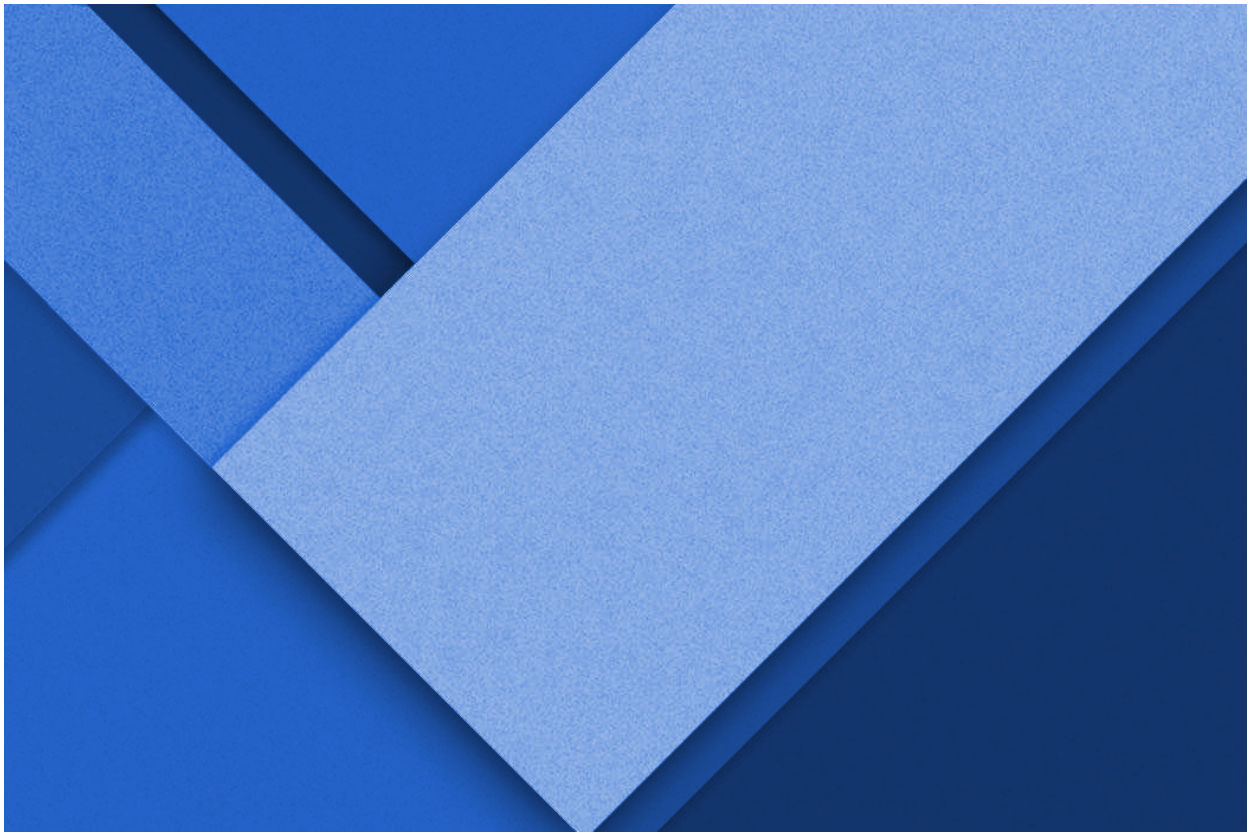

BITCOIN

Project Report

Link to our repository - [DSA-Team18-Bitcoin]

Team 18

Overview

In this project, we simulate the working of Bitcoin and do various operations.

Description

In this Bitcoin system, transactions are stored in data blocks in the following way - Each Block contains 50 transactions, and it is structured as follows. Assuming PreviousBlockHash of the first block is 0.

For calculating the PreviousBlockHash, we designed our Hash Function, which accepts BlockNumber, Transactions, PreviousBlockHash and Nonce of the Previous block. We store the user details using a hashtable. We store details like Unique ID (uID), Wallet Balance, Transaction History, Join Date and Time.

Functions

1. **Add user to the system (**`addUser`**)** - It assigns a randomly generated unique ID and adds a new user to store the initial details.
2. **Transact(**`transact`**)** - Takes sender and receiver's user ID and the amount to be transacted as input, verifies if the requested transaction is valid, and transfers the amount from sender to receiver and makes few necessary changes to the user details.
3. **Create block (**`createBlock`**)** - If the number of transactions crosses 50, it creates a new block and adds the block to the chain of blocks.
4. **Attack(**`attack`**)** - Randomly generates a number from 1-50, and if such block number exists, it modifies its nonce; if such block number does not exist, the attack fails.
5. **Validate Block-Chain(**`validate`**)** - Checks if the previous block hash of each block is valid or not. If it is not valid,it means an attack has occured. In such a case, it modifies the nonce of the attacked block to its original value.

6. **View transaction history(**`inquire_transactions`**)** - Takes a user ID, and prints the user's transaction history.
7. **View Wallet Balance(**`inqure_bal`**)** - Takes a user ID and prints the user's current wallet balance.
8. **View BlockChain(**`displayBlockChain`**)** - Displays the block number, nonce and previous block hasof every block in the Blockchain
9. **Display Block by Block number(**`displayBlock`**)** - Takes block number as argument and prints the Nonce, previous block hash, and transactions of that block.

Data Structures used

I. Linked List

A linked list is a linear data structure without the elements being allocated contiguously. Instead, the nodes are linked to the next node with the help of a pointer. The advantage of using a linked list over an array is that the number of nodes does not have to be constant, and the program keeps making new nodes whenever needed, saving memory. However, the downside to this is that a node has to be accessed serially instead of directly.

In our Project

The blockchain is a kind of a linked list with each node storing the details of a given block.

II. Hash table

A hash table is a data structure type that stores the information as key-value pairs. The key is sent to a hash function that performs arithmetic operations on it. The result of the hash function is the index of the key-value pair in the hash table. The advantage of a hash table over an array is the speed of access.

A possible downside to hash tables is the existence of collisions. In our project, that has been dealt with by using separate-chaining to accommodate multiple users being mapped to the same key.

In our Project

User objects like Unique ID, Wallet Balance, Transaction History, Join Date and Time are stored in a Hash table using the Separate-Chaining method.

Algorithm used - SHA1 algorithm

We used this algorithm to create a hash from block number, transactions and previous block after converting it to a string. The first two elements of the hash were then XORed with the nonce.

Complexity of algorithm

This algorithm takes constant time as the string length is a constant, and SHA1 has a complexity O(n), n being the length of the string. The longest part of the string is the transactions, and if that were variable, rather than being 50 for each block, the complexity would have been O(N), N being the number of transactions in the block.

Division of work (in chronological order)

I. Dheeraja made the ADT for the data types that we used.
II. Mohammed made a basic template for the UI.
III. Praneetha made the AddUser function.
IV. Mohammed made a Transact function, made an array to store users, added balance/transaction history inquiry, and cleaned the UI.
V. Dheeraja made the hashing process and nonce.
VI. Dheeraja made the CreateBlock function.
VII. Praneetha converted the array to store users into a hash table and added hash functions, and made some required changes.
VIII. Dheeraja bridged the changes Praneetha made to make the code functional.
IX. Dheeraja tested the code and fixed bugs.