**Practical – 4**

**Aim: Write a program to create a block class which can hold 3 transactions. Also create a genesis block**

**Code:**

***lib.rs***

pub mod client;

pub mod transaction;

pub mod block;

***main.rs***

use simple\_blockchain\_rs::client::Client;

use simple\_blockchain\_rs::transaction::Transaction;

use simple\_blockchain\_rs::block::Block;

fn main() {

let utsav = Client::new();

let bhupendra = Client::new();

let jash = Client::new();

let mut block0 = Block::genesis\_block(&utsav);

println!("Genesis Block: {:#?}", block0);

println!("utsav public key: {}", utsav.identify());

println!("bhupendra public key: {}", bhupendra.identify());

println!("jash public key: {}", jash.identify());

println!("");

let mut transaction1 = Transaction::new(utsav.public\_key, bhupendra.public\_key, 10.0, None);

transaction1.sign\_transaction(&utsav);

println!(

"Transaction 1 signature validation: {:#?}",

transaction1.is\_valid\_transaction()

);

block0.verified\_transactions.push(transaction1);

let mut transaction2 = Transaction::new(bhupendra.public\_key, jash.public\_key, 10.0, None);

transaction2.sign\_transaction(&bhupendra);

println!(

"Transaction 2 signature validation: {:#?}",

transaction2.is\_valid\_transaction()

);

block0.verified\_transactions.push(transaction2);

let mut transaction3 = Transaction::new(jash.public\_key, utsav.public\_key, 10.0, None);

transaction3.sign\_transaction(&jash);

println!(

"Transaction 3 signature validation: {:#?}",

transaction3.is\_valid\_transaction()

);

block0.verified\_transactions.push(transaction3);

println!("Block 0: {:#?}", block0);

}

***block.rs***

use super::transaction::\*;

use crate::client::\*;

/// A block in the blockchain.

///

/// `index` contains the index of the block.

/// `nonce` contains the nonce of the block that is used to find a valid hash.

/// `previous\_hash` contains the hash of the previous block.

/// `hash` contains the hash of the block.

/// `verified\_transactions` contains the transactions that are verified in the block.

#[derive(Debug)]

pub struct Block {

pub index: u64,

pub nonce: u64,

pub previous\_hash: Option<String>,

pub hash: String,

pub verified\_transactions: Vec<Transaction>,

}

impl Block {

/// This method creates a new block.

pub fn new(index: u64, previous\_hash: Option<String>) -> Self {

Self {

index,

nonce: 0u64,

previous\_hash,

hash: String::new(),

verified\_transactions: vec],

}

}

/// This method generates genesis block.

pub fn genesis\_block(receiver: &Client) -> Self {

let genesis = Client::new();

let initial\_transaction =

Transaction::new(genesis.public\_key, receiver.public\_key, 1000.0, None);

let mut genesis\_block = Block::new(0, None);

genesis\_block

.verified\_transactions

.push(initial\_transaction);

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

genesis\_block

}

/// This method verifies the transactions in the block.

pub fn has\_valid\_transactions(&self) -> bool {

for tran in &self.verified\_transactions {

if !tran.is\_valid\_transaction() {

return false;

}

}

return true;

}

/// This method serializes the block into a string.

pub fn serialize(&self) -> String {

let transactions = self

.verified\_transactions

.iter()

.fold(String::new(), |acc, x| acc + &x.serialize());

format!(

"{}{}{}{}",

self.index,

self.nonce,

self.previous\_hash.as\_ref().unwrap\_or(&String::new()),

transactions

)

}

/// This method calculates the hash of the block using SHA256.

pub fn calculate\_hash(&self) -> String {

crypto\_hash::hex\_digest(crypto\_hash::Algorithm::SHA256, &self.serialize().as\_bytes())

}

}

***client.rs from practical - 1***

***transaction.rs from practical – 2***

**Genesis Block**

**![A picture containing table

Description automatically generated**

**Public keys of Utsav, Bhupendra, Jash**

****

**Transactions between the above three clients inside the block**

****

**Verified transactions inside the block**

**Table

Description automatically generated**