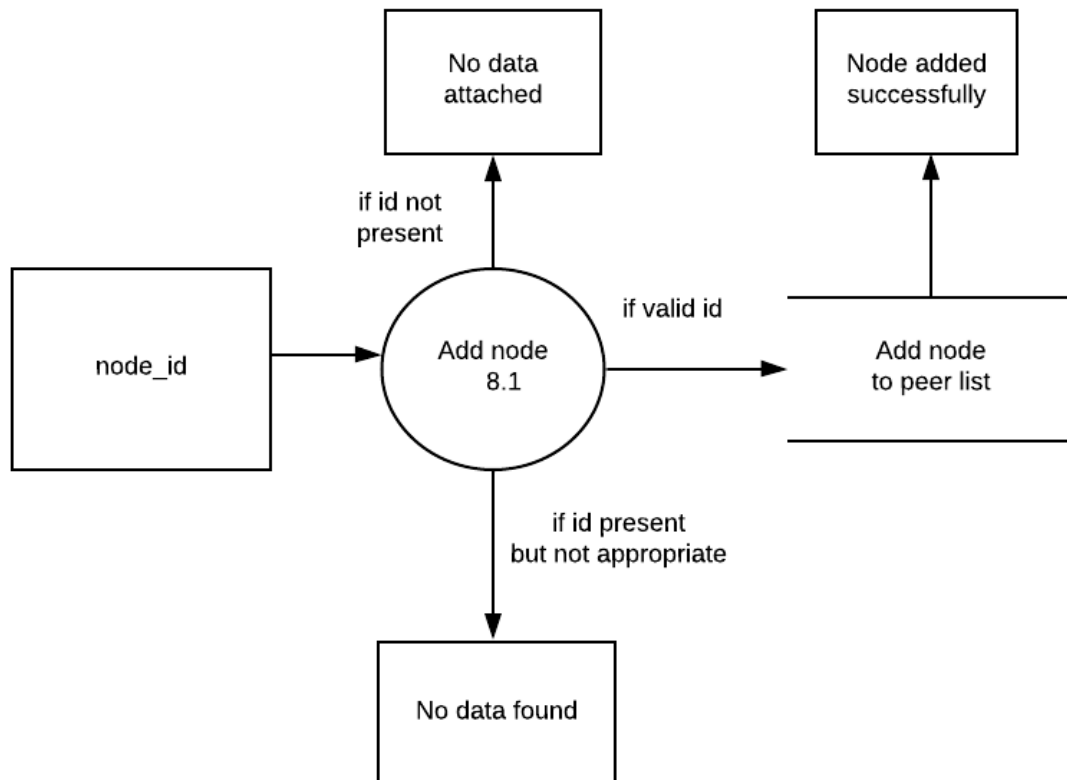
Level 3:

Figure 12: Level 3 DFD for add node

Add node needs a parameter 'node_id', it makes the following checks,

- If no data sent, prints 'no data attached'.
- If data sent is sent but not appropriate, print 'node not found'.
- Valid Node, add a node to the local file and print 'node added successfully'.

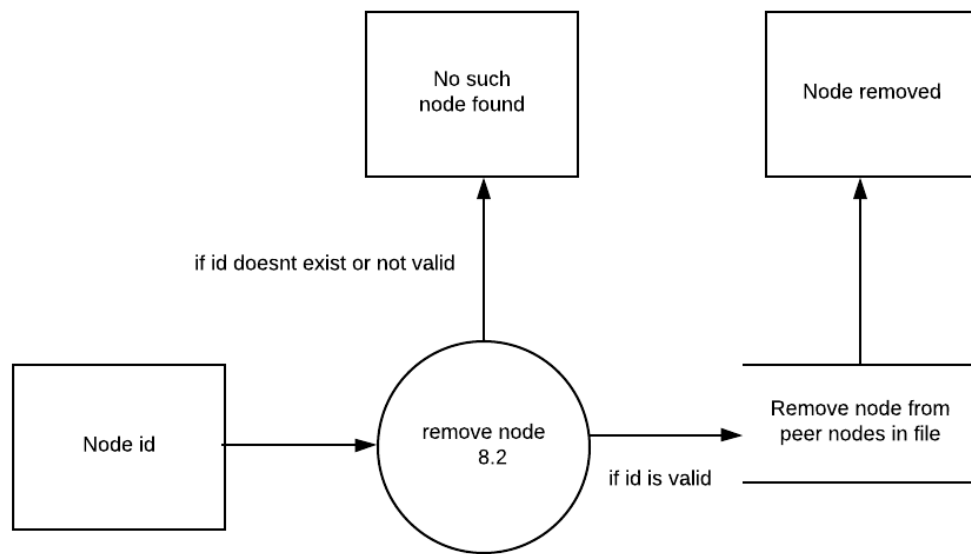


Figure 13: Level 3 DFD for remove node

Remove node needs a parameter 'node_id', it makes the following checks,

1. If data sent, print 'no data attached'.
2. If data sent is sent but is not appropriate, print 'node not found'.
3. Valid Node, Remove the node from the local file and print 'node removed successfully'.

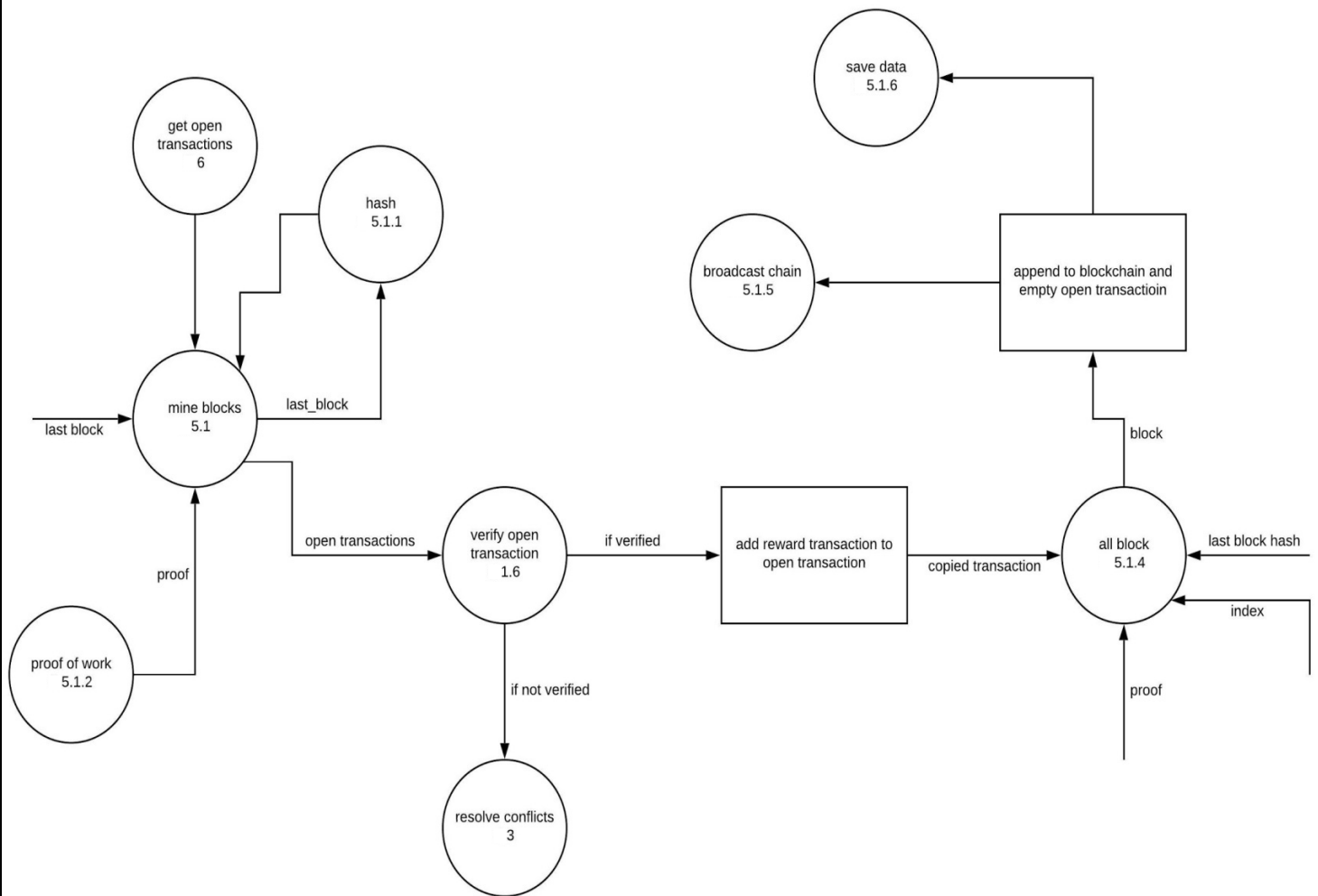
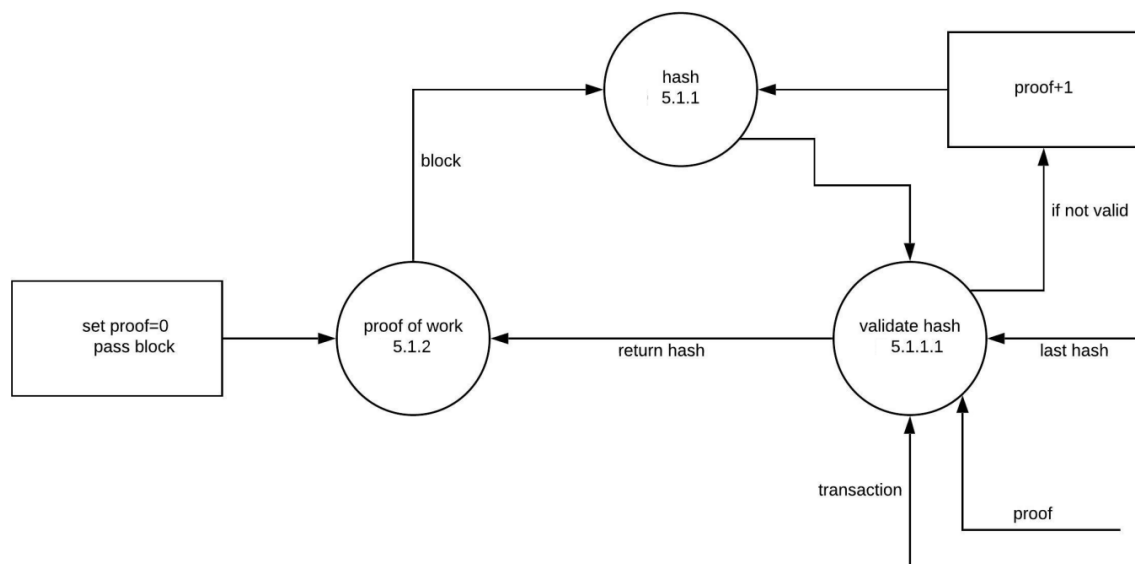


Figure 14: Level 3 DFD for mine blocks

This function fetches the last block of the chain as well as the open transactions. It calculates the hash of the last block and calculates the proof. The open transactions are then verified by their signatures. If the transactions are valid then the mining transaction is appended to the chain of open transaction. These transactions along with the index, hash and proof are passed to add block which yields a block which is appended to the chain and the chain is saved and the broadcasted to the peer nodes.

Level 4:**Figure 15: Level 4 DFD for proof of work**

This function calculates the proof. Initially the proof is set to 0 and the block is hashed. The hash is checked if it meets the difficulty level which by default is first two characters as 00. If the hash meets the difficulty level, proof is returned otherwise the proof is incremented and hash is again hashed until it meets difficulty level. This proof is used to validate the block, the proof stored in the block is used by hashing the block the 'proof' number of times and should match the difficulty level.

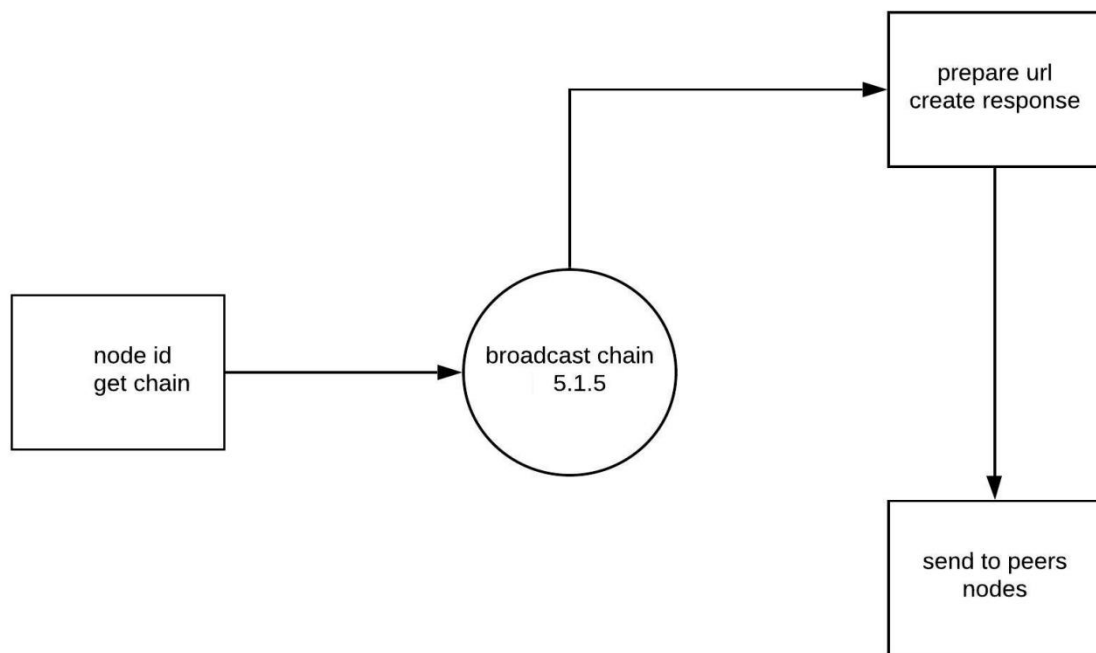


Figure 16: Level 4 DFD for broadcast chain

The broadcast chain function, as the name suggests, is used to broadcast the chain to the entire network. This function is used in resolving conflicts, adding transactions and in adding a new block.

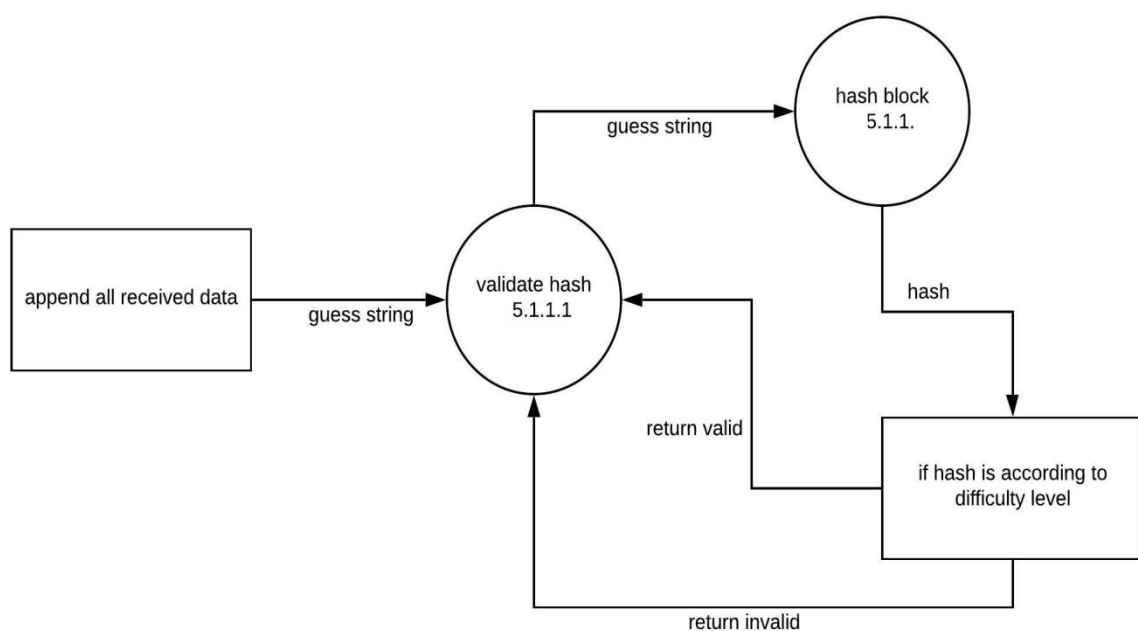


Figure 17: Level 4 DFD for Validate hash

This function is used to validate the hash that has been created. The validate hash function, checks the first two bits of the generated hash and compares it with '00'. If it matches then the hash is said to be valid, otherwise, invalid. The '00' is the default difficulty level provided by the framework for a faster demonstration. While implementing the framework for a system, the developer is free to set a difficulty level according to his needs.

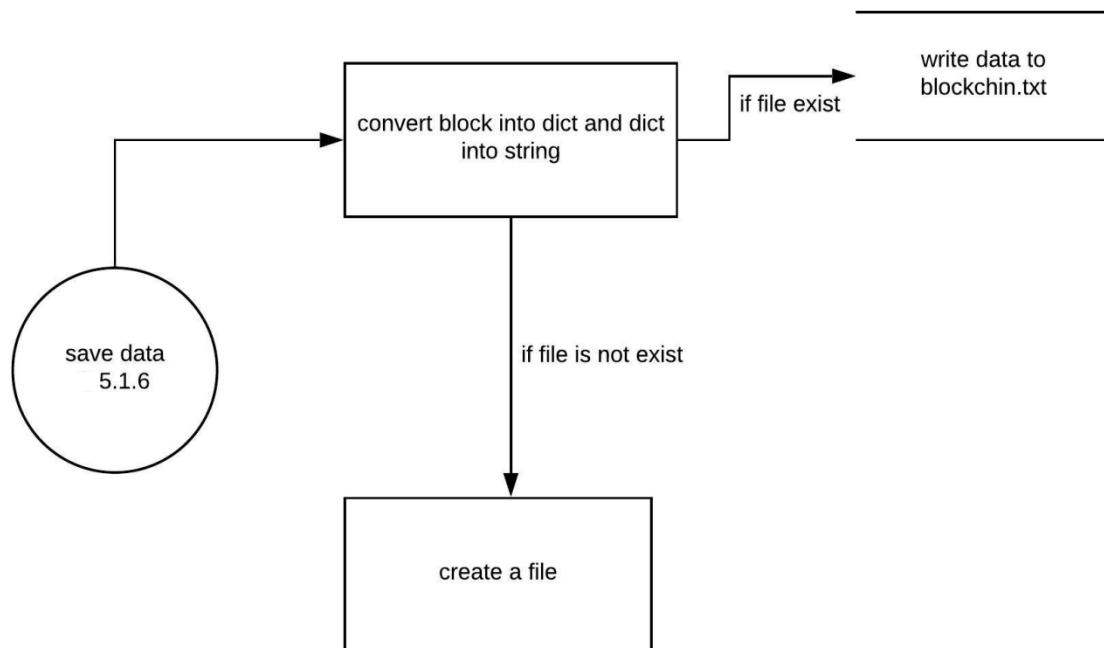


Figure 18: Level 4 DFD for save data.

This function, is used to save a copy of the blockchain, open transactions and the list of peer nodes locally to the system. It converts all the block objects to dictionaries which are then converted to strings using the json package. To avoid errors, it checks if a file already exists or a new file has to be created.

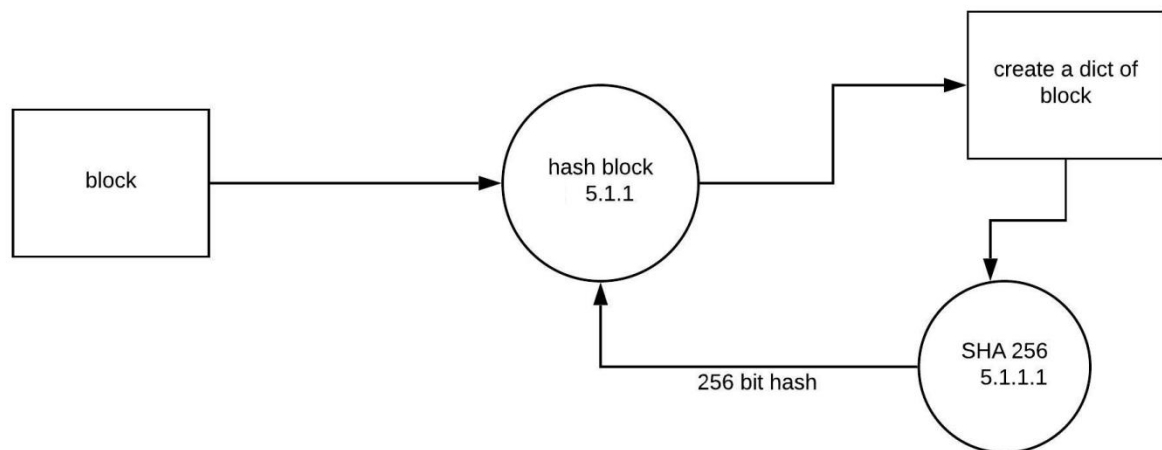


Figure 19: Level 4 DFD for hash block.

The hash block function is used to create a hash for the block. It uses the dictionary representation of the block which is then passed to a sha-256 function which yields a 256-bit binary hash that is converted to a hexadecimal format and is returned by this function.

4. PROGRAM CODE

File: block.py

```
from time import time
```

```
from utility.printable import Printable
```

```
class Block(Printable):
```

```
    """A single block of our blockchain.
```

```
    Attributes:
```

```
        :index: The index of this block.
```

```
        :previous_hash: The hash of the previous block in the blockchain.
```

```
        :timestamp: The timestamp of the block (automatically generated by  
        default).
```

```
        :transactions: A list of transaction which are included in the block.
```

```
        :proof: The proof of work number that yielded this block.
```

```
    """
```

```
    def __init__(self, index, previous_hash, transactions, proof, time=time()):
```

```
        self.index = index
```

```
        self.previous_hash = previous_hash
```

```
        self.timestamp = time
```

```
        self.transactions = transactions
```

```
        self.proof = proof
```

File : blockchain.py

```
from functools import reduce
```

```
import hashlib as hl
```

```
import json
```

```
import pickle
```

```
import requests
```

```
# Import two functions from our hash_util.py file. Omit the ".py" in the import
from utility.hash_util import hash_block
from utility.verification import Verification
from block import Block
from transaction import Transaction
from wallet import Wallet
```

```
# The reward we give to miners (for creating a new block)
MINING_REWARD = 10
```

```
print(__name__)
```

```
class Blockchain:
```

```
    """The Blockchain class manages the chain of blocks as well as open
    transactions and the node on which it's running.
```

```
    Attributes:
```

```
        :chain: The list of blocks
```

```
        :open_transactions (private): The list of open transactions
```

```
        :hosting_node: The connected node (which runs the blockchain).
```

```
    """
```

```
    def __init__(self, public_key, node_id):
```

```
        """The constructor of the Blockchain class."""
```

```
        # Our starting block for the blockchain
```

```
        genesis_block = Block(0, "", [], 100, 0)
```

```
        # Initializing our (empty) blockchain list
```

```
        self.chain = [genesis_block]
```

```
        # Unhandled transactions
```

```
        self.__open_transactions = []
```



```
self.public_key = public_key
self.__peer_nodes = set()
self.node_id = node_id
self.resolve_conflicts = False
self.load_data()

# This turns the chain attribute into a property with a getter (the method
# below) and a setter (@chain.setter)
@property
def chain(self):
    return self.__chain[:]

# The setter for the chain property
@chain.setter
def chain(self, val):
    self.__chain = val

def get_open_transactions(self):
    """Returns a copy of the open transactions list."""
    return self.__open_transactions[:]

def load_data(self):
    """Initialize blockchain + open transactions data from a file."""
    try:
        with open('blockchain-{}.txt'.format(self.node_id), mode='r') as f:
            # file_content = pickle.loads(f.read())
            file_content = f.readlines()
            # blockchain = file_content['chain']
            # open_transactions = file_content['ot']
            blockchain = json.loads(file_content[0][-1])
            # We need to convert the loaded data because Transactions
            # should use OrderedDict
```

```
updated_blockchain = []
for block in blockchain:
    converted_tx = [Transaction(
        tx['sender'],
        tx['recipient'],
        tx['signature'],
        tx['amount']) for tx in block['transactions']]
    updated_block = Block(
        block['index'],
        block['previous_hash'],
        converted_tx,
        block['proof'],
        block['timestamp'])
    updated_blockchain.append(updated_block)
self.chain = updated_blockchain
open_transactions = json.loads(file_content[1][:-1])
# We need to convert the loaded data because Transactions
# should use OrderedDict
updated_transactions = []
for tx in open_transactions:
    updated_transaction = Transaction(
        tx['sender'],
        tx['recipient'],
        tx['signature'],
        tx['amount'])
    updated_transactions.append(updated_transaction)
self.__open_transactions = updated_transactions
peer_nodes = json.loads(file_content[2])
self.__peer_nodes = set(peer_nodes)
except (IOError, IndexError):
    pass
finally:
```

```
print('Cleanup!')

def save_data(self):
    """Save blockchain + open transactions snapshot to a file."""
    try:
        with open('blockchain-{}.txt'.format(self.node_id), mode='w') as f:
            saveable_chain = [
                block.__dict__ for block in
                [
                    Block(block_el.index,
                        block_el.previous_hash,
                        [tx.__dict__ for tx in block_el.transactions],
                        block_el.proof,
                        block_el.timestamp) for block_el in self.__chain
                ]
            ]
            f.write(json.dumps(saveable_chain))
            f.write('\n')
            saveable_tx = [tx.__dict__ for tx in self.__open_transactions]
            f.write(json.dumps(saveable_tx))
            f.write('\n')
            f.write(json.dumps(list(self.__peer_nodes)))
            # save_data = {
            #     'chain': blockchain,
            #     'ot': open_transactions
            # }
            # f.write(pickle.dumps(save_data))
    except IOError:
        print('Saving failed!')

def proof_of_work(self):
    """Generate a proof of work for the open transactions, the hash of the
```

```

previous block and a random number (which is guessed until it fits)."""
last_block = self.__chain[-1]
last_hash = hash_block(last_block)
proof = 0
# Try different PoW numbers and return the first valid one
while not Verification.valid_proof(
    self.__open_transactions,
    last_hash, proof
):
    proof += 1
return proof

def get_balance(self, sender=None):
    """Calculate and return the balance for a participant.
    """
    if sender is None:
        if self.public_key is None:
            return None
        participant = self.public_key
    else:
        participant = sender

    # Fetch a list of all sent coin amounts for the given person (empty
    # lists are returned if the person was NOT the sender)
    # This fetches sent amounts of transactions that were already included
    # in blocks of the blockchain
    tx_sender = [[tx.amount for tx in block.transactions
                    if tx.sender == participant] for block in self.__chain]

    # Fetch a list of all sent coin amounts for the given person (empty
    # lists are returned if the person was NOT the sender)
    # This fetches sent amounts of open transactions (to avoid double
    # spending)
    open_tx_sender = [

```

```

        tx.amount for tx in self.__open_transactions
        if tx.sender == participant
    ]
    tx_sender.append(open_tx_sender)
    print(tx_sender)
    amount_sent = reduce(lambda tx_sum, tx_amt: tx_sum + sum(tx_amt)
                          if len(tx_amt) > 0 else tx_sum + 0, tx_sender, 0)
    # This fetches received coin amounts of transactions that were already
    # included in blocks of the blockchain
    # We ignore open transactions here because you shouldn't be able to
    # spend coins before the transaction was confirmed + included in a
    # block
    tx_recipient = [
        [
            tx.amount for tx in block.transactions
            if tx.recipient == participant
        ] for block in self.__chain
    ]
    amount_received = reduce(
        lambda tx_sum, tx_amt: tx_sum + sum(tx_amt)
        if len(tx_amt) > 0 else tx_sum + 0,
        tx_recipient,
        0
    )
    # Return the total balance
    return amount_received - amount_sent

def get_last_blockchain_value(self):
    """ Returns the last value of the current blockchain. """
    if len(self.__chain) < 1:
        return None
    return self.__chain[-1]

```

```
# This function accepts two arguments.
# One required one (transaction_amount) and one optional one
# (last_transaction)
# The optional one is optional because it has a default value => [1]
```

```
def add_transaction(self,
                    recipient,
                    sender,
                    signature,
                    amount=1.0,
                    is_receiving=False):
    """ Append a new value as well as the last blockchain value to the blockchain.
```

Arguments:

```
:sender: The sender of the coins.
:recipient: The recipient of the coins.
:amount: The amount of coins sent with the transaction
(default = 1.0)
```

```
"""
```

```
# transaction = {
#   'sender': sender,
#   'recipient': recipient,
#   'amount': amount
# }
# if self.public_key == None:
#     return False
transaction = Transaction(sender, recipient, signature, amount)
if Verification.verify_transaction(transaction, self.get_balance):
    self.__open_transactions.append(transaction)
    self.save_data()
    if not is_receiving:
```

```
for node in self.__peer_nodes:
    url = 'http://{}/broadcast-transaction'.format(node)
    try:
        response = requests.post(url,
                                  json={
                                      'sender': sender,
                                      'recipient': recipient,
                                      'amount': amount,
                                      'signature': signature
                                  })
        if (response.status_code == 400 or
            response.status_code == 500):
            print('Transaction declined, needs resolving')
            return False
    except requests.exceptions.ConnectionError:
        continue
    return True
return False

def mine_block(self):
    """Create a new block and add open transactions to it."""
    # Fetch the currently last block of the blockchain
    if self.public_key is None:
        return None
    last_block = self.__chain[-1]
    # Hash the last block (=> to be able to compare it to the stored hash
    # value)
    hashed_block = hash_block(last_block)
    proof = self.proof_of_work()
    # Miners should be rewarded, so let's create a reward transaction
    # reward_transaction = {
    #     'sender': 'MINING',
```

```
# 'recipient': owner,
# 'amount': MINING_REWARD
# }
reward_transaction = Transaction(
    'MINING', self.public_key, "", MINING_REWARD)
# Copy transaction instead of manipulating the original
# open_transactions list
# This ensures that if for some reason the mining should fail,
# we don't have the reward transaction stored in the open transactions
copied_transactions = self.__open_transactions[:]
for tx in copied_transactions:
    if not Wallet.verify_transaction(tx):
        return None
copied_transactions.append(reward_transaction)
block = Block(len(self.__chain), hashed_block,
              copied_transactions, proof)
self.__chain.append(block)
self.__open_transactions = []
self.save_data()
for node in self.__peer_nodes:
    url = 'http://{}/broadcast-block'.format(node)
    converted_block = block.__dict__.copy()
    converted_block['transactions'] = [
        tx.__dict__ for tx in converted_block['transactions']]
    try:
        response = requests.post(url, json={'block': converted_block})
        if response.status_code == 400 or response.status_code == 500:
            print('Block declined, needs resolving')
        if response.status_code == 409:
            self.resolve_conflicts = True
    except requests.exceptions.ConnectionError:
        continue
```



```
return block
```

```
def add_block(self, block):
```

```
    """Add a block which was received via broadcasting to the local  
    lockchain."""
```

```
    # Create a list of transaction objects
```

```
    transactions = [Transaction(  
        tx['sender'],  
        tx['recipient'],  
        tx['signature'],  
        tx['amount']) for tx in block['transactions']]
```

```
    # Validate the proof of work of the block and store the result (True  
    # or False) in a variable
```

```
    proof_is_valid = Verification.valid_proof(  
        transactions[:-1], block['previous_hash'], block['proof'])
```

```
    # Check if previous_hash stored in the block is equal to the local
```

```
    # blockchain's last block's hash and store the result in a block
```

```
    hashes_match = hash_block(self.chain[-1]) == block['previous_hash']
```

```
    if not proof_is_valid or not hashes_match:
```

```
        return False
```

```
    # Create a Block object
```

```
    converted_block = Block(  
        block['index'],  
        block['previous_hash'],  
        transactions,  
        block['proof'],  
        block['timestamp'])
```

```
    self.__chain.append(converted_block)
```

```
    stored_transactions = self.__open_transactions[:]
```

```
    # Check which open transactions were included in the received block
```

```
    # and remove them
```

```
    # This could be improved by giving each transaction an ID that would
```

```
# uniquely identify it
for itx in block['transactions']:
    for opentx in stored_transactions:
        if (opentx.sender == itx['sender'] and
            opentx.recipient == itx['recipient'] and
            opentx.amount == itx['amount'] and
            opentx.signature == itx['signature']):
            try:
                self.__open_transactions.remove(opentx)
            except ValueError:
                print('Item was already removed')
self.save_data()
return True

def resolve(self):
    """Checks all peer nodes' blockchains and replaces the local one with
    longer valid ones."""
    # Initialize the winner chain with the local chain
    winner_chain = self.chain
    replace = False
    for node in self.__peer_nodes:
        url = 'http://{}/chain'.format(node)
        try:
            # Send a request and store the response
            response = requests.get(url)
            # Retrieve the JSON data as a dictionary
            node_chain = response.json()
            # Convert the dictionary list to a list of block AND
            # transaction objects
            node_chain = [
                Block(block['index'],
                    block['previous_hash'],
```

```

        [
            Transaction(
                tx['sender'],
                tx['recipient'],
                tx['signature'],
                tx['amount']) for tx in block['transactions']
        ],
        block['proof'],
        block['timestamp']) for block in node_chain
    ]

    node_chain_length = len(node_chain)
    local_chain_length = len(winner_chain)

    # Store the received chain as the current winner chain if it's
    # longer AND valid
    if (node_chain_length > local_chain_length and
        Verification.verify_chain(node_chain)):
        winner_chain = node_chain
        replace = True
    except requests.exceptions.ConnectionError:
        continue

    self.resolve_conflicts = False

    # Replace the local chain with the winner chain
    self.chain = winner_chain

    if replace:
        self.__open_transactions = []
    self.save_data()

    return replace

def add_peer_node(self, node):
    """Adds a new node to the peer node set.

```

Arguments:

```

        :node: The node URL which should be added.
        """

        self.__peer_nodes.add(node)
        self.save_data()

    def remove_peer_node(self, node):
        """Removes a node from the peer node set.

        Arguments:
            :node: The node URL which should be removed.
        """

        self.__peer_nodes.discard(node)
        self.save_data()

    def get_peer_nodes(self):
        """Return a list of all connected peer nodes."""
        return list(self.__peer_nodes)

```

File:node.py

```

from flask import Flask, jsonify, request, send_from_directory
from flask_cors import CORS

from wallet import Wallet
from blockchain import Blockchain

app = Flask(__name__)
CORS(app)

@app.route('/', methods=['GET'])

```

```
def get_node_ui():
    return send_from_directory('ui', 'node.html')


@app.route('/network', methods=['GET'])
def get_network_ui():
    return send_from_directory('ui', 'network.html')


@app.route('/wallet', methods=['POST'])
def create_keys():
    wallet.create_keys()
    if wallet.save_keys():
        global blockchain
        blockchain = Blockchain(wallet.public_key, port)
        response = {
            'public_key': wallet.public_key,
            'private_key': wallet.private_key,
            'funds': blockchain.get_balance()
        }
        return jsonify(response), 201
    else:
        response = {
            'message': 'Saving the keys failed.'
        }
        return jsonify(response), 500


@app.route('/wallet', methods=['GET'])
def load_keys():
    if wallet.load_keys():
        global blockchain
```

```
blockchain = Blockchain(wallet.public_key, port)

response = {
    'public_key': wallet.public_key,
    'private_key': wallet.private_key,
    'funds': blockchain.get_balance()
}

return jsonify(response), 201

else:
    response = {
        'message': 'Loading the keys failed.'
    }
    return jsonify(response), 500

@app.route('/balance', methods=['GET'])
def get_balance():
    balance = blockchain.get_balance()
    if balance is not None:
        response = {
            'message': 'Fetched balance successfully.',
            'funds': balance
        }
        return jsonify(response), 200
    else:
        response = {
            'messsage': 'Loading balance failed.',
            'wallet_set_up': wallet.public_key is not None
        }
        return jsonify(response), 500

@app.route('/broadcast-transaction', methods=['POST'])
```

```
def broadcast_transaction():
    values = request.get_json()
    if not values:
        response = {'message': 'No data found.'}
        return jsonify(response), 400
    required = ['sender', 'recipient', 'amount', 'signature']
    if not all(key in values for key in required):
        response = {'message': 'Some data is missing.'}
        return jsonify(response), 400
    success = blockchain.add_transaction(
        values['recipient'],
        values['sender'],
        values['signature'],
        values['amount'],
        is_receiving=True)
    if success:
        response = {
            'message': 'Successfully added transaction.',
            'transaction': {
                'sender': values['sender'],
                'recipient': values['recipient'],
                'amount': values['amount'],
                'signature': values['signature']
            }
        }
        return jsonify(response), 201
    else:
        response = {
            'message': 'Creating a transaction failed.'
        }
        return jsonify(response), 500
```

```
@app.route('/broadcast-block', methods=['POST'])
def broadcast_block():
    values = request.get_json()
    if not values:
        response = {'message': 'No data found.'}
        return jsonify(response), 400
    if 'block' not in values:
        response = {'message': 'Some data is missing.'}
        return jsonify(response), 400
    block = values['block']
    if block['index'] == blockchain.chain[-1].index + 1:
        if blockchain.add_block(block):
            response = {'message': 'Block added'}
            return jsonify(response), 201
        else:
            response = {'message': 'Block seems invalid.'}
            return jsonify(response), 409
    elif block['index'] > blockchain.chain[-1].index:
        response = {
            'message': 'Blockchain seems to differ from local blockchain.'}
        blockchain.resolve_conflicts = True
        return jsonify(response), 200
    else:
        response = {
            'message': 'Blockchain seems to be shorter, block not added'}
        return jsonify(response), 409

@app.route('/transaction', methods=['POST'])
def add_transaction():
    if wallet.public_key is None:
```



```
response = {
    'message': 'No wallet set up.'
}
return jsonify(response), 400
values = request.get_json()
if not values:
    response = {
        'message': 'No data found.'
    }
    return jsonify(response), 400
required_fields = ['recipient', 'amount']
if not all(field in values for field in required_fields):
    response = {
        'message': 'Required data is missing.'
    }
    return jsonify(response), 400
recipient = values['recipient']
amount = values['amount']
signature = wallet.sign_transaction(wallet.public_key, recipient, amount)
success = blockchain.add_transaction(
    recipient, wallet.public_key, signature, amount)
if success:
    response = {
        'message': 'Successfully added transaction.',
        'transaction': {
            'sender': wallet.public_key,
            'recipient': recipient,
            'amount': amount,
            'signature': signature
        },
        'funds': blockchain.get_balance()
    }
```

```
        return jsonify(response), 201
    else:
        response = {
            'message': 'Creating a transaction failed.'
        }
        return jsonify(response), 500

@app.route('/mine', methods=['POST'])
def mine():
    if blockchain.resolve_conflicts:
        response = {'message': 'Resolve conflicts first, block not added!'}
        return jsonify(response), 409
    block = blockchain.mine_block()
    if block is not None:
        dict_block = block.__dict__.copy()
        dict_block['transactions'] = [
            tx.__dict__ for tx in dict_block['transactions']]
        response = {
            'message': 'Block added successfully.',
            'block': dict_block,
            'funds': blockchain.get_balance()
        }
        return jsonify(response), 201
    else:
        response = {
            'message': 'Adding a block failed.',
            'wallet_set_up': wallet.public_key is not None
        }
        return jsonify(response), 500
```

```
@app.route('/resolve-conflicts', methods=['POST'])
```

```
def resolve_conflicts():
```

```
    replaced = blockchain.resolve()
```

```
    if replaced:
```

```
        response = {'message': 'Chain was replaced!'}
```

```
    else:
```

```
        response = {'message': 'Local chain kept!'}
```

```
    return jsonify(response), 200
```

```
@app.route('/transactions', methods=['GET'])
```

```
def get_open_transaction():
```

```
    transactions = blockchain.get_open_transactions()
```

```
    dict_transactions = [tx.__dict__ for tx in transactions]
```

```
    return jsonify(dict_transactions), 200
```

```
@app.route('/chain', methods=['GET'])
```

```
def get_chain():
```

```
    chain_snapshot = blockchain.chain
```

```
    dict_chain = [block.__dict__.copy() for block in chain_snapshot]
```

```
    for dict_block in dict_chain:
```

```
        dict_block['transactions'] = [
```

```
            tx.__dict__ for tx in dict_block['transactions']]
```

```
    return jsonify(dict_chain), 200
```

```
@app.route('/node', methods=['POST'])
```

```
def add_node():
```

```
    values = request.get_json()
```

```
    if not values:
```

```
        response = {
```

```
        'message': 'No data attached.'
    }
    return jsonify(response), 400
if 'node' not in values:
    response = {
        'message': 'No node data found.'
    }
    return jsonify(response), 400
node = values['node']
blockchain.add_peer_node(node)
response = {
    'message': 'Node added successfully.',
    'all_nodes': blockchain.get_peer_nodes()
}
return jsonify(response), 201

@app.route('/node/<node_url>', methods=['DELETE'])
def remove_node(node_url):
    if node_url == "" or node_url is None:
        response = {
            'message': 'No node found.'
        }
        return jsonify(response), 400
    blockchain.remove_peer_node(node_url)
    response = {
        'message': 'Node removed',
        'all_nodes': blockchain.get_peer_nodes()
    }
    return jsonify(response), 200
```

```
@app.route('/nodes', methods=['GET'])
def get_nodes():
    nodes = blockchain.get_peer_nodes()
    response = {
        'all_nodes': nodes
    }
    return jsonify(response), 200

if __name__ == '__main__':
    from argparse import ArgumentParser
    parser = ArgumentParser()
    parser.add_argument('-p', '--port', type=int, default=5000)
    args = parser.parse_args()
    port = args.port
    wallet = Wallet(port)
    blockchain = Blockchain(wallet.public_key, port)
    app.run(host='0.0.0.0', port=port)
```

File:transaction.py

```
from collections import OrderedDict
```

```
from utility.printable import Printable
```

```
class Transaction(Printable):
```

```
    """A transaction which can be added to a block in the blockchain.
```

```
    Attributes:
```

```
        :sender: The sender of the coins.
```

```
        :recipient: The recipient of the coins.
```

```
:signature: The signature of the transaction.  
:amount: The amount of coins sent.  
""  
  
def __init__(self, sender, recipient, signature, amount):  
    self.sender = sender  
    self.recipient = recipient  
    self.amount = amount  
    self.signature = signature  
  
def to_ordered_dict(self):  
    """Converts this transaction into a (hashable) OrderedDict."""  
    return OrderedDict([('sender', self.sender),  
                        ('recipient', self.recipient),  
                        ('amount', self.amount)])
```

File: wallet.py

```
from Crypto.PublicKey import RSA  
from Crypto.Signature import PKCS1_v1_5  
from Crypto.Hash import SHA256  
import Crypto.Random  
import binascii  
  
class Wallet:  
    """Creates, loads and holds private and public keys. Manages transaction  
    signing and verification."""  
  
    def __init__(self, node_id):  
        self.private_key = None
```

```
self.public_key = None
self.node_id = node_id

def create_keys(self):
    """Create a new pair of private and public keys."""
    private_key, public_key = self.generate_keys()
    self.private_key = private_key
    self.public_key = public_key

def save_keys(self):
    """Saves the keys to a file (wallet.txt)."""
    if self.public_key is not None and self.private_key is not None:
        try:
            with open('wallet-{}.txt'.format(self.node_id), mode='w') as f:
                f.write(self.public_key)
                f.write('\n')
                f.write(self.private_key)
            return True
        except (IOError, IndexError):
            print('Saving wallet failed...')
            return False

def load_keys(self):
    """Loads the keys from the wallet.txt file into memory."""
    try:
        with open('wallet-{}.txt'.format(self.node_id), mode='r') as f:
            keys = f.readlines()
            public_key = keys[0][:1]
            private_key = keys[1]
            self.public_key = public_key
            self.private_key = private_key
        return True
```

```
except (IOError, IndexError):
    print('Loading wallet failed...')
    return False

def generate_keys(self):
    """Generate a new pair of private and public key."""
    private_key = RSA.generate(1024, Crypto.Random.new().read)
    public_key = private_key.publickey()
    return (
        binascii
        .hexlify(private_key.exportKey(format='DER'))
        .decode('ascii'),
        binascii
        .hexlify(public_key.exportKey(format='DER'))
        .decode('ascii')
    )

def sign_transaction(self, sender, recipient, amount):
    """Sign a transaction and return the signature.

    Arguments:
        :sender: The sender of the transaction.
        :recipient: The recipient of the transaction.
        :amount: The amount of the transaction.
    """
    signer = PKCS1_v1_5.new(RSA.importKey(
        binascii.unhexlify(self.private_key)))
    h = SHA256.new((str(sender) + str(recipient) +
        str(amount)).encode('utf8'))
    signature = signer.sign(h)
    return binascii.hexlify(signature).decode('ascii')
```


@staticmethod

def verify_transaction(transaction):

"""Verify the signature of a transaction.

Arguments:

:transaction: The transaction that should be verified.

"""

public_key = RSA.importKey(binascii.unhexlify(transaction.sender))

verifier = PKCS1_v1_5.new(public_key)

h = SHA256.new((str(transaction.sender) + str(transaction.recipient) +
str(transaction.amount)).encode('utf8'))

return verifier.verify(h, binascii.unhexlify(transaction.signature))

File: hash_util.py

import hashlib as hl

import json

__all__ = ['hash_string_256', 'hash_block']

def hash_string_256(string):

"""Create a SHA256 hash for a given input string.

Arguments:

:string: The string which should be hashed.

"""

return hl.sha256(string).hexdigest()

```
def hash_block(block):
```

```
    """Hashes a block and returns a string representation of it.
```

```
    Arguments:
```

```
        :block: The block that should be hashed.
```

```
    """
```

```
    hashable_block = block.__dict__.copy()
```

```
    hashable_block['transactions'] = [
```

```
        tx.to_ordered_dict() for tx in hashable_block['transactions']
```

```
    ]
```

```
    return hash_string_256(json.dumps(hashable_block, sort_keys=True).encode())
```

File: printable.py

```
class Printable:
```

```
    """A base class which implements printing functionality."""
```

```
    def __repr__(self):
```

```
        return str(self.__dict__)
```

File: verification.py

```
"""Provides verification helper methods."""
```

```
from utility.hash_util import hash_string_256, hash_block
```

```
from wallet import Wallet
```

```
class Verification:
```

```
    """A helper class which offer various static and class-based verification
    and validation methods."""
```

```
    @staticmethod
```

```
    def valid_proof(transactions, last_hash, proof):
```

```

"""Validate a proof of work number and see if it solves the puzzle
algorithm (two leading 0s)

```

```

Arguments:

```

```

:transactions: The transactions of the block for which the proof
is created.

```

```

:last_hash: The previous block's hash which will be stored in the
current block.

```

```

:proof: The proof number we're testing.

```

```

"""

```

```

# Create a string with all the hash inputs

```

```

guess = (str([tx.to_ordered_dict() for tx in transactions]
            ) + str(last_hash) + str(proof)).encode()

```

```

# Hash the string

```

```

# IMPORTANT: This is NOT the same hash as will be stored in the
# previous_hash. It's a not a block's hash. It's only used for the
# proof-of-work algorithm.

```

```

guess_hash = hash_string_256(guess)

```

```

# Only a hash (which is based on the above inputs) which starts with
# two 0s is treated as valid

```

```

# This condition is of course defined by you. You could also require
# 10 leading 0s - this would take significantly longer (and this
# allows you to control the speed at which new blocks can be added)

```

```

return guess_hash[0:2] == '00'

```

```

@classmethod

```

```

def verify_chain(cls, blockchain):

```

```

    """ Verify the current blockchain and return True if it's valid, False
    otherwise. """

```

```

    for (index, block) in enumerate(blockchain):

```

```

        if index == 0:

```

```

            continue

```

```

    if block.previous_hash != hash_block(blockchain[index - 1]):
        return False
    if not cls.valid_proof(block.transactions[:-1],
                           block.previous_hash,
                           block.proof):
        print('Proof of work is invalid')
        return False
    return True

    @staticmethod
    def verify_transaction(transaction, get_balance, check_funds=True):
        """Verify a transaction by checking whether the sender has sufficient coins.

        Arguments:
            :transaction: The transaction that should be verified.
        """
        if check_funds:
            sender_balance = get_balance(transaction.sender)
            return (sender_balance >= transaction.amount and
                    Wallet.verify_transaction(transaction))
        else:
            return Wallet.verify_transaction(transaction)

    @classmethod
    def verify_transactions(cls, open_transactions, get_balance):
        """Verifies all open transactions."""
        return all([cls.verify_transaction(tx, get_balance, False)
                    for tx in open_transactions])

```

File : node.html

<!DOCTYPE html>

```
<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Blockchain Management</title>

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
integrity="sha384-
9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddlhOegiu1FwO5qRGvFXOdJZ4"
crossorigin="anonymous">

<style>

.lds-ring {

  display: inline-block;

  position: relative;

  width: 64px;

  height: 64px;

}

.lds-ring div {

  box-sizing: border-box;

  display: block;

  position: absolute;

  width: 51px;

  height: 51px;

  margin: 6px;

  border: 6px solid #fa923f;

  border-radius: 50%;

  animation: lds-ring 1.2s cubic-bezier(0.5, 0, 0.5, 1) infinite;

  border-color: #fa923f transparent transparent transparent;

}
```

```
.lds-ring div:nth-child(1) {  
  animation-delay: -0.45s;  
}  
  
.lds-ring div:nth-child(2) {  
  animation-delay: -0.3s;  
}  
  
.lds-ring div:nth-child(3) {  
  animation-delay: -0.15s;  
}  
  
@keyframes lds-ring {  
  0% {  
    transform: rotate(0deg);  
  }  
  100% {  
    transform: rotate(360deg);  
  }  
}  
</style>  
</head>  
  
<body>  
<div id="app">  
<div class="container">  
<div class="row mb-3">  
<div class="col">  
<h1>Manage your Blockchain</h1>  
</div>  
</div>  
<div class="row">
```

```

<div class="col">
  <ul class="nav nav-pills">
    <li class="nav-item">
      <a class="nav-link active" href="/">Wallet & Node</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/network">Network</a>
    </li>
  </ul>
</div>
</div>
<hr>
<div v-if="error" class="alert alert-danger" role="alert">
  {{ error }}
</div>
<div v-if="success" class="alert alert-success" role="alert">
  {{ success }}
</div>
<div class="row">
  <div class="col">
    <div v-if="!walletLoading">
      <button class="btn btn-primary" @click="onCreateWallet">
        Create new Wallet
      </button>
      <button class="btn btn-primary" @click="onLoadWallet">
        Load Wallet
      </button>
    </div>

    <div v-if="walletLoading" class="lds-ring">
      <div></div>
      <div></div>
    </div>
  </div>

```

```

</div></div>
</div></div>
</div>
</div>
<div class="col text-right">
<h2>CUB Coins: {{ funds.toFixed(2) }}</h2>
</div>
</div>
<hr>
<div v-if="!wallet" class="row">
<div class="col">
<div class="alert alert-warning">Create a Wallet to start sending coins or to mine
coins!</div>
</div>
</div>
<div v-if="wallet" class="row">
<div class="col">
<form @submit.prevent="onSendTx">
<div class="form-group">
<label for="recipient">Recipient Key</label>
<input v-model="outgoingTx.recipient" type="text" class="form-control" id="recipient"
placeholder="Enter key">
</div>
<div class="form-group">
<label for="amount">Amount of Coins</label>
<input v-model.number="outgoingTx.amount" type="number" step="0.001" class="form-
control" id="amount">
<small class="form-text text-muted">Fractions are possible (e.g. 5.67)</small>
</div>
<div v-if="txLoading" class="lds-ring">
<div></div>
<div></div>
<div></div>

```



```

</div></div>

</div>

<button :disabled="txLoading || outgoingTx.recipient.trim() === '' || outgoingTx.amount <=
0" type="submit" class="btn btn-primary">Send</button>

</form>

</div>

</div>

<hr>

<div class="row">

<div class="col">

<ul class="nav nav-tabs">

<li class="nav-item">

<a class="nav-link" :class="{ active: view === 'chain' }" href="#" @click="view =
'chain'">Blockchain</a>

</li>

<li class="nav-item">

<a class="nav-link" :class="{ active: view === 'tx' }" href="#" @click="view = 'tx'">Open
Transactions</a>

</li>

</ul>

</div>

</div>

<div class="row my-3">

<div class="col">

<button class="btn btn-primary" @click="onLoadData">{{ view === 'chain' ? 'Load
Blockchain' : 'Load Transactions' }}</button>

<button v-if="view === 'chain' && wallet" class="btn btn-success" @click="onMine">Mine
Coins</button>

<button class="btn btn-warning" @click="onResolve">Resolve Conflicts</button>

</div>

</div>

<div class="row">

<div class="col">

<div v-if="dataLoading" class="lds-ring">

```

```

</div></div>

</div></div>

</div></div>

</div>

<div v-if="!dataLoading" class="accordion">
  <div class="card" v-for="(data, index) in loadedData">
    <div v-if="view === 'chain'" class="card-header">
      <h5 class="mb-0">
        <button class="btn btn-link" type="button" @click="showElement === index ?
        showElement = null : showElement = index">
          Block #{{ data.index }}
        </button>
      </h5>
    </div>
    <div v-if="view === 'chain'" class="collapse" :class="{ show: showElement === index }">
      <div class="card-body">
        <p>Previous Hash: {{ data.previous_hash }}</p>
        <div class="list-group">
          <div v-for="tx in data.transactions" class="list-group-item flex-column align-items-start">
            <div>Sender: {{ tx.sender }}</div>
            <div>Recipient: {{ tx.recipient }}</div>
            <div>Amount: {{ tx.amount }}</div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

<div v-if="view === 'tx'" class="card-header">
  <h5 class="mb-0">
    <button class="btn btn-link" type="button" @click="showElement === index ?
    showElement = null : showElement = index">
      Transaction #{{ index }}

```

```

</button>

</h5>

</div>

<div v-if="view === 'tx'" class="collapse" :class="{ show: showElement === index}">
  <div class="card-body">
    <div class="list-group">
      <div class="list-group-item flex-column align-items-start">
        <div>Sender: {{ data.sender }}</div>
        <div>Recipient: {{ data.recipient }}</div>
        <div>Amount: {{ data.amount }}</div>
      </div>
    </div>
  </div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  new Vue({
    el: '#app',
    data: {
      blockchain: [],
      openTransactions: [],
      wallet: null,
      view: 'chain',
      walletLoading: false,

```

```
    txLoading: false,
    dataLoading: false,
    showElement: null,
    error: null,
    success: null,
    funds: 0,
    outgoingTx: {
      recipient: "",
      amount: 0
    }
  },
  computed: {
    loadedData: function () {
      if (this.view === 'chain') {
        return this.blockchain;
      } else {
        return this.openTransactions;
      }
    }
  },
  methods: {
    onCreateWallet: function () {
      // Send Http request to create a new wallet (and return keys)
      var vm = this;
      this.walletLoading = true
      axios.post('/wallet')
        .then(function (response) {
          vm.error = null;
          vm.success = 'Created Wallet! Public Key: ' + response.data.public_key + ',
Private Key: ' + response.data.private_key;
          vm.wallet = {
            public_key: response.data.public_key,
            private_key: response.data.private_key
          }
        })
    }
  }
}
```

```
    }

    vm.funds = response.data.funds;

    vm.walletLoading = false
  })
  .catch(function (error) {
    vm.success = null;

    vm.error = error.response.data.message

    vm.wallet = null

    vm.walletLoading = false

  });
},

onLoadWallet: function () {
  // Send Http request to load an existing wallet (from a file on the server)

  var vm = this;

  this.walletLoading = true

  axios.get('/wallet')

    .then(function (response) {
      vm.error = null;

      vm.success = 'Created Wallet! Public Key: ' + response.data.public_key + ',
Private Key: ' + response.data.private_key;

      vm.wallet = {
        public_key: response.data.public_key,
        private_key: response.data.private_key
      }

      vm.funds = response.data.funds;

      vm.walletLoading = false;

    })
    .catch(function (error) {
      vm.success = null;

      vm.error = error.response.data.message;

      vm.wallet = null;

      vm.walletLoading = false;

    });
},
```

```
    },  
    onSendTx: function () {  
        // Send Transaction to backend  
        this.txLoading = true;  
        var vm = this;  
        axios.post('/transaction', {  
            recipient: this.outgoingTx.recipient,  
            amount: this.outgoingTx.amount  
        })  
        .then(function(response) {  
            vm.error = null;  
            vm.success = response.data.message;  
            console.log(response.data);  
            vm.funds = response.data.funds;  
            vm.txLoading = false;  
        })  
        .catch(function (error) {  
            vm.success = null;  
            vm.error = error.response.data.message;  
            vm.txLoading = false;  
        });  
    },  
    onMine: function () {  
        var vm = this  
        axios.post('/mine')  
        .then(function(response) {  
            vm.error = null;  
            vm.success = response.data.message;  
            console.log(response.data);  
            vm.funds = response.data.funds;  
        })  
        .catch(function (error) {
```

```
        vm.success = null;

        vm.error = error.response.data.message;
    });
},
onResolve: function() {
    var vm = this
    axios.post('/resolve-conflicts')
        .then(function(response) {
            vm.error = null;
            vm.success = response.data.message;
        })
        .catch(function (error) {
            vm.success = null;
            vm.error = error.response.data.message;
        });
},
onLoadData: function () {
    if (this.view === 'chain') {
        // Load blockchain data
        var vm = this
        this.dataLoading = true
        axios.get('/chain')
            .then(function (response) {
                vm.blockchain = response.data
                vm.dataLoading = false
            })
            .catch(function (error) {
                vm.dataLoading = false
                vm.error = 'Something went wrong.'
            });
    } else {
        // Load transaction data
```

```
var vm = this
axios.get('/transactions')
  .then(function (response) {
    vm.openTransactions = response.data
    vm.dataLoading = false
  })
  .catch(function (error) {
    vm.dataLoading = false
    vm.error = 'Something went wrong.'
  });
}
}
}
})
</script>
</body>

</html>
```

File : network.html

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Blockchain Management</title>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
integrity="sha384-
9gVQ4dYFwwWSjIDZnLEWnCjeSWFphJiwGPXr1jddlhOegiu1FwO5qRGvFXOdJZ4"
```



```
        crossorigin="anonymous">
</head>

<body>
<div id="app">
<div class="container">
<div class="row mb-3">
<div class="col">
<h1>Manage your Blockchain</h1>
</div>
</div>
<div v-if="error" class="alert alert-danger" role="alert">
    {{ error }}
</div>
<div v-if="success" class="alert alert-success" role="alert">
    {{ success }}
</div>
<div class="row">
<div class="col">
<ul class="nav nav-pills">
<li class="nav-item">
<a class="nav-link" href="/">Wallet & Node</a>
</li>
<li class="nav-item">
<a class="nav-link active" href="/network">Network</a>
</li>
</ul>
</div>
</div>
<hr>
<div class="row">
<div class="col">
```

```

<form @submit.prevent="onAddNode">
  <div class="form-group">
    <label for="node-url">Node URL</label>
    <input v-model="newNodeUrl" type="text" class="form-control" id="node-url"
      placeholder="localhost:5001">
  </div>
  <button :disabled="newNodeUrl.trim() === ''" type="submit" class="btn btn-
    primary">Add</button>
</form>
</div>
</div>
<hr>
<div class="row my-3">
  <div class="col">
    <button class="btn btn-primary" @click="onLoadNodes">Load Peer Nodes</button>
  </div>
</div>
<div class="row">
  <div class="col">
    <ul class="list-group">
      <button v-for="node in nodes" style="cursor: pointer;" class="list-group-item list-group-
        item-action" @click="onRemoveNode(node)">
        {{ node }} (click to delete)
      </button>
    </ul>
  </div>
</div>
</div>
</div>
</div>
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>

```

```
new Vue({
  el: '#app',
  data: {
    nodes: [],
    newNodeUrl: "",
    error: null,
    success: null
  },
  methods: {
    onAddNode: function () {
      // Add node as peer node to local node server
      var vm = this;
      axios.post('/node', { node: this.newNodeUrl })
        .then(function (response) {
          vm.success = 'Stored node successfully.';
          vm.error = null;
          vm.nodes = response.data.all_nodes
        })
        .catch(function (error) {
          vm.success = null;
          vm.error = error.response.data.message;
        });
    },
    onLoadNodes: function () {
      // Load all peer nodes of the local node server
      var vm = this;
      axios.get('/nodes')
        .then(function (response) {
          vm.success = 'Fetched nodes successfully.';
          vm.error = null;
          vm.nodes = response.data.all_nodes
        })
    }
  }
});
```

```
        .catch(function (error) {  
            vm.success = null;  
            vm.error = error.response.data.message;  
        });  
    },  
    onRemoveNode: function (node_url) {  
        // Remove node as a peer node  
        var vm = this;  
        axios.delete('/node/' + node_url)  
            .then(function (response) {  
                vm.success = 'Deleted node successfully.';  
                vm.error = null;  
                vm.nodes = response.data.all_nodes  
            })  
            .catch(function (error) {  
                vm.success = null;  
                vm.error = error.response.data.message;  
            });  
    }  
}  
})  
</script>  
</body>  
  
</html>
```

5. SYSTEM OVERVIEW

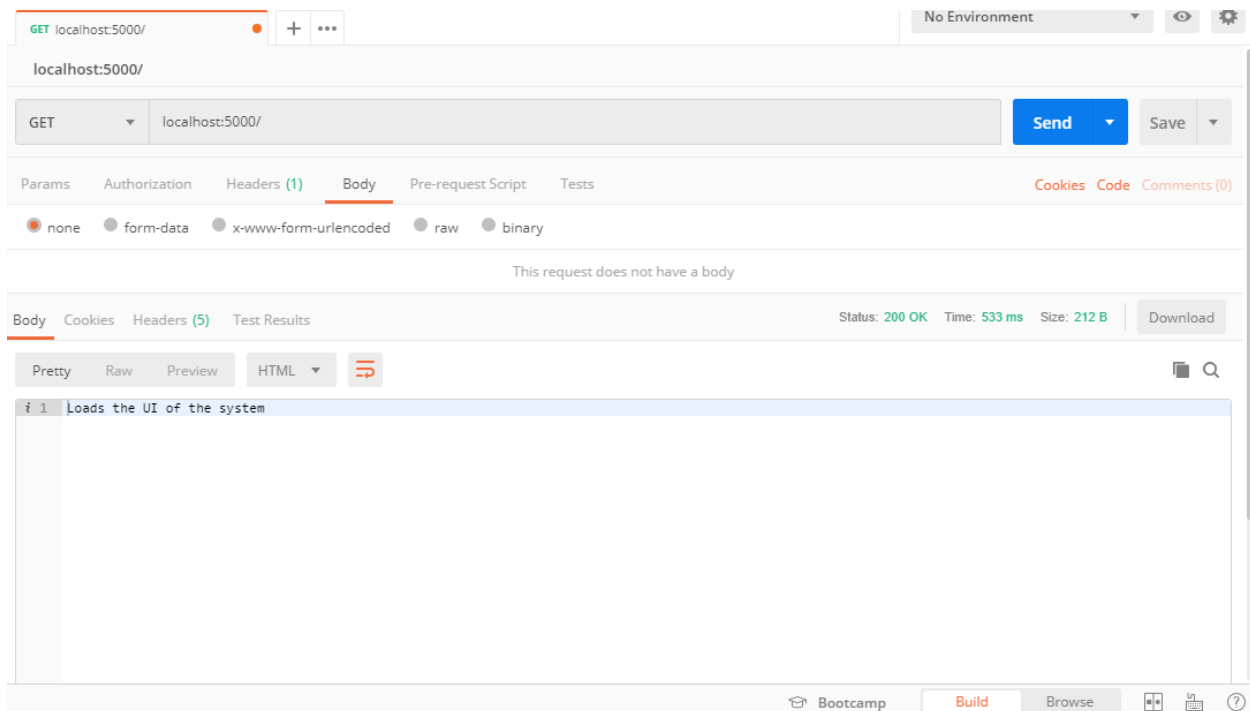


Figure 20: GET request to '/' will load the UI of the system for the user.

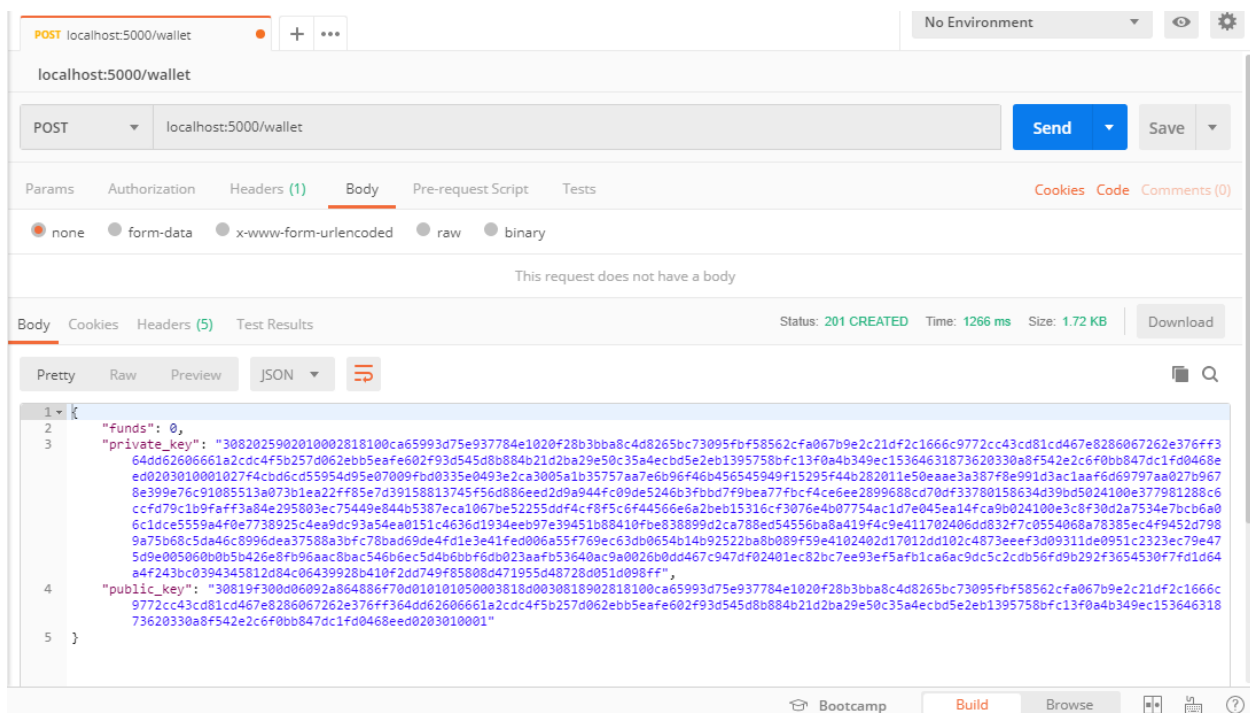


Figure 21: POST request to '/wallet' creates a new pair of keys and initiates funds to 0 and saves keys to a local txt file.

```

1  30819f300d06092a864886f70d0101050003818d0030818902818100ca65993d75e937784e1020f28b3bba8c4d8265bc73095fbf5
8562cfa067b9e2c21df2c1666c9772cc43cd81cd467e8286067262e376ff364dd62606661a2cdc4f5b257d062ebb5eafe602f93d545
d8b884b21d2ba29e50c35a4ecbd5e2eb1395758bfc13f0a4b349ec15364631873620330a8f542e2c6f0bb847dc1fd0468eed0203010001
001
2  3082025902010002818100ca65993d75e937784e1020f28b3bba8c4d8265bc73095fbf58562cfa067b9e2c21df2c1666c9772cc43cd
81cd467e8286067262e376ff364dd62606661a2cdc4f5b257d062ebb5eafe602f93d545d8b884b21d2ba29e50c35a4ecbd5e2eb1395
758bfc13f0a4b349ec15364631873620330a8f542e2c6f0bb847dc1fd0468eed0203010001027f4cbd6cd55954d95e07009fbd0335e
0493e2ca3005a1b35757aa7e6b96f46b456545949f15295f44b282011e50eaae3a387f8e991d3ac1aaf6d69797aa027b9678e399e76
c91085513a073b1ea22ff85e7d39158813745f56d886eed2d9a944fc09de5246b3fbbd7f9bea77fbcf4ce6ee2899688cd70df337801
58634d39bd5024100e377981288c6ccfd79c1b9faff3a84e295803ec75449e844b5387eca1067be52255ddf4cf8f5c6f44566e6a2be
b15316cf3076e4b07754ac1d7e045ea14fca9b024100e3c8f30d2a7534e7bcb6a06c1dce5559a4f0e7738925c4ea9dc93a54ea0151c
4636d1934eeb97e39451b88410f8e838899d2ca788ed54556ba8a419f4c9e411702406dd832f7c0554068a78385ec4f9452d7989a75
b68c5da46c8996dea37588a3bfc78bad69de4fd1e3e41fed006a55f769ec63db0654b14b92522ba8b089f59e4102402d17012dd102c
4873eeef3d09311de0951c2323ec79e475d9e005060b0b5b426e8fb96aac8bac546b6ec5d4b6bbf6db023aaf5b3640ac9a0026b0dd4
67c947df02401ec82bc7ee93ef5afb1ca6ac9dc5c2c2db56fd9b292f3654530f7fd1d64a4f243bc0394345812d84c06439928b410f2d
d749f85808d471955d48728d051d098ff

```

Figure 22: Public Key and Private Key stored in the local file

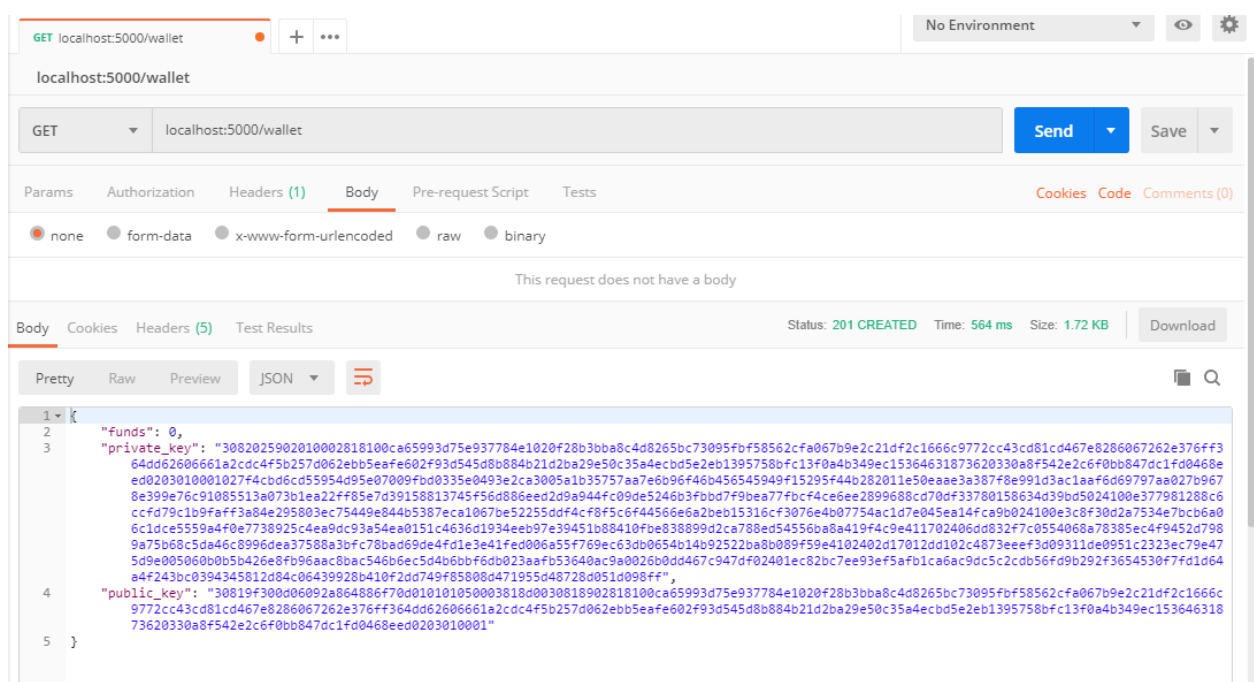


Figure 23: GET request to ‘/wallet’ loads the wallet from the existing local file in user’s system and fetches the balance corresponding to user’s private key

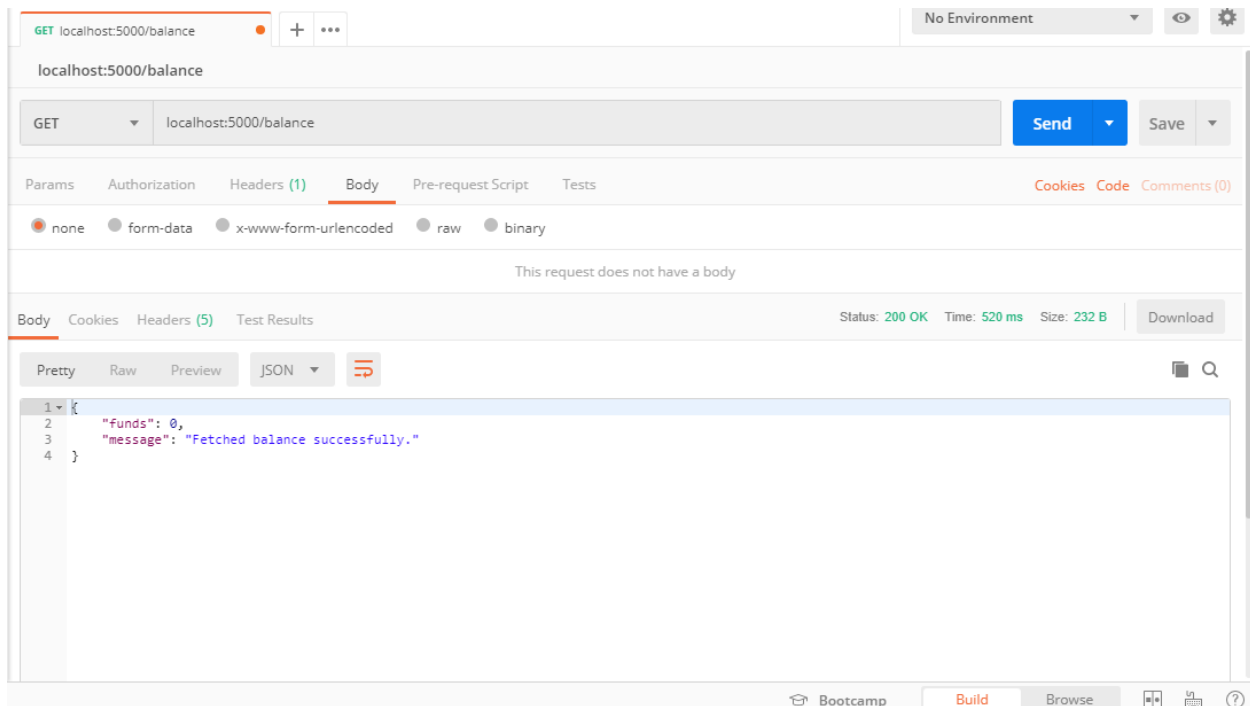


Figure 24: GET request to '/balance' would return the balance of the user.

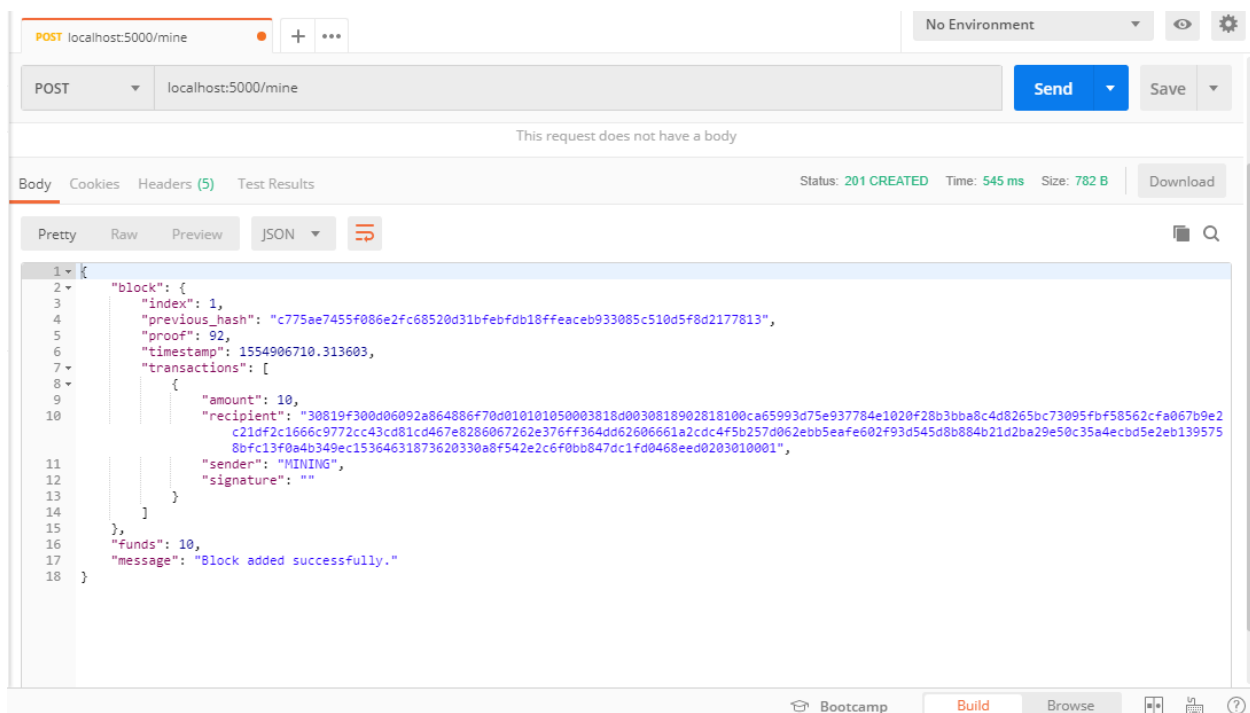


Figure 25: POST request to '/mine' would create a new block and confirm the 'open transactions' and put them into a new block

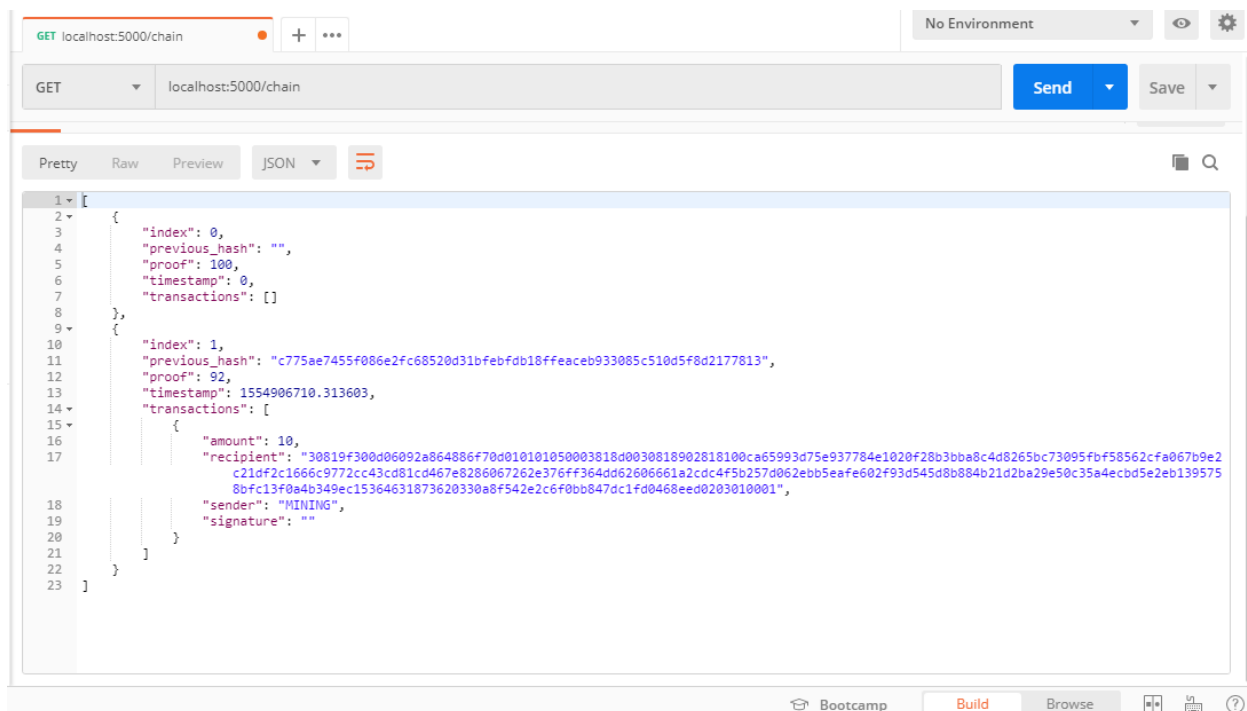


Figure 26: GET request to '/chain' checks the validity of chain and returns the entire chain

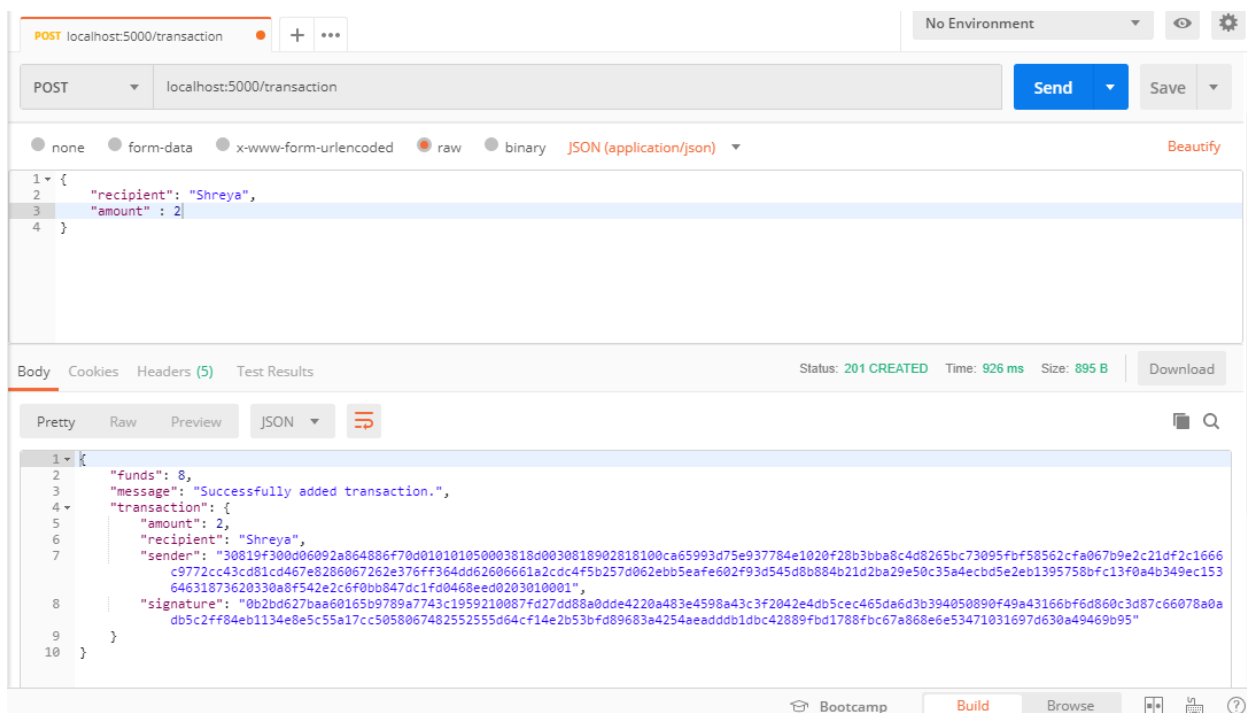


Figure 27: POST request to '/transaction' with the parameters 'recipient' and 'amount' would create a new open transaction and returns the new transaction.

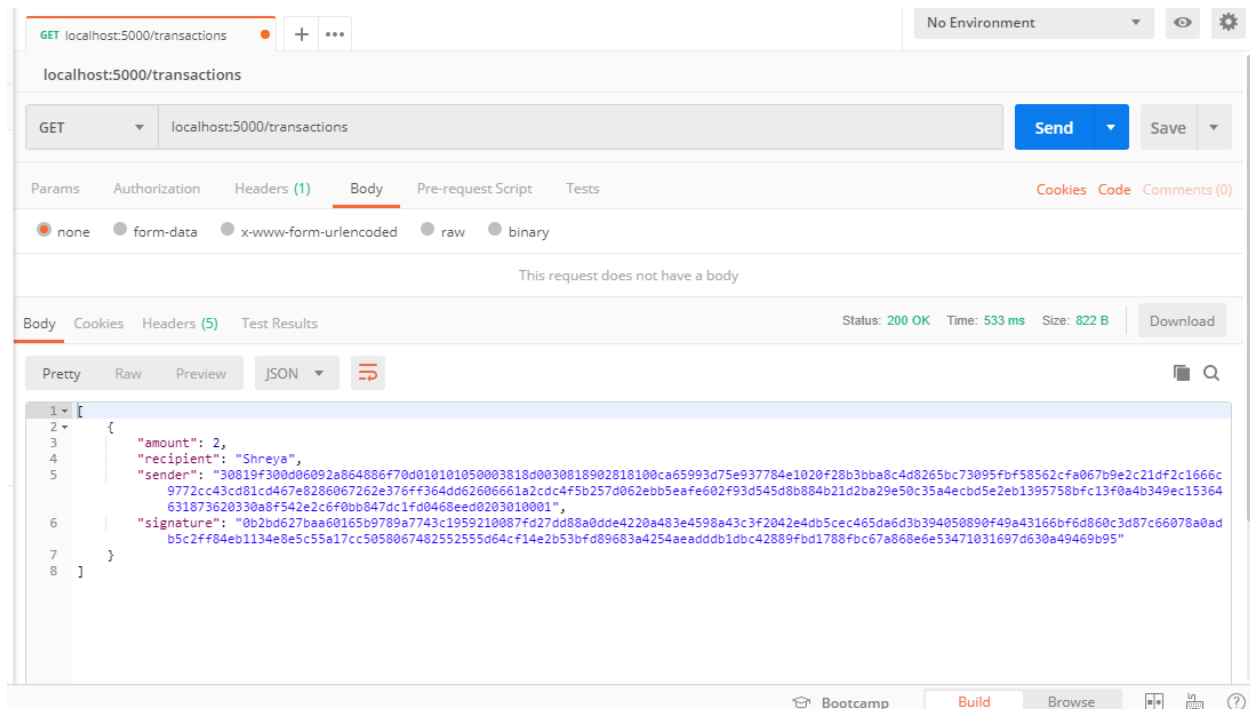


Figure 28: GET request to '/transactions' verifies and prints the list of 'open transactions'

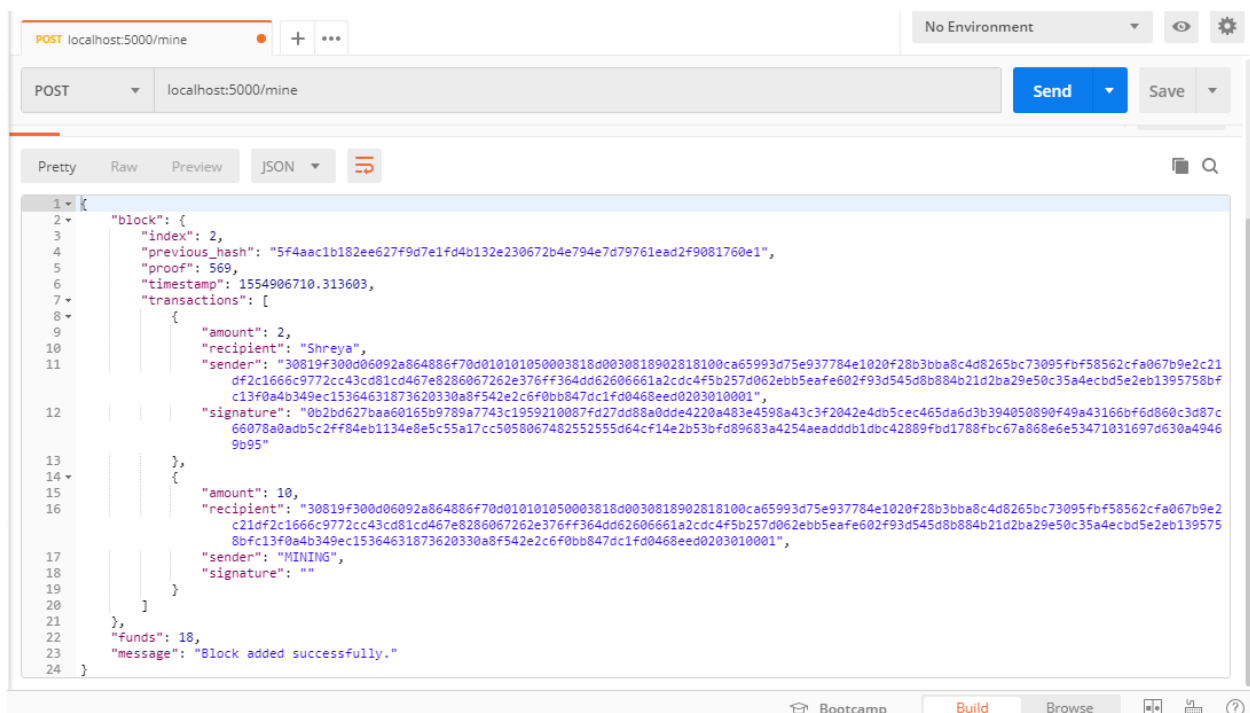


Figure 29: POST request to '/mine'

```

1 [{"index": 0, "previous_hash": "", "timestamp": 0, "transactions": [], "proof": 100}, {"index": 1,
  "previous_hash": "c775ae7455f086e2fc68520d31bfebfdb18fffaceb933085c510d5f8d2177813", "timestamp":
  1554906710.313603, "transactions": [{"sender": "MINING", "recipient":
  "30819f300d06092a864886f70d010101050003818d0030818902818100ca65993d75e937784e1020f28b3bba8c4d8265bc73095fbf
  58562cfa067b9e2c21df2c1666c9772cc43cd81cd467e8286067262e376ff364dd62606661a2cdc4f5b257d062ebb5eafe602f93d54
  5d8b884b21d2ba29e50c35a4ecbd5e2eb1395758bfc13f0a4b349ec15364631873620330a8f542e2c6f0bb847dc1fd0468eed020301
  0001", "amount": 10, "signature": ""}], "proof": 92}, {"index": 2, "previous_hash":
  "5f4aac1b182ee627f9d7e1fd4b132e230672b4e794e7d79761ead2f9081760e1", "timestamp": 1554906710.313603,
  "transactions": [{"sender":
  "30819f300d06092a864886f70d010101050003818d0030818902818100ca65993d75e937784e1020f28b3bba8c4d8265bc73095fbf
  58562cfa067b9e2c21df2c1666c9772cc43cd81cd467e8286067262e376ff364dd62606661a2cdc4f5b257d062ebb5eafe602f93d54
  5d8b884b21d2ba29e50c35a4ecbd5e2eb1395758bfc13f0a4b349ec15364631873620330a8f542e2c6f0bb847dc1fd0468eed020301
  0001", "recipient": "Shreya", "amount": 2, "signature":
  "0b2bd627baa60165b9789a7743c1959210087fd27dd88a0dde4220a483e4598a43c3f2042e4db5cec465da6d3b394050890f49a431
  66bf6d860c3d87c66078a0adb5c2ff84eb1134e8e5c55a17cc5058067482552555d64cf14e2b53bfd89683a4254aeaddb1dbc42889
  fbd1788fbc67a868e6e53471031697d630a49469b95"}, {"sender": "MINING", "recipient":
  "30819f300d06092a864886f70d010101050003818d0030818902818100ca65993d75e937784e1020f28b3bba8c4d8265bc73095fbf
  58562cfa067b9e2c21df2c1666c9772cc43cd81cd467e8286067262e376ff364dd62606661a2cdc4f5b257d062ebb5eafe602f93d54
  5d8b884b21d2ba29e50c35a4ecbd5e2eb1395758bfc13f0a4b349ec15364631873620330a8f542e2c6f0bb847dc1fd0468eed020301
  0001", "amount": 10, "signature": ""}], "proof": 569}]
2 []
3 []

```

Figure 30: The ‘blockchain’, ‘open transactions’ and ‘peer nodes’ are stored in a text file locally

5. TESTING

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is error free. It involves execution of a software component or system component to evaluate one or more properties of interest. Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements.

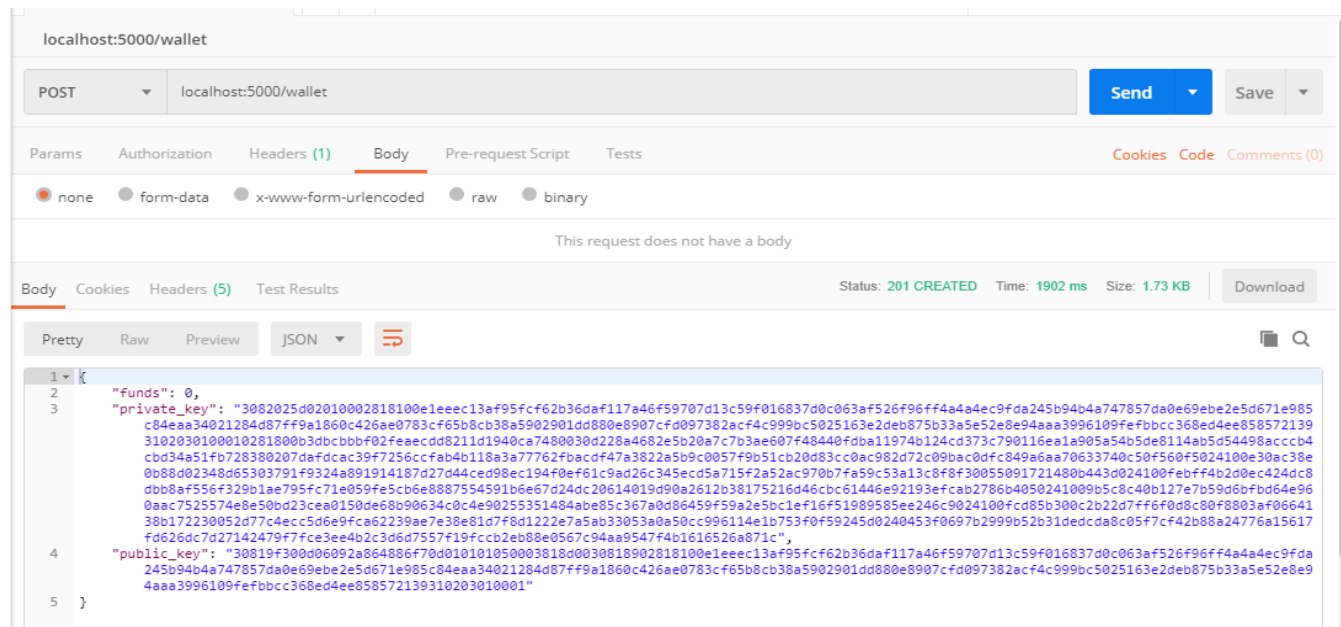


Figure 31: Loading a wallet, initially 0 funds present

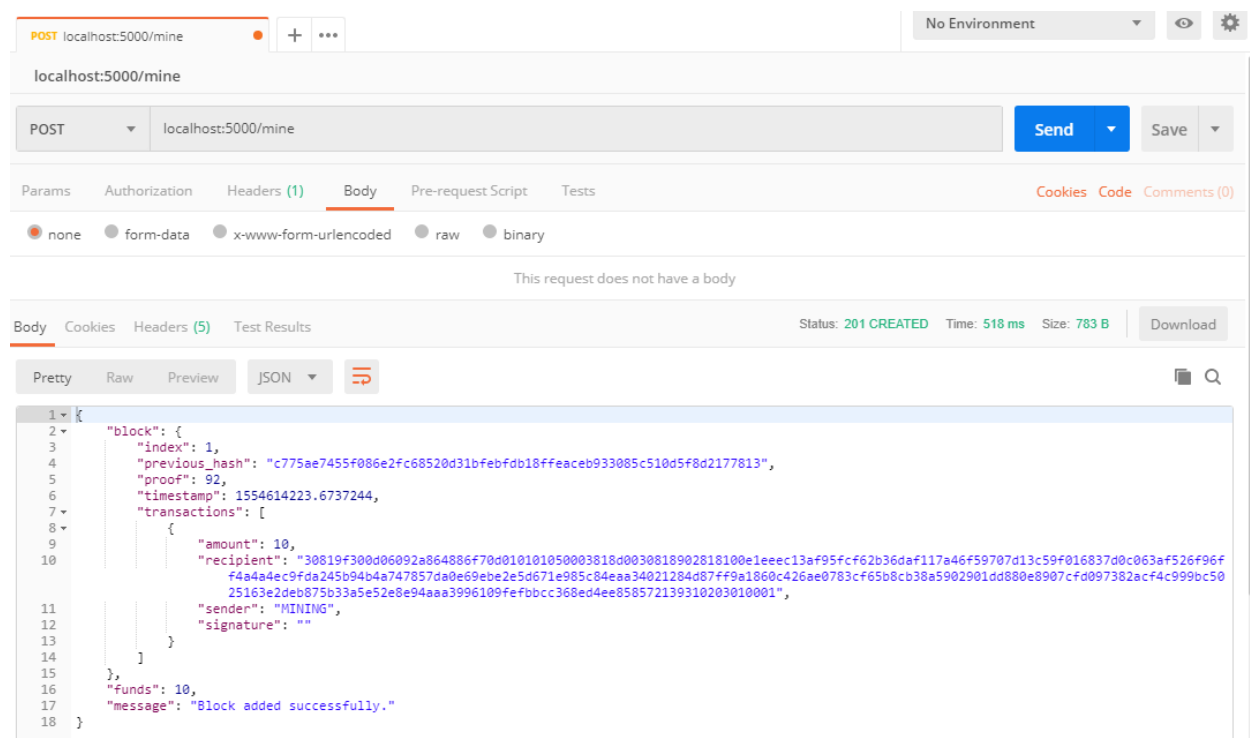



Figure 32: Mining a block to get mining rewards as the fund

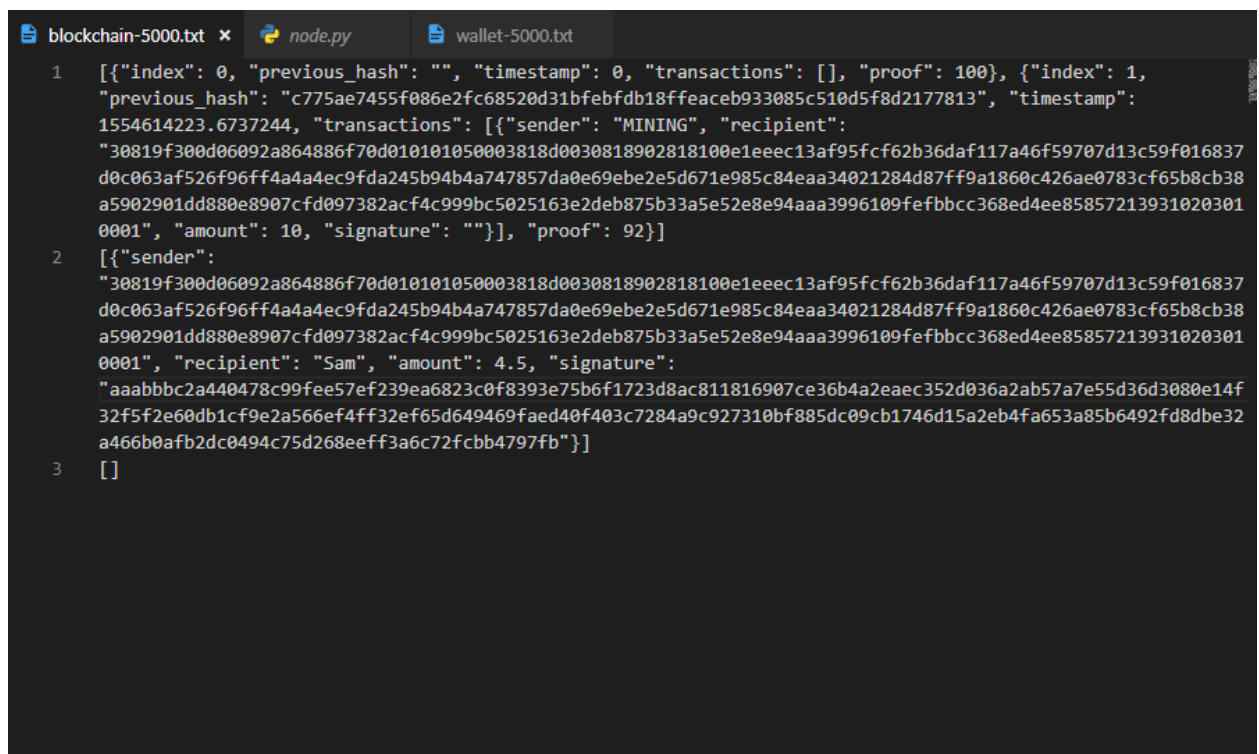


```

1 [{"index": 0, "previous_hash": "", "timestamp": 0, "transactions": [], "proof": 100}, {"index": 1,
  "previous_hash": "c775ae7455f086e2fc68520d31bfebfbdb18fffaceb933085c510d5f8d2177813", "timestamp":
  1554614223.6737244, "transactions": [{"sender": "MINING", "recipient":
  "30819f300d06092a864886f70d0101050003818d0030818902818100e1eeec13af95fcf62b36daf117a46f59707d13c59f016837
  d0c063af526f96ff4a4a4ec9fda245b94b4a747857da0e69ebe2e5d671e985c84eaa34021284d87ff9a1860c426ae0783cf65b8cb38
  a5902901dd880e8907cfd097382acf4c999bc5025163e2deb875b33a5e52e8e94aaa3996109fefbbcc368ed4ee85857213931020301
  0001", "amount": 10, "signature": ""}], "proof": 92}]
2 [{"sender":
  "30819f300d06092a864886f70d0101050003818d0030818902818100e1eeec13af95fcf62b36daf117a46f59707d13c59f016837
  d0c063af526f96ff4a4a4ec9fda245b94b4a747857da0e69ebe2e5d671e985c84eaa34021284d87ff9a1860c426ae0783cf65b8cb38
  a5902901dd880e8907cfd097382acf4c999bc5025163e2deb875b33a5e52e8e94aaa3996109fefbbcc368ed4ee85857213931020301
  0001", "recipient": "Sam", "amount": 4.5, "signature":
  "a99187f2a440478c99fee57ef239ea6823c0f8393e75b6f1723d8ac811816907ce36b4a2eaeac352d036a2ab57a7e55d36d3080e14f
  32f5f2e60db1cf9e2a566ef4ff32ef65d649469faed40f403c7284a9c927310bf885dc09cb1746d15a2eb4fa653a85b6492fd8dbe32
  a466b0afb2dc0494c75d268eeff3a6c72fcbb4797fb"}]
3 []

```

Figure 33: The original information that is stored in the file



```

1 [{"index": 0, "previous_hash": "", "timestamp": 0, "transactions": [], "proof": 100}, {"index": 1,
  "previous_hash": "c775ae7455f086e2fc68520d31bfebfbdb18fffaceb933085c510d5f8d2177813", "timestamp":
  1554614223.6737244, "transactions": [{"sender": "MINING", "recipient":
  "30819f300d06092a864886f70d0101050003818d0030818902818100e1eeec13af95fcf62b36daf117a46f59707d13c59f016837
  d0c063af526f96ff4a4a4ec9fda245b94b4a747857da0e69ebe2e5d671e985c84eaa34021284d87ff9a1860c426ae0783cf65b8cb38
  a5902901dd880e8907cfd097382acf4c999bc5025163e2deb875b33a5e52e8e94aaa3996109fefbbcc368ed4ee85857213931020301
  0001", "amount": 10, "signature": ""}], "proof": 92}]
2 [{"sender":
  "30819f300d06092a864886f70d0101050003818d0030818902818100e1eeec13af95fcf62b36daf117a46f59707d13c59f016837
  d0c063af526f96ff4a4a4ec9fda245b94b4a747857da0e69ebe2e5d671e985c84eaa34021284d87ff9a1860c426ae0783cf65b8cb38
  a5902901dd880e8907cfd097382acf4c999bc5025163e2deb875b33a5e52e8e94aaa3996109fefbbcc368ed4ee85857213931020301
  0001", "recipient": "Sam", "amount": 4.5, "signature":
  "aaabbbcb2a440478c99fee57ef239ea6823c0f8393e75b6f1723d8ac811816907ce36b4a2eaeac352d036a2ab57a7e55d36d3080e14f
  32f5f2e60db1cf9e2a566ef4ff32ef65d649469faed40f403c7284a9c927310bf885dc09cb1746d15a2eb4fa653a85b6492fd8dbe32
  a466b0afb2dc0494c75d268eeff3a6c72fcbb4797fb"}]
3 []

```

Figure 34: Manipulated the first seven bits of the signature in open transaction

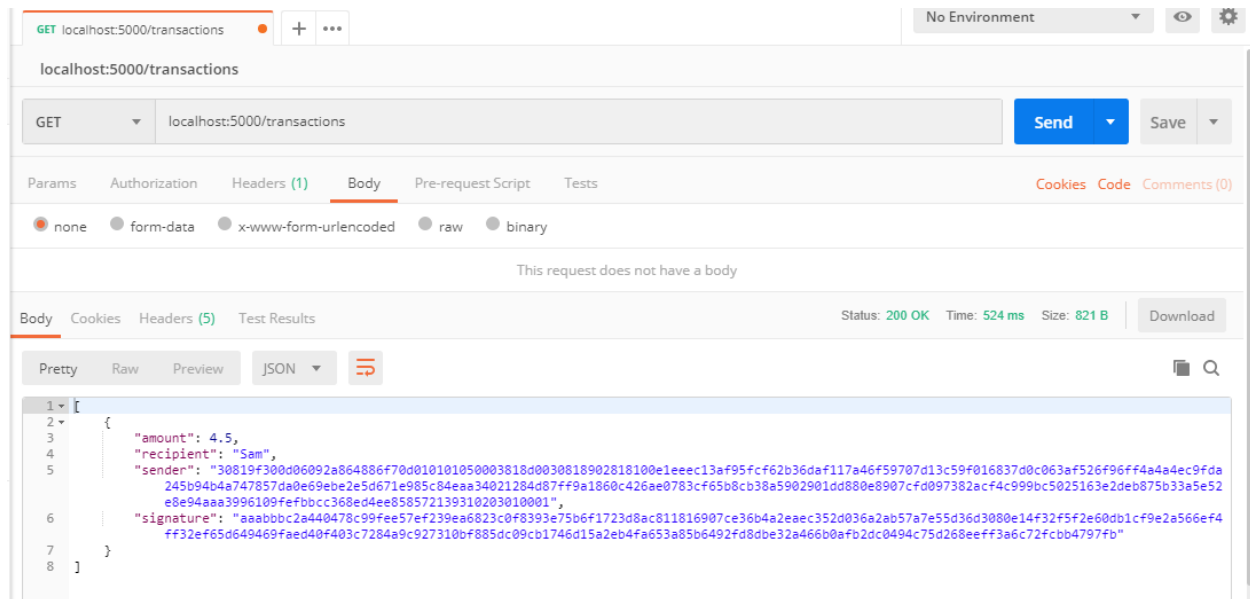


Figure 35: Confirming the manipulation by fetching the open transactions, the first 7 bits are manipulated in the signature, Manipulation Confirmed.

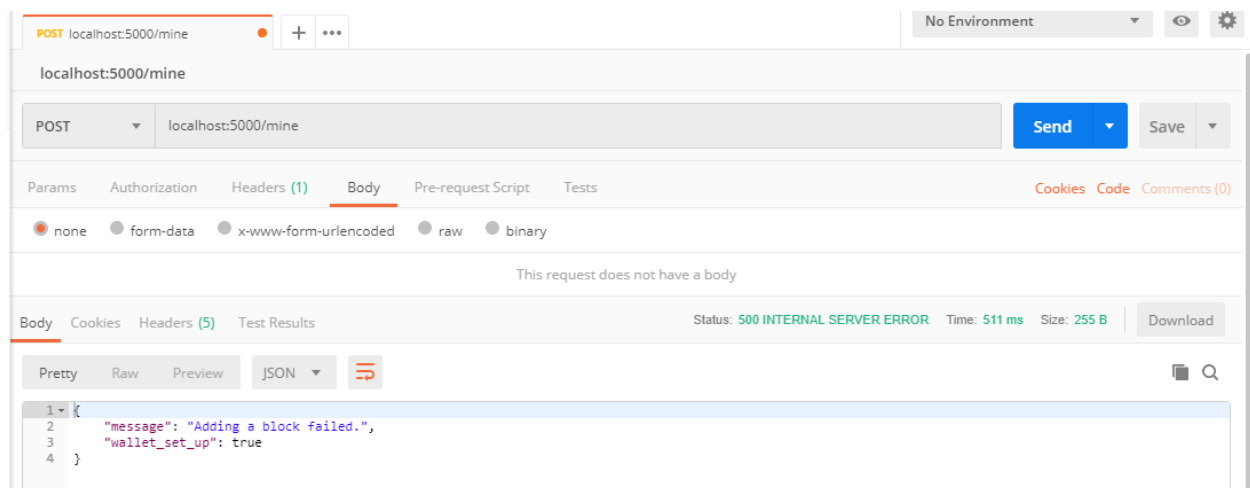


Figure 36: When we try to mine the manipulated transaction to a new block, our framework identifies the error and detects the invalid chain.

6. CONCLUSION

The blockchain framework is implemented using crypto currency transactions. This framework showcases features like creating wallet, mine block, view blockchain, add transaction etc. The distributed ledger functionality coupled with the blockchain security makes this framework potentially stronger to deal with financial and nonfinancial problems. A decentralized model using the concept of blockchain and cryptocurrency is successful demonstrated. Blockchain technology helps in creating a secure audit log that may not be tampered. It will give a proof of log manipulation and nonrepudiation.

7. REFERENCES

1. Michael Crosby, Nachiappan, Pradan Pattnayak, Sanjeev Verma, Vignesh Kalyanaraman, "Blockchain Technology: Beyond Bitcoin", Available at-<https://j2-capital.com/wp-content/uploads/2017/11/AIR-2016-Blockchain.pdf>
2. F. Xavier Olleros, Majilinda Zhegu, "Research Handbook on Digital Transformations", Edward Elgar Publishing
3. Available at- <https://pycryptodome.readthedocs.io/en/latest/src/installation.html>
4. Aaron Wright, Primavera De Filippi, "Decentralised Blockchain Technology and the Rise of Lex Cryptographia", Available at-
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2580664
5. Marcella Atzori, "Blockchain Technology and Decentralised Governance: Is the State still Necessary?", Available at-
<https://poseidon01.ssrn.com/delivery.php?ID=646117100022001071125112104120096104118081007014064089086094006069118088081027097108050122101118013124055088102112074099084101023074053015000069099068018102101019076058082010064108001127016107094107064099069095072099073023107114000065020018108021125082&EXT=pdf>
6. Tomaso Aste, Paolo Tasca, T Di Matteo, "Blockchain Technologies: foreseeable impact on industry and society", Available at-
http://discovery.ucl.ac.uk/10043048/1/Aste_BlockchainIEEE_600W_v3.3_A.doccceptedVersion.x.pdf