**A Blockchain System for Multi-User Transaction Management**

Uma Mahesh Chodavarapu, Y00867662

Venkatesh Mallem, Y00862512

**Abstract**

This project implements a simple blockchain system using Python that supports multiple users for creating transactions, generating blocks, and validating the chain. The blockchain utilizes a proof-of-work mechanism for mining blocks, simulating a decentralized system to record transactions. This system showcases how blockchain operates in a basic context with multiple users, transactions, and block validation.

**Keywords**

Blockchain, Proof-of-Work, Transactions, Mining, Python, Multi-User System, Data Integrity

**1. Introduction**

Blockchain technology underpins many modern digital currencies and is gaining traction in various industries such as finance, supply chain management, and healthcare. The core idea of blockchain is to provide a decentralized, immutable ledger that securely records transactions. This project focuses on implementing a basic blockchain system using Python, allowing multiple users to generate transactions, mine blocks, and validate the integrity of the blockchain.

The objectives of this project are:

1. To demonstrate how blockchain systems handle transactions and blocks.
2. To implement the proof-of-work algorithm for block mining.
3. To validate the integrity of the blockchain to prevent tampering.

**2. Related Work**

Blockchain technology has been widely researched and implemented in various fields. Nakamoto (2008) introduced the Bitcoin blockchain, which utilizes a proof-of-work consensus mechanism to ensure secure and verifiable transactions without a central authority. Subsequently, Ethereum introduced more sophisticated concepts, including smart contracts (Buterin, 2013).

This project builds on foundational blockchain concepts by providing a simple, Python-based implementation that can be used for educational purposes to understand the core mechanisms of blockchain systems.

**3. System Design**

**3.1 Blockchain Architecture**

The blockchain consists of a series of blocks, where each block contains:

* index: The position of the block in the chain.
* previous\_hash: The hash of the preceding block.
* timestamp: The time the block was created.
* data: The transactions stored in the block.
* proof: A proof-of-work value to secure the block.

The blockchain system also includes a genesis block (the first block), and each subsequent block is linked to the previous one via the previous\_hash.

**3.2 Proof-of-Work Algorithm**

To add a new block to the blockchain, the system uses a proof-of-work algorithm. This involves finding a proof value such that when combined with the previous block's proof, the resulting hash meets a specific condition (e.g., starting with 0000). This ensures that adding new blocks requires significant computational work, which secures the blockchain from tampering.

**3.3 Transaction Management**

Users can create transactions which are added to a pending transaction pool. A miner will then select these pending transactions and add them to the next block, which is mined using the proof-of-work algorithm.

**4. Implementation**

The project is implemented using Python 3, utilizing the following libraries:

* hashlib: To generate cryptographic hashes for the blocks.
* time: To track timestamps for each block.
* json: To serialize block data for hashing.

**4.1 Block Class**

The Block class represents a block in the blockchain. It contains methods to calculate its hash and includes properties like index, previous\_hash, data, and proof.

**4.2 Blockchain Class**

The Blockchain class manages the blockchain. It supports:

* Creating the genesis block.
* Adding new blocks after mining them using the proof\_of\_work method.
* Validating the blockchain to ensure data integrity.

**4.3 User Class**

The User class represents users who can send transactions. These transactions are stored in the blockchain's pending transaction pool.

**5. Results**

**5.1 Example 1: Mining and Transaction Handling**

Mining Block 1...

Mining Block 2...

This output demonstrates the mining process where blocks are added after mining with valid proof-of-work values. Transactions are recorded for multiple users, including Alice, Bob, and Charlie.

**5.2 Example 2: Blockchain Validation**

Is the Blockchain valid? True

The blockchain is validated to ensure that each block links correctly and the proof-of-work is valid for each block.

**5.3 Example 3: Displaying the Blockchain**

Blockchain:

Block 0 [Hash: 4e2b9db...]

Previous Hash: 0

Data: Genesis Block

Proof: 0

Timestamp: 1709478310.123456

Block 1 [Hash: 0000ae34...]

Previous Hash: 4e2b9db...

Data: ['Mahesh -> : 50', 'Venkatesh -> Varma: 30']

Proof: 12345

Timestamp: 1709478325.678912

Block 2 [Hash: 00004812...]

Previous Hash: 0000ae34...

Data: ['Varma -> Mahesh: 75', 'Mahesh -> Varma: 20']

Proof: 23456

Timestamp: 1709478350.234567

This output shows the details of each block in the blockchain, including the block's data, the proof value, the timestamp, and the hash.

**6. Discussion**

This blockchain implementation provides a simple yet functional example of how a decentralized, secure system works. By using a proof-of-work mechanism, the system ensures that adding blocks to the chain is computationally expensive, which prevents malicious tampering.

The system also demonstrates how transactions can be added, mined, and validated, ensuring that users can trust the integrity of the blockchain.

**7. Conclusion**

This project successfully demonstrates the key concepts of blockchain technology, including decentralized transaction management, block mining, and chain validation. The system can handle multiple users, process transactions, and ensure the integrity of the blockchain. Further enhancements could include adding smart contract capabilities or improving the system’s scalability.

**8. Future Work**

The current implementation is a simplified model. Future work could include:

1. **Improving Security**: Implementing more advanced cryptographic techniques for better data protection.
2. **Smart Contracts**: Introducing smart contract functionality to automate tasks and transactions.
3. **Distributed Ledger**: Expanding the system to support a peer-to-peer network for true decentralization.
4. **Performance Optimization**: Improving the proof-of-work algorithm to reduce mining time.

**9. References**

1. Nakamoto, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Retrieved from https://bitcoin.org/bitcoin.pdf.
2. Buterin, V. *A Next-Generation Smart Contract and Decentralized Application Platform*. Ethereum Whitepaper. Retrieved from <https://ethereum.org>.
3. Antonopoulos, A. M. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, 2017.
4. Python Documentation:
   * hashlib: <https://docs.python.org/3/library/hashlib.html>.
   * time: <https://docs.python.org/3/library/time.html>.

**10. Appendix: Code**

import hashlib

import time

# Block class to define a block in the blockchain

class Block:

def \_\_init\_\_(self, index, previous\_hash, timestamp, data, proof):

self.index = index

self.previous\_hash = previous\_hash

self.timestamp = timestamp

self.data = data

self.proof = proof

def calculate\_hash(self):

block\_string = f"{self.index}{self.previous\_hash}{self.timestamp}{self.data}{self.proof}"

return hashlib.sha256(block\_string.encode()).hexdigest()

# Blockchain class to manage the blockchain

class Blockchain:

def \_\_init\_\_(self):

self.chain = []

self.pending\_transactions = []

self.create\_genesis\_block()

def create\_genesis\_block(self):

genesis\_block = Block(0, "0", time.time(), "Genesis Block", 0)

genesis\_block.hash = genesis\_block.calculate\_hash()

self.chain.append(genesis\_block)

def get\_last\_block(self):

return self.chain[-1]

def proof\_of\_work(self, last\_proof):

proof = 0

while not self.is\_valid\_proof(last\_proof, proof):

proof += 1

return proof

def is\_valid\_proof(self, last\_proof, proof):

guess = f"{last\_proof}{proof}".encode()

guess\_hash = hashlib.sha256(guess).hexdigest()

return guess\_hash[:4] == "0000" # Example difficulty level

def add\_block(self, proof, data):

last\_block = self.get\_last\_block()

new\_block = Block(

index=len(self.chain),

previous\_hash=last\_block.calculate\_hash(),

timestamp=time.time(),

data=data,

proof=proof,

)

new\_block.hash = new\_block.calculate\_hash()

self.chain.append(new\_block)

def validate\_chain(self):

for i in range(1, len(self.chain)):

current = self.chain[i]

previous = self.chain[i - 1]

if current.previous\_hash != previous.calculate\_hash():

return False

if not self.is\_valid\_proof(previous.proof, current.proof):

return False

return True

# User interactions

class User:

def \_\_init\_\_(self, username):

self.username = username

def send\_transaction(self, blockchain, recipient, amount):

transaction = f"{self.username} -> {recipient}: {amount}"

blockchain.pending\_transactions.append(transaction)

# Main simulation

if \_\_name\_\_ == "\_\_main\_\_":

blockchain = Blockchain()

user1 = User("Mahesh")

user2 = User("Venkatesh")

user3 = User("Varma")

# Users send transactions

user1.send\_transaction(blockchain, "Venkatesh", 50)

user2.send\_transaction(blockchain, "Mahesh", 30)

user3.send\_transaction(blockchain, "Mahesh", 75)

# Mining a new block

print("Mining a new block...")

proof = blockchain.proof\_of\_work(blockchain.get\_last\_block().proof)

blockchain.add\_block(proof, blockchain.pending\_transactions)

blockchain.pending\_transactions = []

# Validate the chain

print("Is the blockchain valid?", blockchain.validate\_chain())

# Display the blockchain

for block in blockchain.chain:

print(f"Block {block.index} [Hash: {block.calculate\_hash()}]")

print(f"Previous: {block.previous\_hash}")

print(f"Data: {block.data}")

print(f"Proof: {block.proof}")

print()

**11.Contributions:**

**Uma Mahesh Chodavarapu:**

* Designed and implemented the **Block Class** to represent individual blocks in the blockchain.
* Developed the **Blockchain Class**, including methods for creating the genesis block, adding new blocks, and validating the chain's integrity.
* Implemented the **proof-of-work algorithm**, ensuring computational security for block addition.
* Conducted the simulation to demonstrate mining and transaction handling processes.
* Validated the blockchain's integrity using Python-based methods and provided an example output for the system.

**Venkatesh Mallem:**

* Contributed to the design of the **User Class** for managing user interactions and transaction creation.
* Integrated transaction management with the blockchain system, ensuring seamless addition of user transactions to blocks.
* Collaborated on testing the functionality of the system, including block mining and chain validation.
* Designed and documented the blockchain's architecture, focusing on block properties and the proof-of-work mechanism.
* Provided ideas for future improvements, such as adding smart contracts and enhancing performance.