



**VIT<sup>®</sup>**  

---

**UNIVERSITY**  
(Estd. u/s 3 of UGC Act 1956)

## **BKT4006-Cryptocurrency Technologies J-Component Project**

Team Members:

20BCE2771 : AYUSH GIRI

20BCE2766 : ADITYA SAPKOTA

20BCE2922 : SARJAK DEVKOTA

20BCE2779 : BAIBHAV RIJAL

20BCE2763 : BIGGYAT PANDEY

Submitted to : Prof. Gopichand G.

# Project: PYCHAIN:OUR OWN MINEABLE CRYPTOCURRENCY.

**Softwares and Languages used:** **Backend:** Python, Flask, Pub/Sub API, Hashing Libraries in Python, **Frontend:** JavaScript , React

## Documentation Contents:

1. Abstract
2. Achievements on the project
3. Proof of Work and 51% Attack Prevention in Our Blockchain System.
4. Preparing the Blockchain for Collaboration
5. The Blockchain Network - Flask API and PUB/SUB
6. The Cryptocurrency-Wallets, Keys and Transactions.
7. Transactions on the Network
8. Connect the Blockchain and Cryptocurrency
9. Frontend Blockchain
10. Frontend Cryptocurrency
11. Implementation
12. Screenshots of Code and Github Link:

## ABSTRACT:

Blockchain is a revolutionary technology that allows people to record transactions on a digital, decentralized, distributed ledger also focusing on decentralized transaction and data management. Some consider this technology as “the trust machine” as it lets people who have no particular confidence in each other collaborate without having to go through a neutral central authority and cryptocurrency is without doubt the most notable by-product of the blockchain revolution. Blockchains and cryptocurrencies are now topics of substantial impact that academia, practitioners and the IT industry need to contemplate, study, research, publish, innovate, exploit and adopt. Blockchain is the technological weapon-of-choice behind the success of Bitcoin and other cryptocurrencies.

## About CryptoCurrencies

Cryptocurrency, an encrypted, peer-to-peer network for facilitating digital barter, is a technology developed thirteen years ago.

Bitcoin, the first and most popular cryptocurrency, is paving the way as a disruptive technology to long standing and unchanged financial payment systems that have been in place for many decades. While cryptocurrencies are not likely to replace traditional fiat currency, they could change the way Internet-connected global markets interact with each other, clearing away barriers surrounding normative national currencies and exchange rates.

To Establish a Peer- to – Peer network (blockchain) and Create our own

CryptoCurrency Coders generally use Structural Programming Language I.e C++, Java , Rust etc . However, In this Project we are aiming to code an entire blockchain and CryptoCurrency using a more dynamical programming language I.e Python

## ACHIEVEMENTS:

1. BUILT A BLOCKCHAIN
2. PROPER HASHING FUNCTIONS WERE ALSO ESTABLISHED.
3. ESTABLISHED A BLOCKCHAIN WITH A MINING MECHANISM
4. CREATED A MINE ABLE ENVIRONMENT
5. ESTABLISHED PROOF OF WORK
6. SETUP AN API TO MINE THE BLOCKCHAIN
7. SEE THE NUMBER OF BLOCKS CREATED
8. MINE THE BLOCK IN THE LOCAL HOST ENVIRONMENT
9. ESTABLISHED CRYPTOCURRENCY WALLET WHICH IS PERSONAL AND PRIVATE TO THE OWNER THAT CAN SHOW ALL TRANSACTION RELATED DETAILS.

## **Proof Of Work and 51% Attack and its prevention in our Blockchain**

- Proof of Work systems is a computational system that requires miners to spend some computational power to mine a block rather than allowing anyone to mine a block. A user has to spend some CPU time and Electricity Cost In order to mine a block into a Blockchain.
- A malicious individual cannot spend so much computational power to rewrite the entire blockchain thus making it tamper free.
- Many Cryptocurrencies like BitCoin use PoW systems to prevent Malicious Attack

### **MINING RATE:**

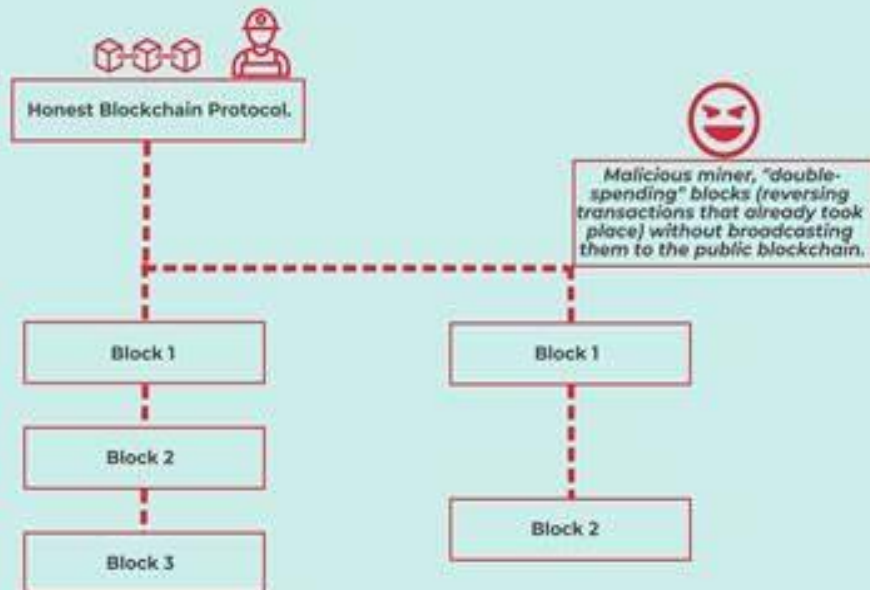
- Blockchain can also control the overall mining rate of the System. For Example: If it's taking a short time for miners to add a new block we will increase the level of difficulty . If its taking a long time for miners to add a new block we will decrease the difficulty.
- By this we can bring the average rate to which the blocks are being added into a rate that is closer to the set amount of Mining Rate i.e. 4 seconds in case of our blockchain.
- This prevents a malicious miner from mining 51% of the blockchain and others will get 49% which allows the malicious miner to get an advantage in the distributed consensus method.

## Level of Difficulty:

- When Miners try to add new Block they will have to find the hash value of the block that matches the level of difficulty.
- Miners will then have to match the number of leading zeros in the Hash value to be able to add it to the new block.
- The idea is to generate new hashes for the same block based of block data as well as adjusting a value known as nonce.
- Also our Hash code is generated by the SHA-256 algorithm which makes the generated Block Cryptographically Secure.
- Once a Block is mined successfully the miner has to submit the nonce last-hash value and difficulty to other blocks to synchronize it amongst peer

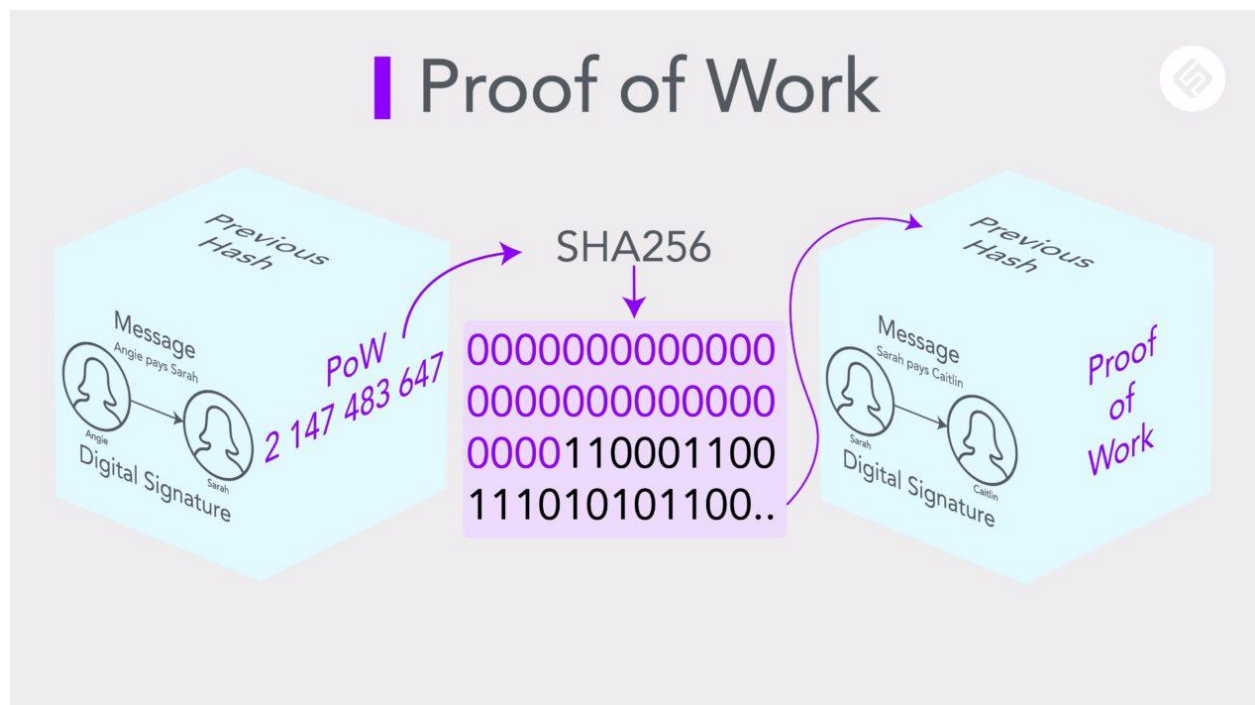
## 51% Attack In A Nutshell

A 51% Attack is an attack on the blockchain network by an entity or organization. The primary goal of such an attack is the exclusion or modification of blockchain transactions. A 51% attack is carried out by a miner or group of miners endeavoring to control more than half of a network's mining power, hash rate, or computing power. For this reason, it is sometimes called a majority attack. This can corrupt a blockchain protocol that malicious attackers would take over.



## Proof of work

It is a mechanism that requires miners to solve a computational puzzle in order to create valid blocks. Solving the puzzle requires a brute-force algorithm that demands CPU power (hence, the analogy of mining the block).



## The leading o's requirement

It is the standard proof of work implementation for finding valid blocks. By adjusting a nonce value in the block, there is an infinite number of tries at generating new hashes. Once it finds a hash with a matching number of leading 0's as the block difficulty, the fields for valid new block have been found.



## Dynamic difficulty

It is a mechanism that increases or decreases the difficulty of the next block based on how long it's taking to mine the new block. If the time is exceeding an established mining rate for the system, the difficulty decreases. Likewise, if the time is still before the mine rate, the difficulty increases. This allows the blockchain to control the rate at which blocks are added. By converting the hashes from their default hexadecimal form to a binary form, a more precise leading 0's requirement can be implemented.

With proof of work done, the core of the blockchain data structure itself has been completed. But the true power of a blockchain system kicks in once there are multiple contributors to the blockchain's growth. Likewise, the blockchain becomes secure once there are multiple validators to check the healthiness of new block data. So now we focus on functionality like the validation and replacement of chains in the next section.

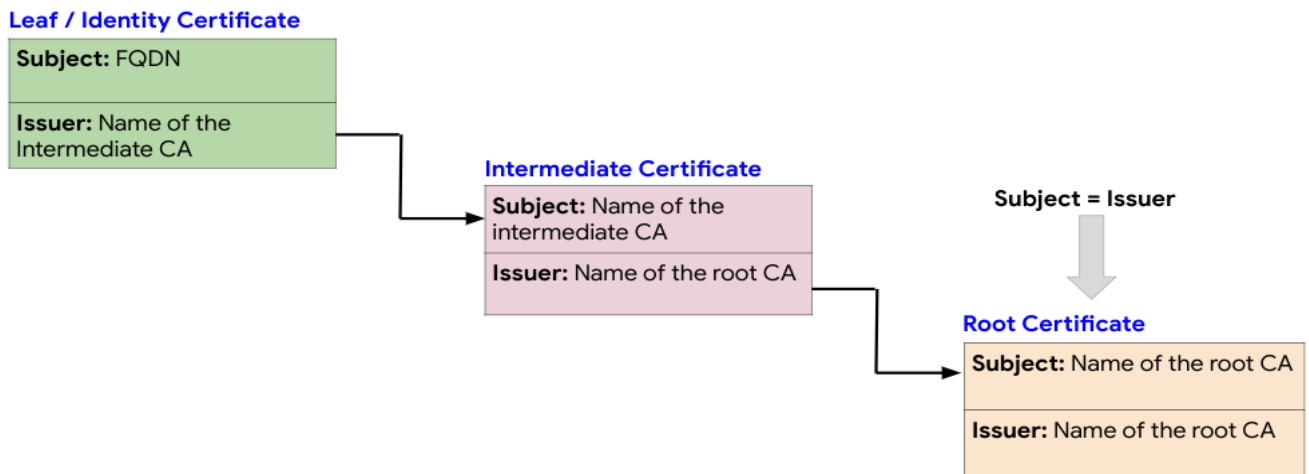
## BLOCKCHAIN COLLABORATION

### Chain validation

This process ensures that the data of an external blockchain is formatted correctly. For the blockchain to be valid, there are multiple rules to enforce. For starters, every block must be valid, with a proper hash based on the block fields, correctly adjusted difficulty, acceptable number of leading 0's in the hash for the proof of work

requirement, and more. Likewise, in the blockchain itself must start with the genesis block, and every block's last hash must reference the hash of the block that came before it.

## Certificate Chain



## Chain replacement

We use the process of substituting the current blockchain data for the data of an incoming blockchain. If the incoming blockchain is longer, and valid, then it should replace the current blockchain. This will allow a valid blockchain, with new blocks, to spread across the eventual blockchain network, becoming the true blockchain that all nodes in the blockchain network agree upon.

With chain validation and replacement completed, we've now prepared the blockchain for collaboration. So, in the next section, the creation of the blockchain network that participants of the system will use to collaborate is going to be processed

## The blockchain network-Flask API and Pub/Sub

An **API**, or Application Programming Interface is a medium that allows external parties to call code within an existing system.

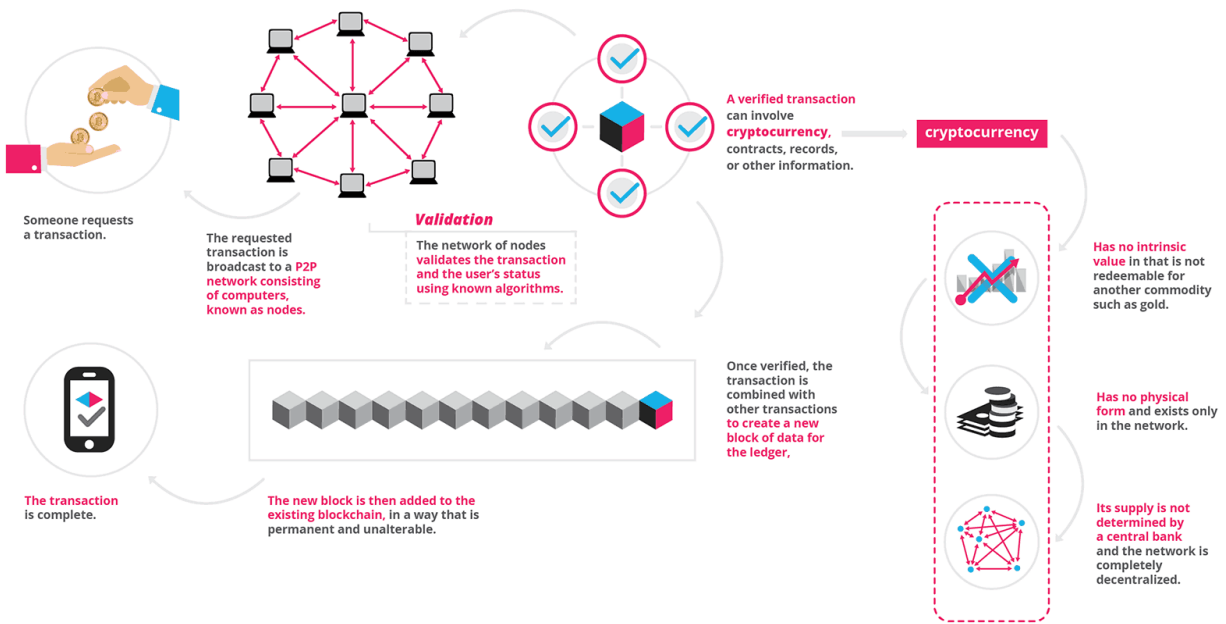
**HTTP** stands for hypertext transfer protocol - and http allows us here to fetch resources over the web. For example, a GET http request is associated with reading data from an API. A POST http request is associated with sending data to the API.

**Flask** is used here as a python module that helps build web servers.

**JSON**, which stands for JavaScript Object Notation, is a format for structuring data that is being used for sharing objects over the web. Even though the name includes JavaScript, it's actually supports multiple languages. This makes it a great format to use for sending objects across applications - for example, from a backend server, to a frontend web application.

**The publisher/subscriber pattern**, or pub/sub for short, is a networking pattern that exposes various communication channels. Publishers broadcast messages on those channels. And subscribers receive messages on those channels.

**Serialization** is the process in which we are converting a complex custom object into a simpler format that can be shared across the web, or perhaps stored in a database more easily. Most often, the end result of serialization is a string representation of the original object. Deserialization would then take the string representation, and convert it back to the complex custom object.



With the execution of the blockchain portion of the backend, we work on the other bigger aspect of the backend: the cryptocurrency itself and exploring wallets, keys, transactions, and all the components of a cryptocurrency

## The Cryptocurrency: Wallets, Keys, and Transaction.

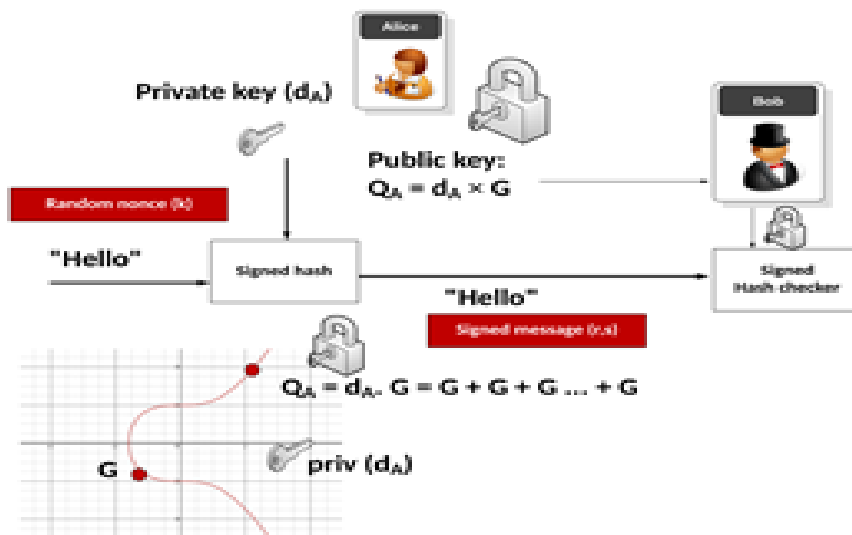
In this section we have implemented and included the following-

- A wallet keeps track of an individual's amount of currency. Each wallet has an address, and a pair of keys (a private and a public key).
- The private key of a wallet must be kept secret. It's used to generate signatures (see signatures below) on behalf of the wallet owner for based on objects of data. For example, a wallet owner will sign a generated transaction to make it official.



- The public key is the other half of the key pair, and it can be publicly shared with other entities.
- Signatures are unique data objects created using the private key of the key pair and an original data object to sign. With the signature, public key, and the original data object, other entities can verify if the signature was generated by the true owner of the public key.

- ECDSA stands for elliptic cryptography digital signature algorithm, and it's the underlying implementation of the cryptography python module used in the project. The mathematics behind the system use elliptic curves to create key pairs and signatures.



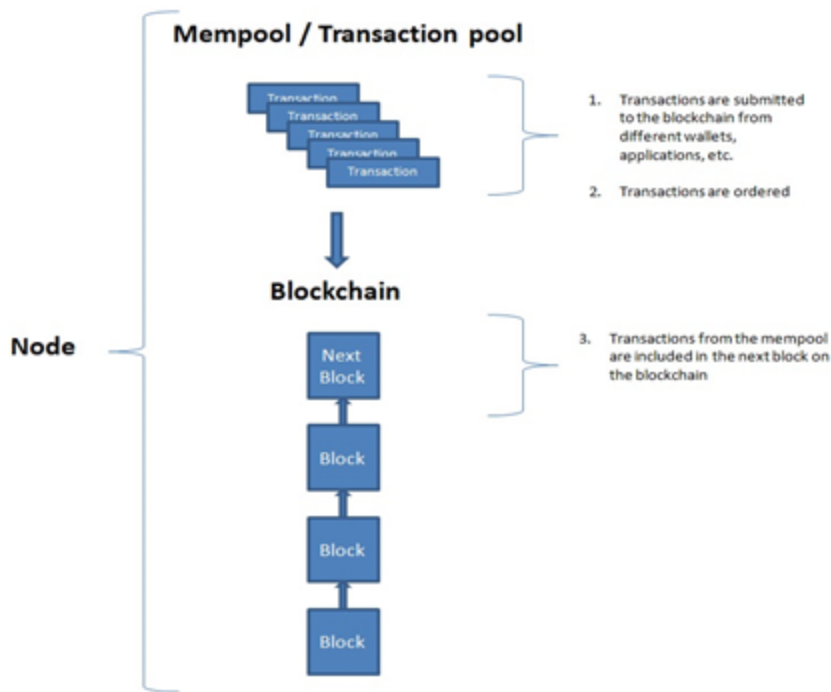
- A transaction consists of an input and output field. The input contains metadata, including the address, public key, and the balance amount of the sender. The input also includes a signature that's generated by the sender, using the transaction output as the underlying data. The output contains a series of entries where recipient addresses will receive certain amounts as a result of the transaction. The transaction can have any number of recipients. At least one of the recipients is the sender address itself, because this details how much currency the sender should have after the transaction is completed.

- Validating transactions involves checking that the total currency sent to the recipients is correct, and that the signature is correct according to the presented public key and transaction output.

### Transactions on the Network.

- The new /wallet/transact endpoint is the first POST request of the API. A POST request allows the requester to send data to the application, for methods that usually create new objects.
- Serializing the wallet's public key took a more complex approach since the public key byte string is not in the utf-8 encoding. Instead, the public key's default format is in an encoding format called PEM, that's defined within the cryptography module. But by encoding and decoding the public key using this format, a public key can be shared across the network, and restored back into a rich public key object when it comes to validating signatures.
- The transaction pool is an object that collects transactions that have been broadcasted across the network. It stores transactions according to their id. The idea is that this transaction pool will collect the transactions that miners will use as the basis of data for new blocks.





- Truthy values in Python act like True when placed in a Boolean context. Any value that is not falsy (values that like False in a Boolean context) is truthy. So it's important to know what values are falsy: None, 0, the empty string, or any empty collection, like an empty list or dictionary.

## Connect the Blockchain and Cryptocurrency

- By mining transactions into the blockchain, they become official in the cryptocurrency. This means that they can be used to affect wallet balances.
- When a new block of transactions is added to the blockchain, those transactions should be removed from the transaction pool to prevent duplicate transactions from appearing in newly attempted blocks.

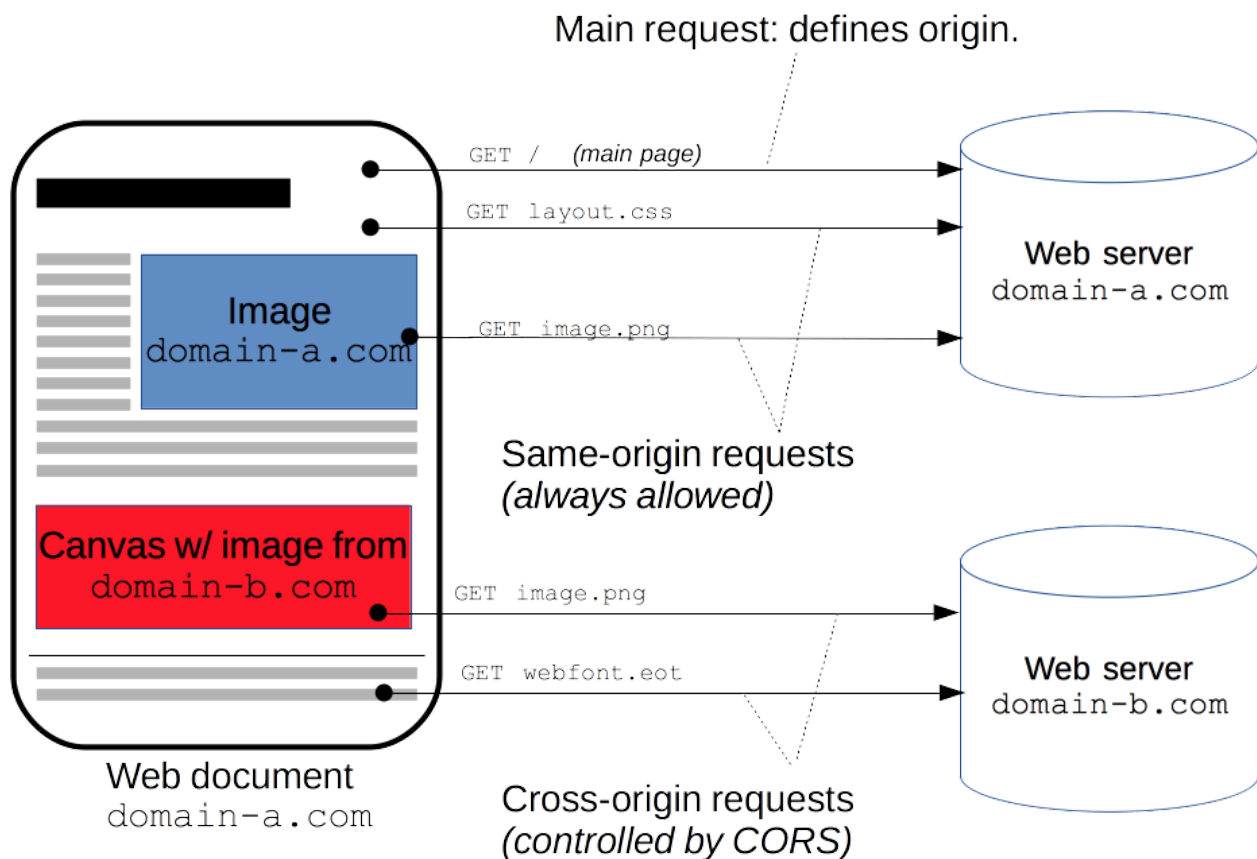
- To calculate a balance, you sum the output values sent to a given address since that wallet's most recent transaction. Every time a wallet conducts a new transaction, they create a new "starting point" for the algorithm to calculate a balance.
- The mining reward, a transaction with a standard amount of cryptocurrency, incentivizes miners to continue investing CPU power to mine new blocks. That way, the cryptocurrency's backing blockchain will grow at a steady pace and continue to make transactions official.

There's a handful of rules to enforce when it comes to a valid blockchain of transactions:

- The blockchain data structure itself must be valid (see `is_valid_chain`).
- There can only be one mining reward per block.
- Transactions cannot appear twice in the blockchain.
- Every transaction input amount must reflect the accurate balance of the wallet at the time the transaction was created.
- Each transaction must be valid (see `is_valid_transaction`).

## Section Summary (14)

In our frontend we are using **CORS(Cross Origin Resource Sharing)** Policy which is a security check that the browser implements to ensure that websites are making authorized requests to backend servers. The browser confirms that the backend has recognized the frontend url as an origin. To recognize the frontend url as an origin, the server decorates each response with an "Allow-Control-Allow-Origin" header that will contain the frontend's url.



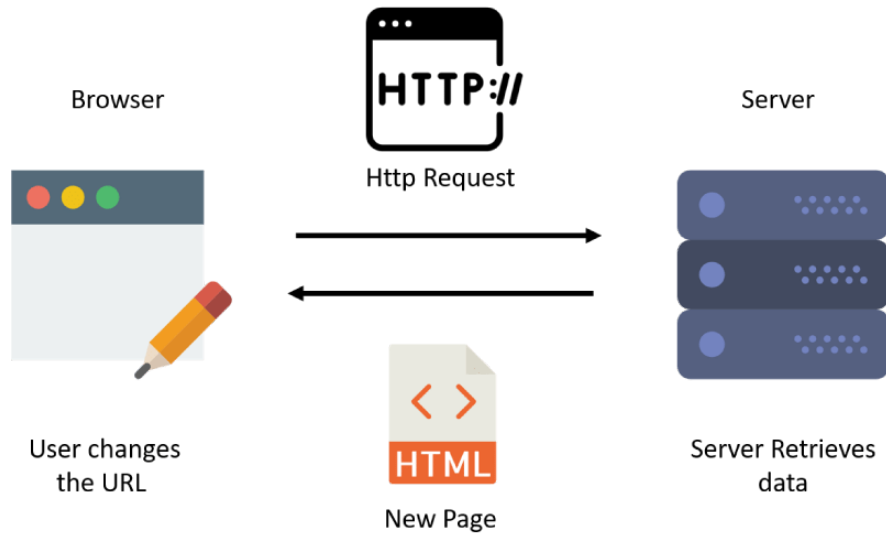
The properties are a way that parent React components can pass data down to child components also in the frontend

section, the approach we use to display long lists of data in our frontend web application is called **pagination**. The reason behind using pagination is that instead of attempting to present the entire list, the frontend allows the user to navigate through the list through slices.

## **Section Summary (15)**

Here we are using a library called **React Router** that allows a React application to have routes. Routes are endpoints on the frontend application that contain their own pages of functionality. They're accessible through paths, like /blockchain, or /transaction-pool (at the end of the url).

The illustration gives us an idea about how React Router works:



The **history** object, instantiated from a `createBrowserHistory` function from the `history` module, in this section allows the React application to programmatically navigate the user to a new path. There are multiple methods to control where the user should go throughout the lifetime of the application. Mainly, we're using the `push` method to send the user to a different path, based on completing certain actions (like going to the transaction-pool page after conducting a transaction successfully).

# IMPLEMENTATION:

## Running Our Application:

### 1. Testing Backend Servers in Python Terminal:

```
(blockchain-env) pranjal@PT:/mnt/d/Giri/python-blockchain$ python3 -m pytest backend/tests
===== test session starts =====
platform linux -- Python 3.8.10, pytest-5.1.2, py-1.11.0, pluggy-0.13.1
rootdir: /mnt/d/Giri/python-blockchain
collected 40 items

backend/tests/blockchain/test_block.py ..... [ 25%]
backend/tests/blockchain/test_blockchain.py ..... [ 55%]
backend/tests/util/test_crypto_hash.py . [ 57%]
backend/tests/util/test_hex_to_binary.py . [ 60%]
backend/tests/wallet/test_transaction.py ..... [ 87%]
backend/tests/wallet/test_transaction_pool.py .. [ 92%]
backend/tests/wallet/test_wallet.py ... [100%]

===== 40 passed in 9.88s =====
```

### 2. Running and Starting our Backend Server.

```
(blockchain-env) pranjal@PT:/mnt/d/Giri/python-blockchain$ python3 -m backend.app
* Serving Flask app "backend.app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

### 3. Running Peer Instance Server in Backend API.

```
(blockchain-env) pranjal@PT:/mnt/d/Giri/python-blockchain$ export PEER=True && python3 -m backend.app

-- Error synchronizing: Cannot replace. The incoming chain must be longer.
* Serving Flask app "backend.app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5353/ (Press CTRL+C to quit)
```

#### 4. Firing Up our Frontend Application using NPM .

```
PS D:\Gini\python-blockchain\frontend> npm start

> frontend@0.1.0 start
> react-scripts start
Starting the development server...

Browserslist: caniuse-lite is outdated. Please run next command `npm update`
Compiled successfully!

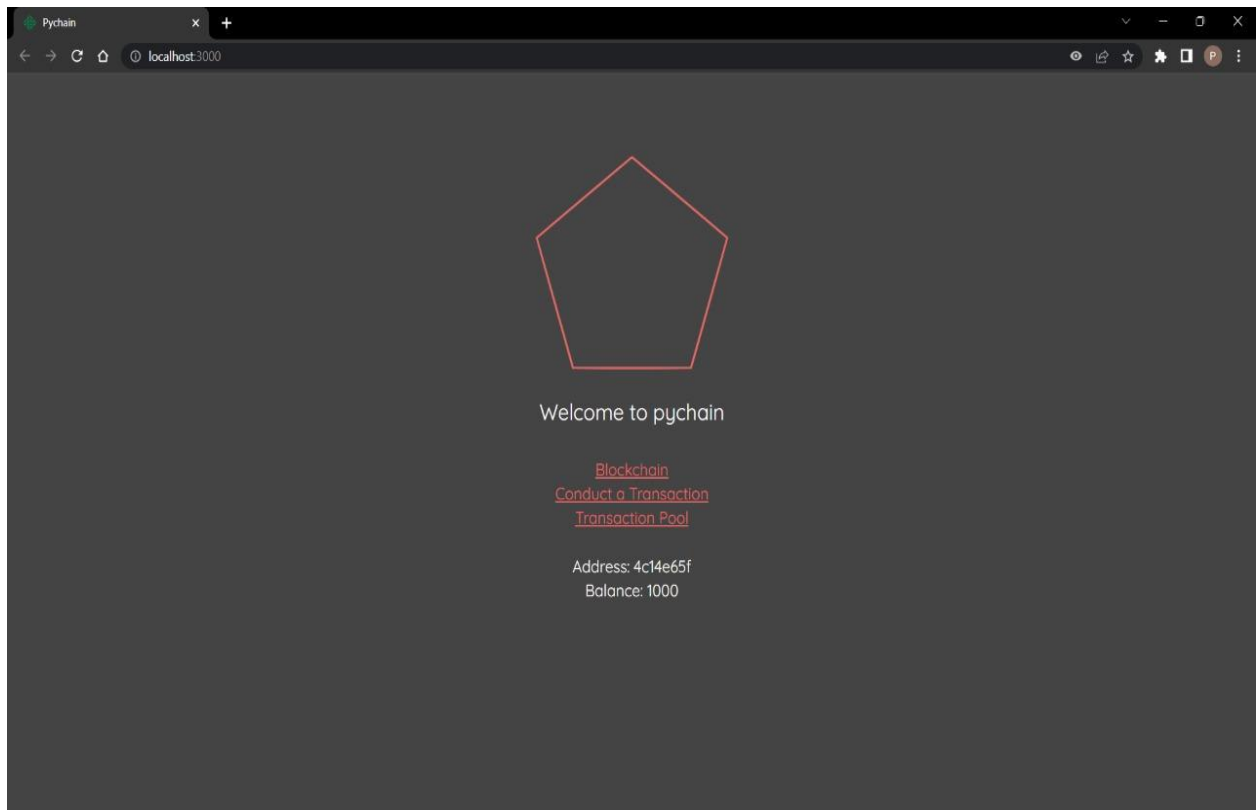
You can now view frontend in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.56.1:3000/

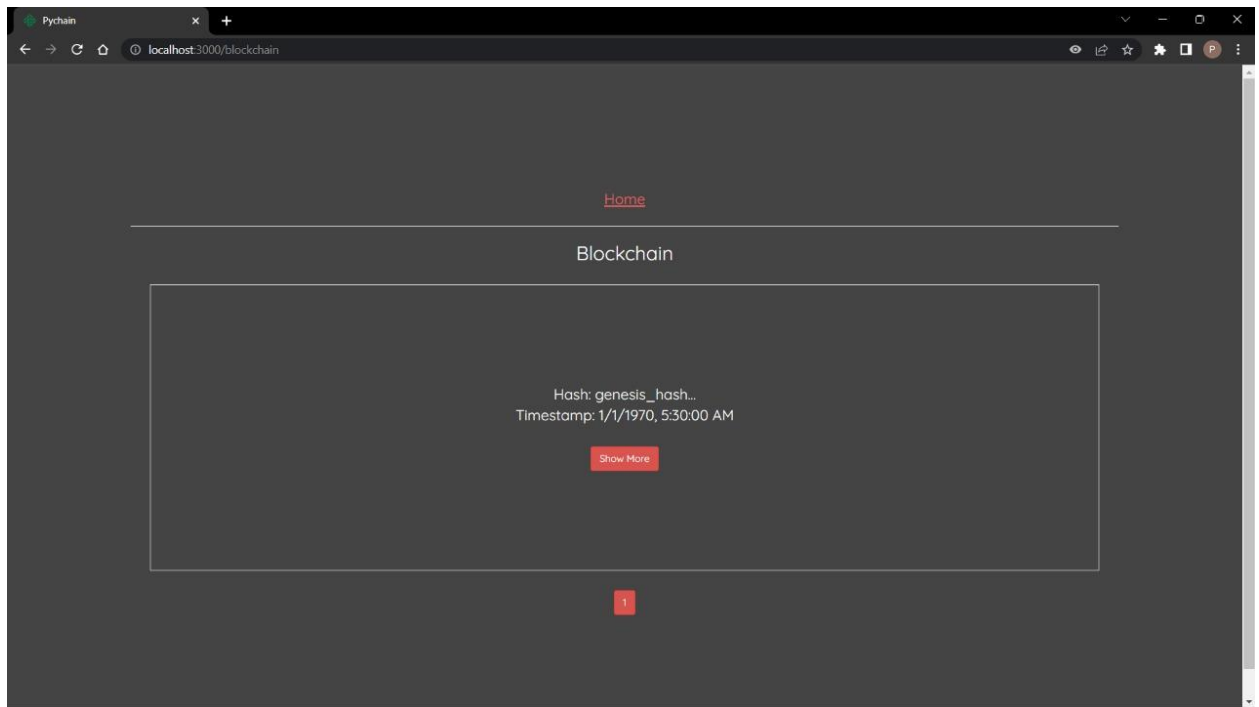
Note that the development build is not optimized.
To create a production build, use npm run build.
```

#### 5. The Frontend View.

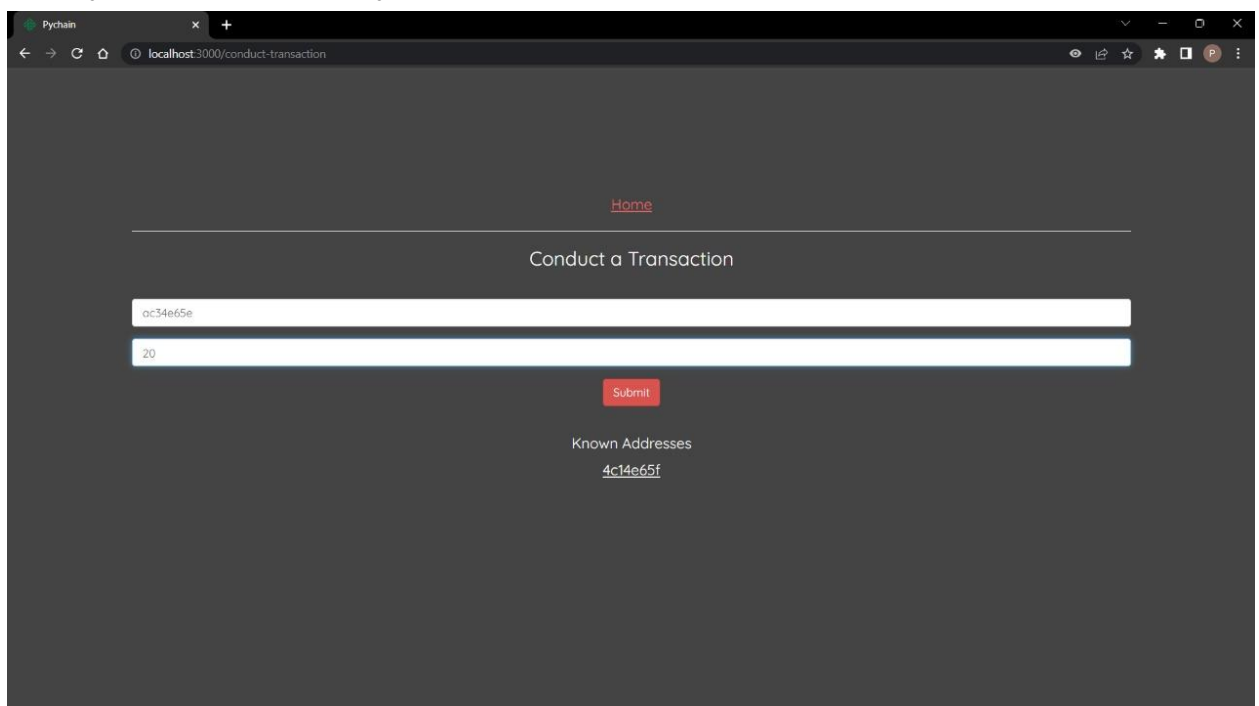
As seen in the Picture Our Frontend Application Users Can View The Blockchain/ Conduct A Transaction and See all the Transaction History Performed By the User.



## 6. Our Blockchain Details without any transactions performed:



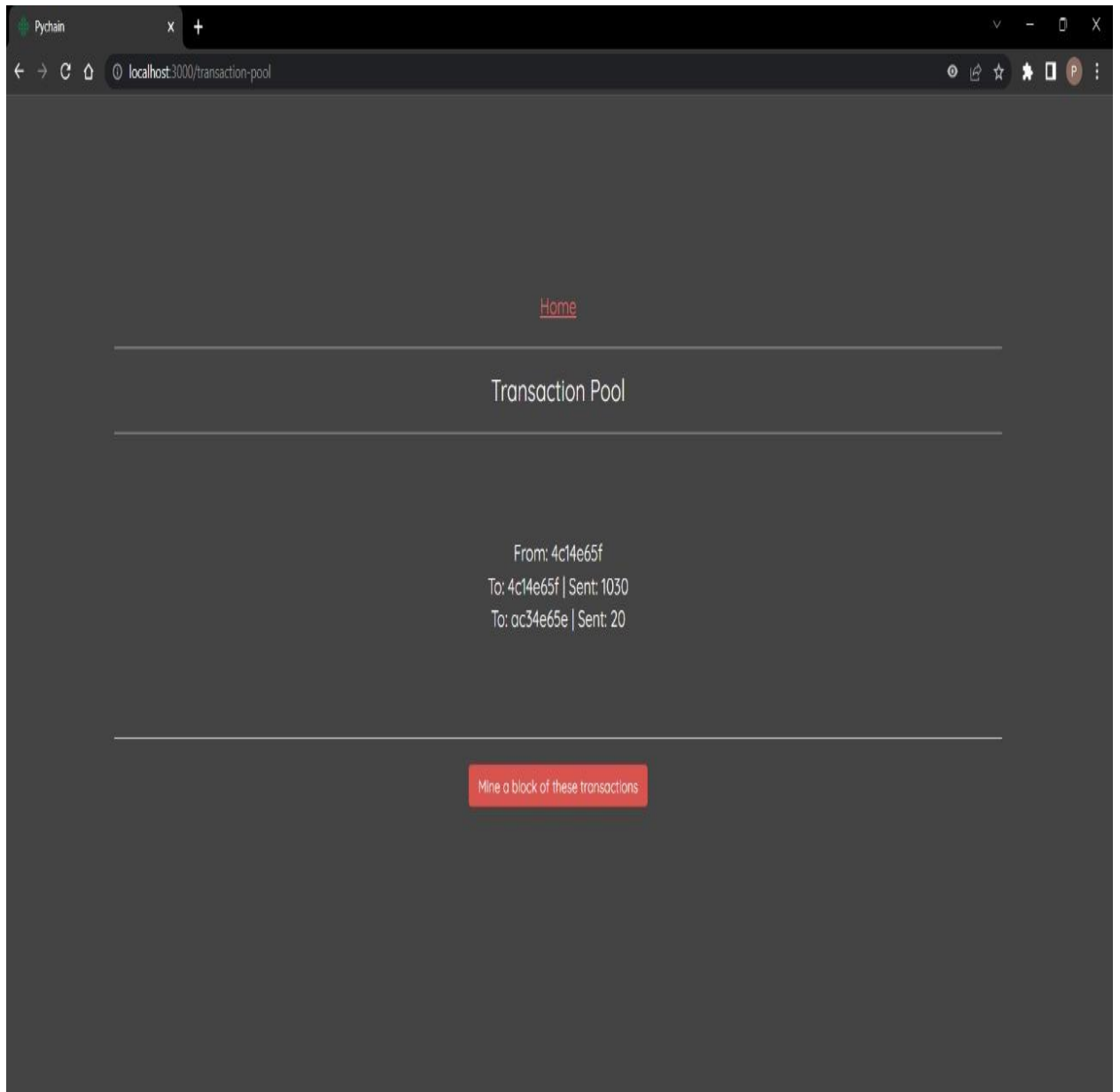
## 7. Performing Transactions to address `ac34c65e` and Sending 20 Pychain Currency to that address





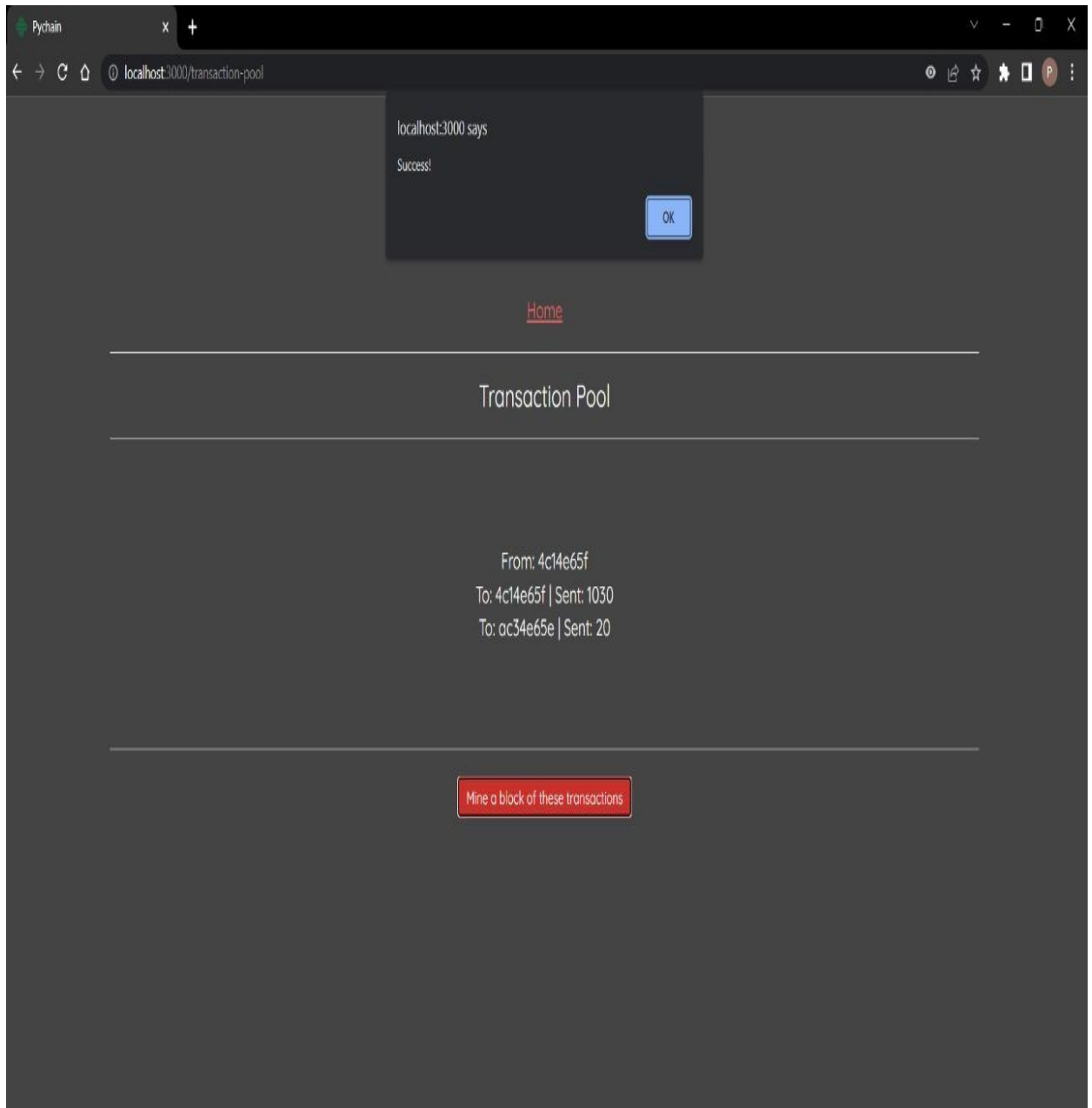
## 8. Transaction Proof:

Here we can see that we have Sent 20 of our Bitcoin to the address which is should in the transaction proof menu of the user login.



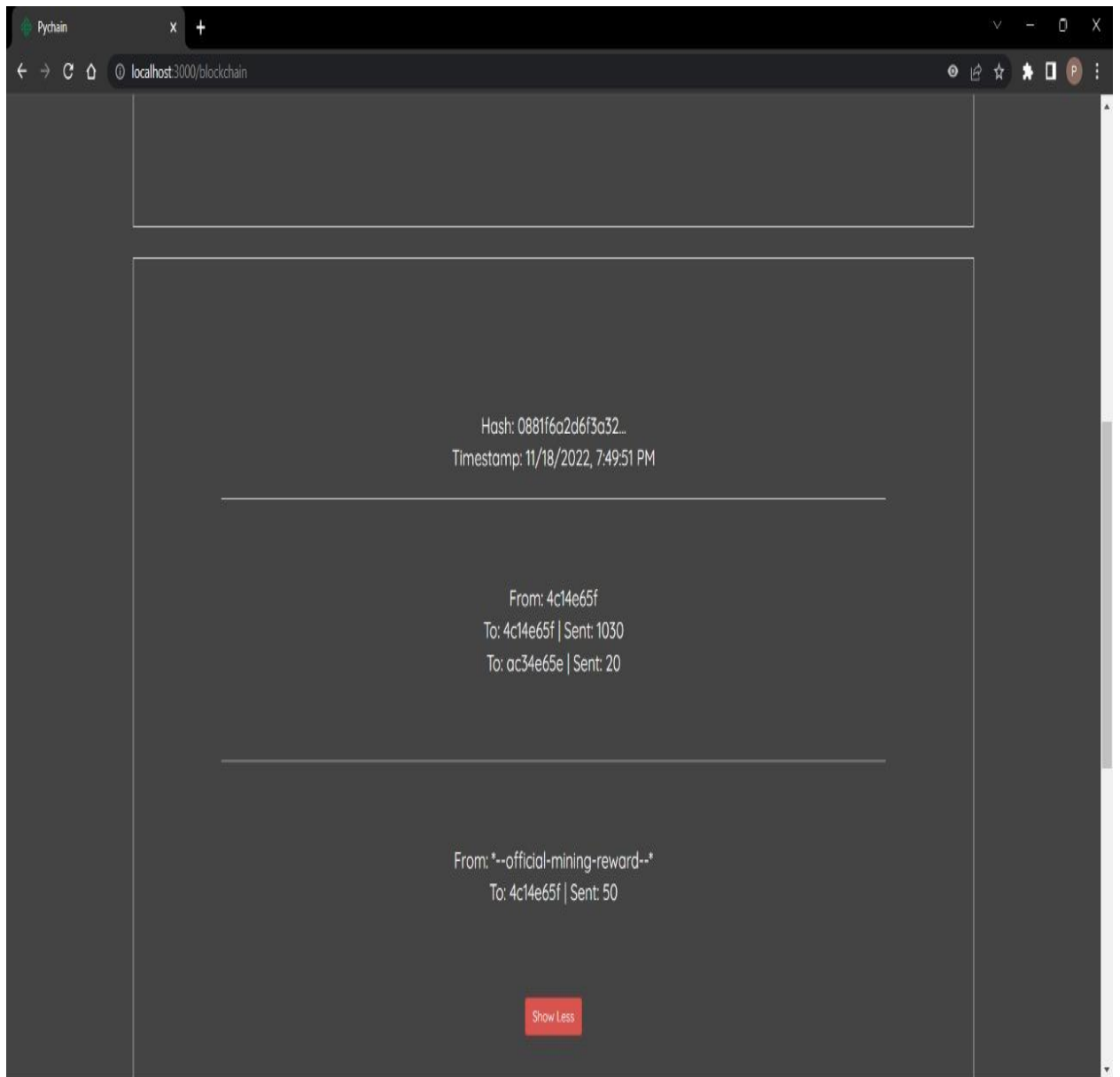
## A MINEABLE ENVIRONMENT FOR EACH TRANSACTIONS:

After every transaction a block is then mined to store all the transaction data as show in the screenshot below.



Generating a reward system after every mine is successfully performed.

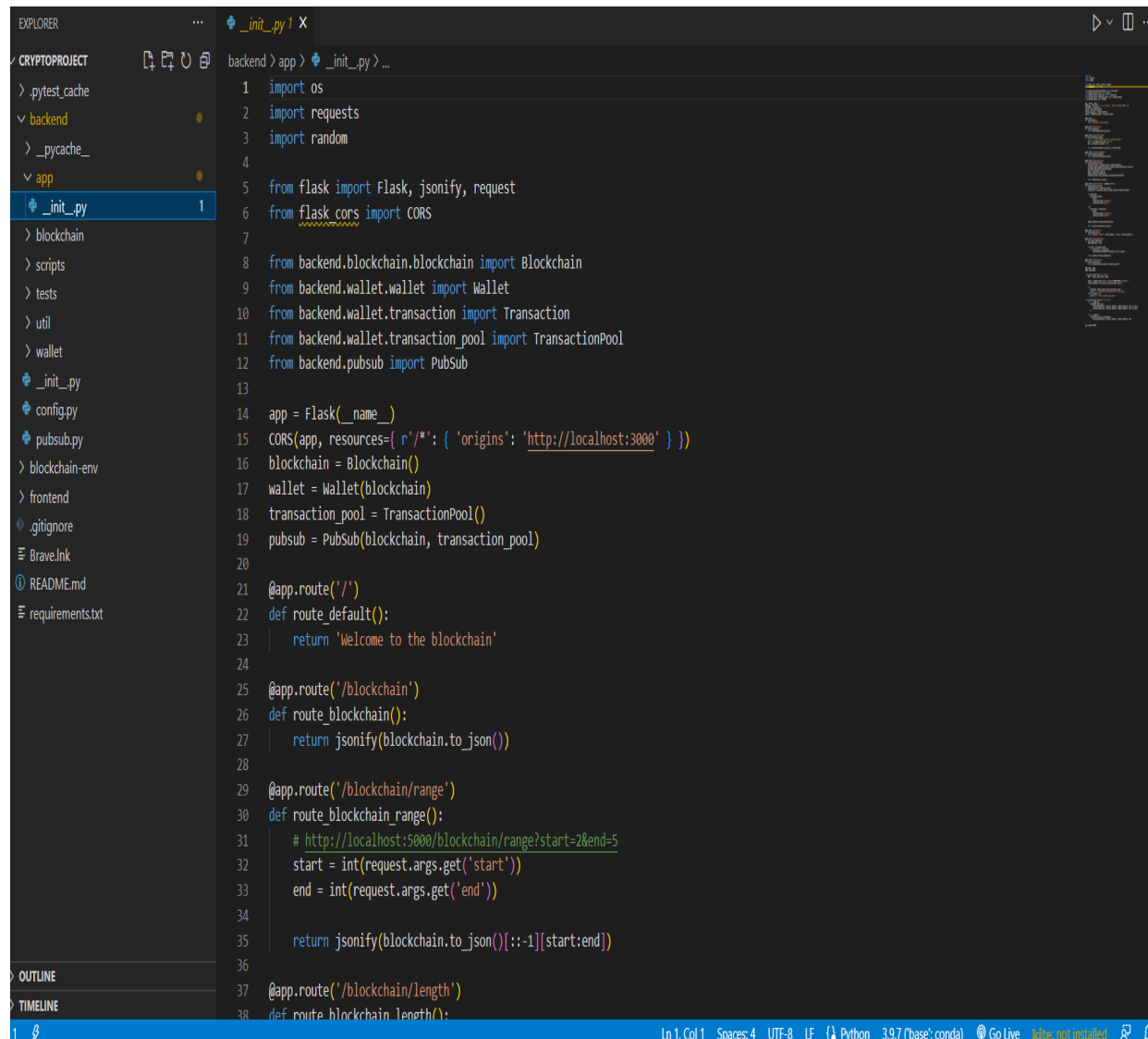
After a user mine a cryptocurrency a reward system is established such that reward is given to increase the traffic of the users.



# CODE SCREENSHOTS:

## 1. BACKEND IMPORTANT CODE SCREENSHOTS:

### a. Backend Application:



The screenshot displays a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like .pytest\_cache, backend, and app, and files like \_init\_.py, config.py, pubsub.py, blockchain-env, frontend, .gitignore, BraveLink, README.md, and requirements.txt. The code editor shows the content of \_init\_.py, which imports various modules and defines the Flask application.

```
1 import os
2 import requests
3 import random
4
5 from flask import Flask, jsonify, request
6 from flask_cors import CORS
7
8 from backend.blockchain.blockchain import Blockchain
9 from backend.wallet.wallet import Wallet
10 from backend.wallet.transaction import Transaction
11 from backend.wallet.transaction_pool import TransactionPool
12 from backend.pubsub import PubSub
13
14 app = Flask(__name__)
15 CORS(app, resources={r'/*': {'origins': 'http://localhost:3000'}})
16 blockchain = Blockchain()
17 wallet = Wallet(blockchain)
18 transaction_pool = TransactionPool()
19 pubsub = PubSub(blockchain, transaction_pool)
20
21 @app.route('/')
22 def route_default():
23     return 'Welcome to the blockchain'
24
25 @app.route('/blockchain')
26 def route_blockchain():
27     return jsonify(blockchain.to_json())
28
29 @app.route('/blockchain/range')
30 def route_blockchain_range():
31     # http://localhost:5000/blockchain/range?start=2&end=5
32     start = int(request.args.get('start'))
33     end = int(request.args.get('end'))
34
35     return jsonify(blockchain.to_json()[::-1][start:end])
36
37 @app.route('/blockchain/length')
38 def route_blockchain_length():
```

```

84         known_addresses.update(transaction['output'].keys())
85
86         return jsonify(list(known_addresses))
87
88     @app.route('/transactions')
89     def route_transactions():
90         return jsonify(transaction_pool.transaction_data())
91
92     ROOT_PORT = 5000
93     PORT = ROOT_PORT
94
95     if os.environ.get('PEER') == 'True':
96         PORT = random.randint(5001, 6000)
97
98         result = requests.get(f'http://localhost:{ROOT_PORT}/blockchain')
99         result_blockchain = Blockchain.from_json(result.json())
100
101         try:
102             blockchain.replace_chain(result_blockchain.chain)
103             print('\n -- Successfully synchronized the local chain')
104         except Exception as e:
105             print(f'\n -- Error synchronizing: {e}')
106
107     if os.environ.get('SEED_DATA') == 'True':
108         for i in range(10):
109             blockchain.add_block([
110                 Transaction(Wallet(), Wallet().address, random.randint(2, 50)).to_json(),
111                 Transaction(Wallet(), Wallet().address, random.randint(2, 50)).to_json()
112             ])
113
114         for i in range(3):
115             transaction_pool.set_transaction(
116                 Transaction(Wallet(), Wallet().address, random.randint(2, 50))
117             )
118
119     app.run(port=PORT)
120

```

## b.Blockchain Establishment.

```

1  from backend.blockchain.block import Block
2  from backend.wallet.transaction import Transaction
3  from backend.wallet.wallet import Wallet
4  from backend.config import MINING_REWARD_INPUT
5
6  class Blockchain:
7      """
8      Blockchain: a public ledger of transactions.
9      Implemented as a list of blocks - data sets of transactions
10     """
11     def __init__(self):
12         self.chain = [Block.genesis()]
13
14     def add_block(self, data):
15         self.chain.append(Block.mine_block(self.chain[-1], data))
16
17     def __repr__(self):
18         return f'Blockchain: {self.chain}'
19
20     def replace_chain(self, chain):
21         """
22         Replace the local chain with the incoming one if the following applies:
23         - The incoming chain is longer than the local one.
24         - The incoming chain is formatted properly.
25         """
26         if len(chain) <= len(self.chain):
27             raise Exception('Cannot replace. The incoming chain must be longer.')
28
29         try:
30             Blockchain.is_valid_chain(chain)
31         except Exception as e:
32             raise Exception(f'Cannot replace. The incoming chain is invalid: {e}')
33
34         self.chain = chain
35
36     def to_json(self):
37         """

```

## Our Cryptocurrency Wallet Code

```
1 import json
2 import uuid
3
4 from backend.config import STARTING_BALANCE
5 from cryptography.hazmat.backends import default_backend
6 from cryptography.hazmat.primitives.asymmetric import ec
7 from cryptography.hazmat.primitives.asymmetric.utils import (
8     encode_dss_signature,
9     decode_dss_signature
10 )
11 from cryptography.hazmat.primitives import hashes, serialization
12 from cryptography.exceptions import InvalidSignature
13
14 class Wallet:
15     """
16     An individual wallet for a miner.
17     Keeps track of the miner's balance.
18     Allows a miner to authorize transactions.
19     """
20
21     def __init__(self, blockchain=None):
22         self.blockchain = blockchain
23         self.address = str(uuid.uuid4())[0:8]
24         self.private_key = ec.generate_private_key(
25             ec.SECP256K1(),
26             default_backend()
27         )
28         self.public_key = self.private_key.public_key()
29         self.serialize_public_key()
30
31     @property
32     def balance(self):
33         return Wallet.calculate_balance(self.blockchain, self.address)
34
35     def sign(self, data):
36         """
37         Generate a signature based on the data using the local private key.
38         """
```

# Transaction Pool Check:

```
class TransactionPool:
    def __init__(self):
        self.transaction_map = {}

    def set_transaction(self, transaction):
        """
        Set a transaction in the transaction pool.
        """
        self.transaction_map[transaction.id] = transaction

    def existing_transaction(self, address):
        """
        Find a transaction generated by the address in the transaction pool
        """
        for transaction in self.transaction_map.values():
            if transaction.input['address'] == address:
                return transaction

    def transaction_data(self):
        """
        Return the transactions of thje transaction pool represented in their
        json serialized form.
        """
        return list(map(
            lambda transaction: transaction.to_json(),
            self.transaction_map.values()
        ))

    def clear_blockchain_transactions(self, blockchain):
        """
        Delete blockchain recorded transactions from the transaction pool.
        """
        for block in blockchain.chain:
            for transaction in block.data:
                try:
                    del self.transaction_map[transaction['id']]
```

# TRANSACTIONS IN THE BLOCKCHAIN:

```
import time
import uuid

from backend.wallet.wallet import Wallet
from backend.config import MINING_REWARD, MINING_REWARD_INPUT

class Transaction:
    """
    Document of an exchange in currency from a sender to one
    or more recipients.
    """
    def __init__(
        self,
        sender_wallet=None,
        recipient=None,
        amount=None,
        id=None,
        output=None,
        input=None
    ):
        self.id = id or str(uuid.uuid4())[0:8]
        self.output = output or self.create_output(
            sender_wallet,
            recipient,
            amount
        )
        self.input = input or self.create_input(sender_wallet, self.output)

    def create_output(self, sender_wallet, recipient, amount):
        """
        Structure the output data for the transaction.
        """
        if amount > sender_wallet.balance:
            raise Exception('Amount exceeds balance')

        output = {}
        output[recipient] = amount
        output[sender_wallet.address] = sender_wallet.balance - amount
```

# HASHING CODES FOR BLOCKS:

```
1 import hashlib
2 import json
3
4 def crypto_hash(*args):
5     """
6     Return a sha-256 hash of the given arguments.
7     """
8     stringified_args = sorted(map(lambda data: json.dumps(data), args))
9     joined_data = ''.join(stringified_args)
10
11     return hashlib.sha256(joined_data.encode('utf-8')).hexdigest()
12
13 def main():
14     print(f"crypto_hash('one', 2, [3]): {crypto_hash('one', 2, [3])}")
15     print(f"crypto_hash(2, 'one', [3]): {crypto_hash(2, 'one', [3])}")
16
17 if __name__ == '__main__':
18     main()
19
```



## 2. FRONTEND CODE:

### HTML THAT MAKES THE USE OF SELF CREATED GOOGLE TIES SHEETS.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Quicksand">
5     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dG
6     <title>Pychain</title>
7   </head>
8   <body>
9     <div id="root"></div>
10  </body>
11 </html>
12
```

## CSS DESIGN FOR THE CODE:

```
body {
  background-color: #444;
  color: #fff;
  text-align: center;
  font-size: 18px;
  font-family: 'Quicksand';
  padding-top: 5%;
  word-wrap: break-word;
}

.logo {
  width: 250px;
  height: 250px;
}

.App {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.WalletInfo {
  width: 500px;
}

.Block {
  border: 1px solid #fff;
  padding: 10%;
  margin: 2%;
}

.Transaction {
  padding: 5%;
}

.Blockchain, .ConductTransaction, .TransactionPool {
```

## JAVASCRIPT CODE:

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import { Router, Switch, Route } from 'react-router-dom';
4  import './index.css';
5  import history from './history';
6  import App from './components/App';
7  import Blockchain from './components/Blockchain';
8  import ConductTransaction from './components/ConductTransaction';
9  import TransactionPool from './components/TransactionPool';
10
11  ReactDOM.render(
12    <Router history={history}>
13      <Switch>
14        <Route path="/" exact component={App} />
15        <Route path="/blockchain" component={Blockchain} />
16        <Route path="/conduct-transaction" component={ConductTransaction} />
17        <Route path="/transaction-pool" component={TransactionPool} />
18      </Switch>
19    </Router>,
20    document.getElementById('root')
21  );
22
```

## **Conclusion and Limitations:**

The synchronization feature of our project is dependent on the root node as it depends on an instance of the application running a port three thousand. However that root node may be down but other nodes will still be running in the system. So if a new peer joins it should request a block chain from one of those other peers rather than always assuming that it can make a successful request to an either running or not running root node.

One way to approach this issue would be to broadcast your own address on startup. If you are a peer then any receiving node could send you this block chain data with POST request as a blocking route. This request might have to be sent in chunks.

## **Major Aspect to Improve:**

An aspect to improve is catching up the block chain if it ever falls behind. If a node rejects too many incoming blocks in a row there may be a situation where its blocking has fallen behind the block chain data that most data that most nodes are using So some kind of pulling block chain that checks the length of neighboring block chains can help determine if your local chain needs to catch up and validate those longer chains. More API endpoints can also be added which will allow us to read more information about the system. Another idea is to

create a separate type of application instance for non miners who aren't interested in mining but could still have their own wallet and would be able to invest actual money though into the system to get currency. They could also exchange money with a miner who already has currency in the system.

There's also an unlimited number of front end improvements around the functionality and styling to make the react application more slick or customized according to your own design.

