Name : Irsa Roshan

Roll No : 14651

Subject : Data Structure And Algorithm

Submitted To : Sir Jamal Abdul Ahad

Date : 26 Dec 2024

Project Name : Blockchain Simulation

Blockchain Simulation Using Linked List:

Overview:

This project demonstrates the core concepts of blockchain using a simple linked list structure..Each “block” contains data, a timestamp, and a hash of the previous block to ensure immutability..This blockchain simulation is implemented in Python.

Introduction:

This project simulates a blockchain using a linked list data structure. Each block in the blockchain contains:

- An index (indicating its position in the chain).
- A timestamp to record the block’s creation time.
- Data, representing transactions or information stored in the block.
- A hash, ensuring the block’s integrity.
- A reference to the previous block’s hash, linking blocks together.

The primary purpose of this simulation is to demonstrate how blockchain ensure data immutability and consistency using cryptographic hashing.

Features:

1. Creation of a genesis block.
2. Adding new blocks to the chain.
3. Validating the integrity of the blockchain.
4. Printing the blockchain structure.

Requirements:

- Python 3.x

- No external libraries required.

Objective:

To simulate a blockchain's functionality using a linked list.

- To illustrate how blocks are chained together using cryptographic hashes.
- To validate the blockchain's integrity by detecting tampered data.

Tools And Environment:

- **Programming Language:**

Python (chosen for simplicity and readability).

- **IDE:**

Visual Studio Code or PyCharm.

- **Library Used:**

hashlib for hashing.

- **System Requirements:**

Any machine capable of running Python 3.7 or above.

Setup Instructions:

1. Install Python (if not already installed) from Python.org.
2. Open the IDE and create a new Python file (e.g., blockchain_simulation.py).
3. Copy and paste the implementation code provided in the Implementation Details section.
4. Run the script to simulate the blockchain.

Network Architecture:

In this simulation:

Each block is represented as a node in the linked list.

A block contains:

1. **Index:**

Position of the block in the chain.

2. **Timestamp:**

When the block was created.

3. **Data:**

Information stored in the block.

4. **Previous_hash:**

Hash of the previous block.

5. **Current_hash:**

Hash of the current block (calculated based on index, timestamp, data, and previous_hash).

Illustration:

[Block 0] -> [Block 1] -> [Block 2] -> [Block 3]

Each block links to the previous block via previous_hash.

Implementation:

Code Implementing:

```
import hashlib

import datetime

class Block:

    def __init__(self, index, timestamp, data, previous_hash):

        self.index = index

        self.timestamp = timestamp

        self.data = data

        self.previous_hash = previous_hash

        self.current_hash = self.calculate_hash()

    def calculate_hash(self):

        # Create a SHA-256 hash of the block's contents

        block_content = f"{self.index}{self.timestamp}{self.data}{self.previous_hash}"

        return hashlib.sha256(block_content.encode()).hexdigest()
```

```
class Blockchain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]
    def create_genesis_block(self):
        # First block in the chain (index 0)
        return Block(0, str(datetime.datetime.now()), "Genesis Block", "0")
    def add_block(self, data):
        # Add a new block to the chain
        previous_block = self.chain[-1]
        new_block = Block(len(self.chain), str(datetime.datetime.now()), data, 
previous_block.current_hash)
        self.chain.append(new_block)
    def is_chain_valid(self):
        # Validate the blockchain's integrity
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i - 1]
            # Check if current block's hash is valid
            if current_block.current_hash != current_block.calculate_hash():
                return False
            # Check if current block's previous hash matches the previous block's hash
            if current_block.previous_hash != previous_block.current_hash:
                return False
        return True
# Simulation
blockchain = Blockchain()
```

```
blockchain.add_block("Block 1 Data")

blockchain.add_block("Block 2 Data")

blockchain.add_block("Block 3 Data")

# Display blockchain

for block in blockchain.chain:

    print(f"Index: {block.index}")

    print(f"Timestamp: {block.timestamp}")

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

    print(f"Previous Hash: {block.previous_hash}")

    print(f"Current Hash: {block.current_hash}\n")

# Validate the blockchain

print("Is blockchain valid?", blockchain.is_chain_valid())
```

Test Cases And Results:

Test Case 1:

- **Adding Blocks**

Input: Add data (“Block 1 Data”, “Block 2 Data”, etc.)

Output: Blocks added successfully, and each block is linked to the previous one.

- **Test Case 2: Validating Blockchain**

Input: Run is_chain_valid() after adding blocks.

Output: Returns True (blockchain is valid).

- **Test Case 3: Tampering Data**

Input: Manually change a block’s data and re-run is_chain_valid().

Output: Returns False (blockchain is invalid).

- **Results:**

Screenshot or log of block details and validation output:

Index: 0

Timestamp: 2024-12-25 20:00:00

Data: Genesis Block

Previous Hash: 0

Current Hash: abcd1234...

Index: 1

Timestamp: 2024-12-25 20:01:00

Data: Block 1 Data

Previous Hash: abcd1234...

Current Hash: efgh5678...

Is blockchain valid? True

Conclusion:

The project successfully simulates the basic structure and functionality of a blockchain using a linked list. It demonstrates how blocks are linked using cryptographic hashes and ensures data integrity. Future work could include implementing advanced features like proof-of-work, consensus algorithms, and peer-to-peer networking.

Reference:

1. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
2. Python hashlib documentation
3. TutorialsPoint: Data Structures – Linked List Basics.